Czech Technical University in Prague

Faculty of Electrical Engineering

Department of Electric Drives and Traction

Master Thesis

Inverter with Digital Signal Processor for Implementation of Control Algorithms for Low-Power AC Drives

Filip Baum

Prague, January 2023

Supervisor: Ing. Ondřej Lipčák, Ph.D.

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení:	**Baum**	Jméno: **Filip**	Osobní číslo: **473673**
Fakulta/ústav:	**Fakulta elektrotechnická**		
Zadávající katedra/ústav:	**Katedra elektrických pohonů a trakce**		
Studijní program:	**Elektrotechnika, energetika a management**		
Specializace:	**Elektrické pohony**		

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Měnič se signálovým procesorem pro implementaci řídících algoritmů střídavých pohonů menších výkonů

Název diplomové práce anglicky:

Inverter with Digital Signal Processor for Implementation of Control Algorithms for Low-Power AC Drives

Pokyny pro vypracování:

1. Vytvořte přípravek pro řízení střídavých pohonů menších výkonů založený na měničové desce STEVAL od STMicroelectronics. Jako řídicí desku použijte LAUNCHXL-F280049C od Texas Instruments.
2. Navrhněte propojovací desku předávající řídící a zpětnovazební signály mezi měničem, procesorem a pohonem.
3. Uveďte základní vektorové rovnice synchronního motoru s permanentními magnety a princip vektorově orientovaného řízení.
4. Pro verifikaci funkčnosti přípravku implementujte základní algoritmus vektorového řízení synchronního motoru s permanentními magnety (PMSM) na procesoru F280049C. Popište implementační kroky a možnosti, které nabízí využitý procesor pro moderní algoritmy řízení střídavých pohonů.
5. Prozkoumejte možnost využití prostředí MATLAB pro sběr dat z procesorové desky přes dedikovaný USB konektor.
6. Proveďte ladění regulátoru použitých v rámci vektorového řízení přes implementované komunikační rozhraní.

Seznam doporučené literatury:

[1] QUANG, Nguyen P., Jörg-Andreas DITTRICH a J. DITTRICH. Vector Control of Three-Phase AC Machines: System Development in the Practice. Berlin, Heidelberg: Springer Berlin / Heidelberg, 2015.
[2] Texas Instruments Technical Reference Manual. TMS320F28004x Real-Time Microcontrollers (SPRUI33E), NOVEMBER 2015 – REVISED OCTOBER 2022.
[3] TIETZE, Ulrich, Christoph SCHENK a Eberhard GAMM. Electronic circuits: handbook for design and application. 2nd. ed. Berlin: Springer, 2008.
[4] Documentation – MATLAB & Simulink. MathWorks [online]. Natick, Massachusetts, USA, 2022. Available from: https://www.mathworks.com/help/

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Ondřej Lipčák, Ph.D. katedra elektrických pohonů a trakce FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **17.10.2022** Termín odevzdání diplomové práce: **10.01.2023**

Platnost zadání diplomové práce: **24.09.2024**

Ing. Ondřej Lipčák, Ph.D.		prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.	
Datum převzetí zadání	Podpis studenta

Acknowledgement

First of all, I would like to thank my supervisor, Ondřej Lipčák, who always found the time to help me with the thesis. Secondly, I would like to thank Jan Bauer, the head of the Department of Electric Drives and Traction, for his support. Lastly, to my family and friends for their patience and support.

Declaration

I hereby declare that I have written my master thesis on my own and quoted all the sources of information used in accordance with the methodological instructions on ethical principles for writing an academic thesis.

Prague, 5. January 2023

....

Filip Baum

Abstract

Variable speed AC motor drives have become an integral part of our society. With ever increasing range of applications, engineers are pressured to develop novel more sophisticated control algorithms to achieve better performance and efficiency. Over the years, however, motor drives have become increasingly complex systems. Furthermore, extensive supporting software tools for debugging, programming and visualization are indispensable during development. This thesis sets out to create a prototype of a motor drive platform, capable of driving different low power AC machines and accompanied by a MATLAB-based application for visualization of internal control variables. Together they should facilitate future development of complex motor control algorithms. To verify their functionality, a sensored field-oriented control algorithm is programmed into a microprocessor in the C language.

First chapters give a brief overview of permanent magnet synchronous machines and the most common type of switching converter used in variable speed motor drives – the two-level voltage-source inverter. Finally, subsequent chapters deal with the design process of the motor drive platform, including the design of a custom signal interface board. The board isolates the inverter from the microprocessor and ensures a proper transfer of digital and analog signals. Special attention is given to the used microprocessor, its peripherals, advanced features, and proposed configuration. The development process of the MATLAB-based application is described next. To close out the thesis, the implementation process of the control algorithm is described and experimental results are presented.

Keywords: AC Motor Drive, Permanent Magnet Synchronous Machine, Field-Oriented Control, Digital Signal Processor, Serial Communication, Texas Instruments TMS320280049C, Voltage-Source Inverter

Abstrakt

Pohony se střídavými motory jsou v dnešní době nedílnou součástí naší společnosti. Ruku v ruce s nárůstem popularity roste potřeba vývoje nových sofistikovanějších řídicích algoritmů, díky kterým bude možné dosáhnout vyššího výkonu a lepší účinnosti. Moderní elektrický pohon je komplexní systém, pro jehož vývoj je nutné použit širokou škálu podpůrných softwarových nástrojů pro programování, ladění a vizualizaci.

Cílem této diplomové práce je navrhnout prototyp elektrického pohonu, který bude schopen řídit různé střídavé stroje nízkého výkonu. Pohon je doplněn o aplikaci vyvinutou v prostředí MATLAB pro monitorování a vizualizaci řídicích veličin. Společně vytváří univerzální nástroj, který v budoucnu může usnadnit vývoj složitých algoritmů pro řízení pohonů. S cílem otestovat funkčnost navrženého řešení, je do signálového procesoru implementován algoritmus vektorového řízení synchronního motoru s permanentními magnety v jazyce C.

Úvodní kapitoly se věnují vlastnostem a způsobu řízení synchronního stroje s permanentními magnety a dvouúrovňového napěťového střídače, který je nejběžnějším výkonovým měničem používaným v elektrických pohonech. Následné kapitoly se věnují praktické realizaci pohonu, včetně návrhu desky plošného spoje. Navržená deska galvanicky odděluje měnič od procesoru a zajišťuje přenos řídicích signálů. Zvláštní pozornost je věnována použitému mikroprocesoru, jeho periferiím, pokročilým funkcím a navržené konfiguraci. Dále je popsán vývoj monitorovací aplikace a její fungování. Závěrečná kapitola se věnuje implementaci vektorového řízení a experimentálním výsledkům.

Klíčová slova: Střídavý elektrický pohon, Synchronní motor s permanentními magnety, Vektorové řízení, Digitální signálový procesor, Sériová komunikace, Texas Instruments TMS320280049C, Napěťový střídač

Table of Contents

Figures

Tables

Introduction

Historically, applications requiring variable speed control were exclusively dominated by brushed DC machines. The speed control was done predominantly by varying the armature voltage, which is simple and straightforward. The voltage was controlled using either a resistor bank or a rotatory converter. The control was, however, lossy. The introduction of semiconductor devices led to sizable improvements as it was now possible to replace the resistor banks with either a thyristor rectifier or a DC-DC converter. Nevertheless, the mechanical commutator of the DC machine was prone to malfunctions, decreasing the overall reliability of the drive system.

AC machines, on the other hand, offer superior efficiency, reliability, and power density. Their notorious disadvantage has been complicated speed control, which requires variation of the supply frequency. Consequently, more complex control algorithms with greater demand on computational power are required. The rapid development in the field of power electronics and microprocessors within the last decades finally provided the means to overcome these issues, commencing a widespread introduction of AC machines into variable speed motor drives. Soon AC machines became the driving force behind various modern technologies and inventions. One can expect to find them in about every place imaginable, from home appliances to robotics, aviation, electric cars, high-speed trains or even ships and submarines. With the recent rapid rise in popularity of personal electric mobility such as scooters, longboards, bikes etc., medium to low power variable speed AC motor drives have become even more prevalent throughout our everyday lives. Despite them being often overshadowed by the final products they are part of, AC motor drives are, in a way, the driving force behind our modern society that places increased faith in technology to solve the many challenges lingering over the horizon.

The variable speed drives of old were from an electrical point of view simple and presented much greater mechanical undertaking. Conversely, the modern motor drives are a much more sophisticated electrical system. Typically featuring a power semiconductor converter with sophisticated auxiliary circuitry, powerful microprocessor sometimes with multiple cores – all linked together with a complex set of signal interfaces. Ensuring signal integrity of all the digital and feedback analog signals in the presence of a hard switching converter is a tremendous challenge. Moreover, the drive unit is required to be able to communicate with other systems, adding additional layer of complexity with high-speed serial communication.

This thesis sets out to create a prototype of a motor drive platform, capable of driving different low power AC machines. With the ability of being easily connectable to a computer, thus enabling convenient programming and debugging. The ability to visualize internal system variables is key to developing a reliable motor control system. A MATLAB-based application for real-time visualization is developed to supplement the motor drive platform. Finally, a sensored field-oriented control of a permanent magnet synchronous machine is developed and programmed into a microprocessor in the C language to asses the functionality of the designed motor drive platform.

First chapter gives a brief overview of permanent magnet synchronous machines, with emphases placed on the mathematical description and field-oriented control. Second chapter presents the most common type of switching converter used in variable speed motor drives – the two-level voltage-source inverter. Finally, subsequent chapters deal with the design process of the motor drive platform. Special attention is given to the used microprocessor, its peripherals, advanced features, and proposed configuration. The development process of the MATLAB-based communication interface is described next, along with the developed communication protocol for sending data between a computer and the microprocessor over an UART. To close out the thesis, experimental results obtained from controlling a motor via the designed platform are presented.

Chapter 1

Permanent Magnet Synchronous Motor

The permanent magnet synchronous motor (PMSM) has emerged as one of the most common types of electric machine used in modern variable speed motor drives. Such popularity can be attributed to its inherent advantages such as simple structure, mechanical robustness (thanks to the lack of a mechanical commutator), high power density, high efficiency, low noise, and fast dynamic response. The rapid advancement in the field of power electronic and digital signal processors in the last decades has also made a significant contribution to its rise in popularity. Nowadays, the PMSM can be found in all kinds of applications ranging from household appliances to electric cars and robotics [1], [2].

The literature sometimes classifies the brushless DC motor (BLDC) also as a PMSM. While both machine structures share many similarities, there are few key differences. Arguably the most important difference is the shape of the back EMF. The back EMF of a BLDC motor has a characteristic trapezoidal shape, while the PMSM has a sinusoidal back EMF. Moreover, BLDC has higher torque ripple, cogging torque and, consequently, greater acoustic noise [3]. To avoid any future confusion, only a PMSM with sinusoidal back EMF is considered in this thesis.

1.1 Construction

Generally, most PMSMs used today are made of an outer stator with three phase sinusoidally distributed winding and an inner rotor with permanent magnets. The stator yoke is made of laminated steel to minimize the eddy current losses. Based on the rotor structure, the PMSMs can be put into two categories: non–salient pole PMSMs, i.e., surface-mounted PMSMs (SPMSM), and salient pole PMSMs, i.e., interior-mounted PMSMs (IPMSM) [1], [2].

Figure 1.1-1a depicts a typical cross section of a SPMSM. The permanent magnets are mounted on the surface of the rotor core, and because the relative permeability of the magnets is roughly one, i.e., $\mu_\mathrm{r} \approx 1$, the SPMSM is considered to have an uniform air gap. As a result, the synchronous inductances in both direct and quadrature axes are equal, resulting in no reluctance torque produced by the machine. The SPMSM has a simpler structure and permits smaller rotor diameters with lower inertia, making it particularly suitable for servo applications. On the other hand, the SPMSM has a limited field-weakening capability due to the increased risk of demagnetization of its permanent magnets due to their direct exposure to the armature field. Furthermore, the maximum speed of the machine must be limited to preserve the structural integrity of the rotor. These limitations render the SPMSM impractical for traction applications where an extensive field-weakening capability and high-speed operation are required [1], [2], [3].

The IPMSM, on the other hand, has its magnets effectively buried within the rotor core, see figure 1.1-1b, shielding them from the armature filed, thus mitigating the risk of demagnetization. Making it well suited for field-weakening operation. The presence of permanent magnets inside the rotor causes saliency, i.e., $L_d \neq L_q$, specifically for IPMSM $L_q > L_d$. The level of saliency can be controlled during the design process and allows the machine to also produce the reluctance torque. Inserting the magnets into the rotor core, however, complicates the production process, hence the IPMSM is generally more expensive than its non–salient counterpart. Nevertheless, the structural integrity of the rotor is not at risk of being compromised even at extremely high speeds. Extensive field–weakening capability, large toque and high-speed operation make the IPMSM a popular choice in traction applications [2].

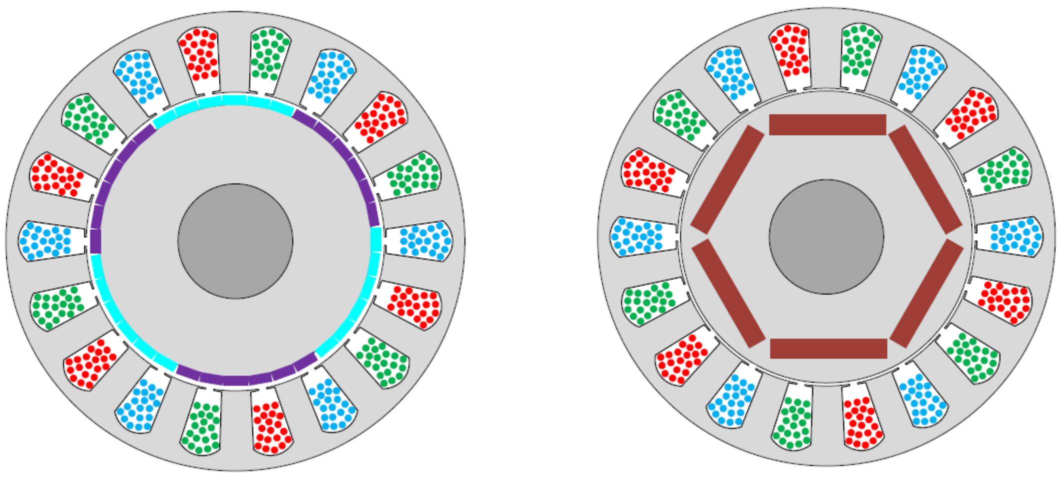

(a) SPMSM (b) IPMSM

Figure 1.1-1: Cross sections of typical structures of PMSMs

1.2 Mathematical Model

To derive a mathematical model of the PMSM, a set of simplifying assumptions must be established. The PMSM presents a complicated nonlinear system, hence the assumptions are required to simplify the system and make it possible to derive a linear model, which is more practical to work with. The following list of assumptions is considered [4]

- sinusoidal distribution of the magnetic flux density of the permanent magnets $\vec{B}_{\mathrm{M}}$ inside the air gap,
- sinusoidal distribution of the three-phase stator winding along the air gap,
- symmetric stator windings, i.e., resistances, self inductances and mutual inductances are identical,
- the effect of stator slotting is neglected,
- iron losses are neglected,
- magnetic saturation is neglected,
- for SPMSM a uniform air gap is assumed,
- for IPMSM the phase inductances depend on the rotor position.

Since SPMSM is used further throughout the thesis, its model is given in greater detail while, for the sake of completeness, only the final set of equations for IPMSM is presented.

We begin with a set of voltage and flux linkage equations for each phase of the stator winding. The voltage equations are simply [4]

$$v_k = R_k i_k + \frac{\mathrm{d}\psi_k}{\mathrm{d}t}, \qquad \text{where } k = \mathrm{a}, \mathrm{b}, \mathrm{c}. \tag{1.2-1}$$

While for the flux linkages we can write [4]

$$\begin{aligned}
\psi_\mathrm{a} &= L_\mathrm{a} i_\mathrm{a} + M_\mathrm{s} \cos\left(\frac{2\pi}{3}\right) i_\mathrm{b} + M_\mathrm{s} \cos\left(-\frac{2\pi}{3}\right) i_\mathrm{c} + \psi_\mathrm{M} \cos\left(\vartheta\right), \\
\psi_\mathrm{b} &= L_\mathrm{b} i_\mathrm{b} + M_\mathrm{s} \cos\left(\frac{2\pi}{3}\right) i_\mathrm{c} + M_\mathrm{s} \cos\left(-\frac{2\pi}{3}\right) i_\mathrm{a} + \psi_\mathrm{M} \cos\left(\vartheta - \frac{2\pi}{3}\right), \\
\psi_\mathrm{c} &= L_\mathrm{c} i_\mathrm{c} + M_\mathrm{s} \cos\left(\frac{2\pi}{3}\right) i_\mathrm{a} + M_\mathrm{s} \cos\left(-\frac{2\pi}{3}\right) i_\mathrm{b} + \psi_\mathrm{M} \cos\left(\vartheta + \frac{2\pi}{3}\right),
\end{aligned} \tag{1.2-2}$$

where L is the self inductance of each phase, M_s is the mutual inductance between phases, ψ_M is the permanent magnet flux and ϑ is the rotor position.

A common method used in analysis of AC machines for reducing the number of equations is the Clarke transform which transforms a real three-phase quantity into a complex one, often called a space vector, denoted as $\underline{x}$. The transform is defined as [4], [5]

$$\begin{pmatrix} x_\alpha \\ x_\beta \end{pmatrix} = K \begin{pmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{pmatrix} \begin{pmatrix} x_\mathrm{a} \\ x_\mathrm{b} \\ x_\mathrm{c} \end{pmatrix} \tag{1.2-3}$$

and its inverse

$$\begin{pmatrix} x_\mathrm{a} \\ x_\mathrm{b} \\ x_\mathrm{c} \end{pmatrix} = \frac{1}{K} \begin{pmatrix} \frac{2}{3} & 0 \\ -\frac{1}{3} & \frac{1}{\sqrt{3}} \\ -\frac{1}{3} & -\frac{1}{\sqrt{3}} \end{pmatrix} \begin{pmatrix} x_\alpha \\ x_\beta \end{pmatrix}. \tag{1.2-4}$$

The transformation constant K can be in principle chosen arbitrarily, however the most common choices are $K = 1$, $K = \frac{2}{3}$ or $K = \sqrt{2/3}$ [5]. Transforming equations 1.2-1 and 1.2-2 yields a set of two complex vector equations [4], [5]

$$\begin{aligned}
\underline{v}_\mathrm{s}^\mathrm{s} &= R_\mathrm{s} \underline{i}_\mathrm{s}^\mathrm{s} + \frac{\mathrm{d}\underline{\psi}_\mathrm{s}^\mathrm{s}}{\mathrm{d}t}, \\
\underline{\psi}_\mathrm{s}^\mathrm{s} &= L_\mathrm{s} \underline{i}_\mathrm{s}^\mathrm{s} + \underline{\psi}_\mathrm{M}^\mathrm{r} \mathrm{e}^{\mathrm{j}\vartheta}.
\end{aligned} \tag{1.2-5}$$

The superscript $^\mathrm{s}$ denotes that the variable is coupled with the stationary coordinate system $\alpha\beta$, whilst $^\mathrm{r}$ denotes coupling with the synchronous coordinate system dq. The number of equations was reduced but the space vectors are time dependent because the flux linkage vector $\underline{\psi}_\mathrm{s}^\mathrm{s}$ depends on the rotor position ϑ.

This is addressed by another coordinate transformation, the Park transform, which transforms equations 1.2-5 into the synchronous coordinate system dq. Flux linkages, voltages and currents become DC quantities in the dq coordinates, making them much more suitable for controller design. The Park transform is defined as [4], [5]

$$\begin{pmatrix} x_d \\ x_q \end{pmatrix} = \begin{pmatrix} \cos\left(\vartheta\right) & \sin\left(\vartheta\right) \\ -\sin\left(\vartheta\right) & \cos\left(\vartheta\right) \end{pmatrix} \begin{pmatrix} x_\alpha \\ x_\beta \end{pmatrix} \tag{1.2-6}$$

and its inverse

$$\begin{pmatrix} x_\alpha \\ x_\beta \end{pmatrix} = \begin{pmatrix} \cos(\vartheta) & -\sin(\vartheta) \\ \sin(\vartheta) & \cos(\vartheta) \end{pmatrix} \begin{pmatrix} x_d \\ x_q \end{pmatrix}. \tag{1.2-7}$$

Transforming equations 1.2-5 into the synchronous coordinates yields the final set of equations which can be used to design controllers or simulate the machine behavior [4], [5]

$$\begin{aligned} v_{\mathrm{sd}} &= R_{\mathrm{s}} i_{\mathrm{sd}} + \frac{\mathrm{d}\psi_{\mathrm{sd}}}{\mathrm{d}t} - \omega\psi_{\mathrm{sq}}, \\ v_{\mathrm{sq}} &= R_{\mathrm{s}} i_{\mathrm{sq}} + \frac{\mathrm{d}\psi_{\mathrm{sq}}}{\mathrm{d}t} + \omega\psi_{\mathrm{sd}}, \\ \psi_{\mathrm{sd}} &= L_{\mathrm{s}} i_{\mathrm{sd}} + \psi_{\mathrm{M}}, \\ \psi_{\mathrm{sq}} &= L_{\mathrm{s}} i_{\mathrm{sq}}. \end{aligned} \tag{1.2-8}$$

Furthermore, the torque is given as [4], [5]

$$T = \frac{3}{2} p_{\mathrm{p}} \psi_{\mathrm{M}} i_{\mathrm{sq}}. \tag{1.2-9}$$

Detail derivation of equation 1.2-8 can be found, for example, in [4]. Equation 1.2-8 lays the foundation for all modern control strategies of SPMSM.

The IPMSM is described by a similar set of equations, the main difference is that $L_d \neq L_q$ and, consequently, the reluctance torque [4]. The differences from SPMSM equations are shown in red color

$$\begin{aligned} v_{\mathrm{sd}} &= R_{\mathrm{s}} i_{\mathrm{sd}} + \frac{\mathrm{d}\psi_{\mathrm{sd}}}{\mathrm{d}t} - \omega\psi_{\mathrm{sq}}, \\ v_{\mathrm{sq}} &= R_{\mathrm{s}} i_{\mathrm{sq}} + \frac{\mathrm{d}\psi_{\mathrm{sq}}}{\mathrm{d}t} + \omega\psi_{\mathrm{sd}}, \\ \psi_{\mathrm{sd}} &= L_d i_{\mathrm{sd}} + \psi_{\mathrm{M}}, \\ \psi_{\mathrm{sq}} &= L_q i_{\mathrm{sq}}, \\ T &= \frac{3}{2} p_{\mathrm{p}} \left(\psi_{\mathrm{M}} i_{\mathrm{sq}} + \left(L_d - L_q\right) i_{\mathrm{sq}} i_{\mathrm{sd}}\right). \end{aligned} \tag{1.2-10}$$

1.3 Field-Oriented Control

The field-oriented control (FOC) of PMSM is based on the vector equations 1.2-8, 1.2-9, hence it is also sometimes called the vector control. It utilizes the aforementioned fact that in the synchronous coordinate system *dq* coupled with rotor, voltages, currents and flux linkages are DC quantities, which permits us to use simple PI controllers to control the speed/position and torque of the machine, instead of more sophisticated controller structures. Furthermore, thanks to equation 1.2-9 it is possible to achieve a decoupled control of the machine torque and flux, similarly to a separately excited DC machine [5].

Critical parameter affecting the performance of the FOC is the rotor angle that is required to perform the Park and inverse Park transforms. Imprecise knowledge of the rotor angle causes the FOC detuning, severely compromising the drive performance. The angle can be either measured with a position sensor such as an incremental encoder, absolute encoder, or resolver, or calculated using an observer [5].

This thesis implements only the sensored variant, which is more straightforward to implement, making it ideal to test the designed inverter platform. A principal control structure is depicted in figure 1.3-2, while implementation-specific details are given later in the thesis.

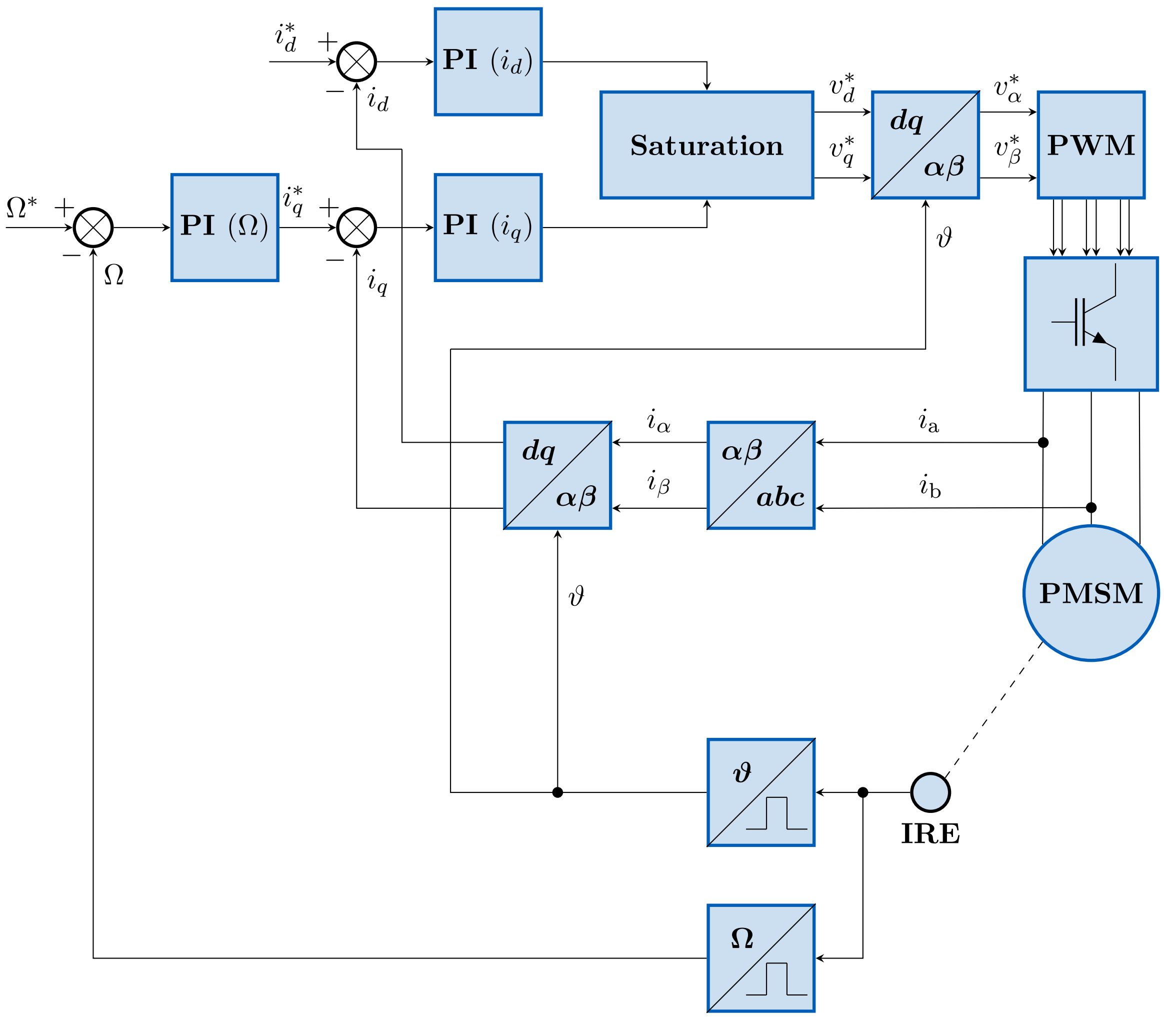


Figure 1.3-2: Block diagram of the position-sensored FOC of PMSM.

First, the phase currents are measured. A minimum of two currents have to be measured since a balanced three-phase system is assumed, i.e., $i_\mathrm{a} + i_\mathrm{b} + i_\mathrm{c} = 0$, so the third current can be computed. The number of measured currents is mainly dictated by the capabilities of the control microprocessor. The currents are then transformed to the synchronous coordinates dq, using the Clarke (1.2-3) and Park (1.2-6) transforms. Transformed currents are subtracted from their corresponding reference values and the resulting control errors are fed to the current controllers. Typically, the field producing component i_d is kept at zero and only the torque producing component i_q is controlled to generate the desired torque. Only during field-weakening a negative i_d is applied to weaken the magnetic flux from the permanent magnets, thus allowing the motor to operate above its nominal speed [5]. Additionally, if an IPMSM operates under the MTPA control strategy, i_d is controlled such that the machine produces the maximum torque for a given current magnitude, details can be found for example in [6], [7].

The reference voltages v_d^*, v_q^* coming from the current controllers are first saturated, then transformed to the stator coordinates $\alpha\beta$ via the inverse Park Transform (1.2-7). These voltages are then used by the PWM modulator to generate switching patterns for the inverter supplying the motor [5]. The control scheme depicted in figure 1.3-2 utilizes an incremental encoder to measure the position and speed of the rotor and operates in a speed loop, matching the implemented variant later in the thesis. The incremental encoder can be replaced by a resolver or an absolute encoder. The latter offers great precision but at the cost of a noticeable increase in price. Resolver, on the other hand, is analog in nature, requiring an external sinusoidal supply and auxiliary circuitry to process the output signals. Alternatively, the speed controller can be replaced by a position controller, allowing for precise control of the rotor position in servo applications.

Chapter 2

Two Level Voltage Source Inverter

Integral part of a modern AC motor drive is a power inverter supplying the motor with voltage of desired magnitude and frequency. By far the most common topology used today is the three-phase voltage-source inverter. Same topology is used further in the design part of the thesis; hence a brief overview of its signature properties is given in the following sections.

2.1 Principle of Operation

A typical arrangement is depicted in figure 2.1-1. The core of the inverter are the six semiconductor switches (S_1 to S_6), arranged into three separate legs with each leg having two switches. Here, the switches are shown as IGBTs, however power MOSFETs can also be used with voltages up to $\approx$ 800 V. One or more capacitors in series are connected across the DC link of the inverter to filter out voltage transients from the supply [5], [8], [9]. The DC supply is not shown in figure 2.1-1 but is usually realized as either a battery or a rectifier. In the past, three-phase full bridge diode rectifiers were used, nowadays these are being phased out in favor of the PWM rectifiers capable to control the amount of reactive power drawn from the AC supply [8].

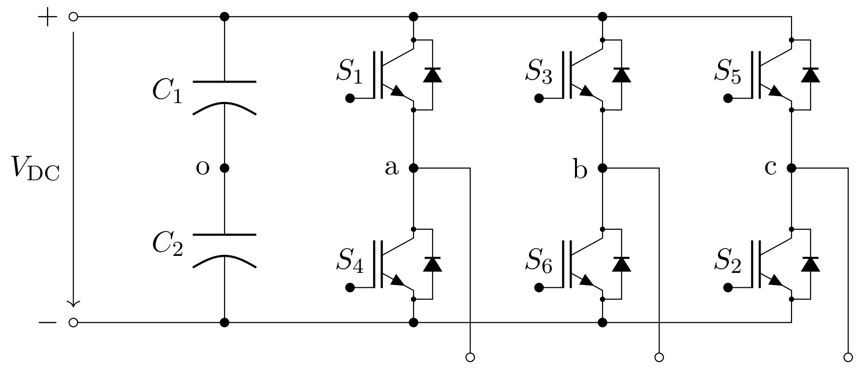


Figure 2.1-1: Three phase voltage-source inverter.

The control signals for the transistors are generated by the PWM modules of the control computer. A 3.3 V outputs of the computer are not powerful enough to drive the transistors, hence driver circuits must be used to generate the appropriate voltage and current. Furthermore, to ensure safe operation of the inverter, under no circumstances can both switches in one leg be turned on simultaneously. Otherwise, the DC link is shorted and the transistors are destroyed. Hence a dead-time is added to the control signals to ensure that a switch starts to turn on only when the other one has turned off. The dead-time can be added either by the microcontroller or the driver circuit [5], [8], [9].

With the total of six switches there exist $2^3 = 8$ possible switching combinations, corresponding to eight voltage vectors in the complex $\alpha\beta$ plane. Table 2.1-1 shows which switching combination produces which voltage vector. State "1" stands for the upper switch being on and lower one off, while "0" stands for the opposite. Vectors $\underline{v}_1 - \underline{v}_6$ are the active vectors, conversely $\underline{v}_0$, $\underline{v}_7$ are the zero vectors [5], [8], [9].

Fundamental voltage vectors can also be arranged in the complex plane, see figure 2.1-2. The voltage vectors divide the four quadrants $Q_1 - Q_4$ of the complex plane into further six sectors $S_1 - S_6$, with each sector being 60° wide. This division becomes important for generating an arbitrary voltage vector [5], [8], [9].

Phase	$\underline{v}_0$	$\underline{v}_1$	$\underline{v}_2$	$\underline{v}_3$	$\underline{v}_4$	$\underline{v}_5$	$\underline{v}_6$	$\underline{v}_7$
a	0	1	1	0	0	0	1	1
b	0	0	1	1	1	0	0	1
c	0	0	0	0	1	1	1	1

Table 2.1-1: Fundamental voltage vectors with corresponding switching states [5].

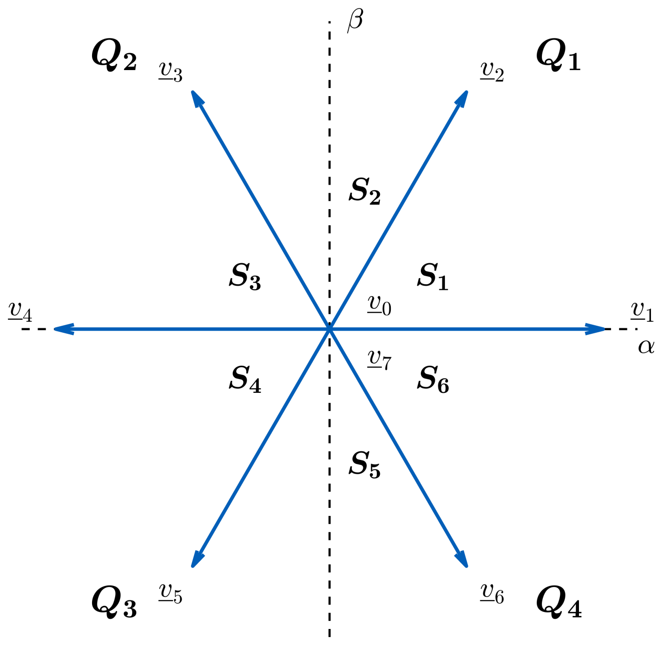


Figure 2.1-2: Fundamental voltage vectors.

2.2 Space Vector Modulation

The magnitude of all fundamental voltage vectors is [5], [8], [9]

$$|\underline{v}_{\max}| = |\underline{v}_1| = |\underline{v}_2| = |\underline{v}_3| = |\underline{v}_4| = |\underline{v}_5| = |\underline{v}_6| = \frac{2}{3}V_{\mathrm{DC}}. \tag{2.2-1}$$

Using only the discrete fundamental vectors would result in the square-wave mode of operation where each switch is turned on and off only once per period of the output voltage. This becomes beneficial in high power applications where the semiconductor switches have generally slow turn-on and turn-off speeds. However, the square-wave mode does not allow to control the voltage magnitude, if required, it has to be done on the DC side which is impractical [8].

One way to generate an arbitrary voltage vector is as a vector sum of two vectors whose direction is the same as the fundamental vectors, while the magnitude is chosen to produce the desired vector [5], [8], [9]. Such technique is called the space vector modulation and the principle is shown in figure 2.2-3, showing the situation when the desired vector $\underline{v}^*$ is in sector 1, for other sectors the principle is analogous.

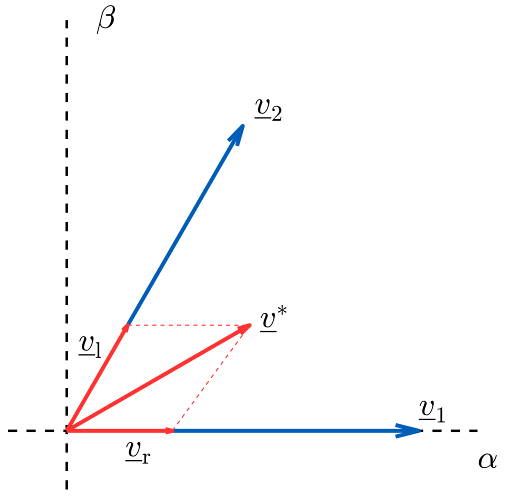


Figure 2.2-3: Construction of an arbitrary voltage vector from fundamental vectors.

The desired vector $\underline{v}^*$ is generated as the sum of vectors $\underline{v}_\mathrm{r}$, $\underline{v}_\mathrm{l}$. These vectors are generated by varying the duration of the fundamental vectors $\underline{v}_1$, $\underline{v}_2$ during the switching period T_PWM. Let T_l be the duration of $\underline{v}_\mathrm{l}$ and T_r be the duration of $\underline{v}_\mathrm{r}$, then we can write [5], [8], [9]

$$\begin{aligned} T_\mathrm{l} &= T_\mathrm{PWM}\frac{|\underline{v}_\mathrm{l}|}{|\underline{v}_\mathrm{max}|} \\ T_\mathrm{r} &= T_\mathrm{PWM}\frac{|\underline{v}_\mathrm{r}|}{|\underline{v}_\mathrm{max}|}. \end{aligned} \tag{2.2-2}$$

The remaining duration of the switching period $T_\mathrm{PWM} - (T_\mathrm{l} + T_\mathrm{r})$ is occupied by one of the zero vectors. The vector $\underline{v}^*$ is then obtained as [5], [8], [9]

$$\underline{v}^* = \frac{1}{T_\mathrm{PWM}}\left[T_\mathrm{r}\underline{v}_1 + T_\mathrm{l}\underline{v}_2 + T_\mathrm{PWM} - (T_\mathrm{l} + T_\mathrm{r})\,\underline{v}_0 \text{ or } \underline{v}_7\right]. \tag{2.2-3}$$

Analogous equations can be developed for the remaining sectors. One possible switching sequence is the seven-segment modulation depicted in figure 2.2-4, which is practical when an up-down counter is used in the microcontroller, since the pulse pattern is symmetric along its maximum value. Different patterns with emphasis on minimizing the THD or switching losses can be found, e.g., in [8], [9].

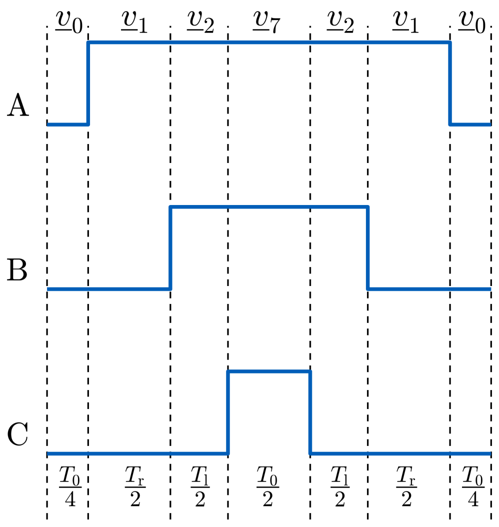


Figure 2.2-4: Pulse pattern for sector 1.

2.2.1 Switching Time Calculation

To calculate the duration of the corresponding fundamental vectors, knowledge of $|\underline{v}_\mathrm{r}|$, $|\underline{v}_\mathrm{l}|$ is required, see equation 2.2-2. One possible way to calculate $|\underline{v}_\mathrm{r}|$, $|\underline{v}_\mathrm{l}|$ is to use the $\alpha\beta$ components of the reference voltage vector from the current controllers. Utilizing the segmentation of the complex plane into sectors and trigonometry, table 2.2-2 can be put together [5], [9].

Quadrant	Sector	$\lvert\underline{v}_\mathrm{r}\rvert$	$\lvert\underline{v}_\mathrm{l}\rvert$
Q_1	S_1	$\lvert v_\alpha\rvert - \frac{1}{\sqrt{3}}\lvert v_\beta\rvert$	$\frac{2}{\sqrt{3}}\lvert v_\beta\rvert$
	S_2	$\lvert v_\alpha\rvert + \frac{1}{\sqrt{3}}\lvert v_\beta\rvert$	$-\lvert v_\alpha\rvert + \frac{1}{\sqrt{3}}\lvert v_\beta\rvert$
Q_2	S_2	$-\lvert v_\alpha\rvert + \frac{1}{\sqrt{3}}\lvert v_\beta\rvert$	$\lvert v_\alpha\rvert + \frac{1}{\sqrt{3}}\lvert v_\beta\rvert$
	S_3	$\frac{2}{\sqrt{3}}\lvert v_\beta\rvert$	$\lvert v_\alpha\rvert - \frac{1}{\sqrt{3}}\lvert v_\beta\rvert$
Q_3	S_4	$\lvert v_\alpha\rvert - \frac{1}{\sqrt{3}}\lvert v_\beta\rvert$	$\frac{2}{\sqrt{3}}\lvert v_\beta\rvert$
	S_5	$\lvert v_\alpha\rvert + \frac{1}{\sqrt{3}}\lvert v_\beta\rvert$	$-\lvert v_\alpha\rvert + \frac{1}{\sqrt{3}}\lvert v_\beta\rvert$
Q_4	S_5	$-\lvert v_\alpha\rvert + \frac{1}{\sqrt{3}}\lvert v_\beta\rvert$	$\lvert v_\alpha\rvert + \frac{1}{\sqrt{3}}\lvert v_\beta\rvert$
	S_6	$\frac{2}{\sqrt{3}}\lvert v_\beta\rvert$	$\lvert v_\alpha\rvert - \frac{1}{\sqrt{3}}\lvert v_\beta\rvert$

Table 2.2-2: Determination of boundary vectors based on v_α, v_β.

In actuality, table 2.2-2 contains only three different terms, i.e., [5], [9],

$$\begin{aligned} a &= |v_\alpha| + \frac{1}{\sqrt{3}}|v_\beta|\,, \\ b &= |v_\alpha| - \frac{1}{\sqrt{3}}|v_\beta|\,, \\ c &= \frac{2}{\sqrt{3}}|v_\beta|\,. \end{aligned} \tag{2.2-4}$$

Signs of v_α, v_β are used to determine the quadrant, and subsequently the sing of b is used to determine the sector [5], [9].

2.2.2 Physical Limitations

The set of all realizable voltage vectors lies within a hexagon whose vertexes are the endpoints of the fundamental vectors. While the full hexagon might be utilized to reach the maximum voltage, moving the endpoint of the voltage vector along the hexagon produces non-sinusoidal currents. Distortion of the motor currents can deteriorate the drive performance if not accounted for properly. When the full hexagon is used, the converter is said to operate in the overmodulation region [5], [9].

To generate sinusoidal currents, only the inscribed circle with radius of $1/\sqrt{3}V_{\mathrm{DC}}$ can be used. Further limitations arise in the regions where the circle touches the hexagon. As the circle nears an edge of the hexagon, the duration of the zero vectors reduces. Every transistor has a finite turn-on and turn-of time making pulses below a certain minimal duration unrealizable [5], [9]. The minimal pulse duration depends on the type of transistor used. Generally, silicon IGBTs have their turn-on and turn-of times roughly few hundred nanoseconds. MOSFETs tend to be faster but cannot withstand such high voltages and currents as IGBTs can. Modern transistors based on GaN or SiC technology can reach a few dozen nanoseconds, however high price and power limitations remain an issue [10].

Hence the maximum realizable voltage must be reduced with respect to the minimal pulse duration from $|v_{\mathrm{max}}| = 1/\sqrt{3}V_{\mathrm{DC}}$ to $|v_{\mathrm{max}}| = \lambda/\sqrt{3}V_{\mathrm{DC}}$. Where λ can be expressed as

$$\lambda = 1 - \frac{T_{0\mathrm{min}}}{T_{\mathrm{PWM}}} < 1. \qquad (2.2\text{-}5)$$

Final set of all realizable voltage vectors is depicted in figure 2.2-5 as the inner inscribed circle.

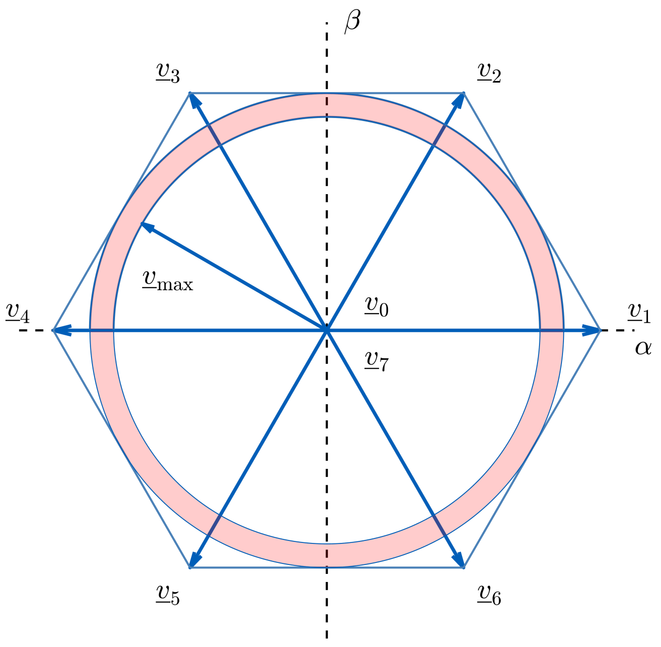


Figure 2.2-5: Realizable set of voltage vectors.

Chapter 3

Inverter Platform Design

The main objective of this thesis was to design and build an inverter platform that can be easily powered from the 230 V/50 Hz power line and includes a powerful digital signal processor for implementing motor control algorithms. The processor should also be easily connectable to a computer for programming and debugging. The maximum power output should be around 1 kW, allowing to supply a variety of low power three phase AC machines. Insight into the design process is given in the following sections.

3.1 General Concept

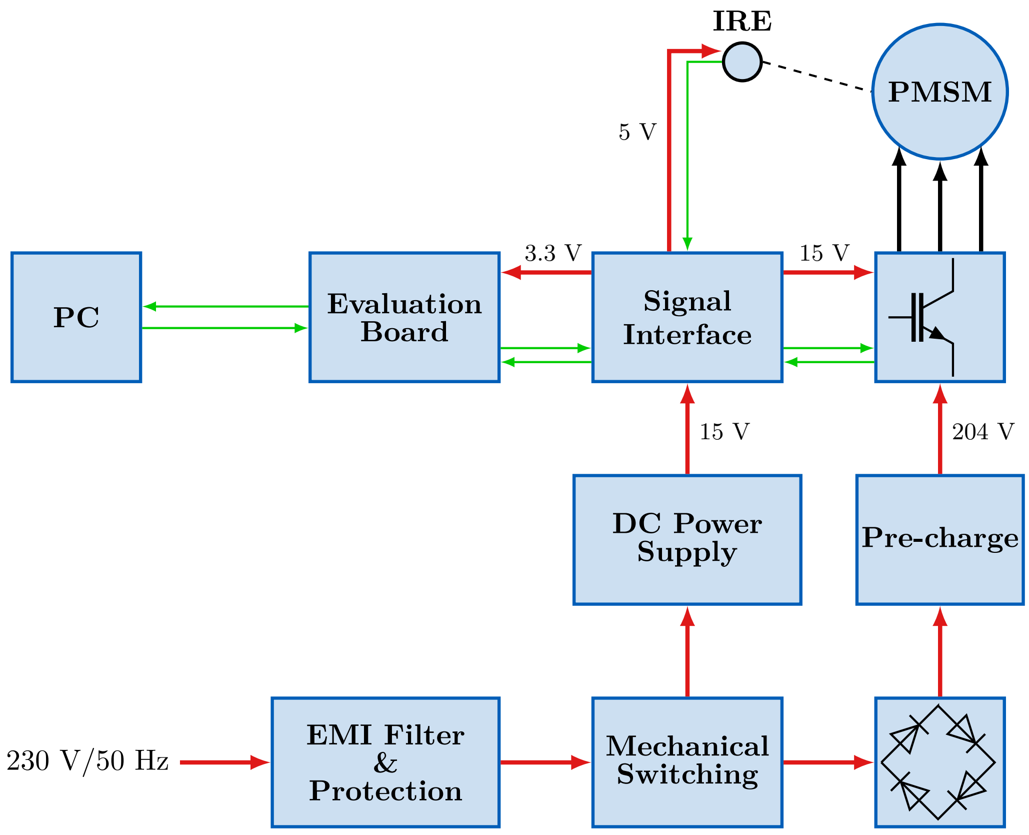


Figure 3.1-1: Proposed structure of the inverter platform.

A general structure of the platform is depicted in figure 3.1-1. The setup is powered from a power outlet through a standard IEC C13 connector. The power then goes through an EMI filter whose main purpose is to block the high amount of switching noise from reaching the power line. Then it passes through a fuse to reach a self-holding circuit made up of a turn-on and turn-off mechanical switches plus a relay. The AC power is then fed to a switch-mode power supply, converting the 230 V/50 Hz into 15 V DC. This DC voltage is used to directly power the drivers of the inverter module. Furthermore, it gets stepped down by linear voltage regulators to 5 V and 3.3 V to power all the integrated circuits on the signal interface board as well as the microprocessor. Lastly, the AC power is fed to a full bridge diode rectifier that supplies the DC bus of the inverter. A pre-charging circuit is connected in between the rectifier and the DC bus to limit the capacitor inrush current. The components described so far, can be considered as the input stage of the system.

The STEVAL-IPM15B power board from STMicroelectronics was chosen as the inverter platform for the design. The board is based around the STGIB15CH60TS-L IGBT power module and contains, among other features, three low-side shunts resistors for motor current measurement and a 34-pin connector that allows for smoothless connection to an external control board [11].

One inconvenient design feature of the chosen power board is that it shares a common ground between its DC link and the rest of the circuits. By powering the DC bus from a rectifier, the negative terminal of the DC bus essentially rides on the AC power line as rectifier voltage is only DC between its outputs, not if it is referenced to earth ground. This makes the set up dangerous to work with as touching any ground point on the power board and every other board, such as a control board, could harm the user or any equipment connected to earth ground like a computer or an oscilloscope. The oscilloscope issue can be solved by using a battery powered one whose probe grounds are isolated from earth ground and each other. However, the danger for the user would still persist. Of course, powering the full setup from an isolation transformer solves the problem all together but such transformers are heavy and expensive. Hence it was decided that an interface board would be designed. The board makes use of capacitive coupling isolators to transfer digital signals and optocouplers to transfer analog signals between the power board and the evaluation board. Thus, creating a galvanic isolation between the two domains. Also, a isolated DC/DC switching regulator is used to transfer power between the two domains.

Finally, the LAUNCHXL-F280049C evaluation board form Texas Instruments was selected as a control board for the system. The board contains the TMS320F280049C digital signal processor which is a powerful 32-bit floating point microprocessor with clock rate of up to 100 MHz and a long list of peripherals essential for real-time control applications. The board comes with four 20-pin headers that include ADC channels, PWM channels, GPIOs and communication peripherals making it convenient to connect to external boards. The launchpad also comes with the XDS110 debug probe that allows for easy debugging and programming through an on-board micro-USB connector [12].

3.2 Input Stage

The setup is protected by a 16 A aR class semiconductor fuse. The fast fuse was picked to also protect the IGBTs of the inverter module from short-circuits. To block some of the switching noise from reaching the grid, a EMI filter is placed at the input, right after the fuse. The circuit diagram of the filter is depicted in figure 3.2-2. The filter was bought already assembled inside a package from the supplier, therefore the author did not deal with its design. The power rating was picked to be 230 V AC, 16 A, details, including the frequency response, can be found in [13].

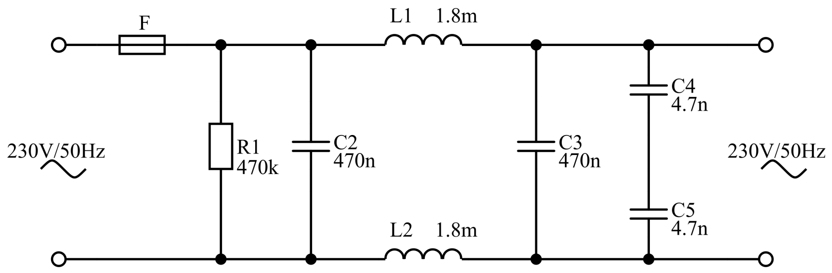


Figure 3.2-2: Input of the platform with a fuse and EMI filter.

To conveniently turn the platform on and off, a simple self-holding circuit is used, see figure 3.2-3. Two mechanical switches are used, normally closed S1 for turn-off and normally open S_2 for turn-on and a relay K_A. Initially, the set up is off, to turn it on, S_2 is pressed first which energizes the relay coil. With the relay coil energized, S_2 is bypassed by K_{A1} keeping the setup powered even after we let go of S_2. To turn it off, S_1 is pressed. After opening S_1, the relay coil is de-energized, causing K_{A1} to open, thus permanently disconnecting the platform from power until S_2 is pressed again, repeating the process.

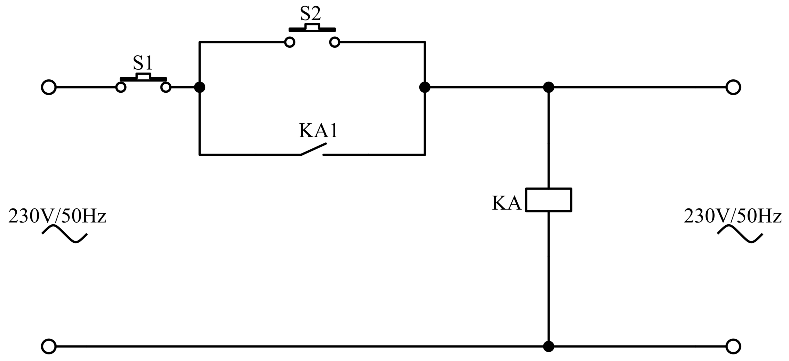


Figure 3.2-3: Switching stage with a self-holding circuit.

The DC bus of the inverter is supplied from a full bridge diode rectifier. The bridge was bought as a integrated module with the ability of being screw-mounted onto a heat sink. The parameters of the rectifier were chosen as

- $V_{\mathrm{RRM}} = 400$ V,
- $V_{\mathrm{RMS}} = 280$ V,
- $I_{\mathrm{F(AV)}} = 25$ A.

The maximum average forward current $I_{\mathrm{F(AV)}} = 25$ A is over dimensioned for this application. However, due to the limited availability of components on the market at the time of making the thesis, this model was chosen.

To address the issue of high inrush current of the DC link capacitor during initial power up, simple pre-charge circuit is used. The circuit is made up of just a timer relay and a wire-wound power resistor, see figure 3.2. During power up, the resistor R_{10} is connected in between the rectifier and the DC bus, limiting the charging current of the fully discharged DC bus capacitor C_1. After a 1 s delay, the relay contacts switch from the initial position 1-3 to position 1-2, connecting the rectifier directly to the DC bus. The peak inrush current is limited to

$$I_{\text{max}} = \frac{V_{\text{max}}}{R_{10}} = \frac{230\sqrt{2}}{50} = 6.5 \text{ A} \tag{3.2-1}$$

and drops with time constant

$$\tau = R_{10}C_1 = 50 \cdot 820 \cdot 10^{-6} = 41 \text{ ms}. \tag{3.2-2}$$

Based on the time constant, the relay could in theory switch much earlier than 1 s. However, the chosen relay does not support delay times below 1 s. The extra losses in the resistor are negligible as the only current flowing through it, after the capacitor is charged, is the capacitor leakage current. Which, according to manufacturer, is roughly $I_{\text{leak}} \leq 2.4$ mA.

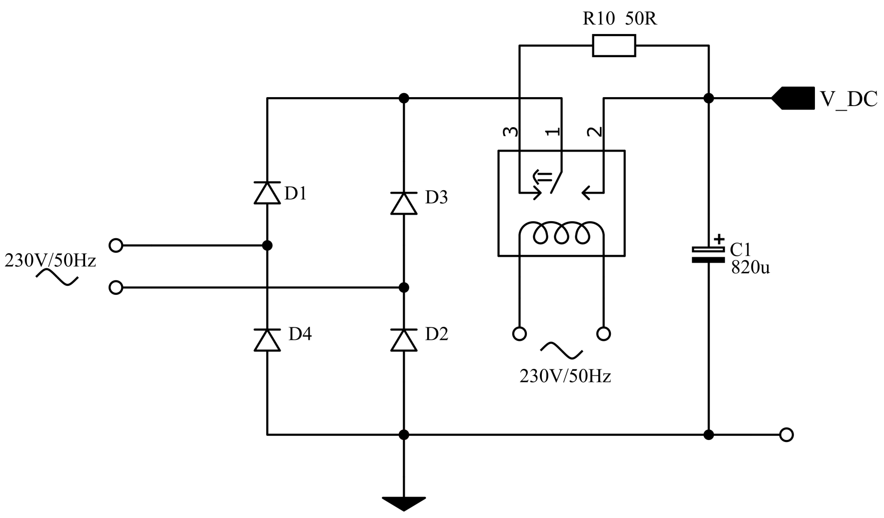


Figure 3.2-4: Full bridge rectifier with a pre-charge circuit.

3.3 Inverter Board

The STEVAL-IPM15B power board from STMicroelectronics serves as the final power stage of the designed set up. Its architecture can be seen in figure 3.3-5. The inverter module is hidden underneath the pcb to allow for an easier connection to a heat sink. The main features of the power board are [11]

- 125-400 V DC input voltage,
- 1000 W nominal power,
- 9 A nominal current,
- 15-20 V DC auxiliary voltage,
- 34 pin control connector,
- Three shunt resistors with op-amp network for current measurement,
- STGIB15CH60TS-L IGBT power module with temperature measurement.

The board comes without the DC link capacitor, therefore it must be provided and mounted by the user. For this application an 820 μF, 450 V aluminum electrolytic capacitor was chosen. Since the DC bus is supplied by a single-phase full bridge rectifier, even larger capacitance would be preferable to smooth out the DC bus voltage as much as possible. However, due to size constrains as the capacitor must fit into the predefined position and availability issues at the time of making this thesis, the aforementioned capacitor was chosen.

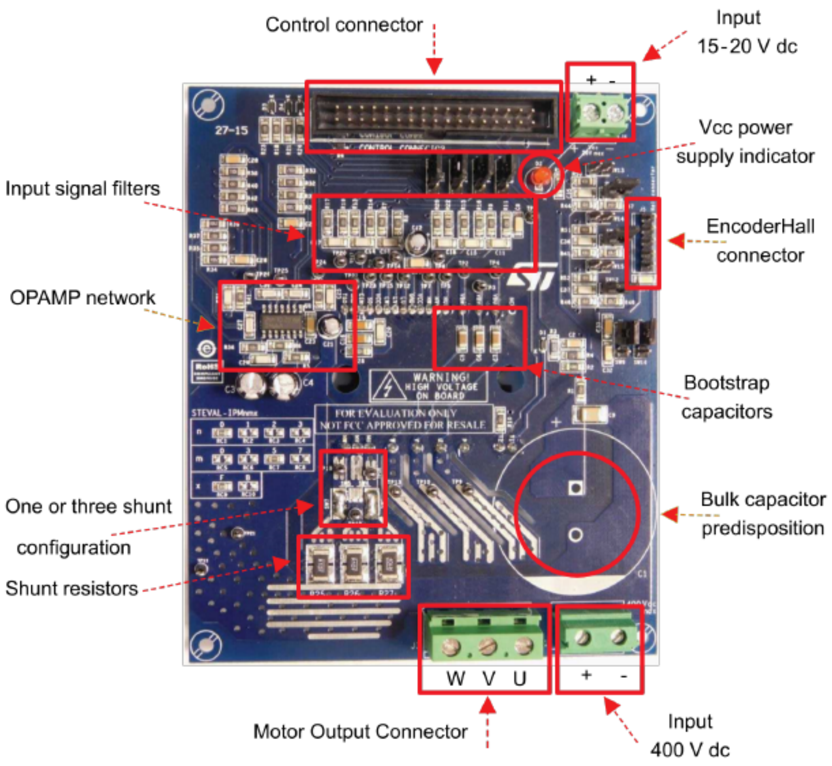


Figure 3.3-5: The STEVAL-IPM15B power board architecture [11].

3.3.1 STGIB15CH60TS-L IGBT Power Module

The power module features six IGBTs with freewheeling diodes connected into a standard three-phase inverter bridge. High-side and low-side drivers are also included. The die temperature is measured by a temperature sensor that outputs voltage proportional to the temperature. The sensor output is available on one of the pins, and, thus, can easily be monitored by a microprocessor. The recommended operating conditions are summarized in table 3.3-1, only the most important ones are listed, while a full list of parameters can be found in the datasheet [14].

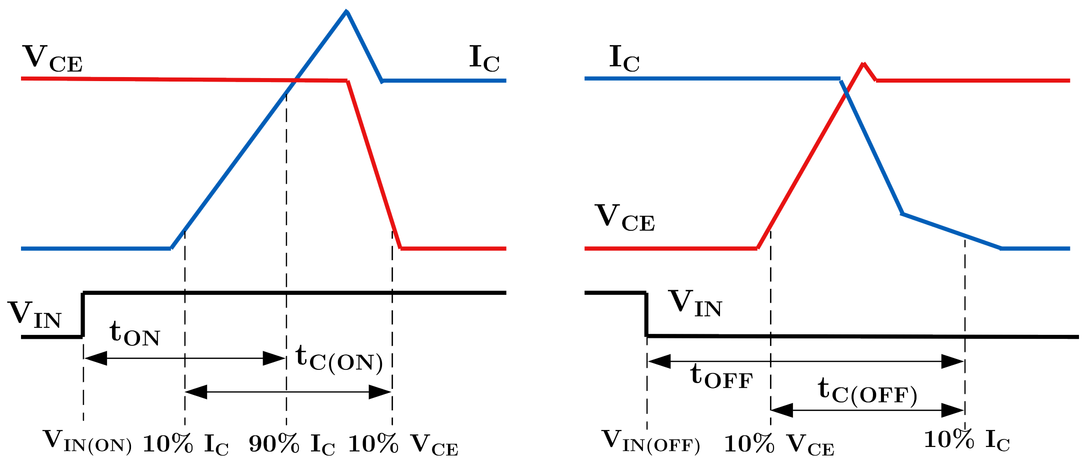
Parameter	Minimum value	Typical value	Maximum value
Supply voltage V_{DC}		300 V	400 V
Control voltage V_{CC}	13.5 V	15 V	18 V
Dead time t_{dead}	1 μs		
PWM frequency f_{PWM}			20 kHz
Case temperature T_C			100 °C

Table 3.3-1: Recommended operating conditions [14].

Furthermore, the typical switching times are given in table 3.3-2. Since the switching times of an IGBT depend on the operating conditions, mainly the collector current I_C, the values in table 3.3-2 are for $V_{DC} = 300$ V, $V_{CC} = 15$ V and $I_C = 15$ A. The different switching time intervals are defined in figure 3.3-6. By looking at the data we can immediately conclude that the switching times are rather long, hence the maximum PWM frequency can only be 20 kHz [14].

Switching time	Typical value
Turn-on time t_{ON}	320 ns
Cross-over time on $t_{C(ON)}$	160 ns
Turn-off time t_{OFF}	510 ns
Cross-over time off $t_{C(OFF)}$	102 ns

Table 3.3-2: Typical switching times for inductive load [14].



(a) turn-on (b) turn-off

Figure 3.3-6: Switching time definition [14] (edited).

3.3.2 Current and DC Bus Voltage Measurement

The motor phase currents are measured using a shunt resistor connected to the emitter of the low-side IGBT of each phase. The low-side shunt resistors have certain practical implications for the FOC algorithm as the current can only be sampled when all the low-side IGBTs are on, i.e, the vector $\underline{v}_0$ is applied to the inverter. Furthermore, the duration of the vector must be long enough to allow the current to reach its steady-state value. This poses a challenge when the modulation reaches the edges of the hexagon, see figure 2.2-5, as the duration of the zero vector shrinks. When the duration drops below the physical limit of the used transistors, the pulse is omitted completely, making it impossible to measure all three currents at the same time. If the full hexagon or even just the inscribed circle of radius $1/\sqrt{3}V_{DC}$ are required, special current reconstruction procedure must be used. Since the FOC developed later in the thesis does not operate under such conditions, these limitations are not an issue. Further details on the reconstruction procedure can be found, e.g, in [15].

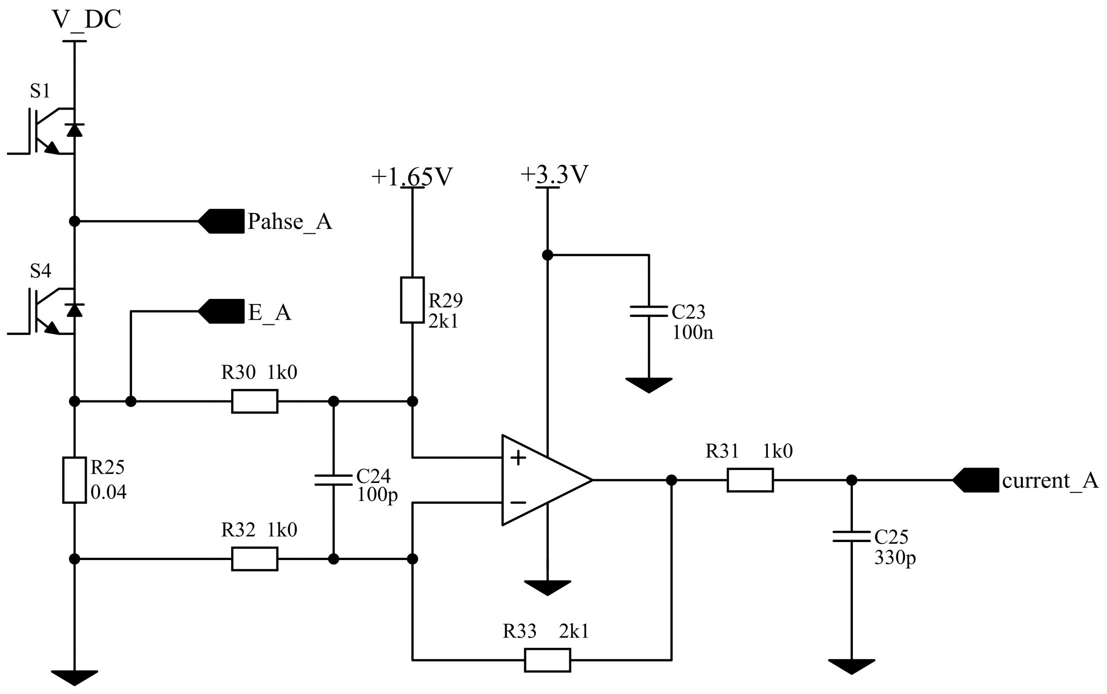


Figure 3.3-7: Current sensing circuit [11].

The current sensing circuit used by the STEVAL power board is depicted in figure 3.3-7. The example is given only for phase A as the circuit for the other phases is the same. The circuit must allow for bidirectional current sensing; hence the non-inverting input of the operational amplifier is offset by 1.65 V which corresponds to zero current. The board uses 0.04 Ω shunt resistors, which would allow us to measure a maximum current of [11]

$$I_{\max} = \frac{\Delta V}{R_{\text{shunt}}} = \frac{1.65}{0.04} = 40 \text{ A}. \tag{3.3-3}$$

Nevertheless, the maximum current of the used IGBTs is 20 A, hence the amplifier has a gain of [11]

$$G = \frac{R_{33}}{R_{30}} = \frac{2.1}{1} = 2.1 \tag{3.3-4}$$

to maximize the resolution of the measurement. The output of the operational amplifier is filtered by a low-pass RC filter with a cut-off frequency of [11]

$$\omega_{\text{c}} = \frac{1}{R_{31}C_{25}} = \frac{1}{10^3 \cdot 330 \cdot 10^{-12}} \approx 3 \text{ MHz}. \tag{3.3-5}$$

The amplified voltage signal proportional to the phase current is available at the 34-pin control connector and can be directly connected to an ADC channel of the microprocessor. Alternatively, the user can access directly the voltage drop on the shunt resistor and process it with a custom-made operational amplifier circuit or the integrated amplifiers of a microprocessor.

The amplifiers operate from a single 3.3 V supply which has to be provided externally to the power board. The 3.3 V is also used to create the 1.65 V offset voltage for the non-inverting inputs, using a simple voltage follower circuit, see figure 3.3-8.

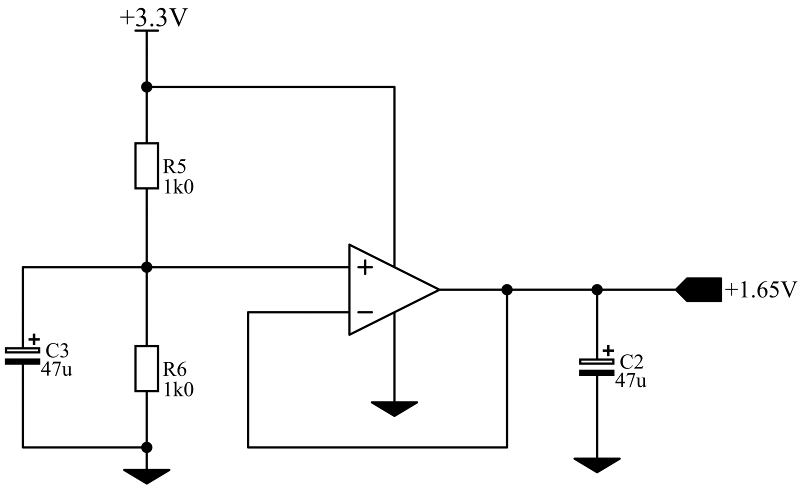


Figure 3.3-8: Voltage follower circuit to create an offset for current measurement.

To achieve a proper functionality of any FOC-based control algorithm, the DC bus voltage must also be measured. This can be easily achieved with a voltage divider, see figure 3.3-9. The divider is calculated as

$$V_{\mathrm{DC(max)}} = 3.3\frac{R_1 + R_2 + R_4}{R_4} = 3.3\frac{2 \cdot 470 + 7.5}{7.5} = 417\ \mathrm{V}. \tag{3.3-6}$$

The output of the divider is clamped to 3.3 V through a diode D_1 which ensures that no more than 3.3 V is sent to the microprocessor.

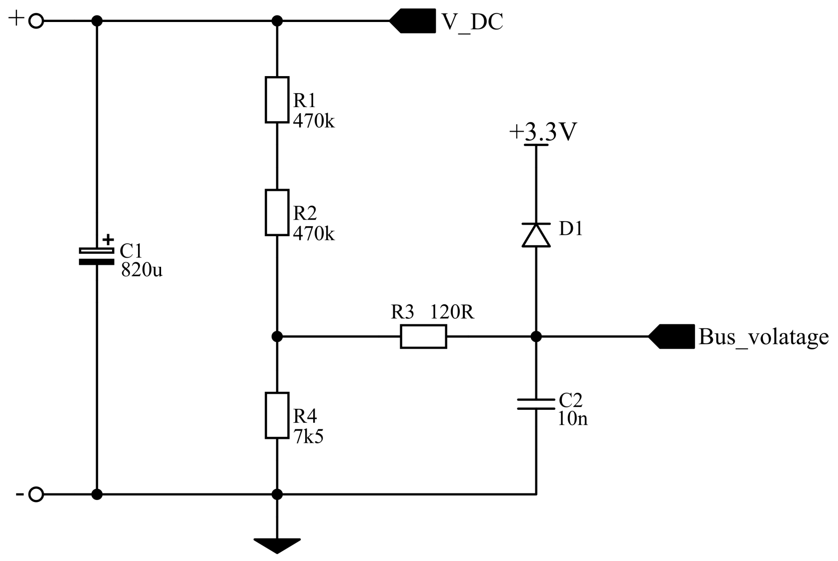


Figure 3.3-9: DC bus voltage sensing circuit [11].

3.4 Evaluation Board

The LAUNCHXL-F280049C evaluation board serves as the control board of the system, its structure together with the main features are shown in figure 3.4-10. The board is built around the TMS320F280049C microprocessor and it is meant to be used with the Booster Packs. Those are compact motor drive boards from Texas Instruments capable of driving small AC motors. They can be connected via the two 20-pin connectors, the launchpad is capable of supporting a total of two of them [12]. These pin connectors can also be used to connect the board to a custom set up and are used as such in this thesis. The TMS320F280049C microprocessor is covered in a separate chapter, hence it is not covered here.

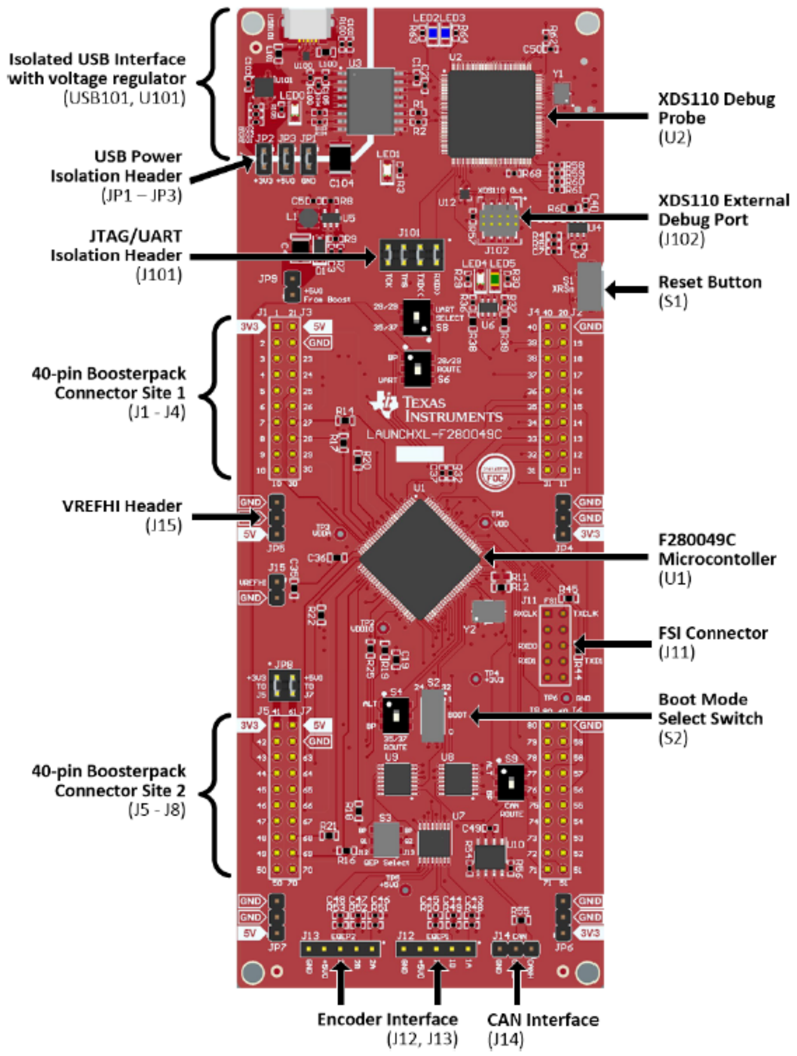

Isolated USB Interface with voltage regulator (USB101, U101)
USB Power Isolation Header (JP1 – JP3)
JTAG/UART Isolation Header (J101)
40-pin Boosterpack Connector Site 1 (J1 - J4)
VREFHI Header (J15)
40-pin Boosterpack Connector Site 2 (J5 - J8)
XDS110 Debug Probe (U2)
XDS110 External Debug Port (J102)
Reset Button (S1)
F280049C Microcontoller (U1)
FSI Connector (J11)
Boot Mode Select Switch (S2)
Encoder Interface (J12, J13)
CAN Interface (J14)
TEXAS INSTRUMENTS
LAUNCHXL-F280049C

Figure 3.4-10: The LAUNCHXL-F280049C evaluation board [12].

3.4.1 Debugging Capabilities

The board features the XDS110 debug probe that enables live debugging of the TMS320F280049C. The core of the probe is the MSP432E401YTPDT 120 MHz, 32-bit Arm Cortex microprocessor, pre-programmed with a debugging software. The probe is connected to the main microprocessor via the compact JTAG (cJTAG) interface. Unlike the standard JTAG (IEEE 1149.1) that uses five signals [12]

1. TDI (Test Data In),
2. TDO (Test Data Out),
3. TCK (Test Clock),
4. TMS (Test Mode Select),
5. TRST (Test Reset),

the cJTAG uses only two signals

1. TMS (Test Serial Data),
2. TCK (Test Clock).

Moreover, the debug probe also creates an auxiliary virtual COM port in the computer to which it is connected. The virtual COM port can be connected to an UART channel of the main microprocessor. Thus, enhancing the user's communication capabilities with the microprocessor [12]. This feature is used further in the thesis to create a real time monitoring interface that receives and sends data to the microprocessor. The debug probe converts both the JTAG and UART signals into USB signals, which can be accessed through the on-board micro-USB connector.

3.4.2 Connectivity

As mentioned previously, the board is equipped with four 20-pin connectors which include GPIOs, analog inputs as well as power and ground pins. The GPIOs can be mapped to PWM channels, various communication peripherals such as UART, I2C, or SPI or used as inputs to the eQEP unit for speed and position measurement [12].

Furthermore, the board includes a dedicated CAN interface, two encoder interfaces and a connector for external voltage reference for the ADC. The on-board debug probe can be disconnected from the main microprocessor and used as a standalone debug probe. To debug an external device, the user has to disconnect the main microprocessor by removing jumpers form header J101 and connect to the external device via the J102 debug port, see figure 3.4-10 [12].

To communicate with an external computer the board uses a micro-USB connector. The USB interface has the capability of being isolated from the rest of the board, and, thus, protecting the connected computer. To enable the USB isolation, three jumpers JP1-JP3 must be removed. When the USB interface is isolated, the board cannot be powered from the computer and must be powered from an external source [12].

3.4.3 Other Features

The board also features a 20 MHz quartz crystal oscillator that can be used as clock source for the microprocessor instead of its internal oscillators. The crystal oscillator is essential if, for example, CAN is used as the internal oscillators are not capable of meeting its critical timing requirements. Moreover, multiple on-board switches can be used to route the UART signals either to the debug probe or the pin headers. Similarly, the eQEP signals can be taken from the dedicated encoder connectors J12, J13 or the pin headers. Additionally, two user LEDs (LED4, LED5) are present on the board and can be freely used by the application software [12].

3.5 Signal Interface Board

To isolate the evaluation board from the inverter, thus making the set up safer to work with, an interface board was designed. The board transfers digital signals, i.e., PWM signals, analog signals, i.e., phase currents, DC bus voltage and inverter module temperature and power through a galvanically isolated interface. Furthermore, it processes the differential signals from the incremental encoder and converts them to single-ended ones with appropriate 3.3 V logic level. To ease orientation in the circuit diagrams, the suffix INV denotes that the signal is referenced to the inverter ground IGND, while MCU denotes that the signal is referenced to the microprocessor ground MGND. The designed board with the main subsystems highlighted is depicted in figure 3.5-11. The following chapters explain each subsystem in detail. If a subsystem contains multiple copies of the same circuit, the corresponding circuit diagram is only shown once. The complete board layout can be found in the attachment.

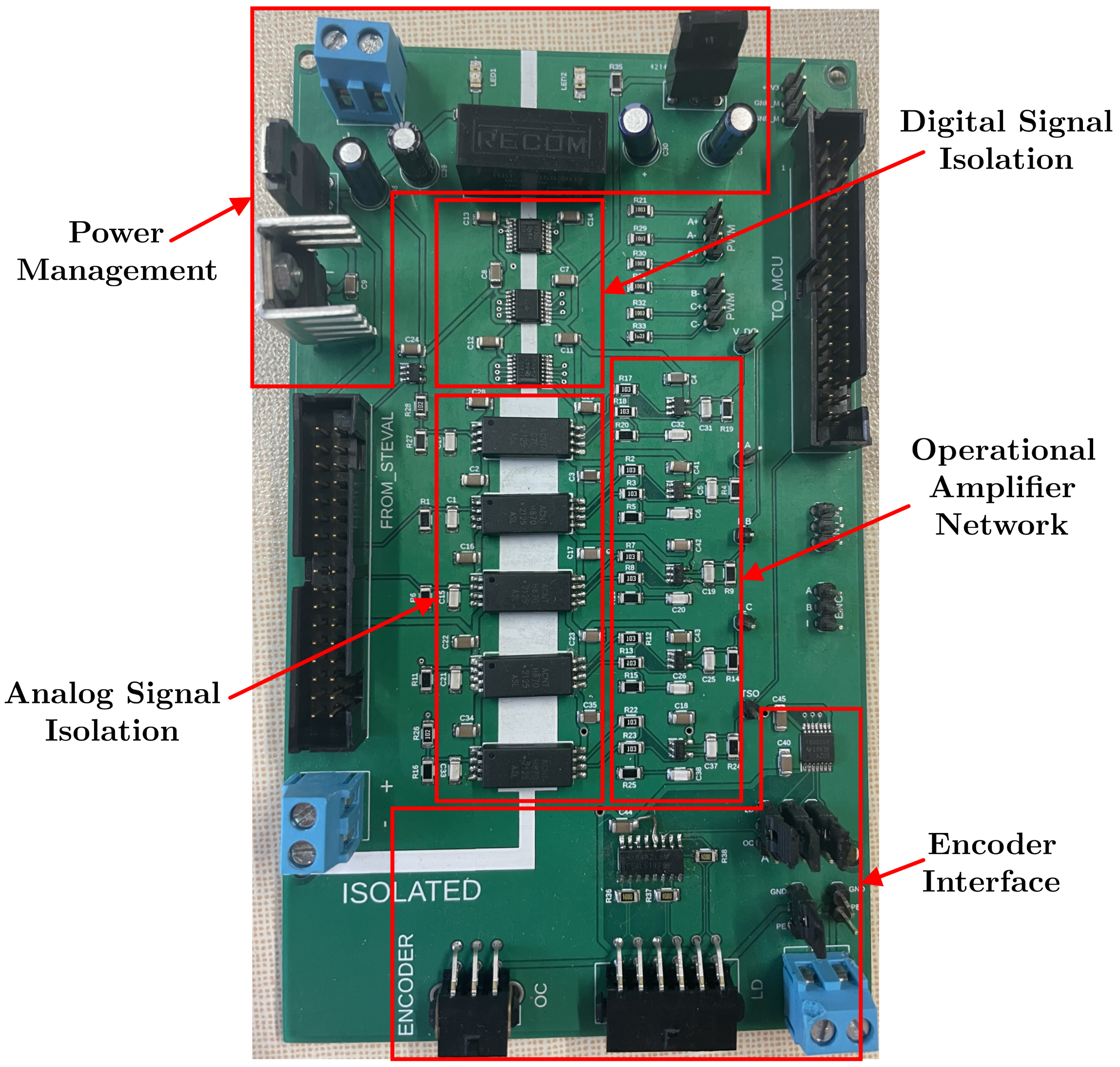


Figure 3.5-11: Designed signal interface board.

3.5.1 Power Management

Power to the board is brought on the inverter side from the 15 V switch-mode DC power supply. The supply voltage is routed to a set of output connectors to pass it to the inverter. This is done purely to ease the wiring process. The isolation amplifiers used to transfer all the analog signals require a 5 V supply on the input side, while the digital isolators as well as the operational amplifiers on the power board need 3.3 V. Two linear voltage regulators are used to step down the input voltage of 15 V to 5 V and 3.3 V. The circuits are depicted in figure 3.5-12.

A standard 7805 regulator is used to step down the 15 V supply to 5 V output voltage. Decoupling capacitors are placed at both input and output pins as given in the datasheet [16]. A BA033ST voltage regulator steps down the 5 V supply further to 3.3 V. The regulator could also use directly the 15 V supply like the 7805, this, however, would lead to much greater losses, hence the 5 V is used as the input. Furthermore, the datasheet [17] does not give any specifications on capacitors at the input or output. Since the regulator is placed physically close to the 7805, a decoupling capacitor should not be needed at the input. Whereas a 22 μF electrolytic capacitor is placed at the output to prevent any fluctuations. To signal that the inverter side is powered up, an LED is used.

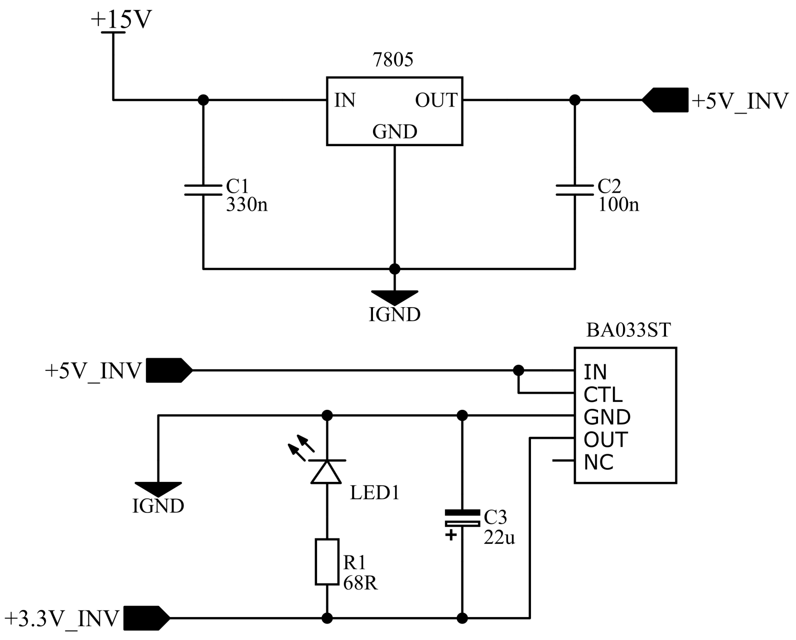


Figure 3.5-12: Power management on the inverter side.

To transfer the power from the inverter side to the microcontroller side, a RSE-2405SZ/H2 step down DC-DC switching converter is used, see figure 3.5-13. The converter is supplied from the 15 V supply, a 47 μF electrolytic capacitor is placed at the input to ensure a stable supply voltage. The converter is also equipped with an active-low enable signal which is connected to ground to keep the converter enabled at all times. To ensure a stable output voltage, a 22 μF electrolytic capacitor is placed across the output. The capacitor values were chosen to match the maximum capacitance constrains from the manufacturer datasheet [18]. The maximum output current of the converter is 400 mA, giving it a total output power of 2 W, which should suffice. Pretty much all the circuits on the MCU side including the microprocessor itself operate from a 3.3 V supply rail. The exact same circuit as on the inverter side with the BA033ST voltage regulator is used to step down the 5 V to 3.3 V, see figure 3.5-13, an LED indicates that the power is present on the MCU side.

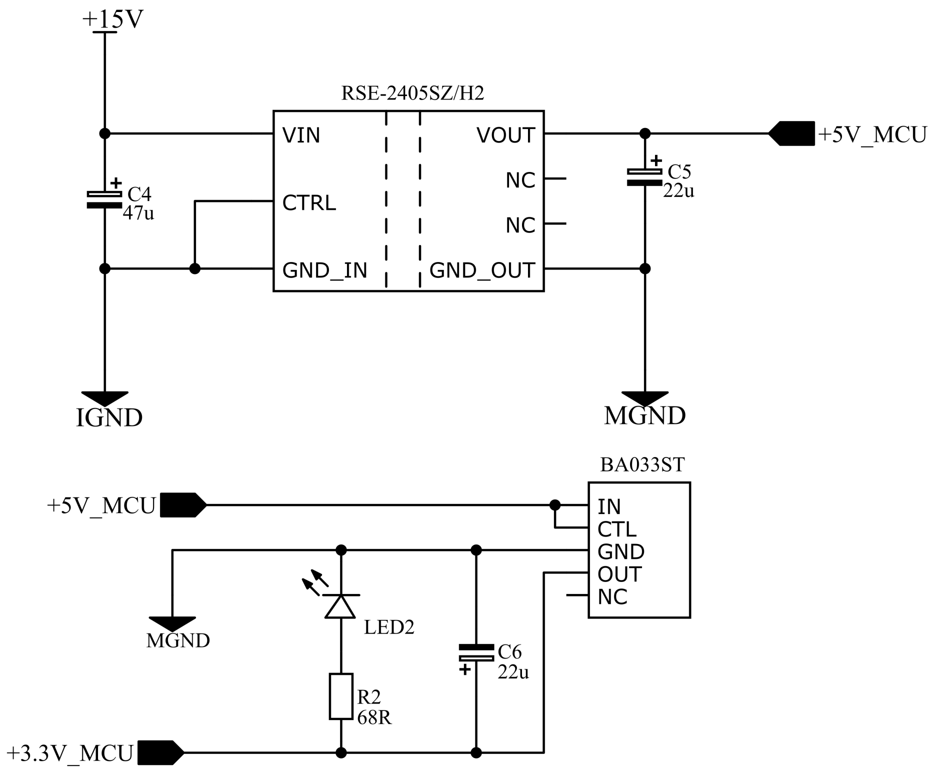


Figure 3.5-13: Power management on the MCU side.

3.5.2 Isolating Digital Signals

To be able to control the inverter, a total of six PWM signals need to be passed from to MCU side to the inverer side. To do so, two ISO7730QDBQRQ1 triple-channel digital isolators were used. Each channel supports up to 100 Mbps data rate with a propagation delay of 11 ns, thus being more than capable of transferring the rather low frequency PWM signals. Detailed information about the circuit, including its ratings, specifications as well as principle of operation, can be found in the manufacturer datasheet [19].

Connecting the circuit is rather simple, see figure 3.5-14, as it requires only a 100 nF decoupling capacitor on each side power supply. The output side also features an active-high enable pin which is routed to the 3.3 V supply to enable the circuit as soon as it is powered up. After some testing it became clear that the input pins must have a pull-down resistor. The reason is that when the MCU is powered up, all its GPIO pins are in a high impedance state by default, therefore no logic level is guaranteed and any voltage from 0 up to the 3.3 V supply can appear on them at any point in time. Any voltage that is greater than 0.7 V_{CC} is considered as logic 1 by the isolators, causing the corresponding output to be pulled high. Since the inverer module does not have a built-in dead time, both transistors in one of its legs could be accidentally switched on simultaneously, shorting the DC bus and burning the invereter. The pull-down resistors ensure that a well-defined voltage level is always present at the input pins of the isolators.

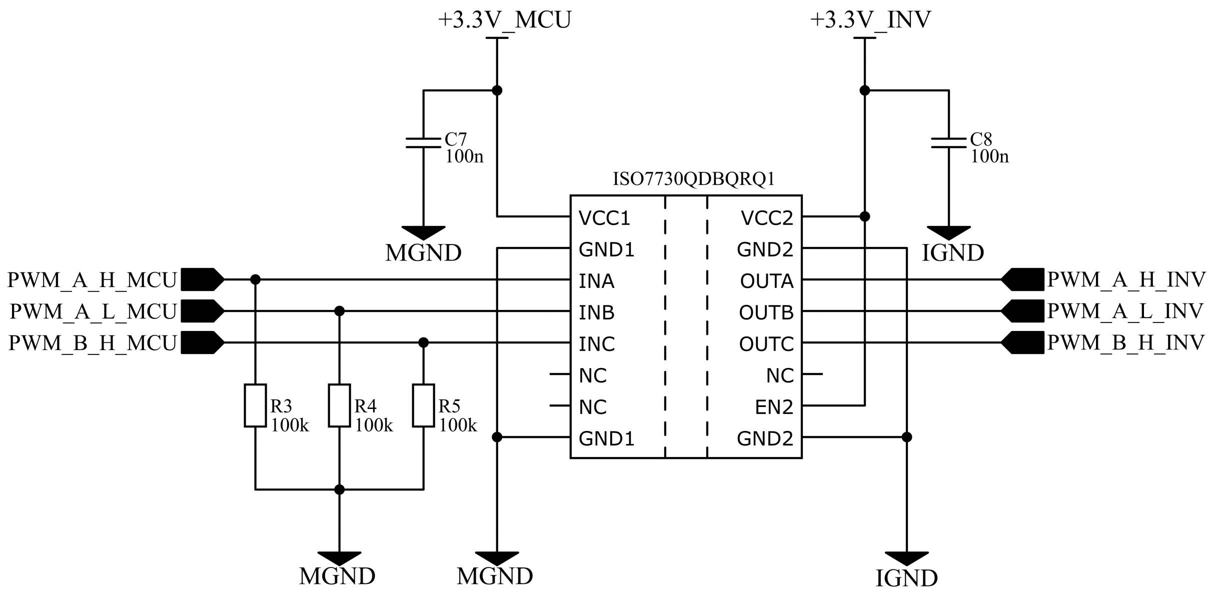


Figure 3.5-14: Galvanic isolation of PWM signals.

3.5.3 Isolating Analog Signals

While isolating the digital signals is straightforward, isolating the analog signals is more complicated mainly due to the limitations posed by the used power board. The main issue lies in the current measurement that is already present on the board, see figure 3.3-7. Isolating both the DC bus voltage measurement as well as the output of the temperature sensor is less complicated, hence a brief research into possible ways to implement a galvanically isolated current measurement is presented.

3.5.3.1 Possible Solutions to Realize Isolated Current Measurement

Using a magnetically coupled current sensor would seem like a natural choice as the measurement by design galvanically isolates the measured circuit from the rest. Such sensors are now available in a compact package that can be easily mounted on a PCB. An example of such sensor is the TLI4971 from Infineon. The sensor can operate in three different modes, i.e., single-ended mode, fully-differential mode and semi-differential mode, the latter one is the default operating mode. The semi-differential mode behaves essentially like the circuit in figure 3.3-7. The sensor outputs a voltage proportional to the measured current, the quiescent voltage is set to 1.65 V and the sensitivity depends on the full-scale range of the sensor. The operating modes as well as the measurement range can be programmed by the user, details are given in [20]. This, in the author's opinion, is the most convenient way to implement an isolated current measurement. Unfortunately, using them is not possible in this thesis as they cannot be integrated to the already made inverter board.

Another option, more suitable for this thesis, is an optically isolated amplifier as it is suitable for shunt measurement. A principal block diagram of such amplifier is depicted in figure 3.5-15. The amplifier first measures the single-ended input voltage over the shunt resistor, the analog voltage is then converter to a digital bit stream using a sigma-delta modulator. The digital signal is transmitted across the optically coupled pair made of a LED and a photodetector. The detector converts the received optical signal back to electrical signal. The photodetector outputs current proportional to the received light, hence a transimpedance amplifier (TIA), which is just a current-to-voltage converter, is used to convert the photocurrent into voltage. The voltage signal is then decoded and filtered to convert it back to an analog signal. The output voltage, usually in a differential mode for better common-mode noise immunity, is proportional to the input voltage with some gain value specified in the datasheet. Alternatively, the digital bit stream does not have to be decoded and converted back to an analog signal, but rather send as it is to the microprocessor. Most DSPs nowadays feature a delta-sigma demodulator that can be used to extract the information about the measured signal directly. Sending the digital bit stream directly to the DSP is perhaps more convenient. Mainly since the analog differential output must be converted to a single-ended signal using a differential amplifier before passing it to the DSP. Adding additional complexity and delay to the circuit and inevitably extra noise to the signal. Alternatively, two ADC channels per signal would have to be used to process the differential signal [21], [22].

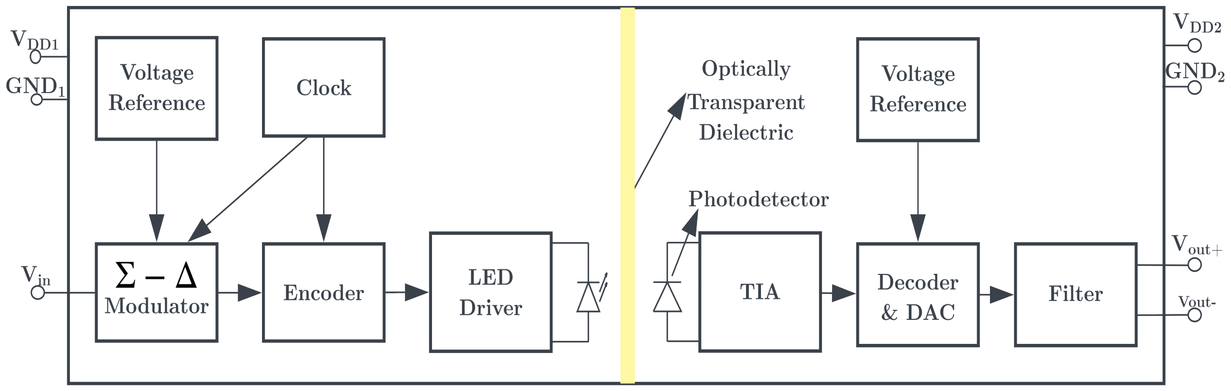


Figure 3.5-15: Functional block diagram of an optically isolated amplifier with an analog output [22] (edited).

Alternatively, an external ADC module could technically be used to sample the currents and send the measured values via some serial communication protocol such as SPI or CAN to the microprocessor. The communication data signals would be isolated as shown in figure 3.5-14. Nevertheless, such solution, in the author's opinion, introduces unnecessary complications for the control algorithm as the motor currents should be sampled either when the PWM counter reaches zero or its maximum value [5]. The control algorithm would have to precisely trigger the external ADC and manage the serial communication. The communication protocol would have to include some kind of check such as a CRC to verify that the received value is correct, adding additional complexity. Otherwise, the corrupted data might cause the control to crash.

3.5.3.2 Implemented Solution

An optically isolated amplifier was chosen for this application. However, all the ones purposefully designed for current measurement can only sustain few hundreds of millivolts at the input. The maximum input voltage is usually around ±200 mV, sometimes even below ±100 mV. The power board uses 0.04 Ω shunt resistors. The board is designed to operate with currents up to 20 A which give us a voltage drop of

$$\Delta V_{\text{shunt}} = R_{\text{shunt}} I = 0.04 \cdot \pm 20 = \pm 0.8\ \text{V}. \tag{3.5-7}$$

Which is clearly way over the limit of the amplifiers. Besides, using directly the voltage drop over the shunt resistor would require to transfer it from the power board to the interface board.

Making it susceptible to noise especially at low currents where the voltage drop is only a few dozen millivolts. Therefore, the amplified signal from the on-board operational amplifiers must be used, see figure 3.3-7.

The ACNT-H870-000E isolation amplifier from Broadcom [23] is used to isolate all the analog signals. The amplifier is designed mainly for voltage measurement; hence it can take up to 2 V at the input and, unlike the current-sensing amplifiers, has an unity gain. A potential downside might be its lower bandwidth of 100 kHz. Nevertheless, the output of the operational amplifiers can swing from 0 V up to 3.3 V, thus the voltage level is first stepped down by a voltage divider before entering the isolation amplifier. The voltage divider is made from the 1 kΩ resistor that is part of the low pass filter at the output of the operational amplifier, see figure 3.3-7, and a 1.54 kΩ resistor on the interface board. The divider ensures that at maximum current, the input of the isolation amplifier is at 2 V since

$$V_{\text{in}} = 3.3\frac{1.54}{1.54+1} = 2\ \text{V}. \tag{3.5-8}$$

The isolation amplifier has a differential output; hence a differential amplifier is used to convert the signal to single-ended one before passing it to the microprocessor. The MAX44259AUK+T rail-to-rail operational amplifier [24] was chosen. The full circuit is depicted in figure 3.5-16, all the bypass and decoupling capacitors were chosen according to the respective datasheets [23], [24]. Furthermore, since the output of the isolation amplifier can only go up to 2 V, the differential amplifier amplifies the signal so that when the input is at 2 V, the output is 3.3 V. Therefore, the gain must be

$$G = \frac{R_{10}}{R_7} = \frac{16.5}{10} = 1.65 = \frac{3.3}{2}. \tag{3.5-9}$$

The other phase currents use the same circuit. The temperature sensor does not have any resistor at the output so the whole voltage divider is implemented on the interface board. Since the DC bus voltage is measured with a voltage divider already, the stepped down voltage is first buffered using a voltage follower circuit and subsequently stepped down further.

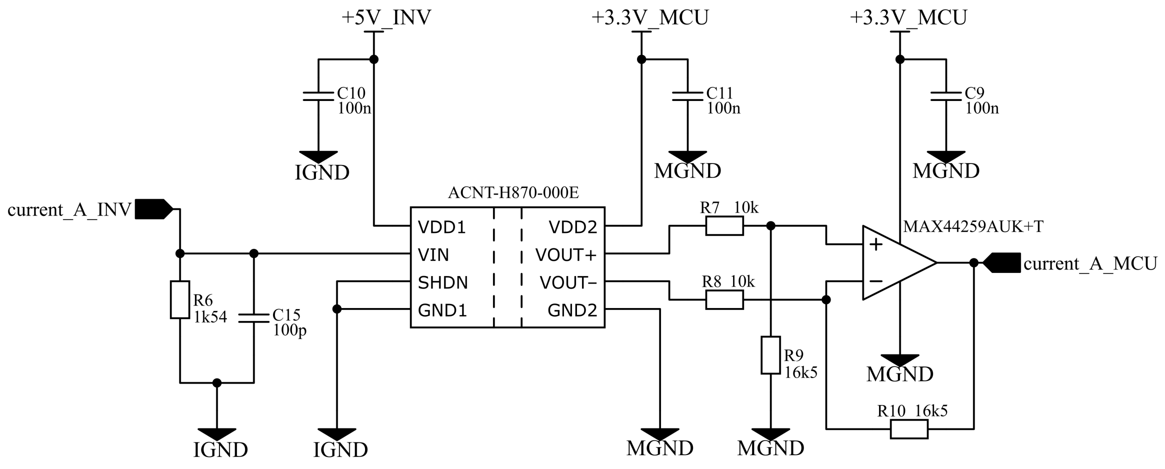


Figure 3.5-16: Isolation of a phase current measurement.

3.5.4 Incremental Encoder Interface

The motor that is going to be controlled later in the thesis utilizes an incremental encoder with differential outputs. Therefore, a proper receiver is required to convert the differential signals into single-ended ones, which can be processed by the microprocessor. The encoder datasheet specifies that a RS-422 type receiver should be used to process the encoder signals, thus a AM26C32CD RS-422 receiver from Texas Instruments [25] was chosen. In order to ensure signal integrity, proper impedance matching of the inputs is required. The datasheet [25] specifies that the termination resistor should be between 80 Ω to 120 Ω, while being within 20% of the characteristic impedance of the cable. Since, however, the characteristic impedance of the encoder cable is unknown, multiple values of the termination resistor were tried and 100 Ω was selected as the final value. The high-level output voltage of the AM26C32CD can be from 3.8 V up to the supply voltage, depending on the differential input voltage. Therefore, a TXU0104PWR voltage-level translator from Texas Instruments [26] is used to match the logic level to those of the microprocessor. The full circuit is depicted in figure 3.5-17.

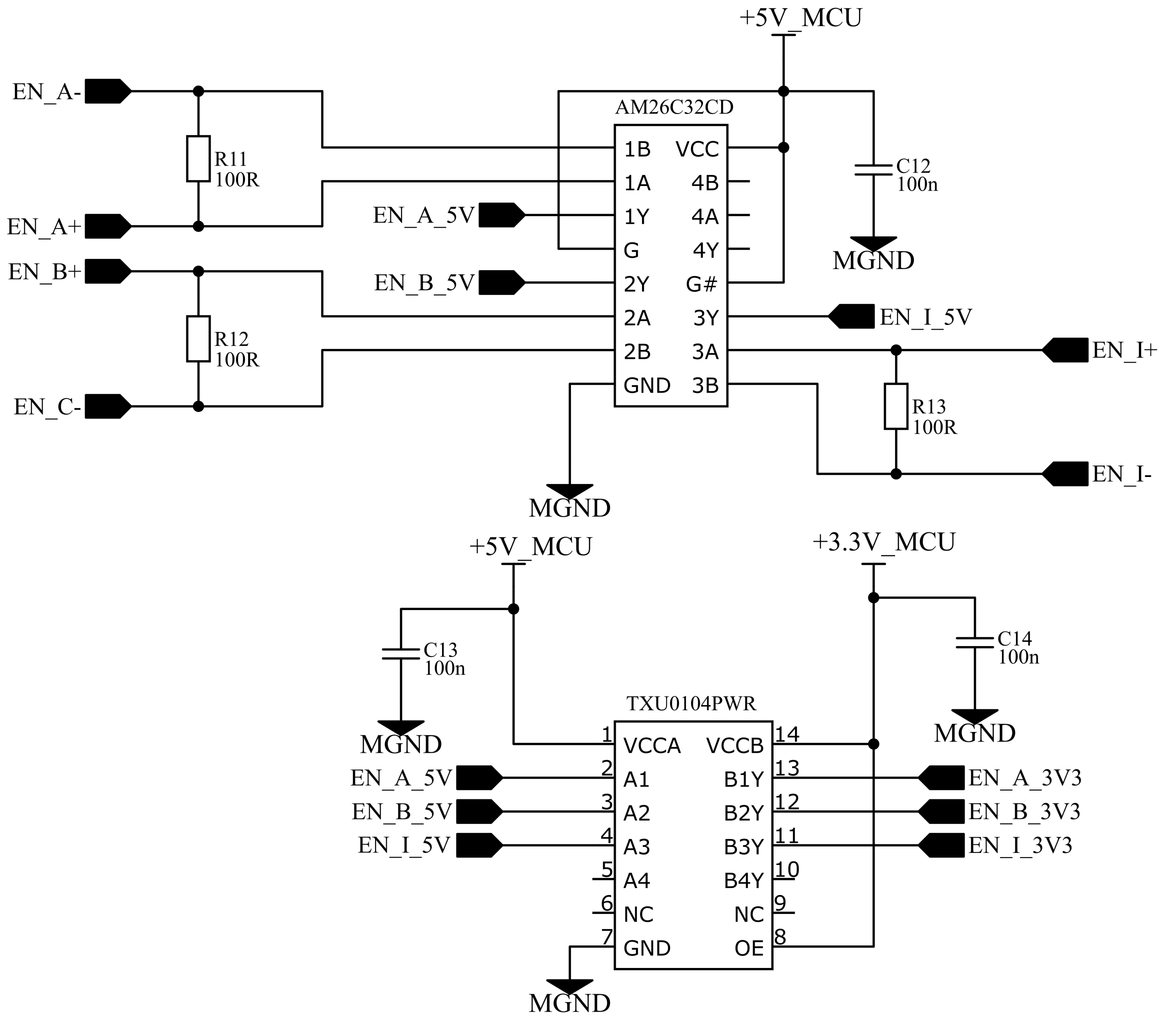


Figure 3.5-17: Incremental encoder interface with a RS-422 type receiver.

3.5.5 Assembled Inverter Platform

The final assembled inverter platform is depicted in figure 3.5-18. All the components are mounted on a board made from an electrically insulating material. A pair of handles was also mounted on the board to ease manipulation. The inverter module cannot be seen as it is located underneath the inverter board to allow for better attachment to a heat sink. Perhaps the only part that has not been covered yet is the Signal transfer board. It plugs on top of the Launchpad board and just transfers the signals form the Signal interface board to the F280049C microprocessor. The board also features a RS-232 driver and a D-sub type female connector. This was meant to be used to connect a computer to the UART port of the microprocessor. However, later it was found out that the UART port can be accessed through the XDS110 Debug probe. So, the original serial connector is redundant. The schematic and layout of the Signal transfer board can be found in the attachment.

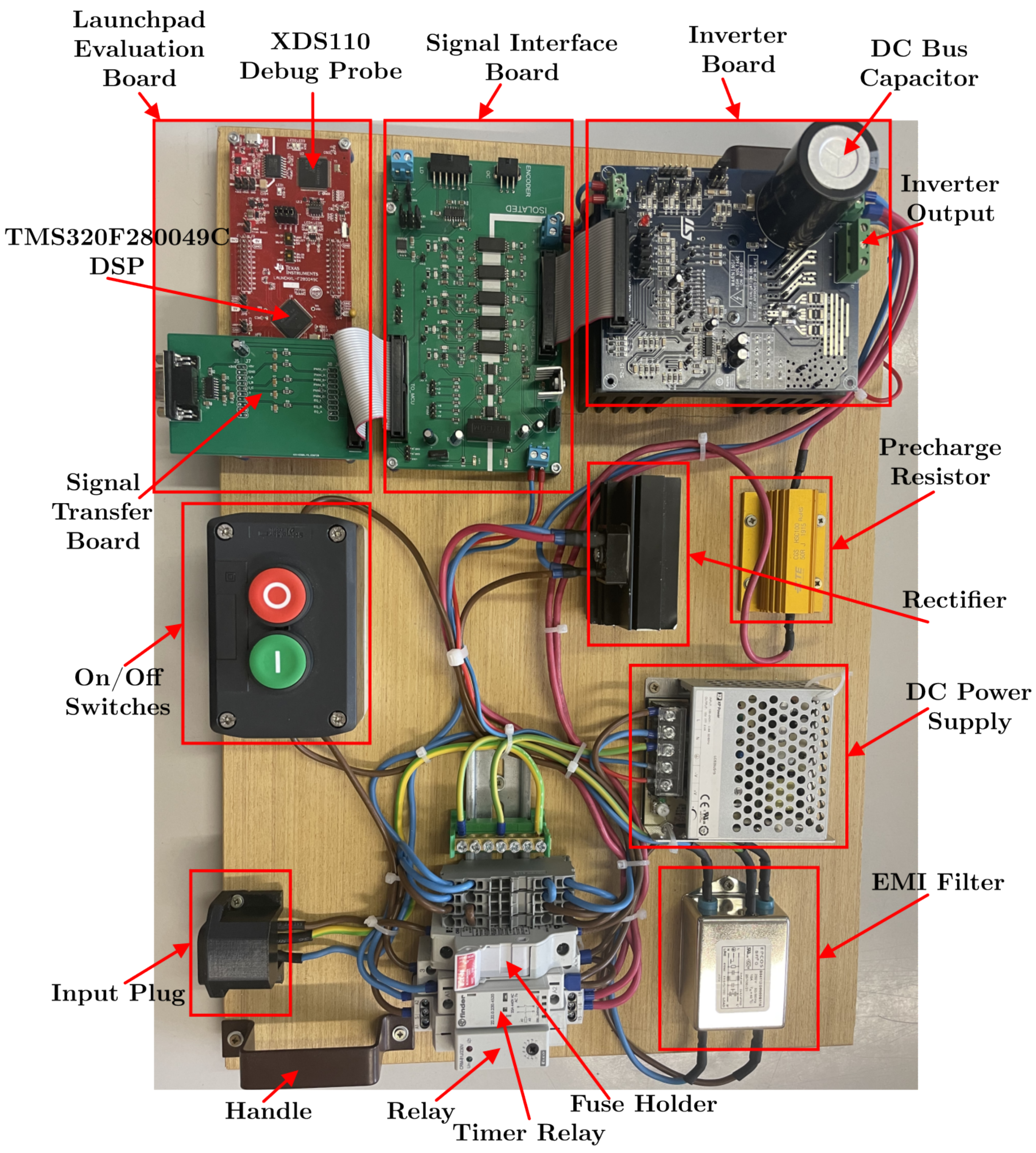


Figure 3.5-18: The final assembled inverter platform.

3.5.6 Complications During the Design Process

It was thought that the encoder does not have to be connected to the inverter side of the interface board. Mainly as it would not be necessary to isolate the signals. Furthermore, the extensive switching noise from the inverter might corrupt the signal. The digital isolator considers everything below $0.3V_{\mathrm{CC}} \approx 1.1$ V as logic 0 and everything above $0.7V_{\mathrm{CC}} \approx 2.1$ V as logic 1. If the input signal contained a large amount of noise of sufficient magnitude, the isolator would essentially administer false pulses. These false pulses would be picked up by the microprocessor, causing an imprecise speed and position measurement. The used encoder has 2500 pulses per revolution, which on one hand enables a precise position measurement, but on the other hand makes the signal resiliency to noise quite poor. So, the decision was made to connect the encoder to the MCU side of the interface board.

By doing so, the motor is essentially connected to two different reference grounds, i.e., the inverter ground and the MCU ground. It might seem that the motor winding is not connected to any ground reference, and that is true if the inverter is not switching. If, however, the inverter starts switching, each time when a low-side IGBT is turned on, the corresponding winding is connected to the inverter ground, while the encoder is always connected to the MCU ground through its negative supply pin.

The encoder is obviously not electrically connected to the winding and likewise the grounds are isolated from each other. That is unfortunately the case only with respect to DC as there exists a parasitic capacitive coupling between all the aforementioned systems, essentially creating a capacitive loop through which an AC current can still pass. Unfortunately, that is exactly what happens in the designed set up. When both the motor phases and the encoder are connected and the invereter starts switching, a current starts to flow through the capacitive loop. The current causes the MCU ground potential to oscillate significantly, corrupting essentially all signals on that side of the board. The microprocessor is unable to connect to a computer for debugging since the ground oscillations corrupt the JTAG signals. The analog signals also contain a significant offset, making it impossible to accurately measure anything.

Regrettably, such eventuality was not foreseen at the time of designing the board. Since there was not enough time to redesign the board while still finishing the thesis in time, the decision was made to connect the inverter and MCU grounds together. By doing so, the galvanic isolation is bypassed but also the capacitive loop is broken. With the grounds connected, no potential difference can be between them, and thus no current can flow. Under these circumstances the set up must be powered from an isolation transformer for safety reasons. The interface board is still in use to show that the concept of the isolation is functional. The board is planned to be redesigned with the encoder interface located on the inverter side which should solve the issue with the capacitive coupling. Furthermore, the differential signaling, and cable shielding seem to provide sufficient noise immunity so isolating the encoder signals should not be much of a problem.

3.5.7 Planned Improvements

Apart from the redesign of the interface board to improve the noise immunity discussed above, the inverter is missing an important part. As of now the inverter does not have any significant braking capability. The diode bridge does not allow to send the excess energy generated during braking to the grid. Therefore, everything is going to go to the DC bus capacitor. While the capacitor is rated for 450 V, it will get charged quickly. Exceeding the rated voltage could destroy the capacitor. To avoid the destruction, a simple braking chopper can be used, see figure 3.5-19.

The chopper is just a transistor and a power resistor with a freewheeling diode. The diode is there to protect the MOSFET from inductive voltage spikes. The MOSFET could be controlled either form the microprocessor or in an analog manner using comparators. Either way the diagram depicted in figure 3.5-19 omits a gate driver circuit. The principle of operation is simple. Whenever the DC bus voltage exceeds a given threshold the MOSFET starts switching and the excess power is being dissipated in the power resistor.

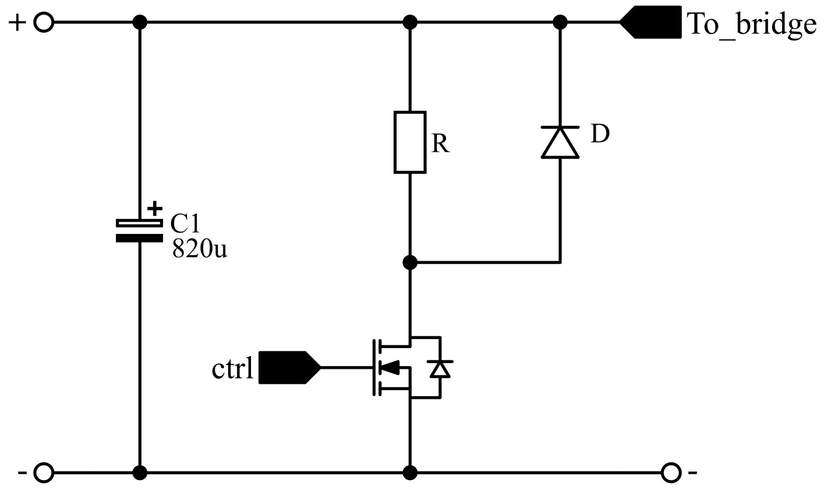


Figure 3.5-19: Circuit diagram of a braking chopper.

Therefore, it is planned to build a simple braking chopper that will be connected to the DC bus. The control will probably be done from the microprocessor as it appears to be the easiest solution. For the initial testing presented later in the thesis the absence of a proper braking capabilities is not an issue since the driven motor will operate under no load. However, for future applications the breaking chopper is undoubtedly necessary.

Chapter 4

TMS320F280049C Digital Signal Processor

The brain of the designed inverter platform is the TMS320F280049C, F280049C for short, microprocessor from Texas Instruments. It belongs to the C2000 family of digital signal processors designated for demanding real-time control applications. The following sections present a brief overview of the microprocessor internal structure, as well as its specific features applicable to real-time control. Lastly, an approach to configuring the microprocessor for a basic FOC is presented. Including the implementation of overcurrent protection utilizing the on-board comparators and setting up simple serial communication over UART for monitoring the control algorithm.

4.1 Structure Overview

The functional block diagram of the F280049C is depicted in figure 4.1-1. The core of the microprocessor is a 32-bit fixed point C28x CPU. The CPU implements a RISC (reduced instruction set computer) architecture which allows the CPU to use single-cycle instructions, register-to-register operations, and a modified Harvard architecture. The architecture enables the CPU to fetch instructions and data from memory simultaneously. Furthermore, to enhance the computational capability of the fixed point C28x CPU, the F280049C features a standalone floating point processor (FPU). The FPU adds instructions and supporting registers to perform IEEE single-precision floating point operations [27], [28].

The maximum supported clock rate is 100 MHz, the clock rate depends on the used oscillator and the subsequent configuration of the internal PLL. The microprocessor features two 10 MHz internal oscillators. Alternatively, an external crystal oscillator can be connected to the designated pins in applications requiring precise timing. The processing power is further enhanced by an eight-stage pipeline, which allows the CPU to execute essentially eight instructions in parallel. The F280049C is equipped with a total of 100 KB of on-chip RAM and 256 KB of on-chip flash memory. It does not, however, support external memory, therefore if the on-chip memory is not sufficient for the application, different microprocessor must be chosen [27], [28].

To process external analog signals, F280049C has three 12-bit analog to digital converters (ADCs). With each module supporting up to sixteen different input channels. Having three ADCs allows for simultaneous sampling of, for example, motor phase currents, enabling the user to implement faster control loops. To utilize the full resolution of the ADCs when a small signal is to be sampled, the signal can be amplified by one of the integrated programmable-gain amplifiers (PGA) before being sent to the ADC. Moreover, the F280049C features seven comparator subsystems (CMPSS). Each subsystem features two analog comparators, each with a reference 12-bit digital to analog converter (DAC). These comparators can be connected to one of the analog pins to implement e.g., fast overcurrent protection or peak current control of a switching converter [27], [28].

To drive an external power converter the microprocessor has eight PWM modules, corresponding to sixteen PWM output channels. Two enhance quadrature capture units are present. These are essential for processing signals from rotatory encoders which are used for measuring speed and rotor position of a electric machine. Furthermore, a variety of industry-standard serial communication interfaces, e.g., CAN, UART, SPI and I2C, are featured. Only the essential features critical for motor control are listed here, the microprocessor offers much more, a thorough description is found in the user manual [28].

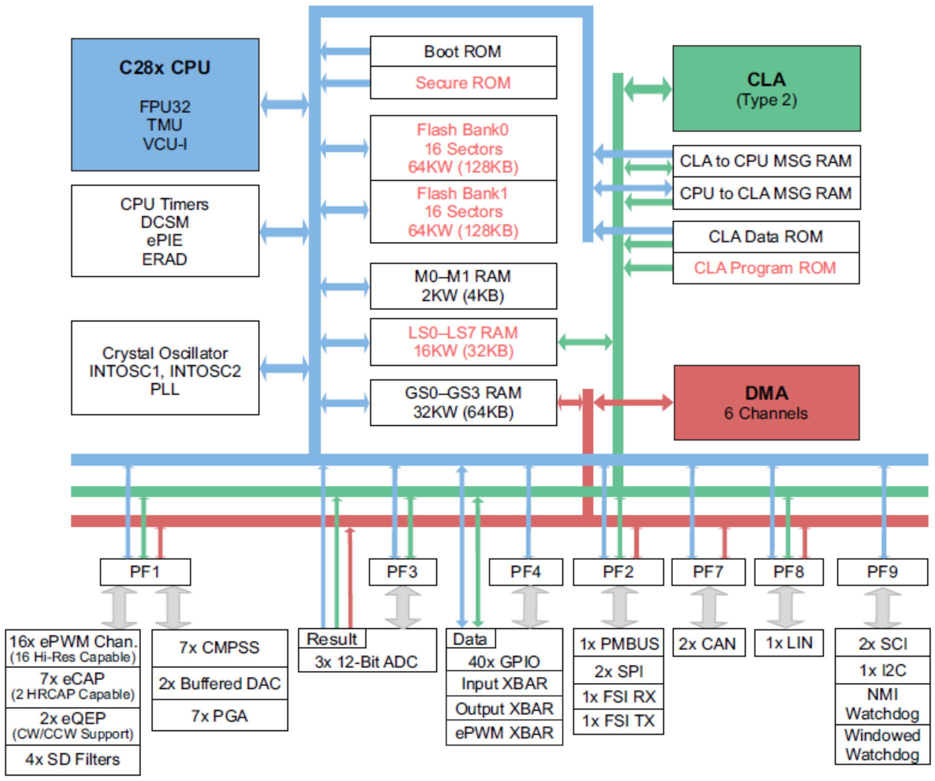


Figure 4.1-1: Functional block diagram of the TMS320F280049C [27].

4.2 Essential Development Tools

To facilitate the development process Texas Instruments offers their proprietary integrated development environment (IDE) Code Composer Studio (CCS). Which is meant to be used with all of their processors and microcontrollers. Furthermore, processors from the C2000 family can utilize the C2000Ware package containing software and other supporting material to further ease the development process.

4.2.1 Code Composer Studio

The Code Composer Studio is an IDE designed for software development in C/C++. It is built upon the Eclipse IDE platform and can be downloaded for free from Texas Instruments website. The environment features C/C++ compilers designated for the different families of processors. Moreover, an extensive debugging environment is included which enables the user to debug their application in real-time. Detailed list of all its features can be found at [29].

4.2.2 C2000Ware

The C2000Ware is an extensive set of software tools designated for the C2000 family of microprocessors. The package includes example projects for the different devices. These are especially helpful when trying to get familiar with the different features of the chosen device. Furthermore, various libraries are included as well. These can be integrated to the user application to optimize e.g., the computational speed of complex mathematical functions. Comprehensive documentation is also featured in the package as well as design files and schematic for all the different evaluation boards such as the Launchpads. The C2000Ware can be downloaded directly from CCS or from the Texas Instruments website [30].

4.3 Advanced Features for Real-Time Control Applications

Apart from the standard control peripherals listed above, the F280049C has a variety of sophisticated features designated for computationally intensive real-time control applications. These features are not limited to just the F280049C and in fact can be found on many other C2000 devices. To find out which ones are included in which device, the user should consult this document [31]. The document contains a comprehensive list of all peripherals included in each different C2000 device. The following sections give a brief introduction into those that are, in author's opinion, the most interesting.

4.3.1 Extended Instruction set

The C28x CPU which forms the core of any C2000 microprocessor is by design a fixed-point CPU. Therefore, its instructions only support integer operations. The full instruction set can be found in [32]. While it is possible to develop applications using only fixed-point arithmetic, it poses additional difficulties as all variables must be properly scaled first before any computations are done. A care must be taken to ensure that a proper scaling is used, otherwise a variable might overflow during a computation, corrupting the result. Furthermore, developing in fixed-point arithmetic is less intuitive for the programmer as the numerical quantities within the program do not immediately correspond to a real physical quantities.

As it has already been mentioned, the F280049C features a dedicated floating point processor (FPU), that adds a hardware support for IEEE singe precision floating point operations. The FPU comes with a set of additional instructions and registers to perform the operations. Furthermore, a special peripheral for computing trigonometric functions, called the Trigonometric Math Unit (TMU), is added. Much like the FPU, the TMU also comes with additional instructions and registers designed specifically to perform trigonometric functions, i.e., sin, cos, tan, atan and so forth. Additionally, to enable much faster serial communication, the F280049C has a dedicated support for Viterbi decoding algorithms, complex math operations and CRC calculation called the VCU unit. All the aforementioned features come with their own set of instructions that can be used in conjunction with the base C28x instruction set to optimize the application [33].

4.3.1.1 Floating Point Unit

To enable the floating point support, the user just has to select the type of FPU unit on the target device, i.e. 32-bit single precision or 64-bit double precision, in the project properties in CCS. With the floating point support enabled, the compiler maps some operations to the FPU instructions directly. Therefore, basic math operations, i.e., addition, multiplication, subtraction and division, involving floating point variables will automatically be performed by the FPU.

The FPU, however, can perform more complex operations than just simple arithmetic. One such operation is finding the fractional part of a floating point number. To do so, the FPU has an instruction `FRACF32 RaH, RbH`. The behavior is straightforward, the fractional part of a number stored in the RbH register is stored in the RaH register. To access this operation inside a C/C++ program, the inline assembly language can be used. This is done using the `asm` statement followed by a single line of assembly code. Consecutive `asm` statements place sequential lines of assembly code. To use the `FRACF32` instruction, one could write the following code [33]:

```
asm("MOF32 R1H,#2.5");
asm("FRACF32 R2H,R1H");
```

First, the number 2.5 is loaded into the R1H register, and then the R2H register is loaded with the value 0.5. Interacting with the FPU in such a way is clunky, however. Since mixing up C/C++ with blocks of assembly language reduces the readability of the code and can lead to errors. More convenient and robust way is to use the intrinsic operators recognized by the compiler. The intrinsics express the meaning of certain assembly statements and behave just like a normal C/C++ function would [33]. Using an intrinsic to access the `FRACF32` instruction is showcased using a simple program:

```
void main(void)
{
  float var1 = 2.5;
  float var2 = 0;
  float var2 = _fracf32(var1);
}
```

We notice that the intrinsic statement is preceded by an "__". This is used to discriminate them from regular C/C++ functions. Some useful intrinsics are summarized in table 4.3-1. The full list can be found in [33].

Intrinsic	Assembly instruction	Description
`float __fmax(float x, float y)`	`MAXF32 x, y`	If x>y, copy x to y.
`float __fmin(float x, float y)`	`MINF32 x, y`	If x<y, copy x to y.
`float __fsat(float x, float max, float min)`	`MAXF32 x, max` `MINF32 x, min`	Returns x if min < x < max, else return min if x < min, else return max if x > max.
`float __einvf32(float x)`	`EINVF32 x`	Returns 1/x to about 8 bits of precision.
`float __eisqrtf32(float x)`	`EISQRTF32 x`	Returns $1/\sqrt{x}$ to about 8 bits of precision.

Table 4.3-1: Examples of the FPU intrinsic statements [33].

4.3.1.2 Trigonometric Math Unit

Real-time control algorithms, especially those controlling an electric machine, have to compute the values of trigonometric functions at run time. Be it to compute the Park transform or to obtain the transformation angle from the rotor flux linkage vector components during a direct

FOC of an induction machine. Thus, a fast and accurate computation of the trigonometric functions is required.

The standard approach using a polynomial approximation is insufficient, due to its long computation time that also depends on the value of the argument. Since the rate of convergence of the polynomial changes with respect to the argument. Therefore, computations for some arguments, especially along the edges of the function domain, might take disproportionally long. A traditional way to circumvent this problem was to use look up tables. These tables were stored in the non-volatile ROM memory of the microprocessor. The tables were paired with a set of optimized functions that would compute the value using a linear interpolation within the table. A special library must be used to gain access to these optimized functions. For example, the famous TMS320F28335 microprocessor uses this approach [33].

The F280049C is equipped with the Trigonometric Math Unit (TMU). The TMU is essentially an add-on to the existing FPU enabling it to compute the trigonometric functions. The TMU adds extra instructions with supporting registers. The complete instruction list can be found in [33]. Two different versions of the TMU can be found inside the C2000 microprocessors, i.e. TMU-0, TMU-1. The TMU-1 has extra instructions for calculating the natural logarithm $\ln(x)$ and the inverse exponential $1/e^x$. To use the TMU in an application, the user must just select the version present on the target device. Provided the device has one in the first place [33].

Interacting with the TMU within a program is best accomplished by using the corresponding intrinsic statements. The most commonly used ones are summarized in table 4.3-2, while the full list can be found in [33].

Intrinsic	Assembly instruction(s)	Description

`float __sin(float x)`	`DIV2PIF32 dst, x`
	`SINPUF32 dst, x`
		Computes the sine of `x`, where `x` is in radians.

`float __cos(float x)`	`DIV2PIF32 dst, x`
	`COSPUF32 dst, x`
		Computes the cosine of `x`, where `x` is in radians.

`float __atan(float x)`	`DIV2PIF32 dst, x`
	`ATANPUF32 dst, x`
		Computes the arc tangent of `x`, where `x` is in radians.

`float __div2pif32(float x)`	`DIV2PIF32 dst, x`	Multiplies `x` by $1/2\pi$.
`float __mpy2pif32(float x)`	`MPY2PIF32 dst, x`	Multiplies `x` by 2π.

Table 4.3-2: Examples of the TMU intrinsic statements [33].

One thing to notice is that the hardware support is only for trigonometric functions whose argument is a per unit value, i.e., the argument is given as a fraction of 2π. Therefore, the `__sin` intrinsic which takes arguments in radians, calls first the `DIV2PIF32` instruction to convert the argument to a per unit value, and then calls the `SINPUF32`. Therefore, if the maximum optimization is desired, the angle should be converted to a per unit value first before calling any trigonometric function. The per unit intrinsics such as `__sinpuf32` or `__cospuf32` should be used to compute the trigonometric functions. This way, the `DIV2PIF32` instruction is only called once as opposed to being called whenever any trigonometric function is called.

4.3.1.3 Viterbi, Complex Math and CRC Unit

All modern real-time control systems make use of sophisticated serial communication protocols to exchange data with other devices within the particular system. To verify that the received message is the one sent by the transmitter, typically a cyclic redundancy check (CRC) is used.

The general idea is to represent the transmitted message as a polynomial over a finite field containing just 0, 1. Then choose a so-called generator polynomial and divide the message by the generator polynomial. The resulting remainder is put at the end of the original message and sent. The receiver divides the whole received message by the generator polynomial, and if the result is zero, the message has been received correctly [34].

However, computing the polynomial division purely in software is time consuming. To accelerate the computations, F280049C has a Viterbi, Complex Math and CRC Unit (VCU). The VCU, much like the TMU, enhances the existing FPU by adding extra instructions that support CRC calculation. The VCU includes standard 8-bit, 16-bit and 32-bit polynomials. The user can also use a custom polynomial, the calculation will, however, be slower. Moreover, the VCU adds instructions for complex arithmetic, reducing the time it takes to compute for example the Fast Fourier Transform (FFT). Many baseband communication applications use the Viterbi decoding algorithm. The VCU adds instructions to speed up its calculation [33].

Unlike the FPU or TMU, the VCU cannot be interacted with using intrinsic statements. The user can either use directly the corresponding assembly instructions or utilize the VCU support library that is a part of the C2000Ware package.

4.3.2 Compiler Optimization

The C2000 C/C++ compiler is able to perform a complex set of optimization techniques to increase the speed or reduce the size of the target application. To invoke any optimization in a CCS project, the user must configure the *Compiler Optimization* tab within the *Project properties* menu. First thing to select is the general aim of the optimization process, i.e., whether the compiler should aim for maximum performance or minimum code size. The user can choose a value from 0 to 5, where 0 indicates that the code is optimized purely for the smallest size with a potentially worse performance and 5 indicates maximum performance optimization with potentially large code size. Any value in between indicates trade-off between the two extremes. Selecting 3 means that roughly the same emphasis is put on both [35].

The actual scale of the optimization performed by the compiler is controlled by selecting one of six possible levels of optimization. Each subsequent level automatically performs all optimizations of the predeceasing levels [35].

- **Level=OFF**
  - No optimization is performed.
- **Level=0**
  - Performs loop rotation.
  - Eliminates unused code.
  - Simplifies expressions.
  - Expands call to functions declared inline.
- **Level=1**
  - Removes unused assignments.
  - Eliminates local common expressions.
- **Level=2**
  - Performs loop optimization.
  - Eliminates global unused assignments.

  - Performs loop unrolling.
- **Level=3**
  - Removes all functions that are never called.
  - Simplifies functions with return values that are never used.
  - Inlines calls to small functions.
  - Propagates arguments into function bodies when all calls pass the same value in the same argument position.
- **Level=4**
  - Additionally performs optimization at link-time.

The link-time optimization enables each source file to be compiled and optimized separately and third-party object files can also participate in the optimization process. To illustrate what kind of optimization is being performed, some techniques are briefly described in the following text. The full list can be found in [35].

4.3.2.1 Expression Simplification

Often times a code might contain arithmetic expressions that are not fully expanded. To reduce the number of instructions required to compute a given expression, the compiler expands the expressions, thus reducing the time it takes to compute them [35]. Assume we have an expression $\mathtt{x} = (\mathtt{a} - 3) + (\mathtt{b} + 5) - 1$. After the optimization is performed, it becomes $\mathtt{x} = \mathtt{a} + \mathtt{b} + 1$.

4.3.2.2 Function Symbol Aliasing

If the compiler finds a function that only contains a call to another function, it looks at their signature. The signature refers to the input arguments and the return value, including their data types and order. If the two functions have matching signatures, the compiler makes the calling function an alias of the called function. That means that at link-time all calls to the calling function are redirected to the called function, thus reducing the overhead time required [35]. Consider the following example:

```c
float func1(int arg1, int arg2);
float func2(int a, int b);
float func1(int arg1, int arg2)
{
  return func2(arg1, arg2);
}
```

If the optimization is performed, all calls to `func1` are redirected to `func2`. The body of function `func1` is deleted and the symbols `func1` and `func2` are defined at the same address.

4.3.2.3 Inline Expansion of Functions

Functions declared using the `inline` keyword are replaced with inline code. Moreover, if the optimization level is set to 3 or more, the compiler might also inline functions that are not explicitly declared as `inline` [35].

If a function is inlined, its body is copied at the point of the call. Inlining can improve performance as it eliminates the function call overhead, however at the cost of increased code size. Additionally function inlining creates opportunities for further optimization and is particularly beneficial for small functions that are being called frequently throughout the code. Declaring a function as `inline` is simply a suggestion from the programmer to the compiler. The final decision whether a function is to be inlined is up to the compiler. For example, the aforementioned intrinsic statements are automatically inlined even if the optimization level is set to OFF [35].

Using `inline` functions as well as other optimization can greatly improve the execution time of the application. However, from experience it makes the debugging difficult, especially using breakpoints and stepping through the code. That is because the code that the programmer sees at a particular line might not in fact be there due to the many rearrangements done during the optimization. Trying to step through the code results in jumping across multiple source files and lines seemingly at random. Therefore, if stepping through the code is desired, all optimization should be disabled.

4.4 Used Configuration for FOC Implementation

As it has already been mentioned, to verify the functionality of the designed inverter platform, a sensored FOC of a PMSM is implemented on the F280049C microprocessor. Before any actual control algorithm can be developed, the microprocessor and its peripherals must be properly configured. Without a proper configuration, the microprocessor will be unable to produce the desired behaviour, rendering any further development pointless.

The following text outlines the underlying philosophy of configuring the F280049C for FOC algorithm. Further details, including the precise configuration of each register can be found in the attached source code. Each peripheral has its own header file with a corresponding source file, e.g. the ADC has *adc_config.h* and *adc_config.c*. The header files contain function prototypes and declaration of variables, mainly structures, associated with the particular peripheral.

4.4.1 Base Initialization of the F280049C

First, the clock rate of the CPU must be configured properly to match the desired value. The F280049C is capable of up to 100 MHz clock rate, which is also the value chosen for this application. After the booting sequence is finished up, the CPU uses by default one of its internal oscillators. Since the Launchpad evaluation board contains a 20 MHz quartz crystal oscillator, the clock source is changed from internal to external. To obtain the desired 100 MHz, the CPU uses a phase-locked loop (PLL) with a programmable gain.

Additionally, the WatchDog is disabled, all GPIOs are configured as a GPIO in input mode, i.e., none are mapped onto any peripheral. Desired peripherals have their clock source enabled and the interrupt vector table is initialized end enabled.

For convenience, the base initialization is done using functions from the C2000Ware package. Namely the `InitSysCtrl, InitGpio` and `InitPieVectTable` functions are used.

4.4.2 CMPSS Configuration

The comparator subsystem module (CMPSS) is used to implement a fast overcurrent protection, to prevent a potential fault from destroying the inverter. The block diagram is depicted in figure 4.4-2. Each submodule contains two analog comparators with supporting circuitry. The left-hand side of the diagram containing a ramp generator and synchronization inputs from the PWM modules is irrelevant to us as it is meant for control applications, such as the peak-current control.

The general idea is to use the analog inputs connected to the current sensing network as the non-inverting inputs of the comparators. The reference DAC connected to the inverting input of each comparator is used to set a threshold at which the comparator goes high. The output of each CMPSS is routed to the PWM modules and when it goes high, the PWM outputs are immediately disabled. Since three currents are measured, three modules are going to be needed. The microprocessor has seven CMPSS modules in total, which ones are going to be used depends on the mapping of the analog imputs found in the datasheet [27]. For this application, modules CMPSS1, CMPSS5 and CMPSS7 are used.

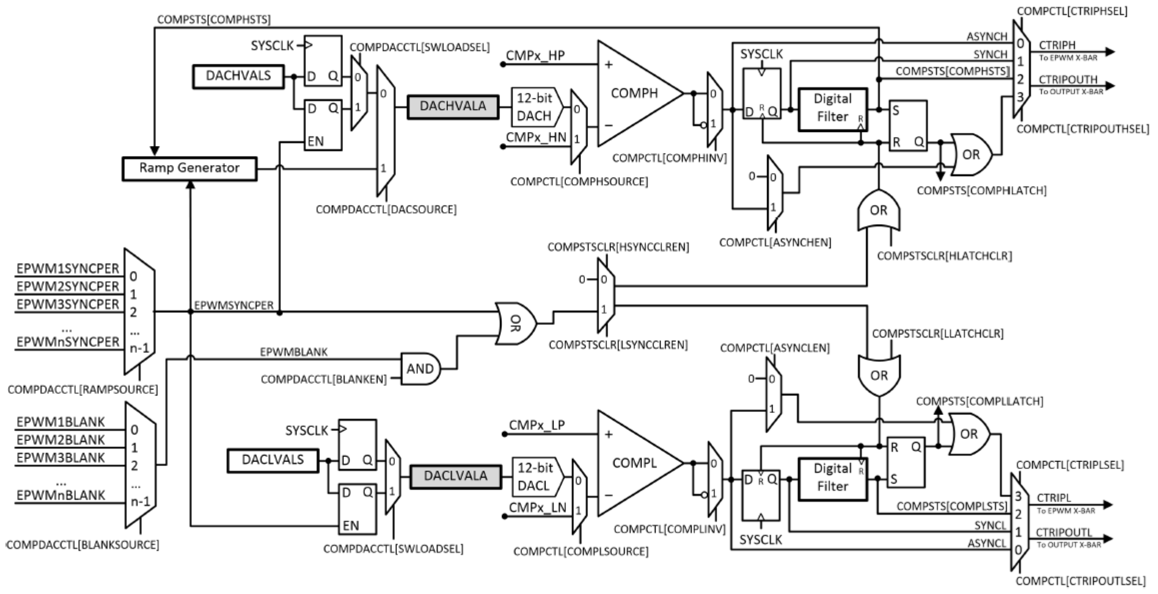


Figure 4.4-2: Block diagram of the comparator subsystem module [28].

The DAC value is chosen to roughly correspond to 90 % of the maximum measurable current which is 90 % $I_{\max} = \pm 18$ A. The DAC can generate voltages from 0 up to 3.3 V, the current is also given as voltage in the same range. To set the threshold to 18 A for the high-side comparator, the DAC value must be

$$DACH = 0.9 \cdot 4095 = 3686. \tag{4.4-1}$$

Similarly, the threshold for the low-side comprator is set to

$$DACL = 0.1 \cdot 4095 = 410, \tag{4.4-2}$$

which corresponds to a current of -18 A. Additionally, the output of the low-side comparator must be inverted so that a fault corresponds to a logic 1. The output of each comparator can be passed through a digital filter to remove potential glitches. The filter stores a given number of samples, and if enough of them are a logic 1, the filer output also becomes 1. Conversely, the output remains unchanged. The user can select the number of samples per frame, the sampling rate and the threshold. To remove glitches that might cause undesired tripping of the PMWs, while also keeping the reaction time of the comparator as low as possible, the frame size is set to 3, threshold to 2 and sampling rate to 100 MHz.

Each comparator generates an output that can be passed to the PWM module. The outputs must be first connected to one of the TRIP signals in the ePWM X-BAR. There are eight TRIP signals in total that can be flexibly mapped. Either each comparator from all three submodules is connected to a unique trip signal, making it six TRIP signals in total or the outputs of both comparators from each submodule are passed through an OR gate. Thus, reducing the number of TRIP signals used by half. The TRIP signals are passed to each PWM module to its digital compare submodule where the signals are processed. The processing of the TRIP signals is explained in the next section dealing with the PWM module configuration.

4.4.3 PWM Configuration

To drive a three-phase motor, three PWM modules are required. Due to a slightly inconvenient pinout of the Launchpad evaluation board, modules PWM1, PWM2 and PWM4 are used. The core of each PWM module is its time-base counter (TBCTR). The frequency of the counter is derived from the main CPU clock using a prescaler. The prescaler is set to 1, thus the TBCTR frequency is set to 100 MHz. The TBCTR is set to operate in the up-down count mode. The period of the TBCTR can be calculated from the desired PWM frequency as [28]

$$TBPRD = \frac{1}{2}\frac{f_{\mathrm{TBCTR}}}{f_{\mathrm{PWM}}}. \tag{4.4-3}$$

For $f_{\mathrm{PWM}} = 10$ kHz and $f_{\mathrm{TBCTR}} = 100$ MHz, the TBCTR period in clock cycles becomes $TBPRD = 5000$. Furthermore, all three PWM modules must be synchronized together, i.e., all TBCTRs must start at the same time. To achieve this, the PWM1 module is selected as the master module. When its TBCTR is zero, a synchronization pulse is sent to the other modules, triggering their respective TBCTR.

The output of the PWM modulator is a set of three compare values, one for each phase, expressed in TBCTR clock cycles. These values are loaded into the compare register of the respective PWM module. The value of the compare register is compared to the TBCTR and when a match occurs, the PWM output changes. Each PWM module has two outputs PWMxA, PWMxB. All three modules are configured to operate in the active-high complementary mode. In this mode, PWMxB outputs the inverted pulse pattern from PWMxA, i.e., when PWMxA=1 then PWMxB=0 and vice versa. Therefore, only the PWMxA output is actively controlled by the application as PWMxB is controlled automatically. The pulse generation is controlled by the action qualifier submodule. Since the TBCTR operates in an up-down count mode, a match occurs twice per period. The PWMxA is forced high when a match occurs during up-count and forced low on a match during down-count. To prevent accidental switching of both transistors in the same inverter leg, a dead time of 1μs is added to the pulse pattern. The dead time for both rising and falling edge is added to the PWMxA output.

To process the TRIP signals from the CMPSS modules signaling that an overcurrent has occurred, the digital compare (DC) and trip zone (TZ) submodules are used. The TRIP signals are passed to the DC submodule first where a corresponding digital compare event is generated and passed to the TZ submodule. The DC can generate four events in total, for the purposes of the overcurrent protection, only the DCAEVT1 is used. The DC events can either be generated on a specific TRIP signal or all relevant TRIP signals are passed through an OR gate whose output drives the chosen DC event. The latter is best suited for overcurrent protection as we only care that a fault has occurred, not where. The TZ is configured to set both PWMxA and PWMxB outputs low when the DCEVT1 occurs. Also, an interrupt is generated. If the user wishes to temporarily disable the overcurrent protection, it is advised to comment out the DC submodule configuration rather than the CMPSS. If the DC remains in use while the CMPSS is disabled, the TRIP inputs are floating which can lead to unexpected behavior. The DCAEVNT1 might be randomly generated, unexpectedly tripping the PWMs.

Since the inverter has the shunt resistors connected between the emitter of the low-side IGBTs and negative rail of the DC bus, the currents must be sampled when all low-side transistors are conducting. Based on the SVM pulse pattern in figure 2.2-4 it happens when the vector $\underline{v}_0$ is applied to the inverter. At exactly half of the duration the TBCTR=0. When that happens, a trigger is sent to the ADC modules to start the conversion sequence.

4.4.4 ADC Configuration

The CPU ROM contains offset trims for each ADC module, these trims were measured during production. Before the ADC can be powered up, the trim value must be loaded from the ROM. C2000Ware function `SetVREF` does this along with setting the reference to internal 3.3 V. After the ADC modules are powered up, a 1 ms delay is inserted to the program to allow the analog circuitry to settle. The delay is done using the `DELAY_US` assembly function.

There are five signals to convert during each sampling sequence; three currents, DC bus voltage and inverter module die temperature. As already mentioned, the PWM1 module sends a start of conversion trigger (SOC) every time the TBCTR=0. Each ADC module can have up to sixteen different SOCs, numbered from 0 to 15. If multiple SOCs occur simultaneously, the one with the lowest number is executed first, while the rest is pending. A single SOC source such as a PWM module can send multiple SOC signals. To convert five signals with three ADC modules, at least two conversion sequences are required. Due to inconvenient pinout of the Launchpad board, both the DC bus voltage and inverter temperature are mapped onto the ADCC module. Since the module can only sample one input channel at a time, additional SCO is needed. So, the PWM1 generates three SOC signals: SOC0 to sample the currents, SOC1 to sample the temperature and SOC2 to sample the DC bus voltage. When the ADCC finishes converting the DC bus voltage an interrupt is generated. During the ADC interrupt the core of the FOC algorithm is executed.

To ensure that a right value is measured, the sample&hold capacitor must have enough time to charge. The datasheet specifies the minimum time being 70 ns. Moreover, the charging time also depends on the internal impedance of the circuit, connected to the particular ADC channel. The duration of the sample&hold window is set to 100 ns which was experimentally verified to be sufficient. Moreover, the ADC uses the successive approximation method to convert the analog voltage to a digital value. According to the datasheet, the duration of each conversion cycle should be 20-200 ns, hence it is set to 50 ns, corresponding to 20 Mhz frequency of the internal ADC clock signal.

4.4.5 eQEP Configuration

To interface with an incremental encoder the enhanced Quadrature Pulse Encoder (eQEP) module is used. The eQEP module directly interfaces with an encoder to obtain speed, position and direction information from its signals. The method used to calculate speed combines two common techniques, i.e., counting the number of encoder pulses in a fixed time interval and measuring the time interval between two consecutive pulses. Details are given later in the text when the FOC implementation is described.

To count the number of pulses, the eQEP position counter is enabled. To further increase the accuracy, the position counter counts each rising and falling edge of the input signals. Quadrupling the number of pulses per revolution since a typical incremental encoder has two quadrature signals shifted by 90°. To enable convenient measurement of the rotor position, the position counter is reset back to zero once it reaches the effective number of pulses per revolution. Which in our case is 10000. However, since the position counter counts from zero the reset value must be set to 9999, otherwise an error is going to accumulate. With this setting the information in the position counter directly corresponds to the rotor angle.

The eQEP unit timer period is set to 550 μs. Every time it counts to the period, the position counter and capture timer are latched to their respective latch registers and interrupt request is sent to the CPU. In the interrupt service routine, the rotor speed is calculated. The unit timer period is set is such a way that the interrupt frequency is a non-integer multiple of the ADC interrupt frequency.

4.4.6 UART Configuration

For evaluation and tuning purposes, the F280049C communicates with a computer using the serial communication interface (SCI) which is a standard two-wire asynchronous serial port, commonly called an UART. The SCI is configured to work in a full-duplex mode, i.e., both the transmitter and receiver are used. The basic unit of data is typically referred to as a character and its format is depicted in figure 4.4-3.

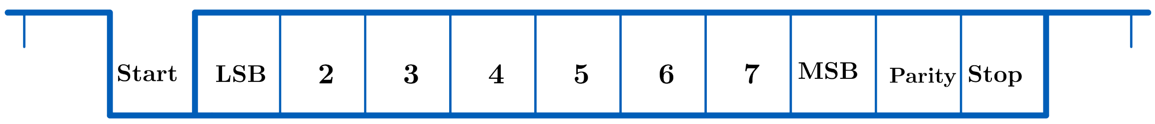


Figure 4.4-3: Typical UART character format [28] (edited).

Each character starts with a start bit, followed by 1 to 8 data bits and a stop bit. Optionally, extra parity bit can be inserted before the stop bit. The parity bit serves as a simple error detecting code. Two types of parity bits are used: even parity and odd parity. As the name suggest the even parity bit ensures that the total number of 1-bits in the message is even, conversely the odd parity ensures odd number of 1-bits in the message. The even parity corresponds to a 1-bit CRC generated by the polynomial $x + 1$. The format used in this thesis uses 8 data bits, 1 stop bit and no parity bit. Since the communication mode is asynchronous, no clock signal is sent between the transmitter and receiver. The synchronization is done at each end using the predefined communication speed. The speed is specified as a baud rate. The baud rate gives the number of signal changes on the transmission medium per second. Since in the SCI case each signal change corresponds to one bit of information the baud rate corresponds to the bit rate and is set to 576000.

The SCI features 16-byte deep FIFO data structures for both the transmitter and receiver. Using them makes managing the communication much easier as no extra interrupts are needed. The user just writes the data byte by byte to the 8-bit transmit buffer register and the SCI automatically loads it to the FIFO buffer and from there to the transmit shift register. Therefore, the user must only make sure to check that the FIFO buffer has enough space using its status register. Received data is loaded automatically to the FIFO buffer from where they can be read byte by byte until the buffer is empty. More details regarding the serial communication including the message format, etc., are given in the next chapter.

Chapter 5

Real Time Monitoring Interface

During the debugging process of any real-time control application, having the ability to visualize all the relevant signals is of tremendous importance. While an oscilloscope is undoubtedly essential, it does have its limitations. Namely speed visualization is complicated and also an oscilloscope cannot in principle show internal control variables. Since any FOC algorithm is based on controlling the d-axis and q-axis components of the stator current vector, is critical to be able to visualize them. An oscilloscope cannot be used as both are a mere mathematical constructs developed for convenient mathematical description of AC machines.

To be able to see the behavior of these internal control variables, typically a real time monitoring (RTM) software is used. The RTM is made up of two subsystems. The first one is a part of the control algorithm running on a microprocessor. Its main purpose is to send the relevant data to a computer using some serial communication protocol, typically an UART. The second part is running on the computer that is connected to the microprocessor. Usually in the form of a graphical user interface (GUI) that receives the data from the microprocessor and visualizes them. The GUI can provide the user with extra tools to process the data, e.g. FFT, cursors, storing the data into a file etc. The RTM can also be designed to not just receive data but to send commands to the microprocessor.

Since no such RTM compatible with the F280049C was available at the time of making the thesis, the author decided to make one that could be used in conjunction with the designed inverter platform. Thus, making the process of developing any future control algorithms easier. The RTM is designed as a application with a GUI using MATLAB, together with a communication module for the F280049C written in C. The module contains a SCI configuration and a set of functions for managing the communication. The designed RTM can receive data either in an offline mode, which is ideal for tuning controllers or in an online mode. The online mode displays the data in real-time, making it useful to show the motor speed and see how the motor reacts when, for example, the load is changed. Also, a simple communication protocol for sending basic commands to the microprocessor was developed.

While using any high-level programming language such as Java, Python or C++ would most likely resulted in better performance and wider options for customization, MATLAB was chosen mainly because the author is fairly familiar with the language and it also makes creating an GUI easy through its interactive development environment called the App Designer. The main downside of using MATLAB is that in order to use the RTM, the user must have MATLAB installed. Furthermore, the app performance depends on the version of MATLAB. The 2022b version was used in the development and the author would suggest to use the same one or at least the version 2021b as it noticeably improves the performance. The improved performance is critical for the online mode as in older versions the axis environment is unable to update fast enough.

5.1 Interacting with Serial Port in MATLAB

Connecting to a serial port in MATLAB is straightforward. First, the user must determine to which virtual COM port the device is connected to. With that the serial port object can be created within MATLAB. To create a serial port object, the `serialport` function/method is used. The function has two mandatory arguments, i.e., the COM port number and baud rate. The optional arguments include the number of parity, stop and data bits, endianness, etc. All of these have a default value assigned; hence it is not necessary to explicitly specify them. The default values are listed in the MATLAB Documentation [36]. To set up connection to a serial port we simply write:

```
port=serialport("COM3",576000);
```

This creates a serial port object called `port` connected to the COM3 port with the baud rate set to 576000. The default values for the optional parameters are used, i.e., no parity bit, 1 stop bit, 8 data bits and little-endian. Which corresponds to how the SCI module of the F280049C is configured. With the connection established, we can now interact with the serial port. Specifically, to read and send data. To send data, there are two methods – `write` and `writeline` [36]. The latter one is specifically designed to send a string of ASCII data and as such it is not of much use to us. The `write` method is more suitable for our application as it sends raw numerical data in the specified format. For example:

```
write(port,5,"uint8");
```

sends the number 5 to the serial port represented by the `port` object as a 8-bit unsigned integer. We might want to send more than just one number, the syntax changes little:

```
write(port,[1 2 3],"uint8");
```

Values 1, 2 and 3 are queued up in the transmit buffer of the computer and once it is given priority from the operating system, the values are sent to the connected device. All development was done under MS Windows; therefore, all the described interactions with a serial port are applicable only to this operating system. Since other operating systems use different task scheduling and prioritization.

Reading data from a serial port is also straightforward. Similarly, two methods are available `read` and `readline` [36]. As the name suggests, the latter one is meant to be used to read strings of ASCII characters, while the former one reads raw numerical data. Reading data can either be done manually or through the use of a callback function. If reading data manually, the `NumBytesAvailable` property of the serial port object is useful as it informs the user about the number of bytes available in the buffer. Once the buffer contains the desired number of bytes, they can be easily read using the `read` method:

```
data=read(port,10,"uint8");
```

The first argument is the serial port object from which the data is to be read, the second is the number of data points of the type specified by the third argument. The data is always received as raw bytes, i.e., `uint8`. However, the `read` method can convert the raw data into different data types, e.g, `int16`, `uint16`, `int32`, `uint32`, `single`, `double` [36]. Nevertheless, the number of data points to read must coincide with the number of bytes in the buffer. Also if anything but `uint8` is to be read, a care must be taken to ensure that, for example, the encoding of negative or decimal numbers is the same on both sides as well as endianness. The author is of the opinion that for maximum reliability the data should be read as `uint8` and converted manually.

Alternatively, the process of reading the data can be automatized using a callback function. The callback function is assigned to a particular event and when it occurs, the callback function is executed. The `BytesAvailable` callback is the best suited for our application as the callback function is executed whenever the buffer contains a predefined number of bytes. The callback function can also be made to trigger when a specific ASCII character or a line terminator appears in the buffer. The callback is configured by calling the `configureCallback` method [36]:

```
configureCallback(port, "byte", 100, @readSerial);
```

This sets the callback of the `port` object to be executed whenever the buffer contains 100 bytes of data and the `readSerial` function is set as the callback function to be executed. A simple program to set up a serial communication and automatically process the received data via a callback function might look something like this:

```
% comunication parameters
numDataToRead=300;
bytesPerDataPoint=2;
numVariables=2;
numBytesToRead=numDataToRead*bytesPerDataPoint*numVariables;

% Serial port object definition
port=serialport("COM3",576000);
configureCallback(port,"byte",numBytesToRead,@readSerial);
```

Whenever the input buffer contains the number of bytes specified by `numBytesToRead`, the `readSerial` function is executed. The function can do anything from just reading and plotting the data to performing, for example, data filtering or spectral analysis. An example use case could be tuning a PI controller. The control software is running in a debug mode through which the user adjusts the controller gains and starts the step response measurement. The system response is sent to a computer which runs the above code and plots the received data. Thus, allowing the user to see how the controller gains affect the system response.

Capabilities of this approach are, however, limited. The communication on both sides must have the exact same parameters, mainly the requested number of data points must be the same. Otherwise, the callback will not be triggered properly. Therefore, if the user wishes to change the number of data points to be plotted, the value must be changed manually on both sides. We might want to also change the sampling rate and so on. Adding additional parameters that need to be adjusted on both sides. Ideally, the parameters are changed only on one side, preferably within the RTM, and automatically sent to the microprocessor. That way the user manages the communication entirely from the RTM, making it more convenient to work with. Such functionality can be achieved only through a GUI.

5.2 Creating Graphical User Interface

To create a graphical user interface (GUI) MATLAB has an interactive environment called the App Designer. The graphical design of an app is done entirely by dragging predefined components such as buttons, check boxes, sliders, edit fields etc. and placing them on a canvas, which accelerates the development process as the user does not have to spend much time designing the layout of the app and can focus on its functionality [36]. The App Designer layout is depicted in figure 5.2-1.

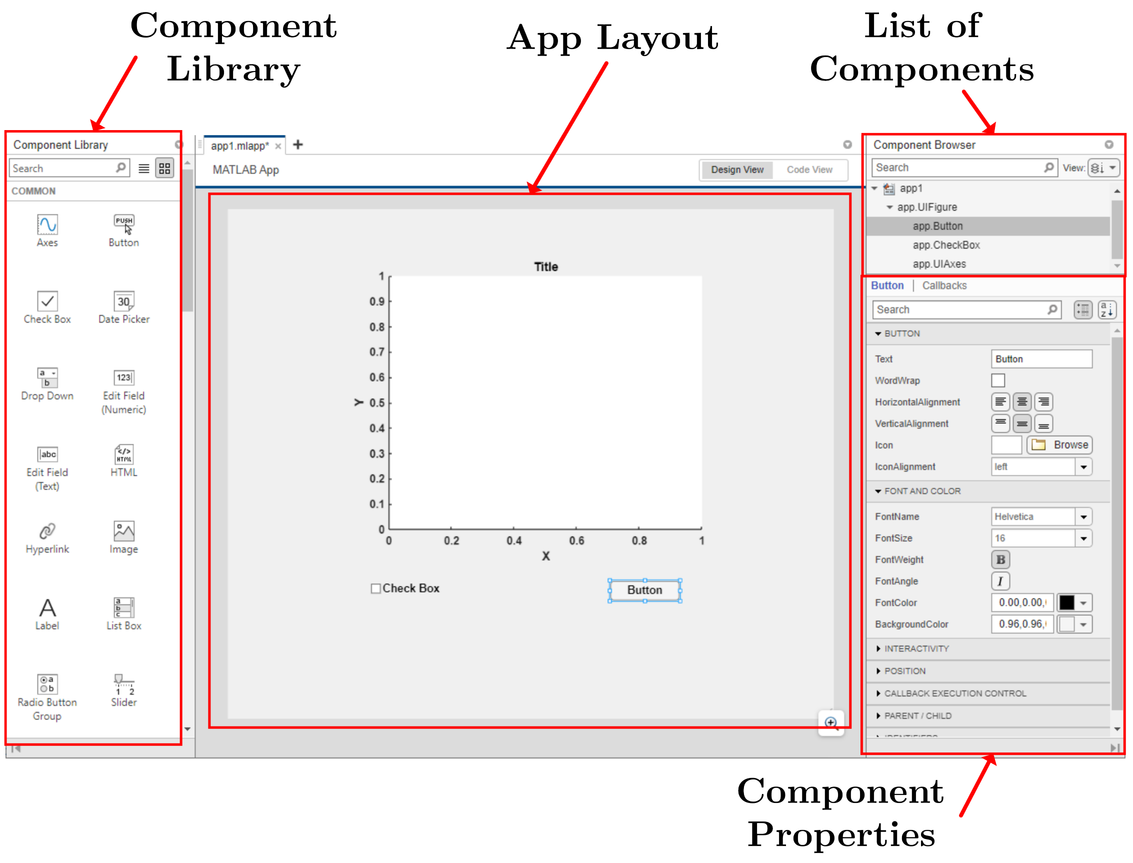
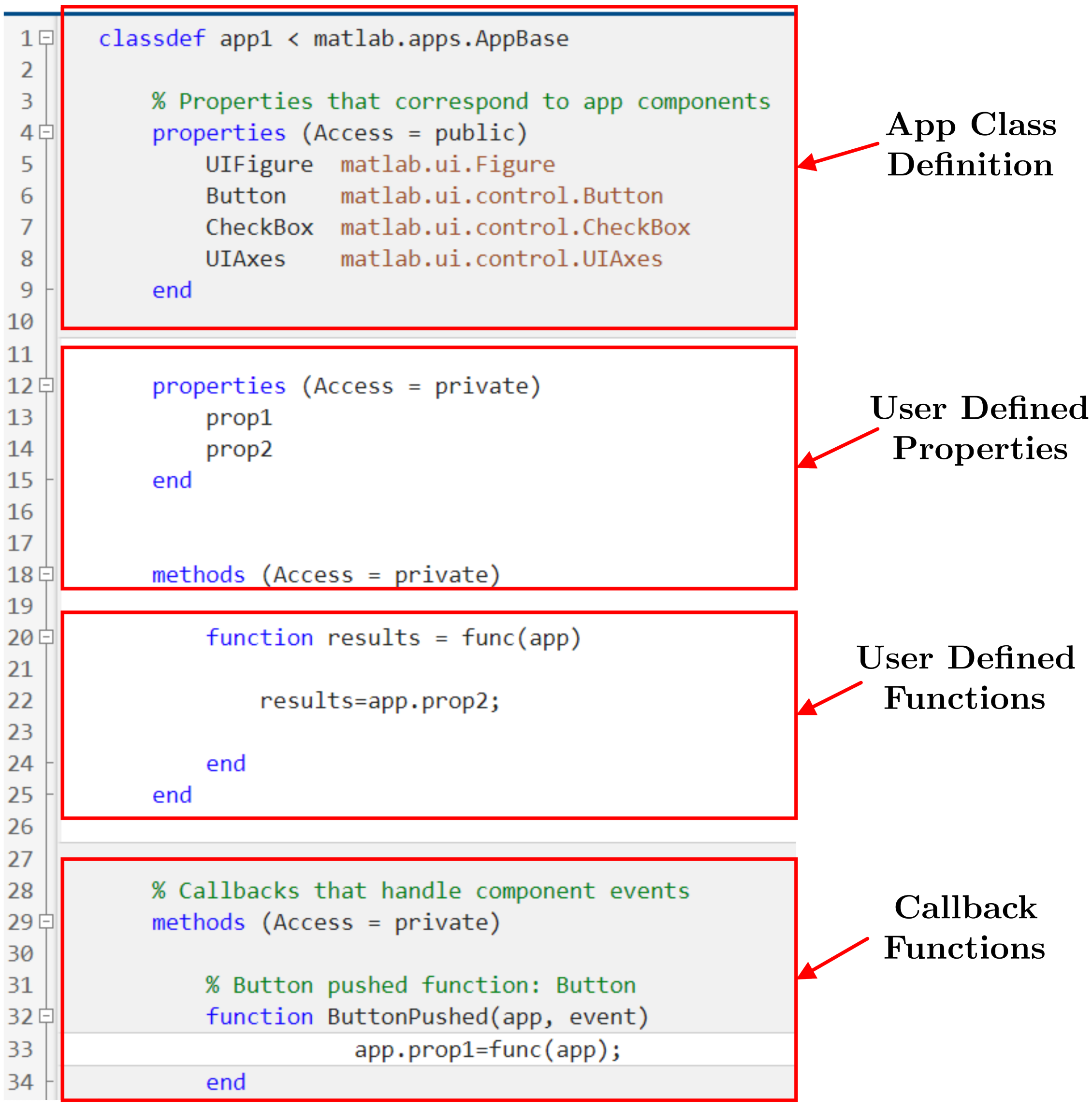


Figure 5.2-1: MATLAB App Designer layout.

To design the layout of an app we simply chose the components we want to use from the Component Library and place them on the premade canvas. If we wish to change the appearance of a component, we can do so in the Component Properties menu. The app can now be executed. However, it will not respond to any input as no functionality has so far been programmed into it.

Everything regarding graphics is handled by an internal MATLAB engine. The user adds functionality to the app by assigning callback functions to the individual components. Thus, whenever, for example, a button is pressed, the callback function assigned to it gets triggered. Programming the app is done in the *Code View* tab in the App Designer which is depicted in figure 5.2-2. Immediately we notice that some parts of the code are on a grey background while others are on a white background. Such code is generated by the App Designer and cannot be modified by the user. Editable parts of the code are placed on a white background. The App Designer automatically generates code that creates the `app` object and associated methods for initializing the GUI. All the different components such as buttons, sliders, check boxes etc., are stored in the `app` object as properties. The value of a check box, for example, can be accessed by tipping `app.CheckBox.Value`. The user can define additional properties to store data or pass additional values to functions. This is done by creating a second list of properties where the `Access` parameter is set to `private`. There is no obvious limit to the number of properties that the `app` object can have. However, if the `app` object gets too big, the app performance might drop. That is because every function within the app must have the `app` object as one of its arguments. So, if the size of the object gets out of hand, it significantly increases the time it takes to copy it into a function workspace.



```
classdef app1 < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure  matlab.ui.Figure
        Button    matlab.ui.control.Button
        CheckBox  matlab.ui.control.CheckBox
        UIAxes    matlab.ui.control.UIAxes
    end


    properties (Access = private)
        prop1
        prop2
    end


    methods (Access = private)

        function results = func(app)

            results=app.prop2;

        end
    end


    % Callbacks that handle component events
    methods (Access = private)

        % Button pushed function: Button
        function ButtonPushed(app, event)
                    app.prop1=func(app);
        end
```

Figure 5.2-2: Code view tab example in App Designer.

Furthermore, the user can also define custom functions that can be used throughout the code. A function can return any number of values but must always have at least the `app` object as an input argument. The custom functions are essentially methods acting on the `app` object. These functions can be accessed in much the same way as the properties, i.e., `app.func(app)`. Whenever a callback is assigned to a component, the App Designer automatically creates a function in the Callback Functions section of the code. A callback function always has two arguments – the `app` object and `event`. The latter represents the particular event triggering the function such as a button being pushed. The number of input arguments cannot be changed so if we want to pass additional arguments to a callback function, they must be part of the `app` object. Apart from the classic callback functions assigned to a component, the App Designer supports two special ones. The `startupFcn` that, as the name suggests, is executed when the app is launched after component creation. Its main purpose is to initialize components and variables. The `UIFigureCloseRequest` which is triggered when the app is closing. It can be used to signal the microprocessor that the communication link has been interrupted.

5.3 Designed RTM Application

The developed RTM Application is comprised of two main subsystems working together to form the complete solution. The first one, written in C, is the API responsible for servicing the communication on the microprocessor. It contains functions for converting variables into individual bytes and vice versa. Functions to send and read data and initialization of the SCI module. The second part is an app build in the App Designer which manages the communication on the computer side. The app can receive as well as send data, visualize the received data, export it to a file and so on.

5.3.1 Communication API

A single structure is used to control the communication and store all relevant data. It contains flags informing the user of the status of the communication, buffers for sending and receiving data and additional auxiliary control variables.

```c
typedef struct{

    // Status flags
    _Bool singleMode;
    _Bool continuousMode;
    _Bool flagComConnected;
    _Bool flagContinuousBufferReady;
    _Bool flagComActive;
    // Auxiliary control variables
    Uint16 bytesReceived;
    Uint16 numDataSend;
    Uint16 numSamples;
    Uint16 dataRead;
    Uint16 divider;
    Uint16 sample;
    Uint16 numBytesToSend;
    Uint16 scaleFactor;
    int16 varA;
    int16 varB;
    int16 valRead;
    // Buffers
    Uint16 bufferPos;
    Uint16 dataBuffer[4];
    Uint16 receivedData[3];
    Uint16 dataBufferContinuous[40];

}T_UART;
```

The variables that might be of interest to visualize such as currents, speed or position are all by nature expressed in floating point. The F280049C uses 32-bit `float` variables. Thus, to send one `float` variable it must be separated into four bytes which can then be sent over UART. Since the data are used mainly for visualization, the precision offered by a 32-bit `float` variable is unnecessary. Furthermore, values of the aforementioned physical quantities will not surpass lower thousands. To speed up the data transfer and subsequent decoding, the floating point values are scaled and type-cast to `uint16`. As such, the number of bytes per message has been halved. Original idea was to just type-cast the floats to `int16` as negative numbers do not require any additional encoding. MATLAB, however, was not able to properly decode the negative numbers automatically. To circumvent the issue, a simple encoding is used for negative numbers.

The variables to be sent are first scaled and type-cast to `int16`, the scaling factor is set to 100 which corresponds to a transmittable range of -327.67-327.67. If, for example, the rotor speed in rpm is to be sent, the scaling factor must be decreased. Afterwards, 65536 is added to the scaled values and the results are stored in auxiliary `uint16` variables. These ensure that all operations are done in $\bmod 2^{16}$ arithmetic. So, if the initial value is positive, the stored value is the same as $2^{16} + x \equiv x \pmod{2^{16}}$. If it is, however, negative, the stored values is $2^{16} - x$. Decoding is simple. Since we know that a 16-bit integer can store values from -32767 to 32767 we can easily determine whether the original value was positive or negative. If the received value is ≤ 32767 the original one must have been positive, and it is obtained by simply dividing the received value by the appropriate scaling factor. On the contrary, if the received value is > 32767, the original one must have been negative. The original value is obtained by subtracting 65536 from the received value and the result is divided by the appropriate scaling factor. The API has two functions `intToBytes` and `bytesToInt` for encoding and decoding, respectively:

```c
void intToBytes(T_UART* com)
{
   Uint16 dataScaled1=(65536+com->varA);
   Uint16 dataScaled2=(65536+com->varB);

    com->dataBuffer[0]=dataScaled1>>8;
    com->dataBuffer[1]=dataScaled1;
    com->dataBuffer[2]=dataScaled2>>8;
    com->dataBuffer[3]=dataScaled2;
}

void bytesToInt(T_UART* com)
{
    Uint16 upperByte;
    Uint16 lowerByte;
    Uint16 sum;

    upperByte=com->receivedData[1]<<8;
    lowerByte=com->receivedData[2];
    sum=upperByte+lowerByte;

    if (sum>32767)
    {
        com->valRead=-(65536-sum)/com->scaleFactor;
    }
    else
    {
        com->valRead=sum/com->scaleFactor;
    }
}
```

The RTM is designed to send two variables with each sample and receive two bytes of data, hence the buffers have four and two elements, respectively. The initial decision to send just two variables was made so that each frame would be send within 100 μs which corresponds to the period of the ADC interrupt with which the FOC algorithm is calculated, so that the communication can be done without extensive data buffering in the microprocessor. Meanwhile, two variables are enough for controller tuning and other debugging purposes. To be exact, with the baud rate set to 576000, the bit duration is 1.736 μs. Each transmitted character has 10 bits, i.e., 1 start bit, 8 data bits and 1 stop bit. So, the byte duration is 17.36 μs. Therefore, it takes $\approx$70 μs to send the four bytes. Thanks to the 16-byte FIFO buffer of the SCI peripheral, we actually do not have to send all the bytes within the 100 μs window. All that is required is for the buffer to have enough space for the next message. The baud rate could also be increased, the author was able to reach about 800 kbps on his set up. Therefore, a future version capable of sending at least four variables at a time is planned.

Two distinct functions `sendMessage` and `sendContinuousData` can be used to write the data into the SCI transmit buffer register. The former one is used when the RTM operates in an offline mode, i.e., the microprocessor sends data each sampling period and after a given number has been sent, the MATLAB GUI plots them. The latter one is used during the online mode of operation:

```c
void sendMessage(T_UART* com)
{
    Uint16 i=0;

    while (SciaRegs.SCIFFTX.bit.TXFFST>=12)
    {

    }

    while (i<com->numBytesToSend)
    {
        SciaRegs.SCITXBUF.all = com->dataBuffer[i];
        i++;
    }
}

void sendContinuousData(T_UART* com)
{
    if (SciaRegs.SCIFFTX.bit.TXFFST==0)
    {

        SciaRegs.SCITXBUF.all = com->dataBufferContinuous[com->
   bufferPos];
        com->bufferPos++;

    }
    if(com->bufferPos==40)
    {

        com->flagContinuousBufferReady=false;
        com->bufferPos=0;
    }
}
```

Due to performance limitations of MATLAB, the continuous mode had to have been adapted. MATLAB has an internally defined period with which it issues an request to the operating system asking whether there are any new data in the serial port buffer. If so, the content of the internal hardware buffer is loaded into the virtual buffer inside MATLAB. The internal period cannot be changed by the user and from experiments it is somewhere between 30 to 35 ms. This also effectively limits the speed at which the virtual buffer can be read. Therefore, if we want to plot data in real-time, the data frames cannot arrive faster than ≈40 ms, since some overhead is required to process and plot the data. The online mode can only display one variable at a time due to performance limitations of the App Designer. If the continuous mode is selected, values are stored into the `dataBufferContinuous` array during the ADC interrupt with a selected prescale. The buffer is hard-coded to have a place for 40 bytes which corresponds to 20 `uint16` values. Therefore, the sampling time should be at least 2 ms or more. When the buffer is full, the `flagContinuousBufferReady` is set and the `sendContinuousData` function is called in the background. When the `dataBufferContinuous` has been emptied, the `flagContinuousBufferReady` flag is set to `false` and new data can be loaded once again. The only timing restriction is that the sampling time must be longer than the time required to transmit the entire buffer, i.e., 694.4 μs. So the aforementioned 2 ms is more than enough. How the app is processing the data is covered in the following section. The process of sending data to a computer is represented by a flowchart shown in figure 5.3-3.

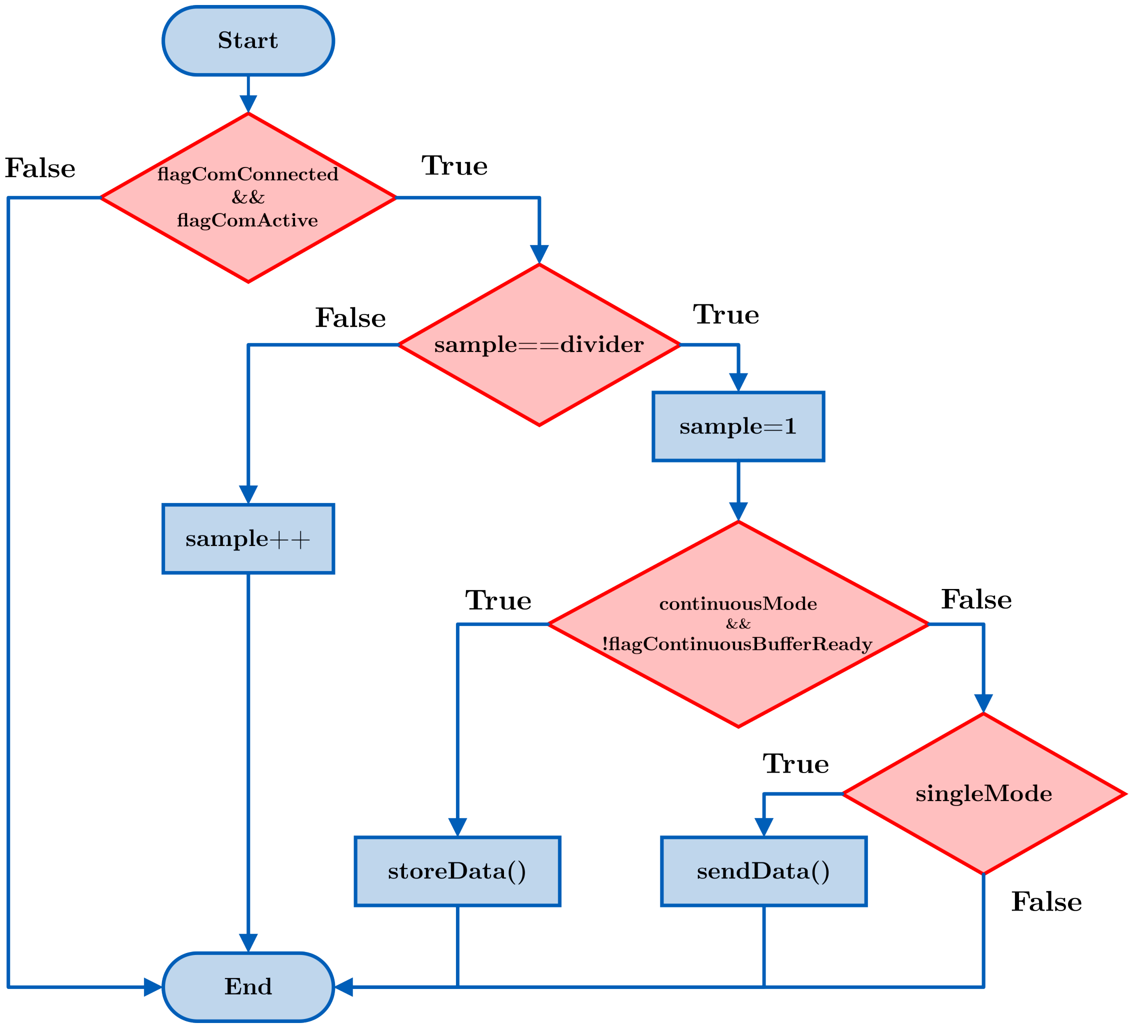


Figure 5.3-3: Flowchart depicting the process of sending data to a computer.

The API also contains a protocol for receiving simple messages from the computer. The messages are predominantly used to control the communication. So whenever a parameter is changed within the app such as the number of samples or sampling rate, the new value is automatically sent to the microprocessor. This ensures that both sides are synchronized all the time. Furthermore, the user can define custom messages that can be used to set, for example, the reference speed. The message format is depicted in figure 5.3-4.

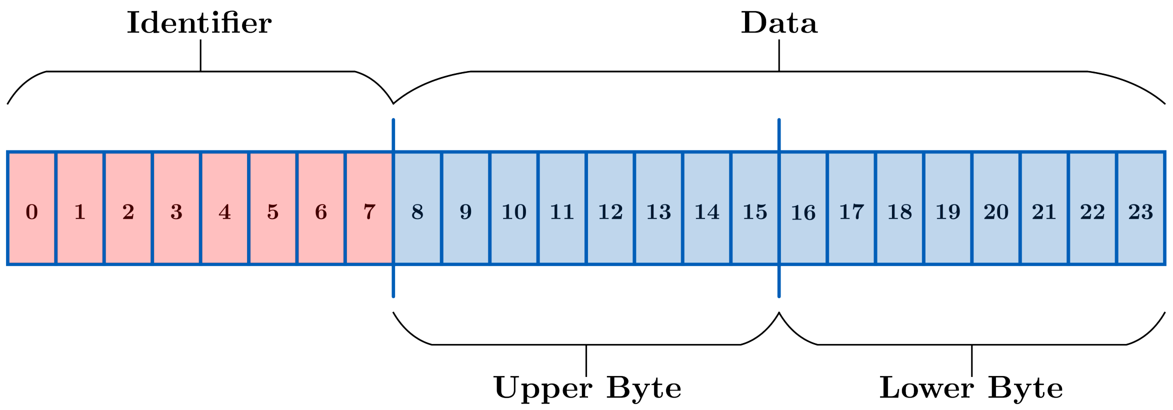


Figure 5.3-4: Incoming message format.

Each message has three bytes, the first byte is the message Identifier (ID). Values from 0 to 255 can be assigned to the message ID, thus the protocol supports up to 256 unique messages. The last two bytes are where the actual message data is stored. The data is of the `int16` type, supporting values from -32767 to 32767. The encoding of decimal and negative numbers is the same as described above. The receive buffer of the microprocessor is continuously checked in the background whether it contains three bytes. If so, then the data is read from the buffer and the message is decoded based on its ID. The message does not contain any error check such as a CRC. That is because the messaging protocol is not being used to send any critical data, so it was deemed unnecessary to increase the length of the message.

5.3.2 Graphical User Interface

5.3.2.1 General Overview

The designed GUI is depicted in figure 5.3-5. For better orientation it can be separated into three distinct sections: the Communication Control Panel, the Auxiliary Panel, and the Visualization. The Communication Control Panel is where all things related to sending and receiving data are located. Connecting to a serial port is done simply selecting the particular COM port and baud rate. The app automatically scans the system for all available COM ports and lists them. If the connection is successful, the COM status lamp turns green, and the COM port is listed as being in use. Otherwise, an error message pops up, informing the user that the connection was unsuccessful. The app also periodically checks whether the connection is still active. If the target device is disconnected, the app automatically updates its status accordingly. The other tabs deal with sending and receiving data and are shown in detail further in the text. The Auxiliary Panel contains features for data export and figure customization. The data can either be exported to the base MATLAB workspace or to a file. Supported file extensions are `.txt`, `.csv` and `.xlsx` and the file is created in the same folder where the app is located. Saving the data to the base workspace is useful for additional processing. The Visualization section is where the received data is plotted. The user can add appropriate axis labels as well as a legend.

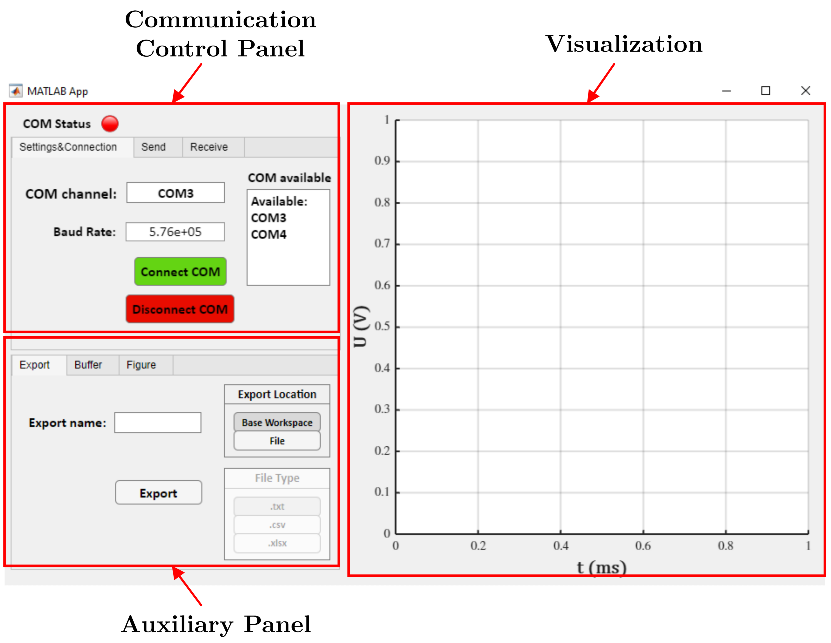
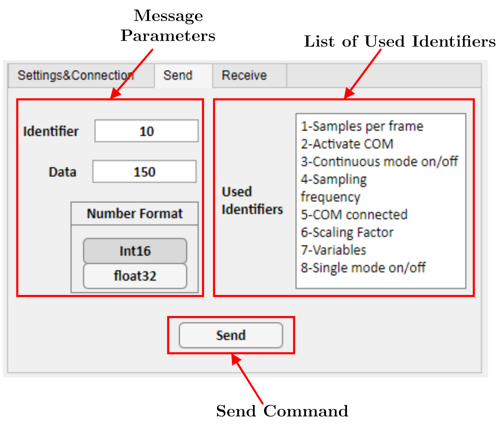


Figure 5.3-5: Designed GUI.

5.3.2.2 Sending Messages

The interface for sending messages is depicted in figure 5.3-6. It implements the message format described above. If a message is to be send, the user must choose its identifier based on which the message will be decoded once it is received by the microprocessor. The identifier must be non-negative integer with the maximum value being 255. Moreover, the chosen identifier must not be already in use by another message to prevent errors during decoding. Otherwise, an error will pop up, informing the user of picking a wrong identifier. To prevent confusion, a text field is added where the user can list all identifiers that are already in use. Before inputting the data that is to be send, the data type must be selected first. The format can be either an integer or a float. This is to optimize the available range of values. Floats whose absolute value is ≤ 327.67 are sent with two decimal digits. Those ≤ 3276.7 are sent with one decimal digit and the rest is rounded to the nearest integer and sent. Integers can vary from -32767 to 32767. If the message does not contain any errors and the app is connected to a COM port, pressing the `Send` button transmits the message to the target device.



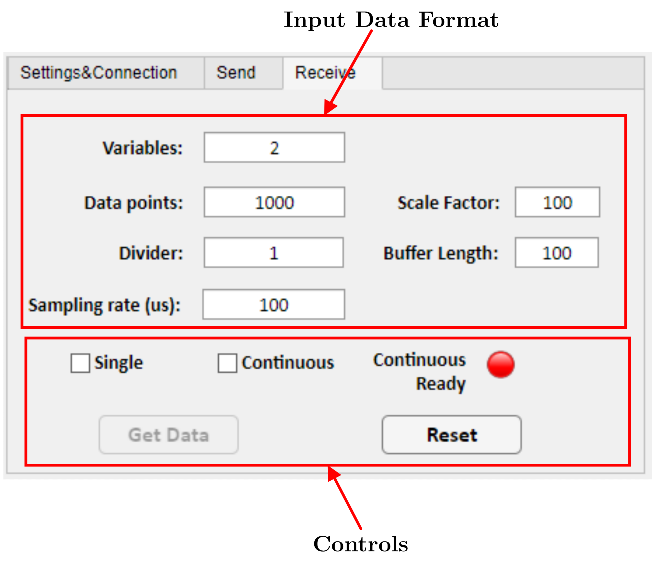
1-Samples per frame
2-Activate COM
3-Continuous mode on/off
4-Sampling frequency
5-COM connected
6-Scaling Factor
7-Variables
8-Single mode on/off

Figure 5.3-6: Interface for sending messages to a microprocessor.

5.3.2.3 Receiving Data

The interface controlling data reception is depicted in figure 5.3-7. Before any data can be received, the user must set the desired format, i.e., the number of variables, number of data points per frame, prescaler, sampling rate and scaling factor. If the continuous mode is to be used, additionally, the buffer length must be selected. The number of variables can for now be only 2 but in a future version it will be possible to select between 1 to 4 at least. Number of data points together with the prescaler (divider) controls the length of the captured frame. The user can flexibly choose a compromise between the length of a frame and the sampling time. The sampling rate is just used to properly scale the horizontal axis and it corresponds to the execution rate of the ADC interrupt. Scale factor is used to properly decode incoming data. If any of these parameters get changed, the app automatically sends the new value to the connected microprocessor. That way both sides remain synchronized. The buffer length parameter exists only within the app and corresponds to the total length of data being shown during the Continuous mode. More details on how the Continuous mode works are given in the next section.

Once the format has been selected, the user can choose between Single and Continuous mode and then press the Get Data button to start the process. Single mode means just one frame of the defined length is received, ideal for tuning a controller. The trigger can also be set up within the control software which is better if it is desired to begin the data logging at a precise point in time. While Continuous mode means that data are being send continuously and the figure is updated with each new data frame.

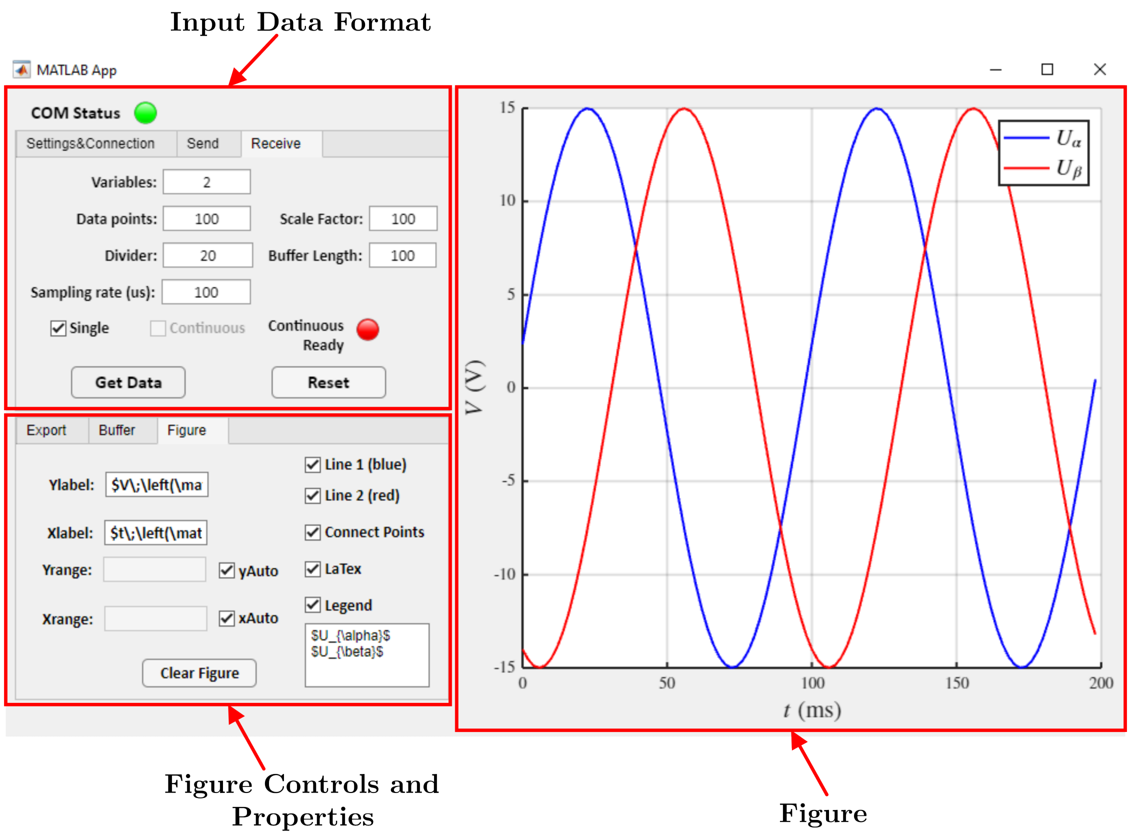


Variables:	2		
Data points:	1000	Scale Factor:	100
Divider:	1	Buffer Length:	100
Sampling rate (us):	100		

Figure 5.3-7: Interface for receiving data from a microprocessor.

5.3.2.4 Visualization

Once data have been received, the app plots them to the `UIAxes` environment, see figure 5.3-8. The user can customize its appearance either before or after receiving the data. The app remembers the settings and uses them for all subsequent figures, unless changed again. The Figure Controls tab has options to add labels to both axes and a legend, both with optional LaTeX support for typesetting. Custom range for both axes can be selected or made `auto`, i.e., MATLAB figures the range on its own. The user can also choose to display just one of the lines or make it so that only the discrete data points are shown.



Variables: 2
Data points: 100
Scale Factor: 100
Divider: 20
Buffer Length: 100
Sampling rate (us): 100
[✓ Single] Continuous
Continuous Ready

Ylabel: $V\;\left(\ma
Xlabel: $t\;\left(\mat
Yrange: [✓ yAuto]
Xrange: [✓ xAuto]
[✓ Line 1 (blue)]
[✓ Line 2 (red)]
[✓ Connect Points]
[✓ LaTex]
[✓ Legend]
U_{α}
U_{β}

Figure 5.3-8: Data visualization example (Single mode).

The Single mode is relatively straightforward as it contains just three operations, i.e., reception of data, decoding and plotting. The Continuous mode on the other had is more complicated. In theory we could just keep adding new data to the already existing ones and plot the enlarged set repeatedly. Such solution would face severe performance issues, however. One issue with the App Designer is the poor performance of the `UIAxes` environment which, unlike the standard `figure` environment written in Java, is written in HTML. So, if we were to plot an ever increasing set of data, eventually just to display the next frame would take unbearable amount of time. The main issue is that if a plot is to be updated, MATLAB replots the whole data set, including the points that had already been plotted. Thus, to achieve reasonable and stable performance the maximum size of the data set must be limited. To achieve this, the Continuous mode functions as sort of a sliding window. The length of the window is set by the aforementioned Buffer Length property. The length is measured in terms of data frames where each frame contains 20 samples. As already mentioned, the frame is buffered inside the microprocessor and then send to the computer all at once. The process of managing the data is depicted in figure 5.3-9.

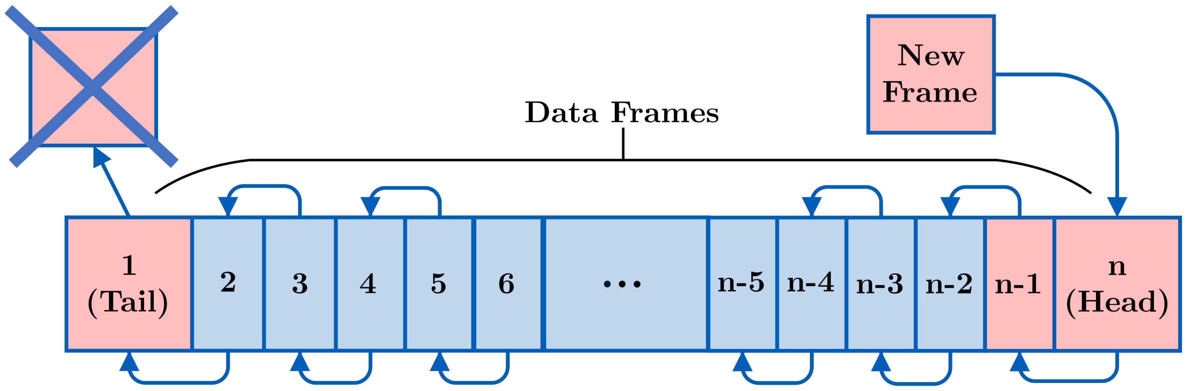
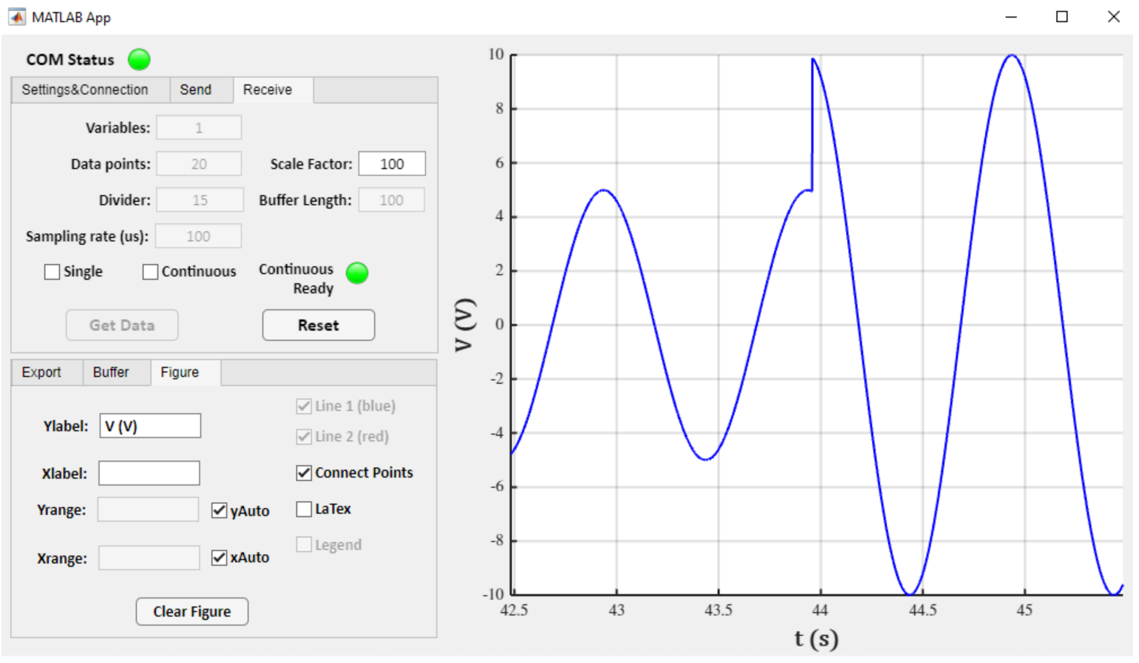


Figure 5.3-9: Data manipulation during continuous mode of operation.

Before the visualization starts, a buffer is created and initialized with `nan` values based on the Buffer Length parameter. Once the first frame is received, it is placed at the start of the buffer which is then plotted. Only the first frame is shown as the `plot` function ignores all `nan` values. The second frame is placed in front of the first one and the result is plotted. This continues until the buffer is full. Once it is full, all frames already inside are shifted to the left which places the oldest frame all the way to the front where it gets replaced by the just received one. As such the buffer size remains constant, thus maintaining reasonable performance. The maximum attainable refresh rate appears to be around 20 frames per second. The shifting operation is illustrated by the following piece of code.

```
pomArray=circshift(app.data,-1*app.DatapointsEditField.Value);
app.data=[pomArray(1:end-1*app.DatapointsEditField.Value) results
    (1:end)];
```

Figure 5.3-10 shows how the Continuous mode looks like. The horizontal axis updated automatically to create the sliding effect. Since the vertical axis range mode is set to `auto` by default, the figure automatically adjusts itself to fit the incoming data.



Variables: 1
Data points: 20
Scale Factor: 100
Divider: 15
Buffer Length: 100
Sampling rate (us): 100
Ylabel: V (V)
Xlabel:
Yrange: ✓ yAuto
Xrange: ✓ xAuto

Figure 5.3-10: Data visualization example (Continuous mode).

Chapter 6

Platform Functionality Verification

To verify functionality of the designed inverter platform, a FOC algorithm of a SPMSM was developed on the F280049C microprocessor, as it is an ideal way to evaluate all the different components of the platform. Mainly the exchange of both digital and analog signals through the isolation barrier as well as the encoder interface and, of course, the inverter module itself. Furthermore, it provides an opportunity to use the developed Real-time monitoring interface and verify that the proposed communication protocol can be used for tuning and debugging a motor control algorithm. The algorithm is written in the C language and was developed in the Code Composer Studio IDE version 11.0.0.00012. The whole CCS project is attached to the thesis. The following sections present an overview of the implementation process and measured results.

6.1 FOC Implementation

The implemented FOC matches the one depicted by the block diagram in figure 1.3-2. It runs the motor in a speed loop and utilizes an incremental encoder to measure the rotor speed and position. No field-weakening is used; thus the machine can only be driven up to its nominal speed. As already mentioned, the core of any motor control algorithm is a proper configuration of the microprocessor and its peripherals. The used configuration of the F280049C has already been discussed in chapter 4. Therefore, this chapter focuses on the actual motor control algorithm and its main components.

6.1.1 Processing of Measured Analog Quantities

6.1.1.1 Current, Voltage and Temperature Measurement

Precise knowledge of motor currents is critical to ensure optimal drive performance. Since the microprocessor is equipped with three ADCs, current in all phases is sampled simultaneously whenever the PWM `TBCTR`=0. If the currents are measured using low-side shut resistors this is necessary to ensure that a current runs thorough all three shut resistors. However, even if the current is measured through different means, it is still sampled either when the `TBCTR`=0 or `TBCTR`=max. Sampling at such occasion ensures in an ideal case that only the fundamental harmonic component is sampled [5], see figure 6.1-1.

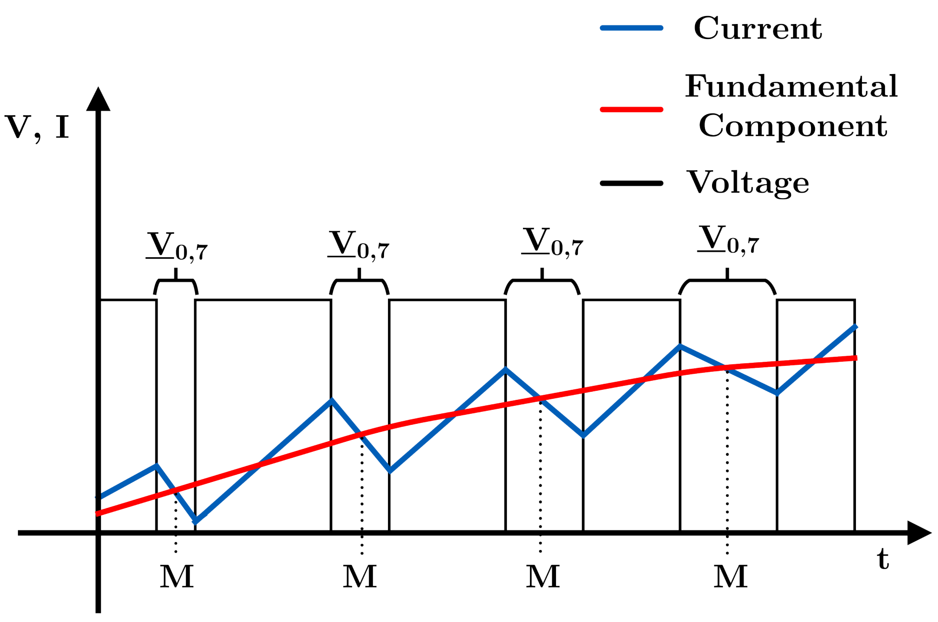


Figure 6.1-1: Timing of current sampling.

To allow for a bidirectional current measurement, the corresponding voltage signal is offset by roughly 1.65 V. The offset is not constant as the output of the operational amplifiers contains noise and also fluctuates with temperature. Therefore, at the start of the program when no current is flowing through the motor, the software automatically measures the offset for all phases. It is done by sampling 1000 values from each phase and calculating their mean value. The offsets are stored and used throughout the code. Moreover, the ADC measures voltage. Of course, to convert the measured voltage into current, the corresponding scaling factor must be known. The most reliable way to obtain the scaling factor is by sending a DC current to one phase at a time and measure the current by an ammeter or a current probe and an oscilloscope. Then, write down the current and the voltage reading of the ADC. Repeat the process for multiple values of current both positive and negative. After all the three phases were measured, the results are plotted and fitted with a first-degree polynomial. The polynomial coefficient corresponds to the current scaling factor. Therefore, the current is obtained as

$$i = k_{\text{current}}(\texttt{ADCRESULTx} - \texttt{OFFSET}). \tag{6.1-1}$$

The SVM requires the value of the DC bus voltage to properly generate the compare values for the PWM modules. The precision, however, is not as critical as in the case of the motor current. Especially when field-weakening is not used. Additionally, the DC bus measurement does not contain any offset as the DC bus voltage cannot be in principle negative. Therefore, the ADC reading is just converted to voltage and multiplied by the inverse of the voltage divider ratio used to step down the DC bus voltage.

Since the IGBT power module has also a die temperature sensor, its output is also being monitored by the microprocessor. The datasheet of the power module contains a voltage-temperature graph. The plot was recreated in MATLAB and fitted with a first-degree polynomial. The polynomial is used within the code to directly convert the measured voltage into temperature in °C.

6.1.1.2 Speed and Position Measurement

Two main approaches can be adopted to calculate the motor speed using an incremental encoder. The first method revolves around counting the number of encoder pulses within a fixed time frame, sometimes referred to as the "M" method. The second one measures the elapsed time between two consecutive encoder pulses, hence referred to as the "T" method. However, the accuracy of both methods depends on the rotor speed. The M method suffers during low speeds when no pulse is counted in the given time frame. Conversely, the T method suffers at high speeds especially when a high-resolution encoder is used as the time between pulses gets too short. If the measured time is too short the result is deteriorated by the resolution of the used counter [28].

The authors of [37] introduced a novel method for speed calculation which is a combination of the two aforementioned methods, hence it is called the "M/T" method. The idea is to count the number of encoder pulses as well as the number of timer counts in a fixed time interval. The accuracy is significantly improved and does not depend on the motor speed. Nevertheless, the "M/T" method still suffers at ultra-low speeds when no encoder pulses are counted in the given time frame.

The speed is calculated using the following formula [37]

$$n = \frac{60 f_\mathrm{c} \Delta m_1}{P \Delta m_2}, \tag{6.1-2}$$

where n is the rotor speed in rpm, f_c is the timer frequency in Hz, P is the number of encoder pulses per revolution, Δm_1 is the number of encoder pulses and Δm_2 is the number of timer counts. To determine the sign, the direction flag of the eQEP unit is used. The "M/T" method is easily realizable using the eQEP unit of the microprocessor. The `calcSpeed` function is used within the code to calculate the motor speed. The speed information is updated every 450 μs. This is to account for the low inertia of the used motor, otherwise calculation rate of around 1 ms should suffice.

Position calculation is straightforward as the position counter is reset every time it reaches the number of pulses per revolution. Therefore, the rotor position can be directly obtained using the current value in the position counter through the following formula

$$\vartheta = \frac{2\pi p_\mathrm{p}}{P}\texttt{QPOSCNT}. \tag{6.1-3}$$

The direction is taken care of automatically as the position counter increments during counter-clockwise rotation and decrements during clockwise rotation. The angle information is updated during every ADC interrupt in order to achieve maximum precision.

6.1.2 Initial Rotor Alignment

A specific feature of all synchronous machines is that the initial rotor position must be known. One way is to align the d-axis of the synchronous reference frame with the α-axis of the stationary reference frame. That way the initial rotor angle is set to zero. The easiest way to accomplish the alignment is to apply a constant voltage vector where $v_\alpha > 0$ and $v_\beta = 0$. Such approach can only be taken if the machine is able to rotate freely and is only under light load. Since that will be the case for the driven machine in this thesis, before the FOC starts, the $(4, 0)$ voltage vector is applied to the machine for 50 ms to set the initial rotor position.

If the machine is loaded but can still rotate freely, a simple PI current controller could be used. That way the alignment current could be controlled and adjusted depending on the load. The controller would output the voltage vector magnitude corresponding to the α component. In applications, where the motor cannot freely rotate, algorithms for detecting the initial rotor position must be used [38].

6.1.3 PI controllers

6.1.3.1 Continuous Time Variant

The implemented FOC utilizes a standard proportional-integral (PI) controllers to control the motor speed as well as the current. The controller is described by the following equation [39]

$$u(t) = K_\mathrm{p}\, e(t) + K_\mathrm{I} \int_0^t e(\tau)\, \mathrm{d}\tau, \tag{6.1-4}$$

where $u(t)$ is the control variable, $e(t)$ is the error value and K_p, K_I are the proportional and integral gains, respectively. Sometimes a derivative term is also used, its purpose is to predict the behavior of the error value. However, since a derivative amplifies all high frequency noise, it is not practical to use it in a PWM controlled system [5], [39].

In practice, every PI controller should have a mechanism in place to limit the wind-up of the integral term, i.e., an anti-windup. The control variable can in principle be anything the controller asks for. However, the output of every actuator is limited. For example, the maximum output voltage of a two level voltage source inverter depends on the available DC link voltage and modulation technique. Therefore, the controller output must be limited to match the capabilities of the corresponding actuator. The control variable, however, does not change instantaneously leading to the increase of the error value when the control variable becomes saturated. The error value is being accumulated in the integral term resulting in a significant overshoot. When the sign of the error value changes, the integral term takes a while to decrease leading to poor dynamic behavior of the system [39].

One possible anti-windup method is the back calculation. The unsaturated control variable is subtracted from the saturated one. The result is multiplied by a gain and added to the integrator input. Block diagram of a PI controller utilizing the back calculation anti-windup method is depicted in figure 6.1-2 [39].

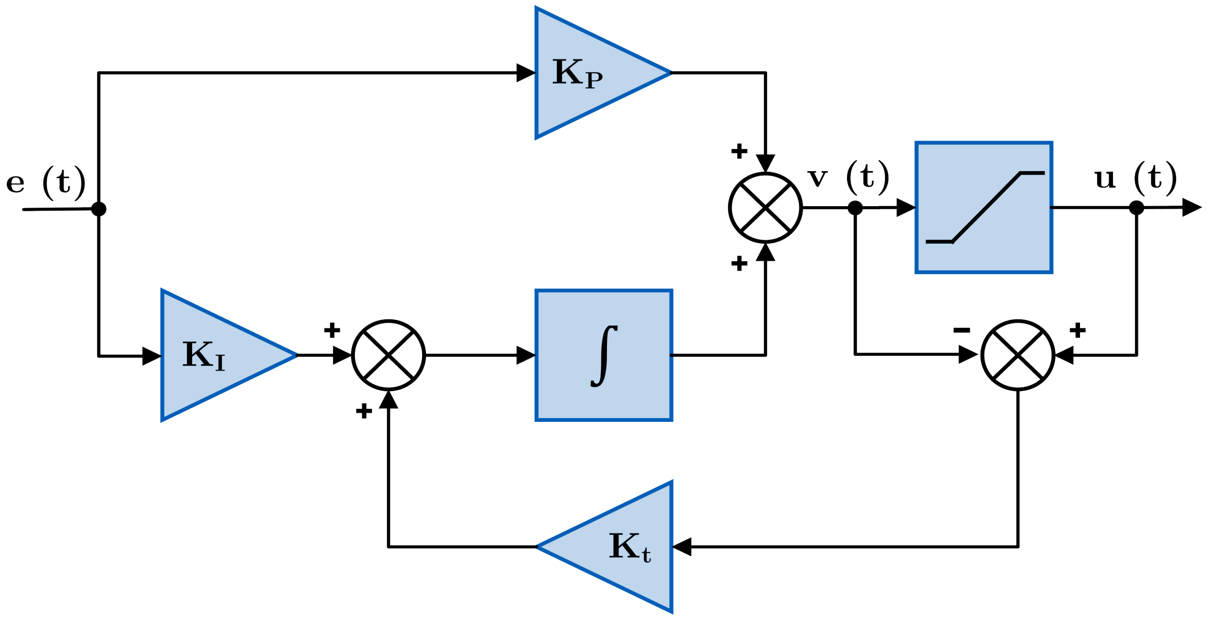


Figure 6.1-2: Block diagram of a parallel PI controller with an anti-windup.

6.1.3.2 Discretization

In order to implement a PI controller on a digital microprocessor, the equation 6.1-4 must be properly discretized. The proportional term remains largely unchanged, however, the integral term must be substituted by a discrete summation. This thesis uses the left rectangle rule to approximate the integral. The approximation is defined as [39]

$$\int_0^{kT} x(t)\,\mathrm{d}t \approx T \sum_{i=0}^{k-1} x(i). \tag{6.1-5}$$

Where k represents the sample number and T is the sampling period. Together they make up the discrete time defined as kT. Substituting equation 6.1-5 into 6.1-4 and changing the continuous time t into the discrete time kT yields [39]

$$u(kT) = K_\mathrm{P}\, e(kT) + K_\mathrm{I} \sum_{i=0}^{k-1} e(i). \tag{6.1-6}$$

Equation 6.1-6 describes the discrete time variant of a PI controller usually referred to as proportional-summation (PS) controller. The following code illustrates the the implementation used within the code. The `__fsat` intrinsic is used instead of a conditional expression to perform the output saturation.

```
float32 piController(Pi_t* controller, float32 refValue, float32
   inputVal)
{
    float32 out=0.0;
    float32 outLimited=0.0;

    controller->error=refValue-inputVal;
    out=controller->errorSum+controller->Kp*controller->error;
    outLimited=__fsat(out,controller->max,controller->min);
    controller->errorSum+=controller->Ki*controller->error+
   controller->Kt*(outLimited-out);

    return outLimited;
}
```

6.1.3.3 Calculating the Limits for Current Controllers

The output of the current controllers must be limited so that the magnitude of the reference voltage vector does not exceed the maximum realizable voltage of the inverter, i.e.,

$$|\underline{v}_\mathrm{s}^*| = \sqrt{{v_\mathrm{sd}^*}^2 + {v_\mathrm{sq}^*}^2} \leq V_\mathrm{max}. \tag{6.1-7}$$

The maximum realizable voltage V_max is, as explained in chapter 2, set to $\lambda V_\mathrm{DC}/\sqrt{3}$. Different approaches exist to ensure that the reference voltage does not exceed the V_max. The approach taken in this thesis is called the dynamic saturation with priority placed on the d-axis current i_{sd}.

The saturation of the i_{sd} controller is set to $\pm 50\,\% V_\mathrm{max}$. Subsequently the saturation limit for the i_{sq} controller is calculated based on the actual value of v_sd^* using the Pythagorean theorem, i.e.,

$$v_\mathrm{sq(max)} = -v_\mathrm{sq(min)} = \sqrt{V_\mathrm{max}^2 - {v_\mathrm{sd}^*}^2}. \tag{6.1-8}$$

6.1.4 Motor Control State Machine

The state machine running the motor control is depicted in figure 6.1-3. The entire motor control is done within the ADC interrupt so both the priority and precise timing are ensured. The sampling period corresponds to the PWM frequency which is set to 10 kHz yielding a sampling period of 100 μs.

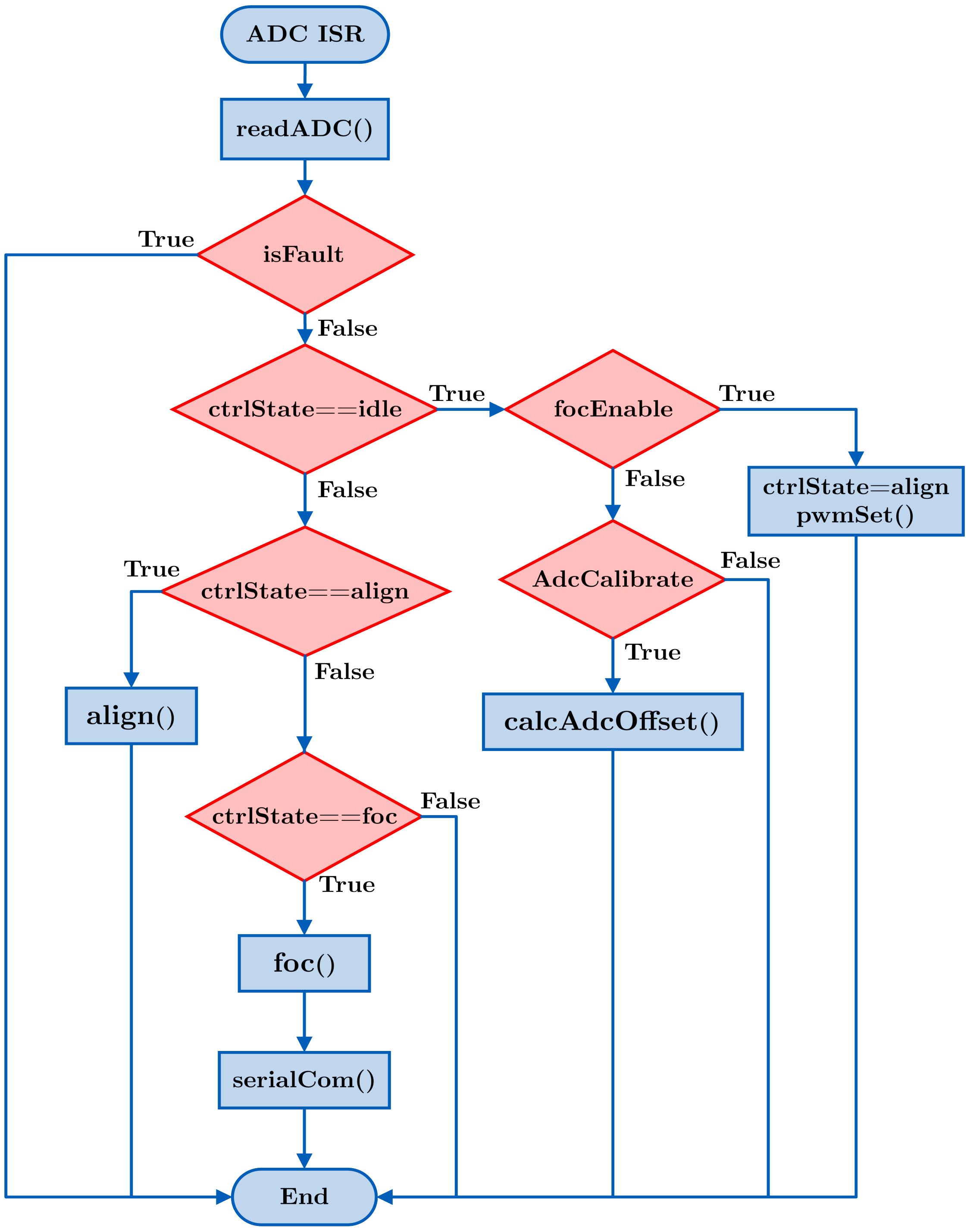


Figure 6.1-3: Flowchart of the motor control state machine.

Upon entering the ADC interrupt service routine (ISR), the program acknowledges the interrupt flag and reads the content of the respective ADCRESULT registers. The read values are converted into physical quantities using the aforementioned techniques. Then, the `isFault` flag is checked if any fault has occurred. The program recognizes three different types of faults, i.e., overcurrent, overtemperature and overspeed. Upon registering either of them, the program forces all PWM outputs low and sets the appropriate flag whilst also setting the general `isFault` flag. So, if a fault has occurred the FOC cannot be executed and the program exits the ADC ISR. If the `isFault` has not been set the program checks the control status which can be `idle`, `align` or `foc`.

If `ctrState=idle` and the `focEnable` flag is also set, then the state machine moves to the next state and all PWM outputs are enabled. Conversely, if the `focEnable` flag is not set then ADC offsets are calibrated. This is done only once and upon completion the `AdcCalibrate` flag is forced low. On the subsequent entries, the program just idles waiting for the `focEnable` flag to be set. The `align` state as the name suggest performs the initial rotor alignment as described above by applying a constant voltage vector $(4, 0)$ for 50 ms. After the time elapses, the control status is automatically changed to `foc` and the software begins to drive the motor.

During the FOC algorithm, the program first calculates the rotor position using the value in the eQEP position counter and calls the speed controller. The speed controller, however, does not have to be executed during every ISR since the speed is updated only every 450 μs. Therefore, it is called only on every fifth ADC ISR. Subsequently the measured motor currents are transformed using the Clarke and Park transforms into the synchronous reference frame. The transforms are just equations 1.2-3 and 1.2-6, respectively, rewritten in C. The TMU intrinsic were used to implement the Park transform, thus greatly accelerating the computation. With the currents transformed, the i_{sd} controller is called first. Subsequently, the saturation limits for the i_{sq} controller are calculated and the controller itself is executed. Finally, the reference voltage vector is transformed back to the stationary coordinates via the inverse Park transform and the PWM modulator is called. The modulator performs the seven-segment SVM and writes the new compare values to the appropriate PWM registers. If data logging is enabled, then serial communication module described earlier is executed.

6.2 Experimental Results

This chapter presents results from experiments preformed on the designed inverter platform. Those include signal integrity verification on the signal interface board as well as a motor start up. The goal was to verify that the platform is capable of implementing a FOC of a low power AC machine. The results are presented in the form of oscillograms, and figures obtained from the real time monitoring interface. Furthermore, parameters of the used motor together with the settings of the controllers is also presented.

6.2.1 Signal Integrity Verification

The main signals of interest are the PWM signals, Encoder outputs and analog signals, especially currents. With the PWM signals we are mainly looking for minimal transmission delay through the isolation barrier and overall, well behaved signal, i.e., fast rise and fall times with minimal oscillations around the edges. Similarly, the encoder signals should have sharp edges with minimal distortion to avoid false readings by the microprocessor. Additionally, the encoder signals should be stepped down properly to 3.3 V in order to avoid damaging the microprocessor. The analog signals should contain as little noise as possible while accurately representing the measured waveform. This is especially true for the current signals as they are AC in nature and their accuracy is crucial for any motor control algorithm. The DC bus voltage measurement is also important but does not have the same accuracy requirements and should not exhibit any rapid changes. The temperature measurement is certainly important for safety purposes, it is not required by the control algorithm itself. Much like the DC bus voltage should not exhibit any dynamic behavior.

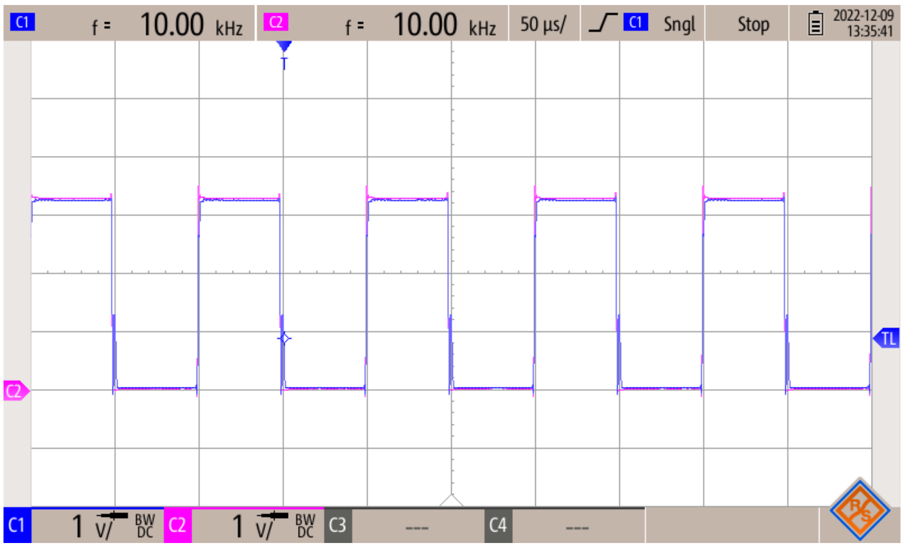


Figure 6.2-4: PWM signal before (magenta) and after (blue) the isolation barrier.

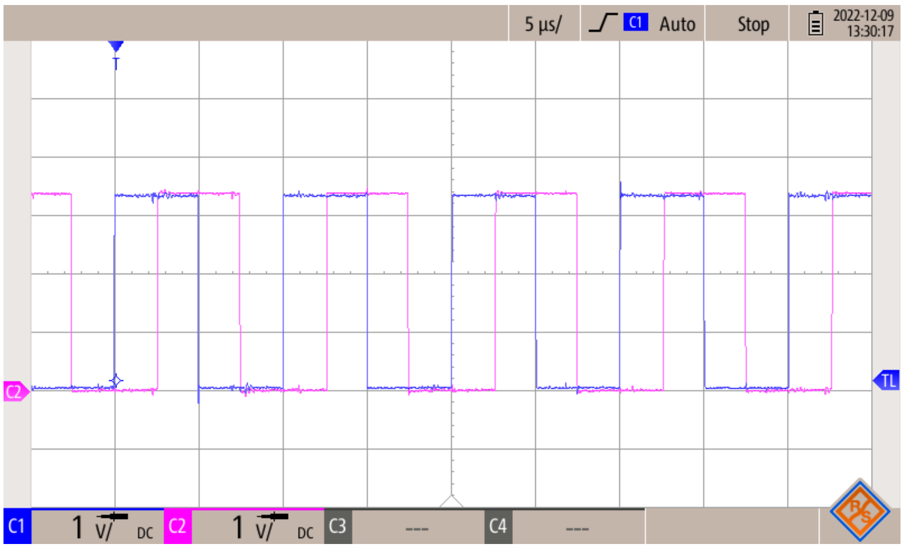


Figure 6.2-5: Processed encoder signals – signal A (blue), signal B (magenta).

By looking at both figures 6.2-4 and 6.2-5 we can conclude that the interface board preserves the integrity of the digital signals in a satisfactory manner. Figure 6.2-4 depicts the PWM signal for the high side IGBT of phase A. The signal before the isolation barrier is shown in magenta while the signal after the barrier is shown in blue. The transfer delay through the isolation barrier is negligible and coincides with the 11 ns specified by the manufacturer. Furthermore, the PWM signal exhibits minimal distortion. Similarly, the encoder signals contain virtually no distortion and the differential 5 V signals coming from the encoder are converted into single-ended 3.3 V signals.

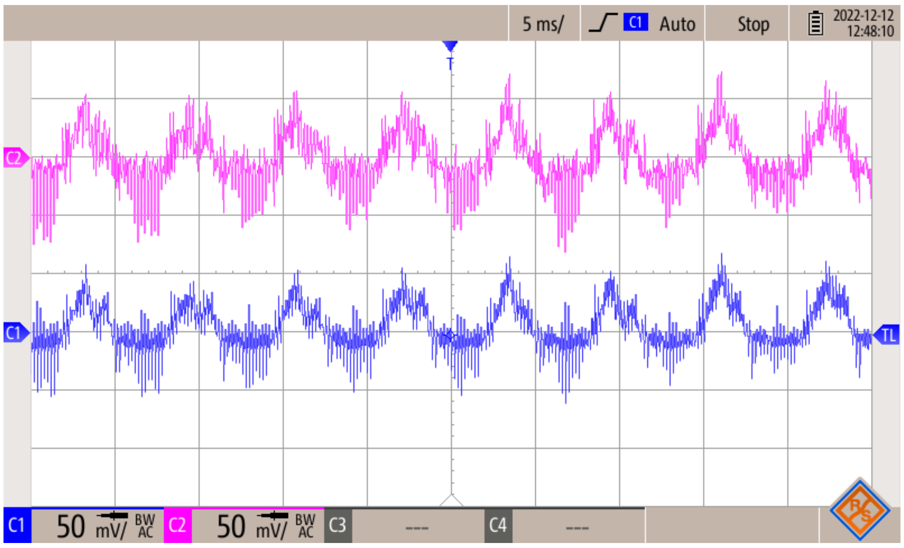


Figure 6.2-6: Output of the current sensing network before (magenta) and after (blue) the isolation barrier.

Finally, the output of one of the current sensing networks is depicted in figure 6.2-6. The output of the isolation amplifier, i.e., the the input to the ADC, is represented in magenta while the output of the operational amplifier connected to the shunt resistor is given in blue. The measurement was done while the motor was spinning at 250 $\text{rad} \cdot \text{s}^{-1}$ under no load. Which implies low stator current, hence the oscillogram only shows the AC component without the DC offset for better resolution. Even at such low currents, the corresponding voltage signal is transferred across the isolation barrier with satisfactory precision. The waveforms appear to be unsymmetrical with only a singe polarity being measured properly. Such behavior, however, is a characteristic feature of a low-side shunt measurement. Current flows through the shunts only when the corresponding low-side transistor is turned on, hence only the current flowing from the machine is captured properly. Current flowing to the machine appears on the shunts only right after the corresponding high-side transistor turns off and the current flows through the freewheeling diode of the low-side transistor. Alternatively, if a zero vector $\underline{v}_0$ is applied to the inverter, the direction of current flow in each phase is preserved by the winding inductance, see figure 6.1-1, allowing the shunt resistor to measure both polarities. The measurement instances are represented by the narrow negative pulses on the oscillogram which are difficult to properly capture especially at such low currents.

This supports the earlier argument that the duration of the zero vector must be long enough to achieve accurate sampling. Therefore, such measurement technique can only be used if a sufficiently long zero vector is guaranteed during each switching period. If, for example, a direct-torque control or a different modulation strategy are used, the current measurement strategy should be adapted accordingly.

6.2.2 Motor Start Up

6.2.2.1 Experimental Set Up

The driven motor is the Siemens 1FL6042 low inertia PMSM intended to be used in servo applications. Its nominal parameters are listed in table 6.2-1.

Parameter	Value
P_n	0.75 kW
T_n	$2.39\ \mathrm{N \cdot m}$
n_n	3000 rpm
P_p	4
I_n	4.7 A
V_n	111 V
J_n	$0.897 \cdot 10^{-4}\ \mathrm{kg \cdot m^2}$

Table 6.2-1: Nominal parameters of the 1FL6042 low-inertia PMSM.

All the presented tests were done under no load conditions. That is because properly loading the machine would be, from a mechanical point of view, complicated and time-consuming. Moreover, loading the machine and testing it was not part of the thesis objectives. The control algorithm had to have been adjusted to account for the no load operation. Specifically, the speed controller. Since the machine has a low inertia which, of course, is desirable in servo applications, its mechanical time constant is relatively small. Thus, it accelerates extremely fast, even more so if not loaded. To keep the acceleration in check, the speed controller was made really slow, and the saturation was set only to 25 % of the peak nominal current. When the current was set too high, the machine would accelerate too fast, essentially displacing the machine. This would also disrupt the highly accurate incremental encoder causing errors in speed calculation, thus compromising the drive performance. All controllers were tuned experimentally using the developed Real-time monitoring interface. The parameters of all controllers are summarized in table 6.2-2.

Controller	K_P	K_I	K_t
Speed controller	0.085	0.007	0.15
i_d controller	5.0	0.7	0.7
i_q controller	5.0	0.7	0.7

Table 6.2-2: Parameters of used controllers.

One thing to note is that the anti-windup gain K_t of the speed controller is much higher than the integral gain K_I. With these settings the drive exhibits much better transient performance. That is most likely because the motor accelerates so fast that it can easily overshoot. Therefore, the more aggressive anti-windup helps to quickly mitigate the overshoot. Nevertheless, the speed controller will have to tuned again once the machine is loaded. Furthermore, the no-load operation also means that the lack of a braking chopper is not an issue.

6.2.2.2 Results

This section presents the measured results during a motor start up. All the data was collected through the monitoring interface, exported to `.csv` files and then plotted. The sampling rate for the speed was set to 500 μs while for both currents it was set to 100 μs. In other words, all data points in the given time frame were collected. The motor operates under no load and responds to a step change of the reference speed from 0 to 250 $\mathrm{rad \cdot s^{-1}}$.

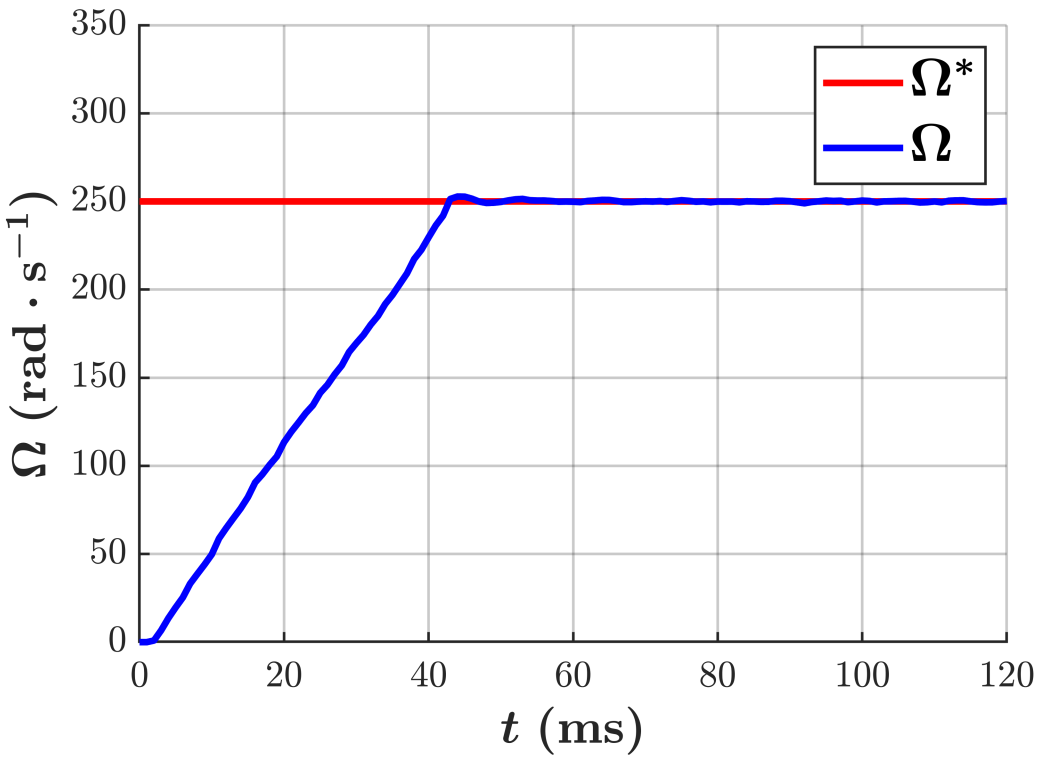
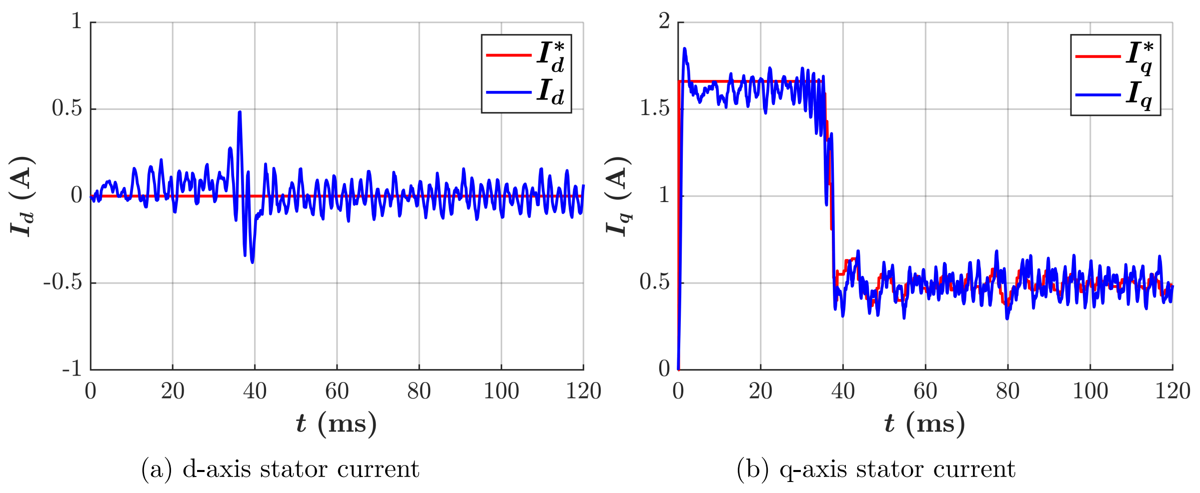


Figure 6.2-7: Mechanical angular speed of the motor during start up.

Figure 6.2-7 depicts the speed response of the motor. We can immediately notice the extremely high dynamic of the motor. Even though the current is limited only to ≈ 1.66 A, the motor reaches the reference speed in roughly 40 ms. This is in accordance with the fact the motor has considerably low inertia. The speed controller exhibits excellent capability of reaching and subsequently holding the reference speed. Virtually no oscillations are present. But, of course, the behavior is going to change significantly under load and with the saturation set to the peak nominal current or even above to allow for some overload capability. The rise time could potentially be shortened even further, but few milliseconds under no load are insignificant.



(a) d-axis stator current (b) q-axis stator current

Figure 6.2-8: Stator current components during motor start up.

Figure 6.2-8 depicts the stator current in the synchronous reference frame. The d-axis component is shown in figure 6.2-8a while the q-axis component is shown in figure 6.2-8b. Since no field-weakening is used, the d-axis current is kept at zero. We can see that the controller is capable of maintaining the reference value. Perhaps the controller should have been made a bit more aggressive as during start up a d-axis current of around 100 mA flows within the motor. This is most likely caused by the internal cross coupling between the d and q axis. Since decoupling is not used, the controllers themselves must compensate it. Therefore, if the controller is not made aggressive enough, it cannot fully compensate the coupling. Perhaps it is even more apparent when the machine reaches the reference speed, a sudden brief oscillation occurs in the d-axis current, see figure 6.2-8a. When the reference speed is reached, the q-axis current drops almost instantaneously. The sudden change in the q-axis current affects also the d-axis current through the cross coupling. Since the i_d controller is not aggressive enough, it takes a while before the coupling is compensated. During steady state, the controller keeps the q-axis current at zero, albeit with some oscillations, which are largely caused by the inevitable noise within the current sensing circuits.

The q-axis current virtually instantaneously reaches the reference value and follows any changes without significant problems. Again, some noise is present within the signal, which again largely comes from the current sensing circuits.

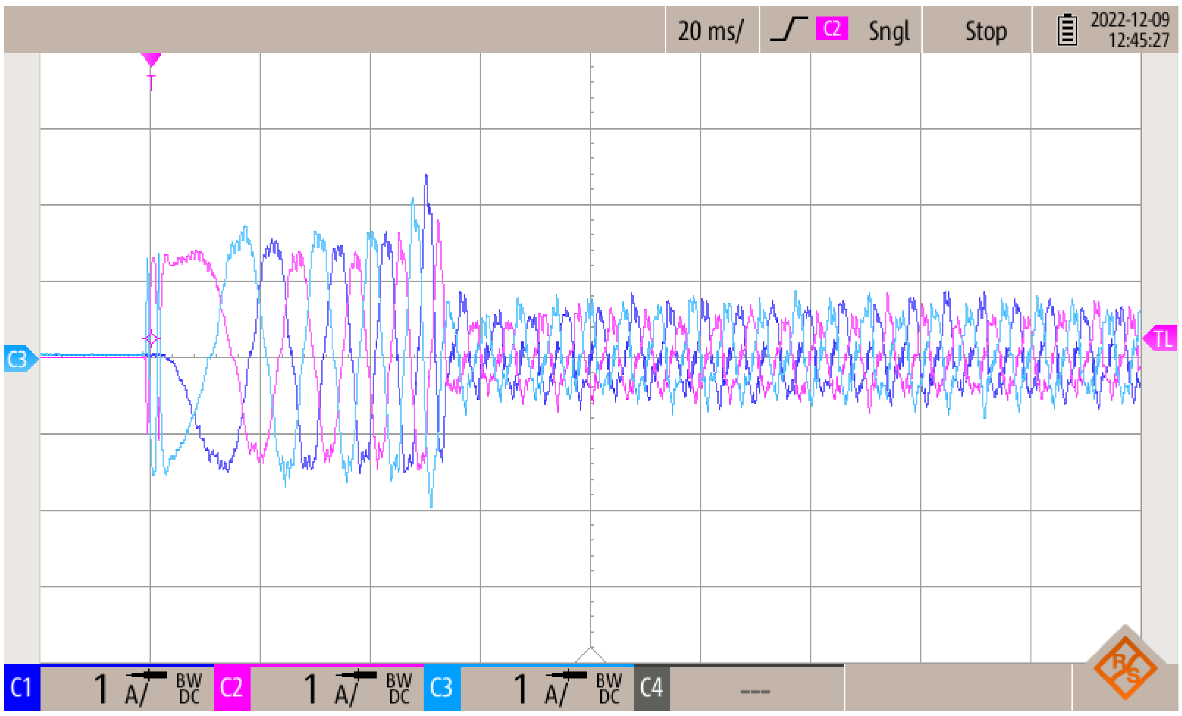


Figure 6.2-9: Three phase motor current during start up.

Finally, an oscillogram of the motor currents is depicted in figure 6.2-9. The currents are relatively sinusoidal especially during start up when the current is higher. Also, we can notice the aforementioned spike in the d-axis current on the oscillogram as well. In steady state under no load, the current waveforms are relatively poor. Most likely even finer tuning of the speed controller would be required to achieve better results as the reference value of the q-axis current oscillates a bit. Perhaps, adjusting the execution rate of the speed controller might lead to some improvements as well. Lastly, noise present in the current measurement could also deteriorate the performance especially in steady-state, where motor currents are low, which leads to smaller signal-to-noise ratio. Nevertheless, in order to thoroughly test the control algorithm, the motor would have to be loaded properly.

Conclusion

Within the presented master thesis, a functioning prototype of a modern variable-speed AC motor drive system was developed. The core of the designed system is a mobile inverter platform, capable of driving a variety of different three-phase AC motors with nominal power up to 1 kW. The platform is powered from a standard 230 V/50 Hz power line and performs all the necessary power conversions to supply all of its underlying components. Specifically, a full bridge rectifier is used to supply the DC bus of the inverter, resulting in roughly 325 V DC across the DC bus capacitor. A simple precharge circuit consisting of a resistor and a timer relay keeps the inrush current of the DC bus capacitor at a reasonable value. Additionally, a switch-mode power supply is used to create a base voltage of 15 V DC for powering all the signal circuits. Essential part of every modern motor drive system is a powerful microprocessor with appropriate peripherals and sufficient processing power to run all the different motor control algorithms. The designed platform features the TMS320F280049C digital signal processor from Texas Instruments. The microprocessor is a part of the Launchxl-F280049C evaluation board which adds extensive debugging and communication capabilities enabling fast and efficient prototyping of various motor control algorithms.

Special attention was given to the issue of signal exchange between the inverter and the microprocessor. Due to safety complications, which arose from the combination of supplying the DC bus from a rectifier and a common power and signal reference ground on the inverter board, a signal interface board was designed. The board implements a complete galvanic isolation between the inverter and the evaluation board. Thus, greatly elevating the safety of the platform during measurements. Both digital as well as analog signals are transferred across the barrier. Moreover, the board provides additional auxiliary functions among which the incremental encoder interface with a differential Rs-422 receiver stands out.

Being able to visualize all the different system variables is of tremendous importance during the development process of any real-time control application. In the case of electric motor drives, the algorithm works with a broad set of internal variables. These variables are typically the result of a mathematical transformation, model, or an observer. Nevertheless, an oscilloscope cannot be used to visualize them as these variables only exist within the algorithm. However, to properly tune the current controllers in a field-oriented control algorithm, having the ability to see the d-axis and q-axis currents is essential. Therefore, a visualization application paired with a communication API was designed. The app was built in MATLAB utilizing the App Designer interface. The API contains functions for receiving and sending data as well as a configuration of the serial peripheral interface of the F280049C microprocessor. The application supports either online or offline mode of data visualization and comes with a straightforward communication protocol for sending commands back to the microprocessor. On top of that it comes with a variety of tools for communication management and data processing. The whole package was designed with portability in mind; therefore it can be used with other C2000 microprocessors with minimal effort. The desktop application could even be used with microprocessors from other manufactures. The user would have to only program the communication within the microprocessor, while following the conventions used within the thesis.

To verify the functionality and applicability for motor control development of the designed inverter platform as well as the real-time monitoring application, a sensored filed-oriented control of a SPMSM was programmed into the F280049C microprocessor. The algorithm was evaluated on a 0.75 kW machine. A stable control algorithm capable of high dynamic drive performance was developed proving the feasibility of the platform for future prototyping and development. Moreover, the real-time monitoring application has proven to be particularly useful during the tuning and debugging process.

Whilst the work presented in this thesis can be considered complete and functional, it presents various opportunities for further improvement. Going forward, the signal interface board is going to be modified by moving the encoder interface to the inverter side of the board, which should solve the issue of high parasitic loop currents. Furthermore, a braking chopper will be added to the inverter to enable the machine to be properly loaded. Moreover, two identical 1FL6042 SPMSMs will be coupled together and an identical second inverter platform will be made to create a complete experimental work space with a controllable load. The work space could be used for laboratory exercises, student theses or even research activities at the department. The real-time monitoring application has already proven useful during the tuning process of a motor control algorithm designated to be used in an e-bike, developed for an industrial partner of the department. Therefore, it has been decided to expand its capabilities by increasing the number of visualizable variables from two to at least four or five, improve the user interface and expand the messaging protocol so that the user can flexibly select which variables to send directly from the GUI without the need for reloading the control software. Additionally, the trigger conditions controlling when the data logging starts should be adjustable from the GUI as well, essentially removing the necessity to use a debugger. As such, it could be used during various future activities at the department.

References

[1] S. Sakunthala, R. Kiranmayi and P. N. Mandadi, *"A study on industrial motor drives: Comparison and applications of PMSM and BLDC motor drives,"* 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS), 2017, pp. 537-540, doi: 10.1109/ICECDS.2017.8390224.

[2] A. Vagati, G. Pellegrino and P. Guglielmi, *"Comparison between SPM and IPM motor drives for EV application,"* The XIX International Conference on Electrical Machines - ICEM 2010, 2010, pp. 1-6, doi: 10.1109/ICELMACH.2010.5607911.

[3] S. Derammelaere, M. Haemers, J. De Viaene, F. Verbelen and K. Stockman, *"A quantitative comparison between BLDC, PMSM, brushed DC and stepping motor technologies,"* 2016 19th International Conference on Electrical Machines and Systems (ICEMS), 2016, pp. 1-5.

[4] P. Vas, *"Electrical Machines and Drives: A Space-Vector Theory Approach,"* Oxford, U.K.: Clarendon, 1993.

[5] QUANG, Nguyen P. a Jörg-Andreas DITTRICH. *Vector Control of Three-Phase AC Machines: System Development in the Practice.* 1st. ed. Berlin, Heidelberg: Springer-Verlag, 2008. ISBN 1612-1287.

[6] C. Lai, G. Feng, K. Mukherjee, J. Tjong and N. C. Kar, *"Maximum Torque Per Ampere Control for IPMSM Using Gradient Descent Algorithm Based on Measured Speed Harmonics,"* in IEEE Transactions on Industrial Informatics, vol. 14, no. 4, pp. 1424-1435, April 2018, doi: 10.1109/TII.2017.2759812.

[7] M. Khayamy and H. Chaoui, *"Current Sensorless MTPA Operation of Interior PMSM Drives for Vehicular Applications,"* in IEEE Transactions on Vehicular Technology, vol. 67, no. 8, pp. 6872-6881, Aug. 2018, doi: 10.1109/TVT.2018.2823538.

[8] TRZYNADLOWSKI, Andrzej M. *Introduction to modern power electronics.* 3rd. ed. Hoboken: Wiley, 2016;2015; ISBN 9781119003212;1119003210;

[9] HOLMES, D. G. and Thomas A. LIPO.*Pulse Width Modulation for Power Converters: Principles and Practice.* John Wiley & Sons, 2003. ISBN 9780471208143;0471208140;

[10] R. Pengelly, S. Wood, J. Milligan, S. Sheppard, W. Pribble. *A review of GaN on SiC high electron-mobility power transistors and MMICs.* IEEE Transactions on Microwave Theory and Techniques - IEEE TRANS MICROWAVE THEORY. 60. 1764-1783. 10.1109/TMTT.2012.2187535.

[11] STMicroelectronic. *1500 W motor control power board based on STGIB15CH60TS-L SLLIMM™ 2nd series IPM*, February 2016 - Revised September 2019. Available at: `https://www.st.com/en/evaluation-tools/steval-ipm15b.html#documentation`

[12] Texas Instruments. *C2000™ Piccolo™ F28004x Series LaunchPad™ Development Kit User's Guide (SPRUII7B)*, June 2018 - Revised April 2020. Available at: `https://www.ti.com/tool/LAUNCHXL-F280049C`

[13] TDK Electronics. *2-line filters, SIFI-G for enhanced insertion loss*, Type: B84112G, January 2021. Available at: `https://www.tdk-electronics.tdk.com/en/179134/products/application-guides/industrial/green-power-generation-photovoltaic-systems/dc-link/emc-filters`

[14] STMicroelectronic. *SLLIMM - 2nd series IPM, 3-phase inverter, 20 A, 600 V, short-circuit rugged IGBT*, June 2014 - Revised October 2021. Available at: `https://www.st.com/en/power-modules-and-ipm/stgib15ch60ts-l.html#documentation`

[15] Kim, Do-Yun, Jung-Hyo Lee, Taeck-Kie Lee and Chung-Yuen Won. *"Phase current sensing method using three shunt resistor for AC motor drive."* 2012 IEEE Vehicle Power and Propulsion Conference (2012): 78-82.

[16] STMicroelectronic. *L78, Positive voltage regulator ICs*, June 2004 - Revised September 2018. Available at: `https://www.st.com/en/power-management/l78.html#documentation`

[17] ROHM Semiconductor. *BA00ST / BA00SFP series, Regulator, low drop-out type with ON/OFF switch.* Available at: `https://cz.mouser.com/ProductDetail/ROHM-Semiconductor/BA033ST?qs=Mqkh4jHMT8zaFb4Pbp3NPg%3D%3D`

[18] RECOM Power. *RSE-Z series, Isolated DC/DC converter*, May 2014 - Revised March 2020. Available at: `https://recom-power.com/en/products/dc-dc-converters/dc-dc-converters-regulated/rec-p-RSE-2405SZ!sH2.html?4`

[19] Texas Instruments. *ISO773x-Q1 High-Speed, Robust-EMC Reinforced Triple-Channel Digital Isolators (SLLSEU3D)*, November 2016 - Revised October 2020. Available at: `https://www.ti.com/product/ISO7730-Q1?keyMatch=ISO7730QDBQRQ1&tisearch=search-everything&usecase=OPN#tech-docs`

[20] Infineon. *XENSIV™ – TLI4971 Magnetic Current Sensor Family*, February 2020 - Revised December 2021. Available at: `https://www.infineon.com/cms/en/product/sensor/current-sensors/tli4971-a050t5-e0001/#!documents`

[21] Lakkireddy GR, Mathe SE,*"A Strategy for Measuring Voltage, Current and Temperature of a Battery Using Linear Optocouplers."* World Electric Vehicle Journal. 2022; 13(12):225. Available at: `https://doi.org/10.3390/wevj13120225`

[22] Broadcom, *Optocoupler Current and Voltage Sensing Brochure*, August 2018. Available at: `https://www.broadcom.com/site-search?q=Broadcom%20Optocoupler%20Current%20and%20Voltage%20Sensing`

[23] Broadcom, *ACNT-H87B, ACNT-H87A, ACNT-H870 Precision Optically Isolated Voltage Sensor in a 15-mm Stretched SO-8 Package*, September 2017 - Revised October 2019. Available at: `https://www.broadcom.com/products/optocouplers/industrial-plastic/isolation-amplifiers-modulators/isolation-amplifiers/acnt-h870`

[24] Maxim Integrated. *MAX44259-MAX44263: 1.8V, 15MHz Low-Offset, Low-Power, Rail-to-Rail I/O Op Amps Data Sheet (Rev. 10)*, June 2011 - Revised August 2019. Available at: `https://www.analog.com/en/products/max44259.html#product-reference`

[25] Texas Instruments. *AM26C32 Quadruple Differential Line Receiver datasheet (Rev. L) (SLLS104L)*, December 1990 - Revised October 2018. Available at: `https://www.ti.com/product/AM26C32?keyMatch=AM26C32CD&tisearch=search-everything&usecase=OPN`

[26] Texas Instruments. *TXU0104 4-Bit Fixed Direction Voltage-Level Translator with Schmitt-Trigger Inputs and 3-State Outputs datasheet (Rev. A) (SCES937A)*, May 2021 - Revised September 2021. Available at: `https://www.ti.com/product/TXU0104/part-details/TXU0104PWR?keyMatch=TXU0104PWR`

[27] Texas Instruments. *TMS320F28004x Microcontrollers datasheet (Rev. F) (SPRS945F)*, January 2017 - Revised February 2021. Available at: `https://www.ti.com/product/TMS320F280049C`

[28] Texas Instruments. *TMS320F28004x Real-Time Microcontrollers Technical Reference Manual (Rev. F)(SPRUI33F)*, November 2015 - Revised December 2022. Available at: `https://www.ti.com/product/TMS320F280049C`

[29] CCSTUDIO Code Composer Studio (CCS) Integrated Development Environment (IDE) — TI.com. *Analog, Embedded Processing, Semiconductor Company, Texas Instruments - TI.com* [online]. © 1995-2022. Available at: `https://www.ti.com/tool/CCSTUDIO`

[30] C2000Ware — TI.com. *Analog, Embedded Processing, Semiconductor Company, Texas Instruments - TI.com* [online]. © 2017-2022. Available at: `https://www.ti.com/tool/C2000WARE`

[31] Texas Instruments. *C2000 Real-Time Control MCU Peripherals-Reference Guide*, June 2003 - Revised October 2022. Available at: `https://www.ti.com/lit/ug/spru566p/spru566p.pdf?ts=1672948874902&ref_url=https%253A%252F%252Fwww.google.com%252F`

[32] Texas Instruments. *TMS320C28x DSP CPU and Instruction Set (Rev. F) (SPRU430F)*, August 2001 - Revised April 2015. Available at: `https://www.ti.com/lit/pdf/spru430`

[33] Texas Instruments. *TMS320C28x Extended Instruction Sets Technical Reference Manual (Rev. C) (SPRUHS1C)*, October 2014 - Revised November 2019. Available at: `https://www.ti.com/lit/pdf/spruhs1`

[34] LEIS, John W. *Communication Systems Principles Using MATLAB.* 1st. ed. Newark: Wiley, 2018. ISBN 9781119470670;1119470676;

[35] Texas Instruments. *TMS320C28x Optimizing C/C++ Compiler v20.12.0.STS User's Guide (Rev. V) (SPRU514Y)*, August 2001 - Revised June 2022. Available at: `https://www.ti.com/lit/pdf/spru514`

[36] Documentation – MATLAB & Simulink. MathWorks [online]. Natick, Massachusetts, USA, 2022. Available at: `https://www.mathworks.com/help/`

[37] OHMAE, Tsutomu et al. *"A Microprocessor-Controlled High-Accuracy Wide-Range Speed Regulator for Motor Drives."* IEEE Transactions on Industrial Electronics. 1982, vol. IE-29, no. 3, s. 207-211. ISSN 0278-0046.

[38] S. Zossak, M. Stulraiter, P. Makys and M. Sumega, *"Initial Position Detection of PMSM," 2018 IEEE 9th International Symposium on Sensorless Control for Electrical Drives (SLED)*, 2018, pp. 12-17, doi: 10.1109/SLED.2018.8486043.

[39] Wittenmark, B., Årzén, K-E., Åström, K. J. (2002). *Computer Control: An Overview.* (IFAC PROFESSIONAL BRIEF). International Federation of Automatic Control.

Appendix A

Nomenclature

General symbols

$\underline{\boldsymbol{x}}$	space vector
x^*	reference value

Symbols and variables

L_d	direct axis inductance (H)
L_q	quadrature axis inductance (H)
R_s	stator resistance (Ω)
ϑ	rotor position (rad)
ψ_M	permanent magnet flux linkage (Wb)
p_P	number of pole pairs (1)
ω	electrical angular frequency $(\mathrm{rad} \cdot \mathrm{s}^{-1})$
Ω	mechanical angular frequency $(\mathrm{rad} \cdot \mathrm{s}^{-1})$
T_PWM	switching period (s)
V_RRM	maximum reverse recovery voltage (V)
$I_\mathrm{F(AV)}$	maximum average forward current (A)

Abbreviations

DSP	digital signal processor
PMSM	permanent magnet synchronous motor
SVM	space-vector modulation
PWM	pulse-width modulation
FOC	field-oriented control
MTPA	maximum torque per ampere
IGBT	insulated-gate bipolar transistor
MOSFET	metal-oxide-semiconductor field effect transistor
AC	alternating current
DC	direct current

ADC	analog-to-digital converter
DAC	digital-to-analog converter
EMI	electromagnetic interference
JTAG	join test action group
UART	universal asynchronous receiver-transmitter
CAN	controller area network
PCB	printed circuit board
PWM	pulse-width modulation
CRC	cyclic redundancy check
CPU	central processing unit
IDE	integrated development environment
FPU	floating point unit
FIFO	first in, first out
API	application programming interface
ISR	interrupt service routine

Appendix B

Contents of Electronic Attachments

The attachments are organized into a single `.zip` file called *thesis_attachments_Baum* which contains the following:

- ***pmsm_foc_project***: CCS project folder containing the source code of the implemented FOC algorithm,
- ***interface_board_manufacture***: ZIP folder containing manufacturing files of the designed interface board,
- ***transfer_board_manufacture***: ZIP folder containing manufacturing files of the designed signal transfer board,
- ***interface_board_layout***: PDF document with the layout of the designed interface board,
- ***transfer_board_layout***: PDF document with the layout of the designed transfer board,
- ***RTM_App***: the real-time monitoring MATLAB application file,
- ***interface_board_schematic***: PDF document with the entire schematic of the designed interface board,
- ***transfer_board_schematic***: PDF document with the entire schematic of the designed transfer board.