Implementation and verification of a hardware-based controller for a three-phase induction motor on an FPGA

by

Marcel Cases Freixenet
BSc

A thesis submitted in partial fulfilment of the requirements for the degree of

Industrial Electronics and Automatic Control Engineering

Supervisors
David Soler Jimenez (UPC)
Aleksander Sudnitsõn (TalTech)

Escola Politècnica Superior d'Enginyeria de Manresa
Universitat Politècnica de Catalunya. BarcelonaTech, and

School of Information Technologies
Tallinn University of Technology. TalTech

Manresa

Submitted March 2019

Implementation and verification of a hardware-based controller for a three-phase induction motor on an FPGA

Marcel Cases Freixenet

Implementation and verification of a hardware-based controller for a three-phase induction motor on an FPGA
Book size: A4
Pages: 64
Word count: 12.530
Document written with LaTeX 2_ε
Compilation date: March 8, 2019

The electronic version of this thesis can be downloaded from:
https://www.marcelcases.com

Abstract

català

L'objectiu d'aquesta tesi és estudiar diverses tècniques de control motor per tal d'implementar i verificar un controlador basat en *hardware* per a un motor d'inducció trifàsic, desenvolupat en llenguatge VHDL i funcionant en una FPGA Artix-7 (Xilinx). Aquest controlador està basat en tècniques de variació de freqüència. Els mòduls que defineixen la descripció de *hardware* funcionen simultàniament entre ells, i permeten agilitzar el sistema, millorant el rendiment i la resposta del motor, en comparació amb un microcontrolador. Aquesta tesi està relacionada amb els sistemes digitals, l'electrònica de potència i els sistemes de control.

eesti keel

Käesoleva töö eesmärk on uurida mootorite juhtimise peamisi tehnikaid, et projekteerida ja rakendada riistvarapõhist kontrollerit kolmefaasilise induktsioonimootori jaoks, mis on välja töötatud VHDL keeles ja töötab Artix-7 FPGA (Xilinx). See kontroller põhineb muutuva sagedusega ajamitehnikatel. Moodulid, mis määratlevad selle kontrolleri riistvara kirjelduse, suhtlevad üksteisega ja võimaldavad mootoril kiiremini reageerida ning parandavad ka selle jõudlust võrreldes mikrokontrolleriga. Käesolev töö on seotud digitaalsüsteemide, võimsuselektroonika ja juhtimissüsteemidega.

English

The aim of this thesis is to study the main techniques of motor control in order to implement and design a hardware-based controller for a three-phase induction motor, developed in VHDL language and running on an Artix-7 FPGA (Xilinx). This controller is based on variable-frequency drive techniques. The modules that define this controller's hardware description run concurrently to each other, and they allow the motor to have a better time response and they also improve its performance compared to a microcontroller. This thesis is related to digital systems, power electronics and control systems.

Dedicat a tothom qui m'ha ajudat

Table of Contents

List of Figures

List of Tables

Listings

Abbreviations

AC	Alternating Current
ACIM	Alternating Current Induction Motor
ADC	Analog-to-digital Converter
ASIC	Application-specific Integrated Circuit
BLDC	Brushless DC electric motor
DC	Direct Current
DTC	Direct Torque Control
FOC	Field-oriented Control
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
IGBT	Insulated-gate Bipolar Transistor
LUT	Look-up Table
PI	Proportional Integral
PMSM	Permanent-magnet Synchronous Motor
RTL	Register-transfer Level
SPWM	Sinusoidal Pulse Width Modulation
SVPWM	Space Vector Pulse Width Modulation
THD	Total Harmonic Distortion
VFD	Variable-frequency Drive
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

Part I

Background

1 Introduction

Thomas Davenport[1] developed the first motor powered by electricity. Soon after, Nikola Tesla[2] discovered Alternating Current (AC) electricity, and he implemented it to power the first Alternating Current Induction Motor (ACIM) in 1887.

AC motors have been improved ever since, both in their assembly and their control methods. They are used in many sectors: in industrial drives, vehicles (cranes, trains, cars, buses) or construction, among others. Nowadays they are boosting the electric mobility day after day as they are being used more frequently in the automotive industry.

It is estimated that about 25% of the world's electrical energy is consumed by electric motors, specifically in industrial applications. The way these motors are commanded has a direct impact in the consumption of energy and the motor's performance. This is why these control techniques have to be optimized, depending on the type of motor they have to control, to avoid energy losses and damaging components built in the motor.

There are many control techniques used to manage their operation. When it comes to three-phase ACIMs, the ones that are used in machines that require high torque, there are two main techniques developed enough to control them: Direct Torque Control (DTC) and Field-oriented Control (FOC).

Both techniques are based on classic Variable-frequency Drives (VFDs), as well as V/Hz control, but they have the ability to adjust some parameters, like the motor voltage magnitude, the current angle of the motor, and the frequency of the shaft so as to control in a more precise way the magnetic flux and the torque of the motor.

FOC is a control technique in which the currents that flow through the stator, the stationary part of the motor system (outer part), are represented as two orthogonal components that can be addressed like a vector. In order to do this, some processes have to be described or programmed. These processes are the Clarke transformation, the Park transformation, and the Inverse Park transformation. Then the output signal is sent to a Space Vector Pulse Width Modulation (SVPWM) module, which has the

[1] Thomas Davenport (9 July 1802 - 6 July 1851) was a Vermont blacksmith who constructed the first American DC electric motor in 1834.

[2] Nikola Tesla (10 July 1856 - 7 January 1943) was a Serbian-American inventor, electrical engineer, mechanical engineer, and futurist who is best known for his contributions to the design of the modern alternating current (AC) electricity supply system.

function to chop the signal so that the power module, or inverter, is able to give to each one of the three phases of the motor the required voltage and current at any moment.

DTC is a less common ACIM control method that calculates and estimates the torque and magnetic flux of the motor in order to directly control the torque of the motor's shaft, and thus its speed. This involves measuring the current and voltage of the motor. The main difference compared to FOC is that it is not necessary to measure or estimate the current rotor position. The SVPWM or any other PWM modules are not necessary.

This thesis is the result of a period of research in the fields of electric motor control techniques and Hardware Description Languages (HDLs) design. The main goal of the experimental part of this project is to implement a VFD method on a FPGA using Vivado Design Suite, a Xilinx's IDE for HDLs, so that each process of this control system run concurrently to each other. This allows us to analyze the impact that it has in the motor: speed response, harmonics reduction, and other improvements compared to the motor control with classical microcontrollers.

The thesis is structured in two clearly differentiated parts: one is the theoretical background (⇨ Part II) where FOC, DTC and general V/Hz (VFD) methods are studied and analyzed as well as every component that pertains to these methods; and the other one (⇨ Part III) is the implementation of the VFD (V/Hz) controller in VHSIC Hardware Description Language (VHDL) on a FPGA and the description of the hardware used: FPGA board, power module, motor, voltage level circuits, and wiring. A study of the performance of this control method is also included.

2 Motivation

My personal motivation when choosing a topic for my Bachelor's thesis was my interest in FPGA technology and the advantages of synthesized hardware over the classic microcontrollers or any other device used to perform the operations involved in the control of Alternating Current Induction Motor methods.

Firstly, my motivation to study the hardware design and FPGAs beyond the contents studied in class was my first approach to what this thesis has ended up being. It was the opportunity I had to attend the course *Digital Systems Design with VHDL* at TalTech that helped me to learn concepts of FPGAs, VHDL and the software suite, Vivado, and to widen my previous knowledge of the VHDL language.

My intention was to find a useful application where FPGAs could have a real improvement of performance and time response. This is why, while defining the experimental part of the thesis, a good option was the implementation of a motor control method in VHDL. This was also a good reason to study in depth the most common used methods of ACIM control, and this has allowed me to develop a research and more theory-oriented part for this thesis. This is why this thesis is multidisciplinary, as it involves topics related to mathematical transformations, hardware design, power electronics, control systems and informatics, to name a few.

Finally, the availability of means was key to the success of the experimental part, this is, the FPGA board (Basys 3), the power module (inverter), the board designed specifically to connect the board with the power module, and the induction motor available in the laboratories of my university.

Part II

Control methods for Alternating Current Induction Motors

3 Variable-frequency Drive

Variable-frequency Drive (VFD), also known as *variable speed drive* or *inverter drive*, is a group of control techniques based on adjustable-speed drive for ACIMs. The methods related to VFD are widely used in electro-mechanical drive systems to control AC motor speed and torque by varying motor input frequency and voltage.

VFDs have many applications in industry. Not only ACIMs can be controlled with these strategies. Some of the purposes of VFD beyond motor control are the commandment of AC-AC and DC-AC drives.

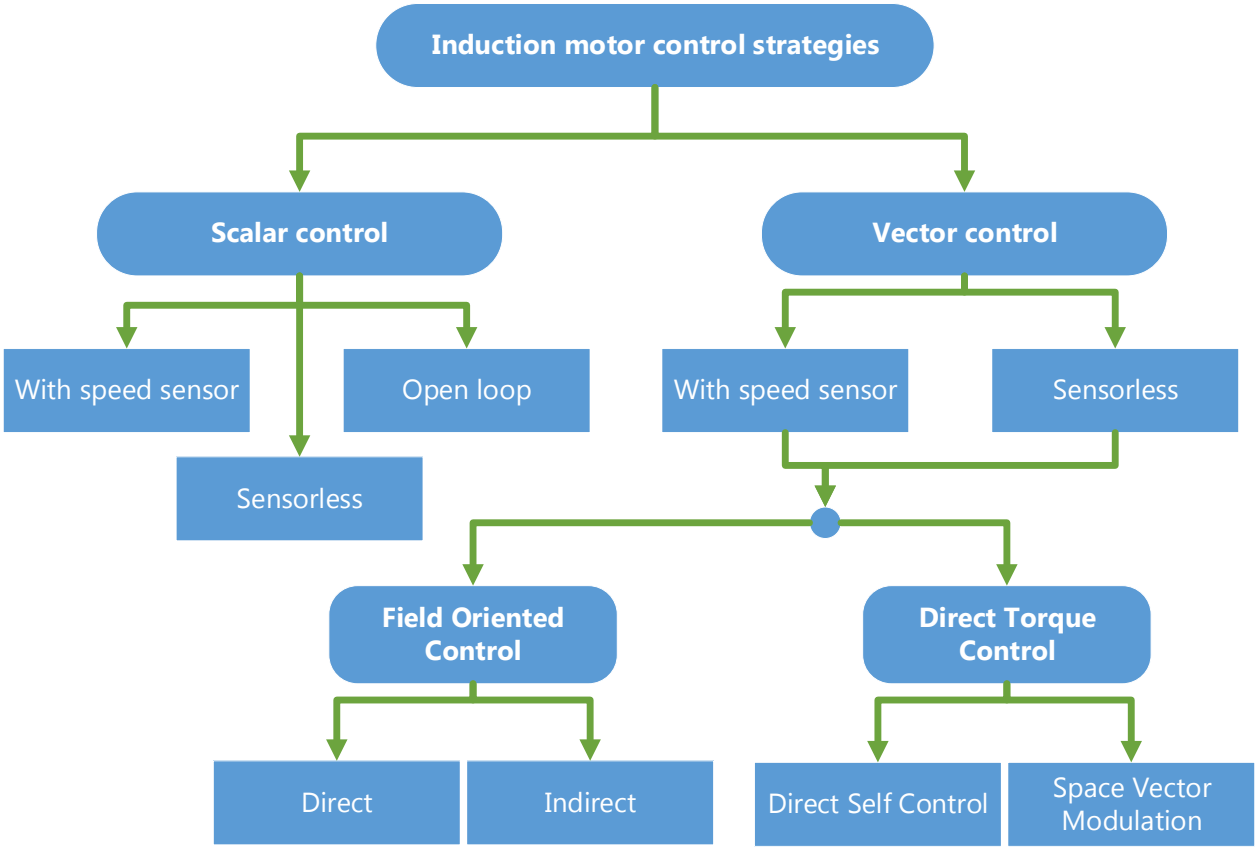


Figure 3.1 *Classification of induction motor control strategies*

This part of the project studies the three principal techniques for ACIM control: V/Hz (scalar control), and Field-oriented Control and Direct Torque Control (vector control).

3.1 V/Hz scalar control

Among the motor control techniques studied in this thesis, VFD Volt/Hertz scalar control is the most simple in concept and design, despite having some drawbacks, the most prominent of them being the impossibility to stabilize the motor at low frequencies (up to 5Hz), and a slow response in speed changes when the motor has a load.

This strategy consists in providing the switches of an inverter (⇨ Figure 3.2) with a signal corresponding to the current state of the switch (ON or OFF), thus defining the desired current orientation of the magnetic field.

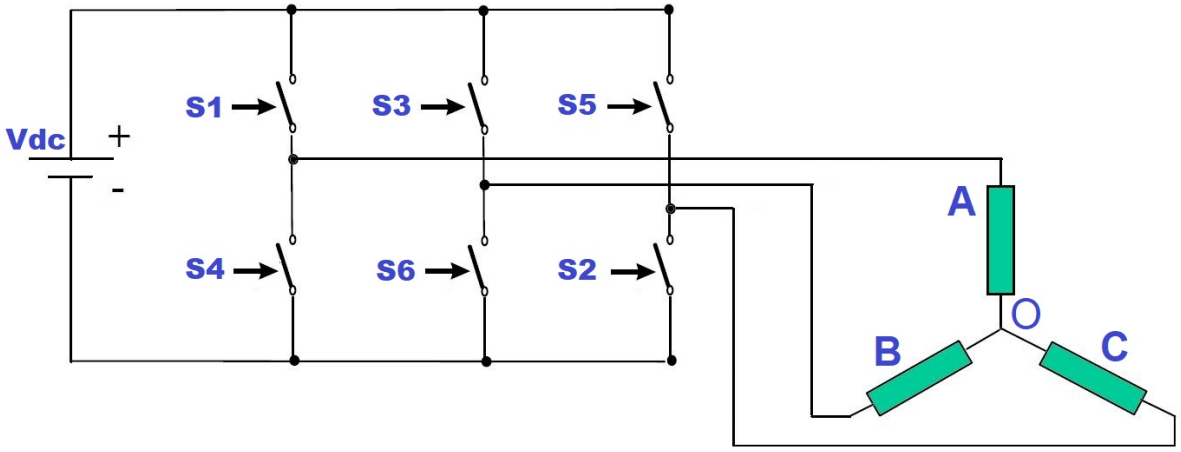


Figure 3.2 *A three-phase inverter and its control signals*

In order to obtain a sinusoidal signal at the outputs to the motor's three phases, the signal driven through the six switches of the inverter's circuit has to be controlled somehow. This can be done by generating a variable PWM signal at a specific frequency, much higher than the rotation frequency of the magnetic field. When this signal is generated, the duty cycle of the PWM wave can be adjusted at real time so that the output signal to the phases of the motor follows a sine pattern.

To produce the corresponding control signal for the Insulated-gate Bipolar Transistors (IGBTs), it has to be compared by a sine signal (in light green) stored in a ROM and a sawtooth signal (in dark blue), and according to this comparison, the duty cycle of the PWM (pink) is established at every moment, as shown on ⇨ Figure 3.3.

This comparison of signals, which generates a variable PWM output, has to be performed for each one of the three phases of the system concurrently, making sure that the sine signals are shifted 120 degrees to each other.

The torque of the motor's shaft when using a scalar control method can be controlled by directly modifying the amplitude of the output sine signals. This is why the sine signal before the comparison with the sawtooth signal is modulated by a factor,

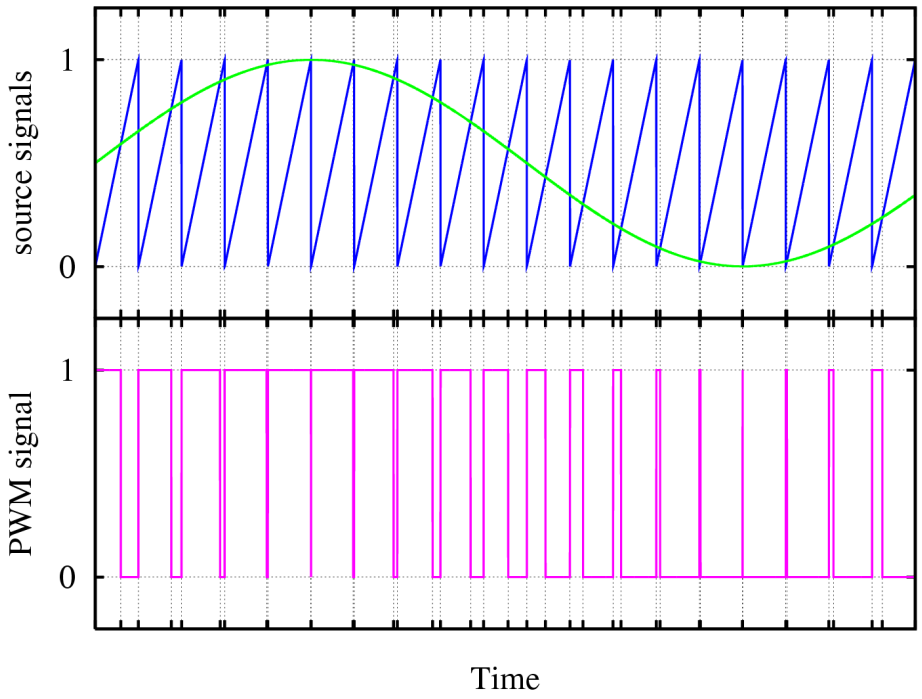


Figure 3.3 *Sine PWM wave generation*

from 0 to 100. This factor is directly proportional to the current set frequency, from 0Hz to the nominal speed of the motor (usually 3000rpm or 50Hz). For speeds greater than the nominal speed, the factor remains 100. This factor will prevent the inverter and the motor from receiving high currents at low speeds, which could be harmful for the system.

4 Field-oriented Control

Field-oriented Control (FOC), commonly referred as *vector control*, is a technique used to control three-phase Alternating Current Induction Motors based on VFD that is stator-centric. This means that all of the operations that have to be performed in order to run a FOC are made from the stator's reference frame, and they are represented by a vector.

In order to run a FOC, the signal coming from the feedback of the system, which is the magnitude of the current on two phases from the power module to the motor, has to be treated and modified by two mathematical transformations: the Clarke transformation and the Park transformation, as shown in ➪ Figure 4.1. These transformations lead to a new reference frame, named *(d, q)*, which is time-invariant.

An ACIM controlled by a FOC, after the required mathematical transformations, can be controlled in a way similar of the DC motors. This allows to control the motor in a more accurate way, whether it is in steady state or transient.

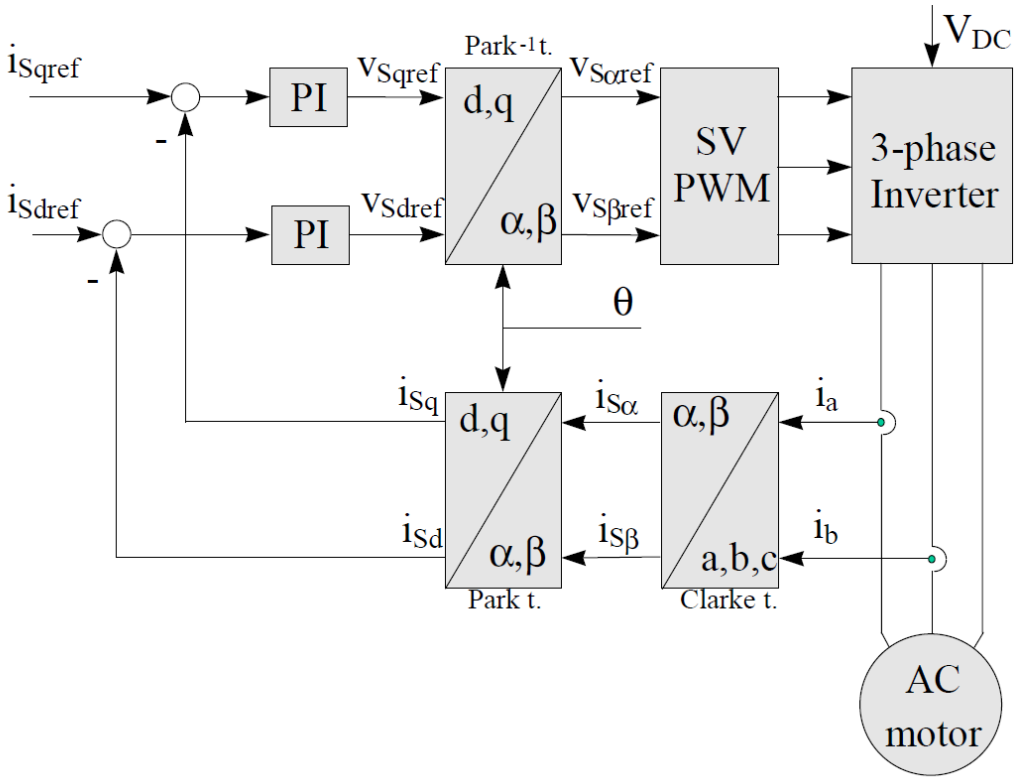


Figure 4.1 *A Field-oriented Control diagram*

4.1 Clarke transformation (the $(a, b, c) \rightarrow (\alpha, \beta)$ projection)

Clarke transformation, also known as *Alpha-beta transformation*, named after Edith Clarke[1], is a mathematical transformation that has the purpose to convert a three-phase system into a bi-dimensional, time-variant system $(a, b, c) \rightarrow (\alpha, \beta)$.

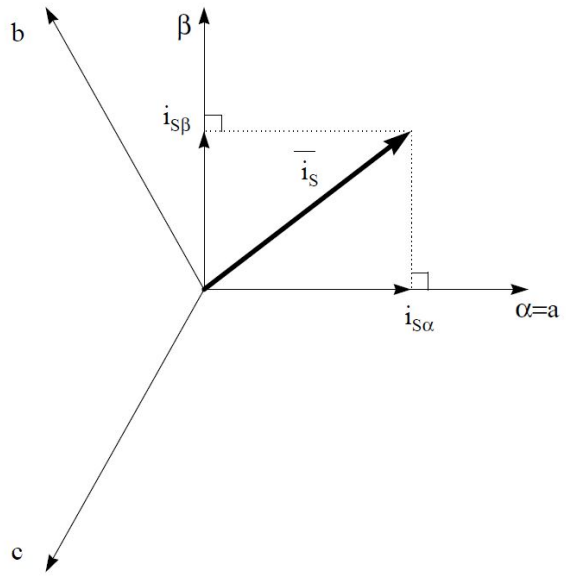


Figure 4.2 *Clarke transformation and its projections*

In a three-phase system, the signals of the three phases (a, b, c) are represented by three different axis separated $\frac{2\pi}{3} rads$ to each other. After the Clarke transformation, the projection of the vectors is represented only by two orthogonal components, (α, β). For this result, it is assumed that the axis a and the axis α are in the same direction.

$$\begin{cases} i_{S\alpha} = i_a \\ i_{S\beta} = \frac{1}{\sqrt{3}} i_a + \frac{2}{\sqrt{3}} i_b \end{cases} \tag{4.1}$$

A product of matrix is used to solve this system:

$$\begin{pmatrix} i_{S\alpha}(t) \\ i_{S\beta}(t) \end{pmatrix} = \sqrt{\frac{2}{3}} \begin{pmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{pmatrix} \begin{pmatrix} i_a(t) \\ i_b(t) \\ i_c(t) \end{pmatrix} \tag{4.2}$$

The resulting system is still time-variant.

[1]Edith Clarke (February 10, 1883 - October 29, 1959) was the first female electrical engineer and the first female professor of electrical engineering at the University of Texas at Austin.

4.2 Park transformation (the $(\alpha, \beta) \rightarrow (d, q)$ projection)

Park transformation, named after Robert H. Park[2], is considered the most critical process of the FOC. This projection transforms a two phase orthogonal system, which is made up by the output signals of the Clarke transformation, into a new rotating reference frame $(\alpha, \beta) \rightarrow (d, q)$, which is a time-invariant system. The only requirement is to consider the d axis aligned with the rotor flux.

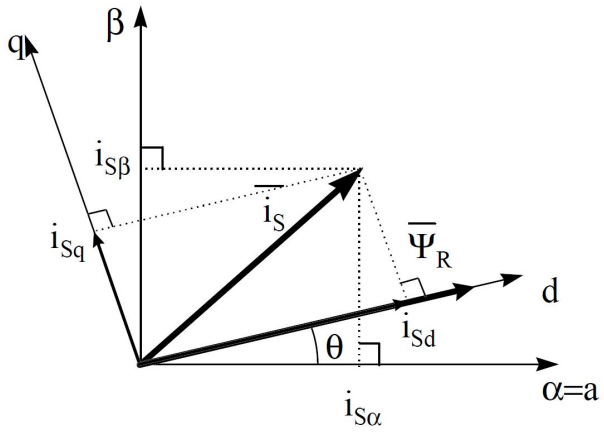


Figure 4.3 *Park transformation and its orthogonal projections*

The result of this projection is sometimes known as *Direct-quadrature-zero transformation* when referring to the product of the Clarke transformation and the Park transformation together.

The flux and torque are components of a new vector. This vector is calculated by solving the linear system of the ⇨ Equation 4.3.

$$\begin{cases} i_{Sd} = i_{S\alpha} \cos\theta + i_{S\beta} \sin\theta \\ i_{Sq} = -i_{S\alpha} \sin\theta + i_{S\beta} \cos\theta \end{cases} \tag{4.3}$$

In this equation, θ is the rotor flux position, and it has to be either measured or estimated. Knowledge of the rotor flux position is the core of the correct performance of FOC.

From this transformation on, the system is treated as if we were rotating inside of the armature or the shaft.

[2]Robert H. Park (March 15, 1902 - February 18, 1994) was an American electrical engineer and inventor.

4.3 Inverse Park transformation (the $(d, q) \rightarrow (\alpha, \beta)$ projection)

The Inverse Park transformation is the process in which the reference signals and the feedback processed signals are transformed again from the rotating reference into a two-phase orthogonal system $(d, q) \rightarrow (\alpha, \beta)$. It transforms voltage signals instead of current in order to command the IGBTs of the inverter.

The equations that define this transformation are the following:

$$\begin{cases} v_{S\alpha ref} = v_{Sdref} \cos\theta - v_{Sqref} \sin\theta \\ v_{S\beta ref} = v_{Sdref} \sin\theta + v_{Sqref} \cos\theta \end{cases} \tag{4.4}$$

The voltage vector, which is the output of the Inverse Park transform block, is then applied as the input vector to the SVPWM component.

4.4 Space Vector Pulse Width Modulation

The Space Vector Pulse Width Modulation (SVPWM) is a technique used to transform an orthogonal reference, represented by a vector, to digital pulses that directly control the inverter that provides the signal to the three phases of an ACIM.

This technique is based on PWM and is applicable to different types of AC motors, such as ACIMs, Brushless DC electric motors (BLDCs) and Permanent-magnet Synchronous Motors (PMSMs).

Some studies of SVPWM reveal that this technique utilizes DC voltage more efficiently and generates less harmonic distortion when compared to the classic Sinusoidal Pulse Width Modulation (SPWM) method. [1]

The commutation of these switches must respect the following conditions:

- ❒ three of the switches must always be ON and three always OFF, and
- ❒ the upper and the lower switches of the same branch have to be driven with two complementary signals in order to avoid short-circuits.

SVPWM can be represented by vectors that divide the plan into six sectors ⇨ Figure 4.4. Depending on the sector that the voltage reference is in, two adjacent vectors are chosen. These two adjacent vectors are time-weighted in a sample period to produce the desired output voltage to the inverter.

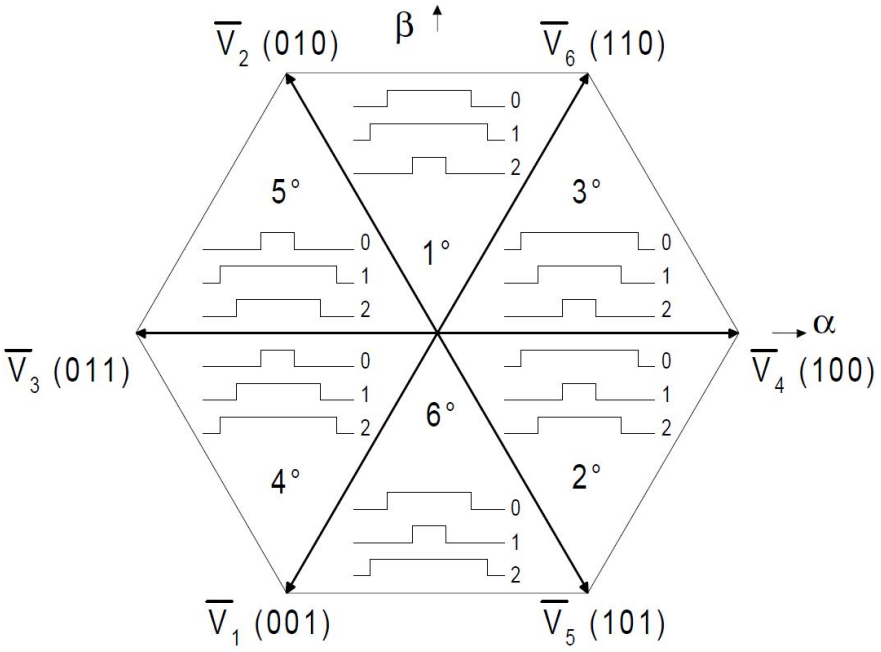


Figure 4.4 *SVPWM sectors*

As an example, if the current orthogonal input vector was located on the 3rd sector, like in ➪ Figure 4.5, then a linear system made from the adjacent vectors would have to be solved (➪ Equation 4.5).

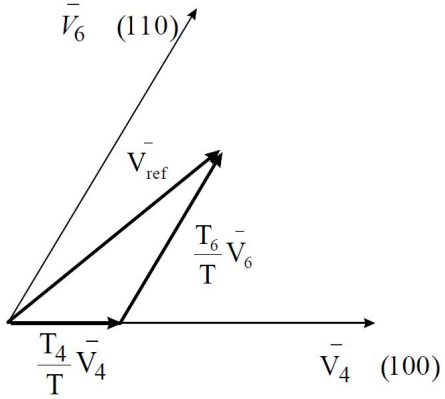


Figure 4.5 *Reference vector as a combination of adjacent vectors*

$$\begin{cases} T = T_4 + T_6 + T_0 \\ \overline{V}_{ref} = \frac{T_4}{T}\overline{V}_4 + \frac{T_6}{T}\overline{V}_6 \end{cases} \tag{4.5}$$

Where T_4 and T_6 are the times during which the vectors V_4, V_6 are applied and T_0 the time during which the zero vectors are applied.

The ➪ Figure 4.6 shows the output time-weighted signal that feeds the inverter. This would be the configuration for the third sector too.

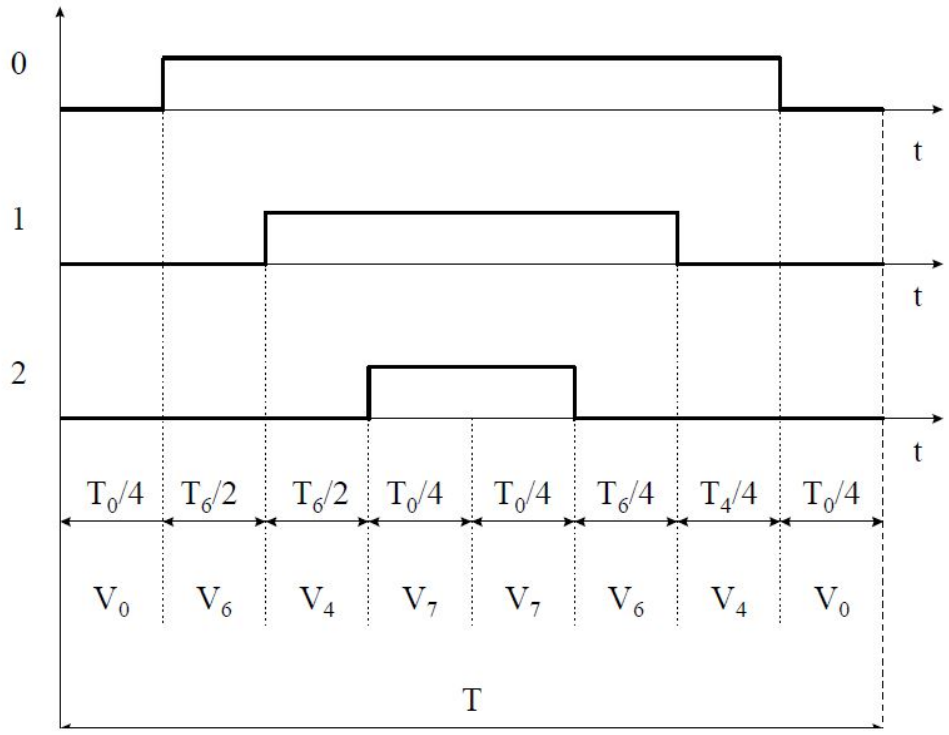


Figure 4.6 *Pattern of SVPWM in the 3rd sector*

4.4.1 SPWM vs SVPWM

Both techniques *Sinusoidal Pulse Width Modulation* and *Space Vector Pulse Width Modulation* have the same objective: feeding the transistors of the inverter to produce a sinusoidal signal than can make the motor turn. However, each system has a different method to achieve this purpose.

While SPWM generates a 3-phase frequency from a simple PWM sine-sawtooth wave comparison, SVPWM is a more sophisticated technique that provides a higher voltage to the motor with lower Total Harmonic Distortion, providing a more efficient use of the supply voltage.

In SPWM, the locus of the reference vector is inside of a circle with a radius of $1/2V_{dc}$, while in SVPWM the locus is located at $1/\sqrt{3}V_{dc}$. This difference shows that the output voltage of SVPWM is 15.47% more efficient than that of SPWM.

4.5 PI regulators

The Proportional Integral (PI) regulators in FOC have the function to regulate the two signals that are the input to the control system: the torque component reference and the flux component reference. They are dependent on the constants that have to be set in order to reach the quickest time response with the lowest overshooting

possible.

Before these two signals are regulated by the PI regulators, they are compared with the output vector of the Park transformation and an error signal is calculated. Finally, the regulated signal is the input to the Park transformation component.

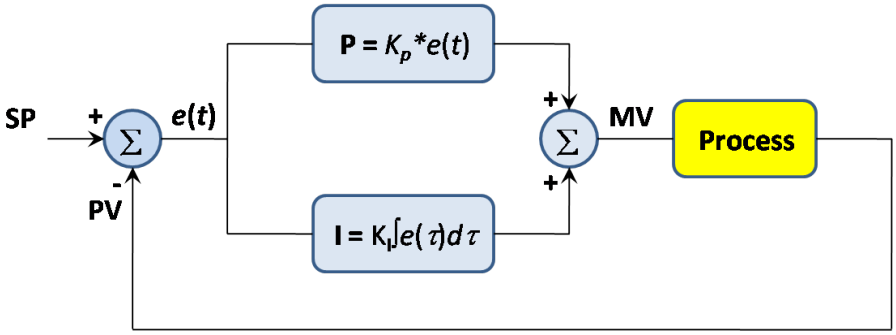


Figure 4.7 *PI control diagram*

5 Direct Torque Control

Direct Torque Control (DTC) is a more recent AC motor control method developed by ABB engineers in 1985. DTC is optimal for ACIMs and PMSMs. This technique is also based on vector VFDs.

The principles of operation of the DTC is to monitor the flux dynamics of the stator and then to directly manipulate the flux that goes through the stator in the form of a vector. This vector is proportional to the torque force at the motor's shaft. Thus, DTC directly controls torque and stator flux, and indirectly controls the stator currents and voltages.

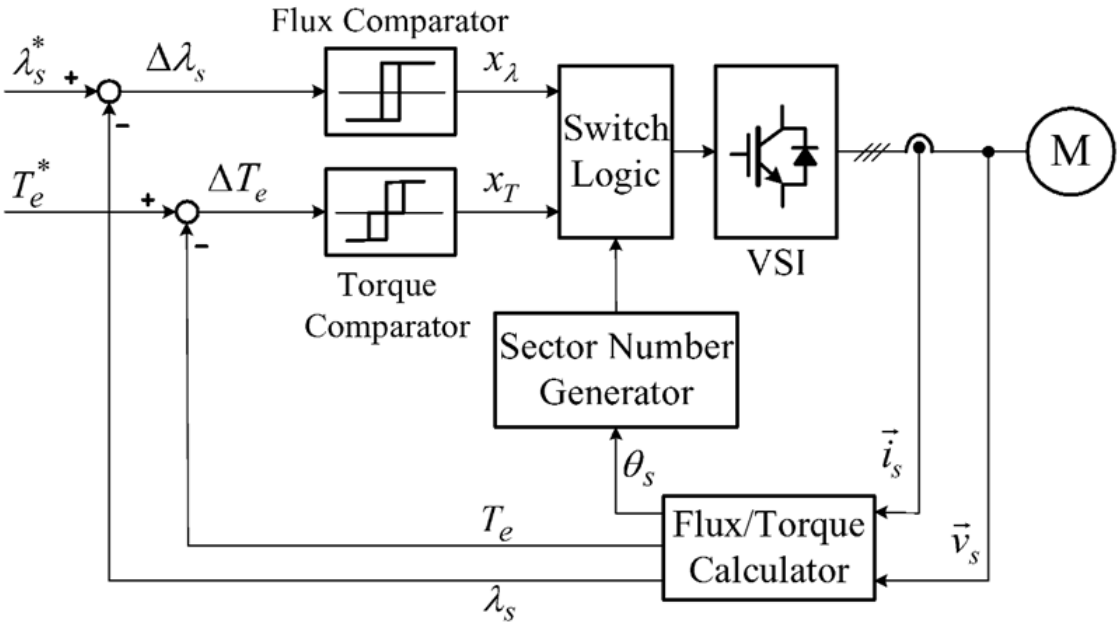


Figure 5.1 *A Direct Torque Control diagram*

In a DTC, torque and flux can be changed very fast by modifying the references. It also has high efficiency and low losses. The step response has virtually no overshoot. No coordinate transforms are needed, all calculations are done in stationary coordinate system. No separate modulator is needed (although it can be used). The hysteresis control defines the switch control signals directly. There is no need for PI current controllers. Thus no tuning of the control is required and the same control can be used in different motors.

According to ABB, the main **advantages** of DTC over other control methods are the following:

- ❒ No need for motor speed or position feedback.
- ❒ Installation of costly encoders or other feedback devices can be avoided.
- ❒ DTC control available for different types of motors, including PMSM and the newer synchronous reluctance motors.
- ❒ Accurate torque and speed control down to low speeds, as well as full startup torque down to zero speed.
- ❒ Excellent torque linearity.
- ❒ High static and dynamic speed accuracy.
- ❒ Absence of co-ordinate transform.

However, DTC has some drawbacks:

- ❒ Possible problems during starting.
- ❒ Requirement of torque and flux estimators, implying the consequent parameters identification.

5.1 DTC vs FOC

Direct Torque Control is considered to have a slight better performance than FOC for the following reasons:

Coordinates reference frame	DTC	(d, q) at stationary reference frame
	FOC	(d, q) at rotor
Controlled variables	DTC	Torque and stator flux
	FOC	Rotor flux, torque current and rotor flux current
Parameter sensitivity	DTC	Stator resistance
	FOC	(d, q) inductances and rotor resistance
Rotor speed measurement	DTC	Not required
	FOC	Required (measured or estimated)
Coordinate transformations	DTC	Not required
	FOC	Required (Clarke and Park)
Switching losses	DTC	Lowest
	FOC	Low
Processing requirements	DTC	Lower
	FOC	Higher

Table 5.1 *DTC vs FOC*

Part III

Workflow: implementing the controller and necessary hardware

6 Field Programmable Gate Arrays

Field Programmable Gate Arrays (FPGAs) are a hardware-based technology designed to performing calculations, routing digital signals, and controlling embedded systems using programmable logic.

FPGAs consist of arrays of logic blocs that are programmable. The chip that contains the FPGA is surrounded by programmable routing resources, which allow the hardware designer to make interconnections between these logic units. Some of these units have ports that connect the internal signals with physical signals.

The first FPGA was introduced by Xilinx in 1985. Nowadays, most of the commercially available FPGAs are manufactured by Xilinx and Altera (Intel), which make up 90% of FPGAs market, but recently more companies have started developing their own FPGA boards.

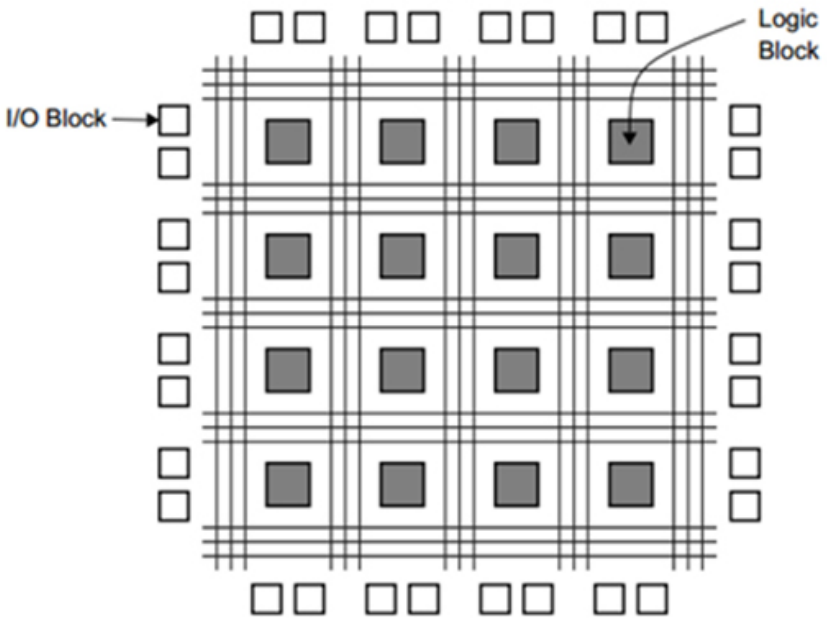


Figure 6.1 *FPGA architecture*

6.1 FPGA over a microcontroller

Microcontrollers have become a dominant component in modern electronic design. They are highly versatile and inexpensive, and can do many of the operations needed in industry, automotive or any other field. They usually are a person's first introduction to the fields related to electronics. Nonetheless, a microcontroller is built around a processor and processors come with fundamental limitations. In some cases, an FPGA can be more useful than a classical microcontroller.

By definition, a processor performs its tasks by executing instructions sequentially, this is, that the processor's operations are inherently constrained: the desired functionality must be adapted to the available instructions and, in most cases, it is not possible to accomplish multiple processing tasks simultaneously, even if the instruction set is designed to be highly versatile.

A good alternative to these limitations would be a hardware-based approach, close to the functionality of Application-specific Integrated Circuits (ASICs). This is where FPGAs have their niche: they have a time performance close to that of ASICs but without the need of having additional hardware. In the case a FPGA is used, its operation will be described in a Hardware Description Language, like VHDL or Verilog, and once the code has been synthesized, the final result will have a behavior like a real hardware component. An advantage is that this code can be rewritten and re synthesized as many times as necessary, and the modules of a top level project are run concurrently to each other, avoiding the sequential operation that microcontrollers have.

6.2 Artix-7 FPGA and Basys 3 board

Basys 3 ➪ Figure 6.2 is a FPGA development board intended for education purposes that includes a FPGA chip (Artix-7) and a complete set of peripherals ready to use together. Hardware descriptions for the Basys 3 can be written in VHDL or Verilog, and synthesized, on Vivado Design Suite. In case of the necessity of a coprocessor for a given project, a simple ALU can be instantiated on the architecture of this project to work as a processor, and a compiled C program can be run on it.

Basys 3 is the board that has been used for this motor control project given the huge amount of cells (more than the necessary for this project), the 12-bit resolution ADC converter and the ability to connect the board with the power module. Its main features are summarized in ➪ Table 6.1

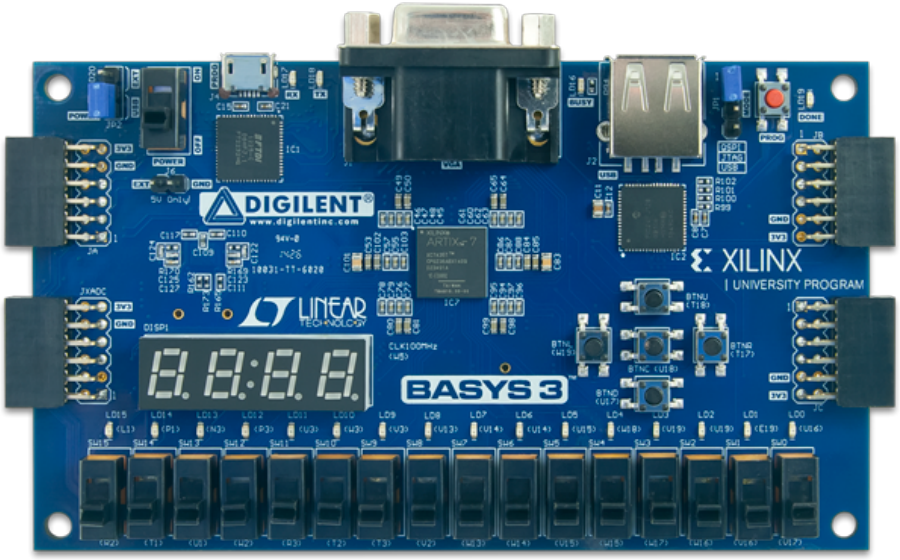


Figure 6.2 *A Xilinx's Basys 3 board*

Artix-7 FPGA Trainer Board Features	
On-chip analog-to-digital converter (XADC, 12-bit resolution)	
Key Specifications	
FPGA part number	XC7A35T-1CPG236C
Logic cells	33,280 in 5200 slices
Block RAM	1,800 Kbits
DSP slices	90
Internal clock	100MHz
Connectivity and Onboard I/O	
Pmod connectors	3
Switches	16
Buttons	5
User LED	16
7-seg displays	4-Digit
VGA	12-bit
USB	HID Host (KB/Mouse/Mass Storage)
Electrical characteristics	
Power	USB (5V in)
Logic level	3.3V output
Physical characteristics	
Width	7.1 cm
Length	12.2 cm

Table 6.1 *Basys 3's main features*

6.3 Xilinx's Vivado Design Suite

Vivado Design Suite by Xilinx is the development platform of hardware description for all types of Xilinx's boards. The whole process of hardware description can be handled in the suite, from highlighting the code, simulating or synthesizing to implementing, programming and debugging the output of the project on the boards.

Usually, a project on Vivado consists in the following steps:

Code The first and most important part of the project consists in the development of the code, in VHDL or Verilog. This is the part where the entity (input and output ports) and the architecture (behavioral model) are defined.

Block design integration Although this part is optional, it consists in integrating components or IP's (Intellectual Property modules) to the main project and to wire them in a graphical, more user-friendly way.

Simulation Simulation is a key part of the hardware description process. it allows us to test every part of the code and to monitor each one of the internal signals so that the developer can easily find mistakes or the source of an unexpected behavior of the code. Usually simulations run according to a *testbench*, which is an independent VHDL piece of code, compiled (not synthesized), that defines the state of the inputs at every moment.

RTL analysis Register-transfer Level (RTL) is a design abstraction which shows a model of a digital circuit in terms of the flow of digital signals (data) between hardware registers, and the logical operations performed on those signals. This analysis generates a sketch of the VHDL code pre-synthesis.

Synthesis Process that transforms high level constructs in human-readable code, which don't have real physical hardware that can be wired up to do your logic, into low level logical constructs which can be literally modeled in the form of transistor logic or look-up tables or other FPGA or ASIC hardware components.

Implementation Vivado implementation includes all steps necessary to place and route the netlist onto device resources, within the logical, physical, and timing constraints of the design.

Programming and debugging Last part of the process. A bitstream file is generated and programmed into the FPGA.

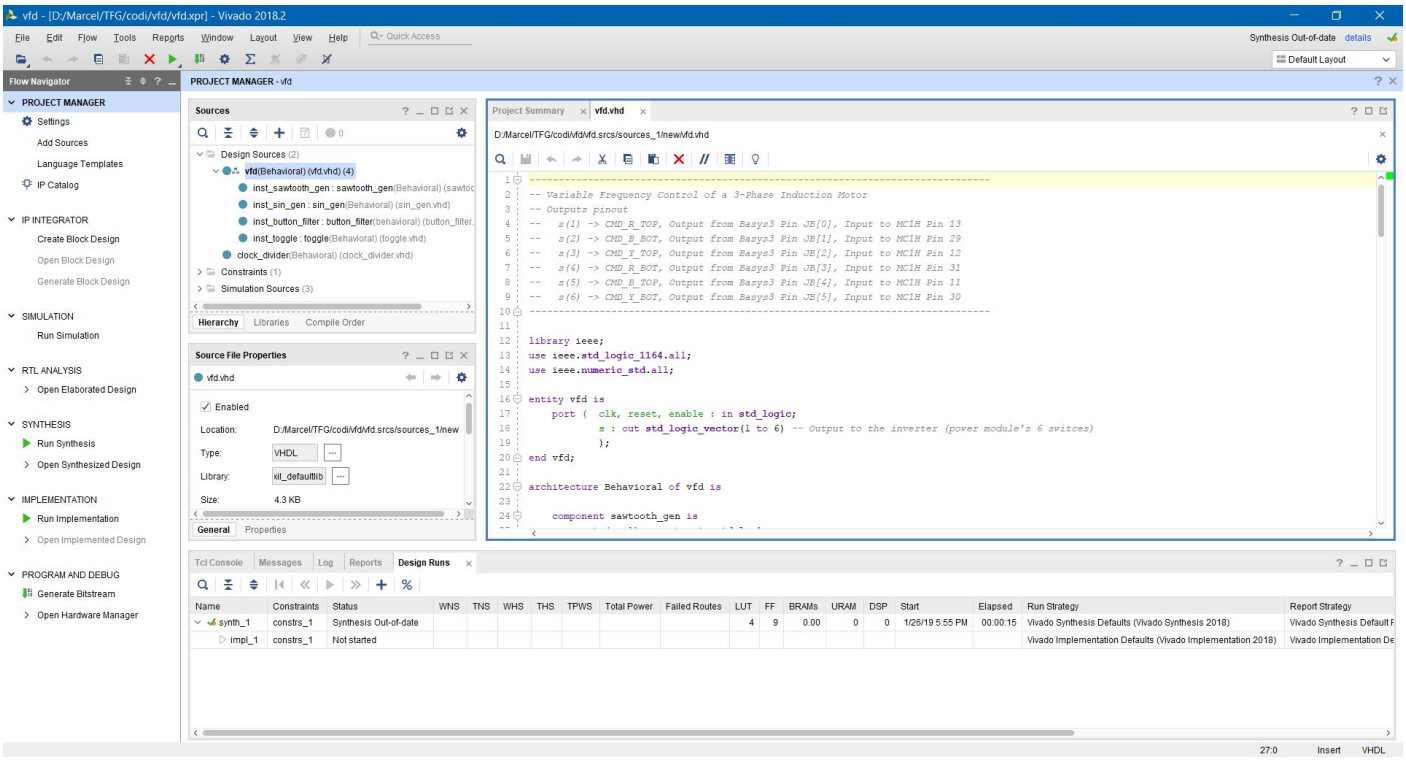


Figure 6.3 *Vivado Design Suite user interface*

6.4 The VHDL language

VHSIC Hardware Description Language (VHDL) is a hardware description language used in digital electronics design. It first appeared in 1980 and is the IEEE's officially supported language for hardware prototyping, hence becoming the standard.

VHDL is a strong-typed language based on the Ada programming language. The biggest advantage of VHDL over classic programming languages is that, when used for systems design, it permits the behavior of the designed system to be described and verified before synthesis tools translate the design into real hardware.

Another benefit of VHDL is that it can handle parallelism, or concurrency. This is useful, if not essential, in many applications that can be programmed or described.

A VHDL description always consists in the following two main modules:

Entity This is the block as seen from the outside of the module. The entity describes the input and output signals, the type of signals, the size of these signals and, as an optional statement, can include generic definitions.

Architecture Definition of the behavior of the internal and external signals of the hardware description. Other components can be instantiated in the architecture. Processes and concurrent statements can be described in this section.

7 Inverter and motor

7.1 Microchip's Power Module

Inverting a signal is the process in which a digital signal is transfered to the physical switches of an inverting circuit in order to amplify the power of that signal (both the voltage and current). The output signal is directly connected to the three phases of the motor. In order to obtain a three-phase inverted wave, the six switches of the inverter's internal circuit have to be controlled following a switching sequence determined by the controller.

According to the datasheet of the Microchip's Power Module used in this project ⇨ Figure 7.3 (3-Phase High Voltage Power Module MC1H), six units of 600V N-Channel IGBT transistors with co-packaged anti-parallel 600V diodes are used as solid-state switches to control the signal through the phases.

Inverting the DC signal is a critical part of the motor control process because it is the transition from a digital power signal (low power) to a high voltage signal (high power). This fact requires the designer to take special precautions in order to avoid electrical hazards.

A concept to take into consideration when designing a motor control is the **dead time**. The dead time is defined as a small interval of time during which both the upper and lower switches in a phase-leg are off. This prevents an hypothetical delay in the switching of a IGBT to result in a harmful short-circuit, as shown in ⇨ Figure 7.2.

The manufacturer specifies that the dead time has to be implemented by the designer of the control system (⇨ Figure 7.1). They also recommend the dead time to be at least $2\mu s$. The highest the dead time is, the worse the Total Harmonic Distortion (THD) will be. For this project, a dead time of $3\mu s$ has been implemented.

Note: No hardware Dead Time is included in the design as it is included as a feature of the Motor Control PWM Module of the dsPIC device. A minimum Dead Time of 2 µs should be used. This applies to both turn on and turn off of both devices.

Figure 7.1 *Dead time requirements*

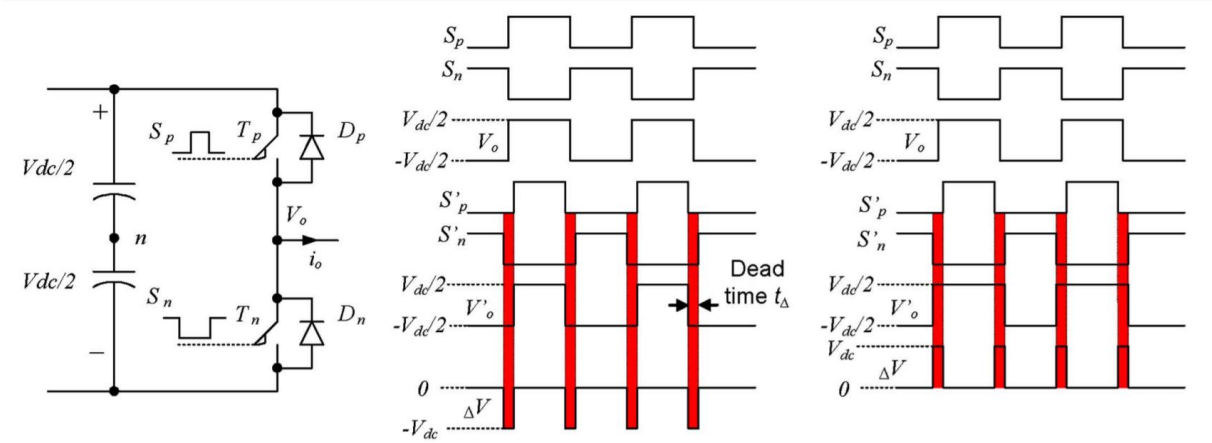


Figure 7.2 *Dead time in a single-phase inverter*

Figure 7.3 *3-Phase High Voltage Power Module MC1H Inverter*

7.2 ABB's three-phase induction motor

Alternating Current Induction Motors (ACIMs), or asynchronous motor, are AC electric motor in which the electric current in the rotor needed to produce torque is obtained by electromagnetic induction from the magnetic field of the stator winding. An induction motor can therefore be made without physical electrical connections to the rotor. Usually, ACIMs are squirrel-cage type, like the one used in this project.

ACIMs are considered asynchronous motors because they always run at a speed lower than the synchronous speed. This difference is measured by the slip.

The rotational speed of the rotating magnetic field is called as synchronous speed, and is defined by the ➪ Equation 7.1.

$$\mathrm{Ns} = \frac{120 \times \mathrm{f}}{\mathrm{P}} \quad (\mathrm{RPM}) \tag{7.1}$$

where f is the frequency of the supply and P is the number of poles.

The difference between the synchronous speed and actual speed of the rotor is called slip and is defined by ➪ Equation 7.2.

$$\mathrm{s} = \frac{\mathrm{Ns} - \mathrm{N}}{\mathrm{Ns}} \times 100 \quad (\mathrm{o/o}) \tag{7.2}$$

where N is the actual speed of the rotor

Three-phase ACIMs have some advantaged over other motors, that can be categorized as follows:

- ❒ They have very simple and rugged (almost unbreakable) construction
- ❒ Reliability and having low cost
- ❒ High efficiency and usually a good power factor
- ❒ Minimum maintenance is required
- ❒ Three-phase induction motor is self starting, so extra starting motor or other special starting arrangements are not necessary

Figure 7.4 *ABB's three-phase motor*

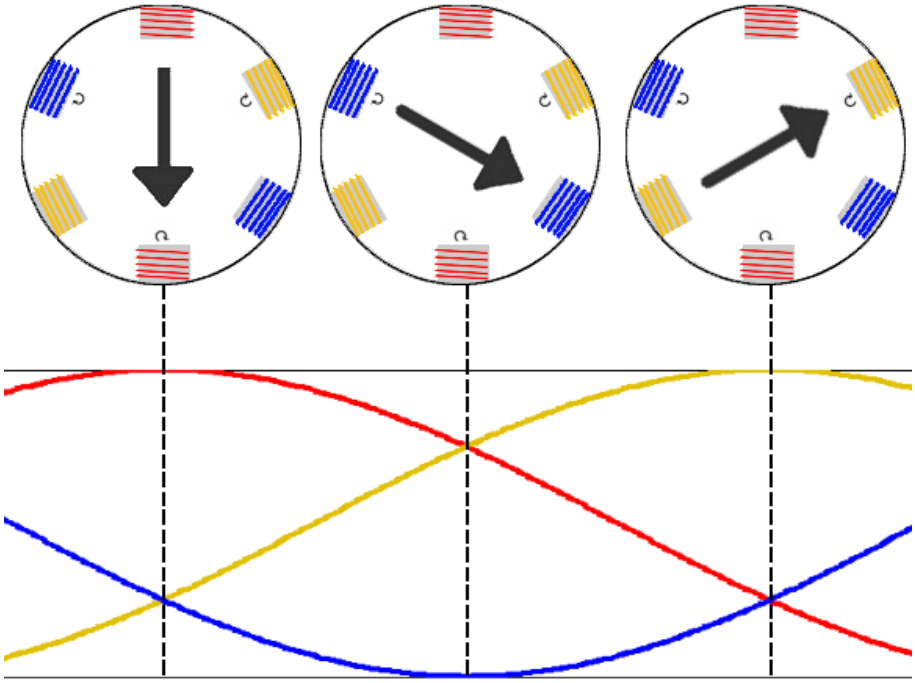


Figure 7.5 *3-phase and corresponding magnetic field orientation*

8 Implementation and Verification of a scalar VFD on an FPGA with VHDL

The following chapters comprise the most important elements (components, processes, functions, ...) of the hardware description of the scalar control method, as well as the testbench used to verify the operation of the technique ad to check that the output signals to the inverter will not create any electrical hazard.

The libraries used in the VFD control are the following:

Listing 8.1: Used libraries in the project

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all; -- Library for sin_gen process
```

All of them are official VHDL libraries, supported by IEEE and reliable after the synthesis has been done.

8.1 Entity

The entity of a project defines the physical ports that will be used to communicate to and from the FPGA. A generic value is defined, *max_freq*, which will define the top value of the frequency reached by the control system. It is recommended to set this value to the nominal speed of the motor, 50Hz in this case. This value can be modified previous to the synthesis.

At the following listing: *clk* comes from the internal oscillator of the Basys 3 board; *reset* is an input signal to the FPGA from a button thatwill reset all of the components, if necessary; *up* and *down* are buttons located on the board that will increase or decrease the frequency one unit; *reset_out* drives the *reset* signal to the power module and will reactivate it after an error has occurred; *pwm_wave_phi_out* is a control signal that shows the inverted wave of all of the three phases; *s* are the output signals to the six IGBTs of the power inverter; *enable_led* will power on the LEDs of the board to warn us when the motor should be running; *cat* and *an* will be routed to

the 7-segment displays so that they show the current set frequency.

Listing 8.2: Entity declaration of the top-level file

```
entity vfd is
    generic (   max_freq : integer := 50 -- 3000rpm, nominal
                );
    port (  clk, -- W5 (internal)
            reset,
            down, -- btnD
            up : in std_logic; -- btnU
            reset_out, pwm_wave_ph1_out, pwm_wave_ph2_out,
               pwm_wave_ph3_out, sync : out std_logic;
            s : out std_logic_vector (1 to 6); -- Output to the inverter
                 (power module's 6 switces)
            enable_led : out std_logic_vector (15 downto 0); -- Enable
               LED indicator
            cat : out std_logic_vector (7 downto 0); -- To 7-seg BCDs
            an : out std_logic_vector (3 downto 0)
            );
end vfd;
```

8.2 Architecture

The process described in ➪ Listing 8.3 generates a sawtooth signal that is compared real time with the sine wave of each one of the three phases. The amplitude of the sawtooth signal is 2^{16} and spans from -2^{15} to 2^{15}. The period of the sawtooth wave is 10us, much higher than the top frequency of the sine waves (50Hz).

Listing 8.3: Sawtooth signal generation

```
proc_sawtooth_gen : process (clk) begin
    if rising_edge(clk) then
        if reset = '1' then
            sawtooth_wave_int <= -2**15;
        elsif (enable = '1') then
            if sawtooth_wave_int > 2**15-1 then --32767
                sawtooth_wave_int <= -2**15;
            elsif (sin_wave_clk_div = '1') then
                sawtooth_wave_int <= sawtooth_wave_int + 2048;--
                   64;--2048;
            end if;
        end if;
    end if;
end process;
sawtooth_wave <= std_logic_vector(to_unsigned(sawtooth_wave_int, 16));
```

The process described in ➪ Listing 8.4 manages the address map that will request a sine value to the instances of the ROM component so that a sine signal can be drawn. This process is considered critical because it defines both the phase shift angle and the frequency of the generated sine signals. The signal *sin_wave_clk_div* is the output of a variable clock divider component that varies its dividing factor according to the value

of the current frequency, thus defining the final frequency of the output sine waves.

Listing 8.4: Three-phase sine signal generation

```
proc_sin_gen : process (clk) begin
    if rising_edge(clk) then
        if (reset = '1') or (enable_pulse = '1') then
            address_ph1 <= (others => '0'); -- (0 degrees)
            address_ph2 <= "1010101010"; --682 (240 degrees)
            address_ph3 <= "0101010101"; --341 (120 degrees)
        elsif (enable = '1') and (sin_wave_clk_div = '1') then
            address_ph1 <= address_ph1 + 1;
            address_ph2 <= address_ph2 + 1;
            address_ph3 <= address_ph3 + 1;
        end if;
    end if;
end process;
```

The listing below allows the user to change the frequency of the motor using the *up* and *down* buttons. It also prevents the user from exceeding the top value of the frequency, set at the entity by the generic constant *max_freq*.

Listing 8.5: Frequency set using the physical buttons

```
proc_set_freq_up_down_counter : process (clk) begin
    if rising_edge(clk) then
        if (reset = '1') then
            current_freq <= 0;
        elsif (up_pulse = '1') and (current_freq < max_freq) then
            current_freq <= current_freq + 1;
        elsif (down_pulse = '1') and (current_freq > 0) then
            current_freq <= current_freq - 1;
        end if;
    end if;
end process;
```

➪ Listing 8.6 is a concurrent assignment to the binary signals *pwm_wave_i*. It is a comparison between the sine and sawtooth signals, and the result is a Sine-PWM signal.

Listing 8.6: Three-phase PWM signals generation

```
pwm_wave_ph1 <= '1' when (to_integer(signed(sawtooth_wave)) < to_integer
    (signed(sin_wave_ph1))) and (enable = '1') else '0';
pwm_wave_ph2 <= '1' when (to_integer(signed(sawtooth_wave)) < to_integer
    (signed(sin_wave_ph2))) and (enable = '1') else '0';
pwm_wave_ph3 <= '1' when (to_integer(signed(sawtooth_wave)) < to_integer
    (signed(sin_wave_ph3))) and (enable = '1') else '0';
```

The following component takes *pwm_wave_phi* as its input and generates the value of the IGBTs for one leg of the inverter (or one phase). This component also generates a delay in the switching of the IGBTs to create a dead time of 2us.

Listing 8.7: Phase signals generation

```
inst_igbt_signals_leg1 : phase_gen
    port map (  clk => clk,
                reset => reset,
                enable => enable,
                pwm_in => pwm_wave_ph1,
                pwm_h => s(1),
                pwm_l => s(4)
                );
```

8.3 Components and functions

8.3.1 16-bit, 1024 address ROM containing sine values

A sine wave has to be generated to be compared with the sawtooth wave. There are many ways to do so. One option is to define a ROM memory containing all the values of a sine wave, defining the amplitude of the output signal. To make a request, an address has to be provided as an input vector.

In order to make the amplitude of the sine wave variable (as explained in ➪ Section 3.1), a *factor* is given. This factor ranges from 0 to 100 and will be taken as a percentage og the maximum amplitude. The output signal is calculated as shown in ➪ Listing 8.8. Part of the hardware description of the ROM memory is found below.

Listing 8.8: Entity of 16-bit 1024 address ROM memory containing the sine values

```
entity sin_rom is
    port (  clk : in std_logic;
            factor : in integer range 0 to 100;
            address : in std_logic_vector (9 downto 0);
            data_out : out std_logic_vector (15 downto 0)
            );
end sin_rom;
```

Listing 8.9: Architecture of 16-bit 1024 address ROM memory

```
process (clk) begin
    if(rising_edge(clk)) then
        data_out <= std_logic_vector(to_unsigned(factor*rom(to_integer(
           unsigned(address)))/100, 16));
    end if;
end process;
```

After the synthesis is performed, the ROM memory described above becomes a hardware component as shown in ➪ Figure 8.1.

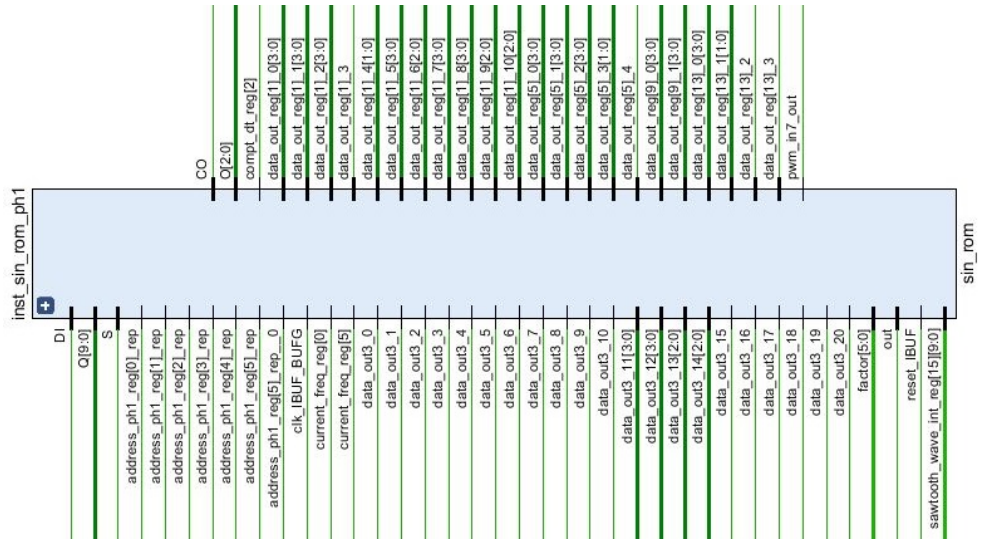


Figure 8.1 *ROM post-synthesis*

8.3.2 Variable clock divider

A clock divider is a component that provides a pulse the size of a clock cycle every time a counter reaches a specified value (*eoc*). In this project, this value has to be modified in real time when the FPGA is running the hardware description. The reason is that this allows us to change the frequency of the generated sine waves as shown in ➪ Listing 8.4.

Listing 8.10: Entity of a variable clock divider

```
entity clock_divider is
    port (  clk : in std_logic;
            reset : in std_logic;
            eoc : in integer;
            clk_div : out std_logic
            );
end clock_divider;
```

Listing 8.11: Architecture of a variable clock divider

```
process (clk) begin
    if rising_edge(clk) then
        clk_div <= '0';
        if reset = '1' then
            counter <= 0;
        elsif counter > eoc then
            counter <= 0;
            clk_div <= '1';
        else counter <= counter + 1;
        end if;
    end if;
end process;
```

8.3.3 Computation of 'end of counting' value

The following function calculates the end of counting (*eoc*) value, used as an input to the variable clock divider component that provides a signal *sin_wave_clk_div* that defines the frequency of the sine signals generation in the process labeled *proc_sin_gen*. This function is a linearization of the current frequency using a factor of 98000.

Listing 8.12: Function gen eoc

```
function get_eoc (current_freq : integer) return integer is
begin
    if (current_freq > 0) then
        return 100*980/current_freq;
    end if;
    return 100*980;
end get_eoc;
```

8.4 Verification of the VFD scalar control

Verification of VHDL hardware descriptions is done by writing a *Testbench*, which is a program also written in VHDL, that automatizes a bunch of conditions and signal assignments that could simulate a real-life performance of the synthesized model.

8.4.1 Testbench of the scalar control

The Testbench described in ➪ Listing 8.13 will run simulating a user pushing 60 times the *up* button. If the description is well written, then the counter should stop at *max_freq*.

Listing 8.13: TestBench: increasing the frequency

```
stimulus : process begin
    wait for 10ms;
    for i in 1 to 60 loop
        up_tb <= '1'; wait for 10ns;
        up_tb <= '0'; wait for 5ms;
    end loop;
    wait;
end process;
```

The statement described in ➪ Listing 8.14 will assert at every moment that the boolean condition is true. If at any moment this condition becomes false, then the console will report a message *"SHORT-CIRCUIT"* and the simulaiton will stop because the severity level is set to *failure* (the worst of the cases).

The boolean condition to assert is that there is not a single moment when the upper and the lower IGBTs of a same leg are driving current, a condition that would

lead to a short circuit.

8.4.2 Assertion of the dead time and other short circuit preventions

Listing 8.14: Short circuit assertion statement

```
assert_short_circuit :
    assert (    not((s_tb(1) = '1' and s_tb(4) = '1') or
                    (s_tb(3) = '1' and s_tb(6) = '1') or
                    (s_tb(5) = '1' and s_tb(2) = '1')
                   )
                )
    report "SHORT-CIRCUIT"
    severity failure;
```

8.5 Simulation and testing

Simulation of a VHDL description is a method to monitor internal and external signals and to check the correct behavior of the system. For the scalar VFD, the simulation is stopped at 300ms, when the frequency has reached 50Hz. From ➪ Figure 8.2, some signals are shown:

- ❒ *current_freq* is the frequency of the magnetic field rotation
- ❒ *sawtooth_wave* is the signal that is compared to the sine waves
- ❒ *sine_wave_phi* is generated for each phase and is compared with the sawtooth_wave
- ❒ *s[1:6]* is the output to the IGBTs of the power inverter
- ❒ *proof* is a proof that some current is driving through at least two phases of the motor

When the system is tested on real hardware, then the external signals of the FPGA can be monitored with an oscilloscope. At ➪ Figure 8.3, two signals are shown when the motor is stable at 50Hz: the higher amplitude one shows the period of the sine signal (used as a synchronization reference), and the chopped wave shows the behavior of an inverted sine after the signal has been converted from Direct Current (DC) to AC.

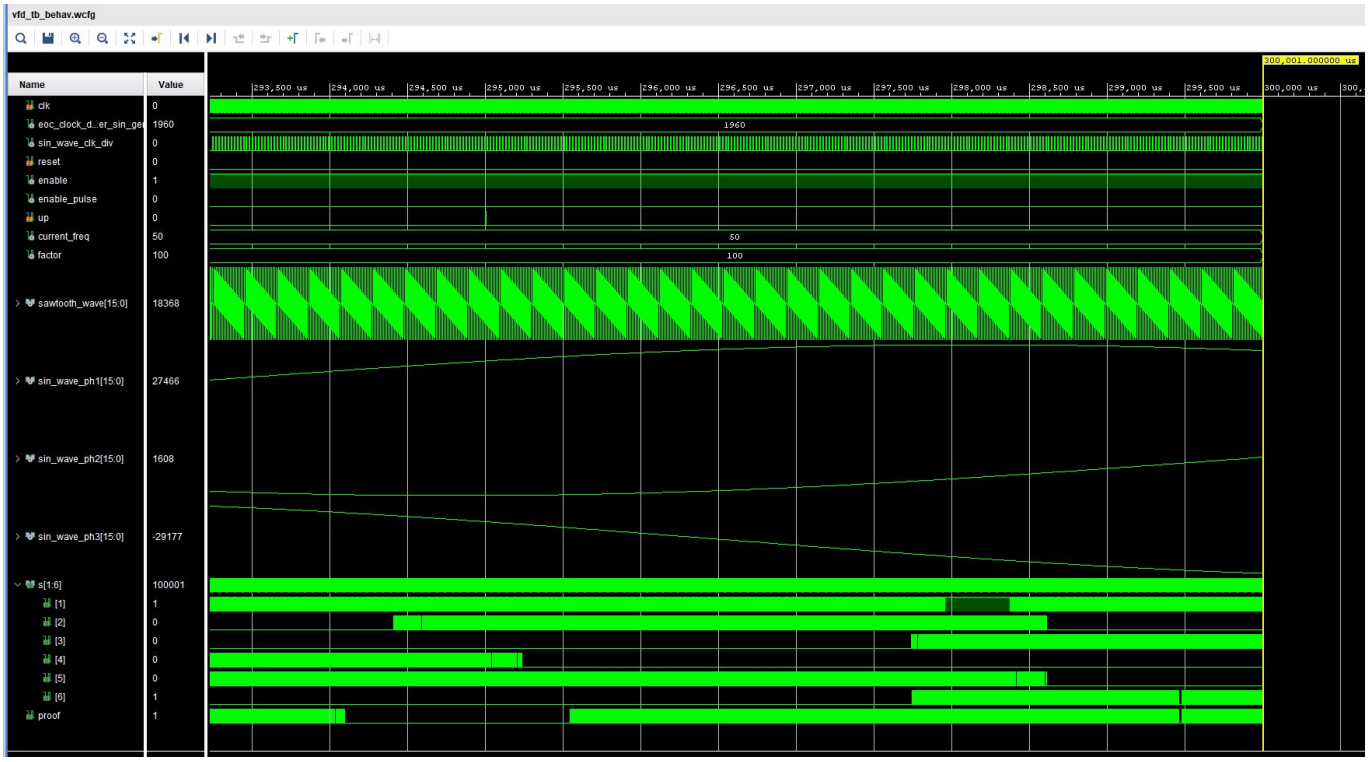


Figure 8.2 *Simulation of the scalar control at 50Hz*

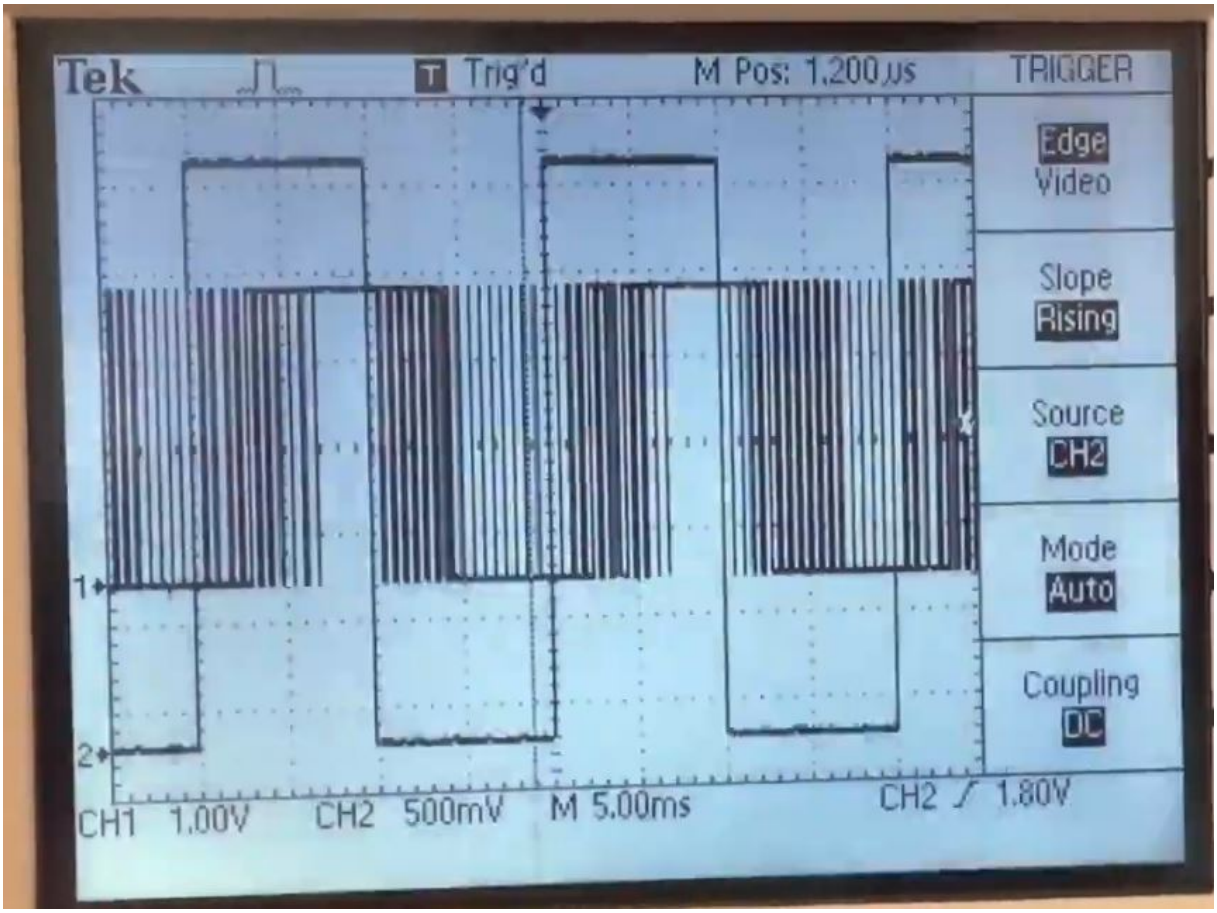


Figure 8.3 *Phase R signals of the scalar control at 50Hz*

8.6 FPGA usage

This is an analysis of the impact that the synthesized design has to the FPGA chip and its power consumption.

8.6.1 Utilization

The synthesized design is transformed to real hardware elements, like look-up tables, flip flops or digital signal processors, as shown in ➪ Table 8.1.

Resource	Utilization	Available	Utilization [%]
LUT	2176	20800	10.46
FF	348	41600	0.84
DSP	3	90	3.33
IO	41	106	38.68
BUFG	1	32	3.13

Table 8.1 *Resources utilization*

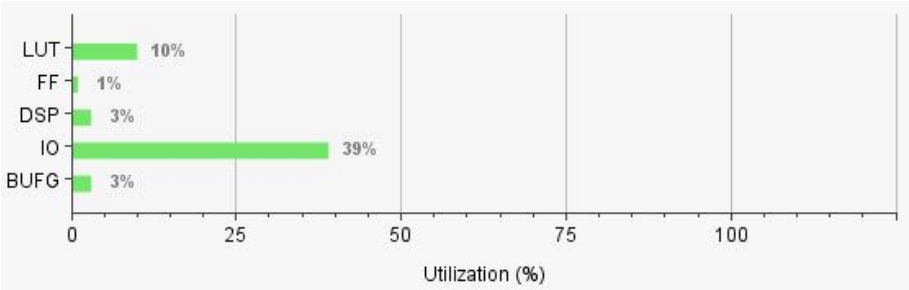
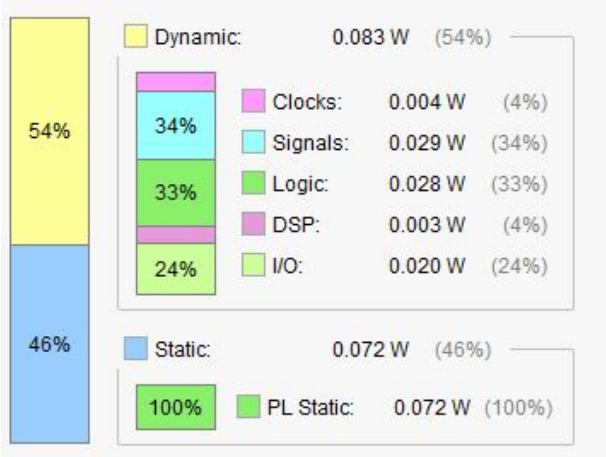


Figure 8.4 *FPGA usage*

8.6.2 Power

The total power required by the hardware elements generated during the synthesis is **0.155W**, broken down as shown in the figure below.



9 Simulation and Verification of a vector FOC on an FPGA with VHDL

This chapter contains the development process of the Field-oriented Control of a three-phase motor with VHDL. The aim of this part of the project is to use, at simulation level, the capabilities of the VHDL language in order to simulate and verify the system, as well as its signals and its behavior. The code written below is not synthesizable due to the use of non supported libraries and the use of a new type intended only for simulation purposes: the *reals*.

9.1 Clarke transformation

Clarke transformation, as studied in ⇨ Section 4.1, is the projection of the three phases of a system into a two-phase based new time-variant reference frame: *(alpha, beta)*.

According to ⇨ Equation 4.1, the new reference frame can be obtained through direct projections of the 3-phase vector, and can be described in hardware as shown in ⇨ Listing 9.1. The signal *beta* has to be handled as real values until just before the concurrent assignation, when it is converted into an integer value, so to avoid losing resolution.

Listing 9.1: Clarke transformation description in VHDL

```
entity clarke is
    port (  a, b: in integer;
            alpha, beta : out integer
            );
end clarke;

architecture behavioral of clarke is begin

alpha <= a;
beta <= integer(0.5774*real(a) + 1.1547*real(b));

end behavioral;
```

9.2 Park transformation

Park transformation, as described in ➪ Section 4.2, is the projection of *(alpha, beta)* into a time-invariant *(d, q)* frame, which is rotating at the same speed as the magnetic field *theta*.

The new reference frame *(d, q)* is generated from ➪ Equation 4.3. A new component, *Trigonometry*, is defined in order to calculate the *sin* and *cos* values of a given *theta*.

Listing 9.2: Park transformation description in VHDL

```
entity park is
    port (  clk, reset : in std_logic;
            alpha, beta : in integer;
            theta : in integer;
            d, q : out integer
            );
end park;

architecture behavioral of park is

    component trigonometry is
        port (  clk, reset : in std_logic;
                address : in integer range 0 to 359;
                sin, cos : out real range -1.0000 to 1.0000
                );
    end component;

    signal sin, cos : real range -1.0000 to 1.0000;

begin

inst_trigonometry : trigonometry
    port map (  clk => clk,
                reset => reset,
                address  => theta,
                sin => sin,
                cos => cos
                );

d <= integer(real(alpha)*cos + real(beta)*sin);
q <= integer(-real(alpha)*sin + real(beta)*cos);

end behavioral;
```

9.2.1 Component: Trigonometry

The component *Trigonometry* is used as a Look-up Table to generate the *sin* and *cos* values of a discrete *theta* value, which spans from 0 to 359 degrees. To generate the value of the cos, an internal theta *(theta_90)* is created to access the memory 90 positions in advance from the sin signal.

Listing 9.3: Trigonometry component description in VHDL

```
entity trigonometry is
    port (  clk, reset : in std_logic;
            address : in integer range 0 to 359;
            sin, cos : out real range -1.0000 to 1.0000
            );
end trigonometry;

architecture behavioral of trigonometry is

    type rom_type is array (0 to 359) of real range -1.0000 to 1.0000;
    constant rom : rom_type := (
    0.0000  ,
    0.0175    ,
    0.0349    ,
    0.0523    ,
    0.0698    ,
    -- . . .
    -0.0349    ,
    -0.0175
    );

    signal address_90 : integer range 0 to 359 := 89;

begin

process (clk) begin
    if(rising_edge(clk)) then
        sin <= rom(address);
        cos <= rom(address_90);
    end if;
end process;

process (clk) begin
    if(rising_edge(clk)) then
        if (address >= 269) then
            address_90 <= address - 269;
        else
            address_90 <= address + 90;
        end if;
    end if;
end process;

end behavioral;
```

From now on, the output time-invariant reference frame *(d, q)*, which are the magnetic flux and the torque signals, can be used to analyze the three-phase AC system like it was a DC system. These signals can be easily compared with a reference and be sent to the Inverse Park module, to further feed the IGBTs of the inverter.

9.3 Inverse Park transformation

Inverse Park transformation, as described in ➪ Section 4.3, is the reverse transform of the Direct-zero quadrature. It works the same way as the Park transformation, but inverting the signals and assigning them to the *(alpha, beta)* reference frame, according to ➪ Equation 4.4.

Listing 9.4: Inverse Park transformation description in VHDL

```
entity invpark is
    port (  clk, reset : in std_logic;
            d, q : in integer;
            theta : in integer;
            alpha, beta : out integer
            );
end invpark;

architecture behavioral of invpark is

    component trigonometry is
        port (  clk, reset : in std_logic;
                address : in integer range 0 to 359;
                sin, cos : out real range -1.0000 to 1.0000
                );
    end component;

    signal sin, cos : real range -1.0000 to 1.0000;

begin

inst_trigonometry : trigonometry
    port map (  clk => clk,
                reset => reset,
                address  => theta,
                sin => sin,
                cos => cos
                );

alpha <= integer(real(d)*cos - real(q)*sin);
beta <= integer(real(d)*sin + real(q)*cos);

end behavioral;
```

9.4 Space Vector Pulse Width Modulation

Space Vector Pulse Width Modulation, as described in ➪ Section 4.4, is the component that feeds the three-phase inverter *(s)* with the *(alpha, beta)* signals coming from the Inverse Park transformation (➪ Listings 9.5, 9.6 and 9.7).

Listing 9.5: SVPWM description in VHDL (Entity)

```
entity svpwm is
    port (  clk, reset : in std_logic;
            v_alpha, v_beta: in integer;
            s : out std_logic_vector(1 to 6)
            );
end svpwm;
```

Listing 9.6: SVPWM description in VHDL (Architecture: state machine)

```
output_decode : process (state) begin
    case state is
        when s0 =>
            pwm_h <= "000";
            pwm_l <= "000";
        when s1 =>
            pwm_h <= "010";
            pwm_l <= "110";
        when s2 =>
            pwm_h <= "100";
            pwm_l <= "101";
        when s3 =>
            pwm_h <= "100";
            pwm_l <= "110";
        when s4 =>
            pwm_h <= "001";
            pwm_l <= "011";
        when s5 =>
            pwm_h <= "010";
            pwm_l <= "011";
        when s6 =>
            pwm_h <= "001";
            pwm_l <= "101";
        when s7 =>
            pwm_h <= "111";
            pwm_l <= "111";
    end case;
end process;

next_state_decode : with sector select
    next_state <=
        s0 when 0,
        s1 when 1,
        s2 when 2,
        s3 when 3,
        s4 when 4,
        s5 when 5,
        s6 when 6,
        s7 when 7;

state_decode : process (clk) begin
    if rising_edge(clk) then
        if reset = '1' then
            state <= s0;
        else
            state <= next_state;
        end if;
    end if;
end process;
```

The finite state machine of the SVPWM switches among 8 possible states. Each state corresponds to a sector (➪ Figure 4.4). Each sector is adjacent to two vectors, which drive to a defined configuration of the IGBTs. The two adjacent vectors of a state only differ in one bit, and this bit is weighted and commuted on *proc_pwm_switching* at a fixed frequency (only for simulation purposes). Then the output signals are driven to the 6-bit vector s which controls the inverter.

Listing 9.7: SVPWM description in VHDL (Architecture: behavioral)

```
sector_determination : process (clk) begin
    if rising_edge(clk) then
        if (reset = '1') then
            sector <= 0;
        elsif (v_beta >= 0) then
            if (v_alpha >= 50) then
                sector <= 3;
            elsif (v_alpha <= -50) then
                sector <= 5;
            else
                sector <= 1;
            end if;
        else
            if (v_alpha >= 50) then
                sector <= 2;
            elsif (v_alpha <= -50) then
                sector <= 4;
            else
                sector <= 6;
            end if;
        end if;
    end if;
end process;

proc_pwm_switching : process begin
    vector <= pwm_h; wait for 5us;
    vector <= pwm_l; wait for 5us;
end process;

s_buff(1) <= vector(0);
s_buff(3) <= vector(1);
s_buff(5) <= vector(2);
s_buff(4) <= not(s_buff(1));
s_buff(6) <= not(s_buff(3));
s_buff(2) <= not(s_buff(5));
s <= s_buff;
```

9.5 Simulation and Verification

On the top level file, the one that collects all of the code parts (functions, components, signals, ports, ...) signal routing among them has to be described. For this purpose, an instance of each one of the components that pertain to the FOC method have to be instantiated (➪ Listing 9.8). All of these modules run concurrently, thus being achieved one of the main purposes of developing a controller in VHDL instead

of software.

Listing 9.8: Routing the components on the top level file

```
inst_clarke : clarke
    port map (  a => i_a,
                b => i_b,
                alpha => i_alpha,
                beta => i_beta
                );

inst_park : park
    port map (  clk => clk,
                reset => reset,
                alpha => i_alpha,
                beta => i_beta,
                theta => theta,
                d => i_d,
                q => i_q
                );

inst_invpark : invpark
    port map (  clk => clk,
                reset => reset,
                d => v_d_ref,
                q => v_q_ref,
                theta => theta,
                alpha => v_alpha_ref,
                beta => v_beta_ref
                );

inst_svpwm : svpwm
    port map (  clk => clk,
                reset => reset,
                v_alpha => v_alpha_ref,
                v_beta => v_beta_ref,
                s => s
                );
```

On the TestBench, the *stimulus* signals are defined. Stimulus signals have the purpose of testing the Field-oriented Control as a whole to check the correct behavior of the system and of all of its components. On (➪ Listing 9.9), a three-phase generation is simulated, and a *theta* is defined and calculated, which defines the period of the signals.

Listing 9.9: Stimulus generation (TestBench) (I)

```
clk_tb <= not clk_tb after 5ns; --half_period

inst_clock_divider_theta : clock_divider
    port map (  clk => clk_tb,
                reset => reset_tb,
                eoc => 100,
                clk_div => clk_div_theta
                );
```

Listing 9.10: Stimulus generation (TestBench) (II)

```
proc_counter_theta : process (clk_tb) begin
    if rising_edge(clk_tb) then
        if (theta_tb >= 359) then
            theta_tb <= 0;
        elsif (theta_120_tb >= 359) then
            theta_120_tb <= 0;
        elsif (theta_240_tb >= 359) then
            theta_240_tb <= 0;
        elsif (clk_div_theta = '1') then
            theta_tb <= theta_tb + 1;
            theta_120_tb <= theta_120_tb + 1;
            theta_240_tb <= theta_240_tb + 1;
        end if;
    end if;
end process;

inst_current_wave_gen_phase_a : trigonometry
    port map (  clk => clk_tb,
                reset => reset_tb,
                address  => theta_tb,
                sin => i_a_real_tb
                );

-- . . . Phases 'b' and 'c' . . .

i_a_tb <= integer(100.0*i_a_real_tb);
i_b_tb <= integer(100.0*i_b_real_tb);
i_c_tb <= integer(100.0*i_c_real_tb);
```

At this point, the controller is implemented at simulation, non-synthesis level. Now a simulation bench is defined to monitor all of the signals involved within the transformations and modules.

On the ➪ Figure 9.1, the signals for the Clarke and Park transformations together, which are referred as Direct-quadrature-zero transformation, are shown. From the currents of the three phases (represented as a phasor), and for a given *theta*, the output of the Clarke transformation is a new reference frame *(alpha, beta)* which are sinusoidal signals moved 90 degrees from one another. These signals are converted to continuous, time-invariant signals at Park transformation (*(d, q)* frame).

On the ➪ Figure 9.2, the same signals are shown for the reverse process: from *(d, q)* frame to (alpha, beta) and then to the six IGBT signals to the inverter. The waves at *s* define, for each phase, a sine signal that will make the magnetic field inside of the stator to move at a frequency defined by *theta*.

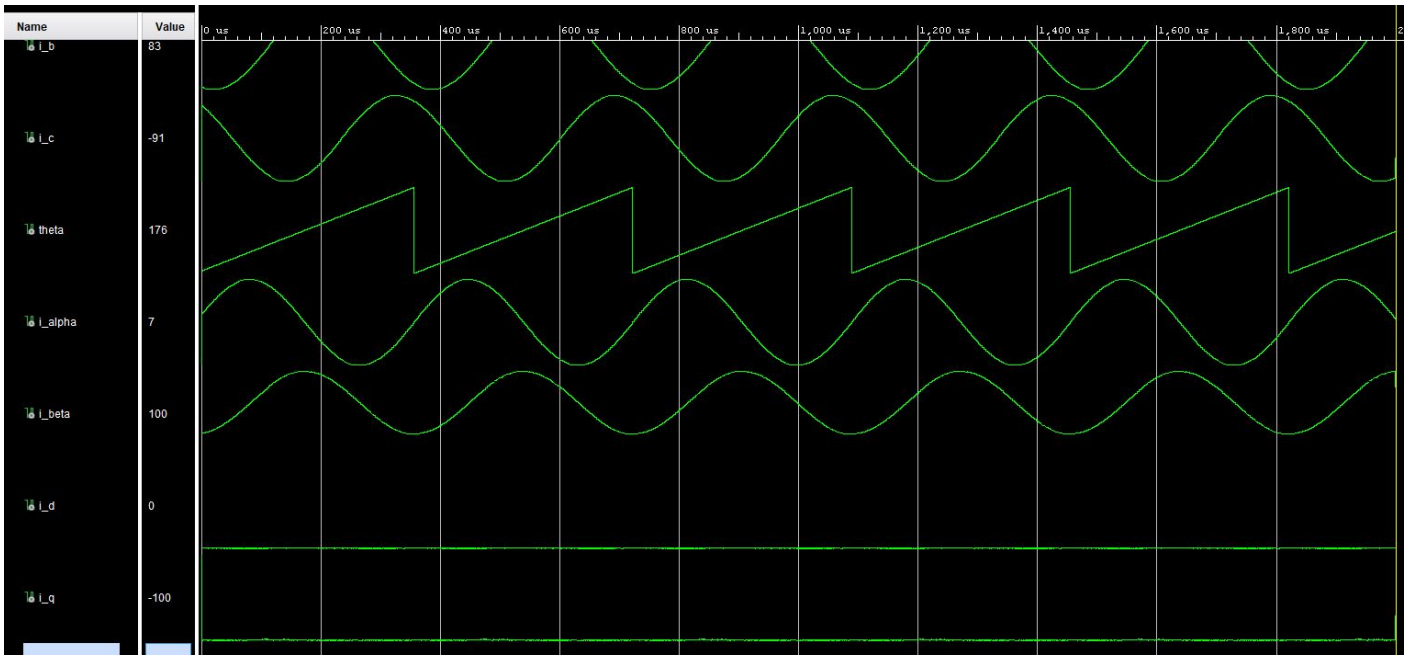


Figure 9.1 *Direct-quadrature-zero transformation waveforms*

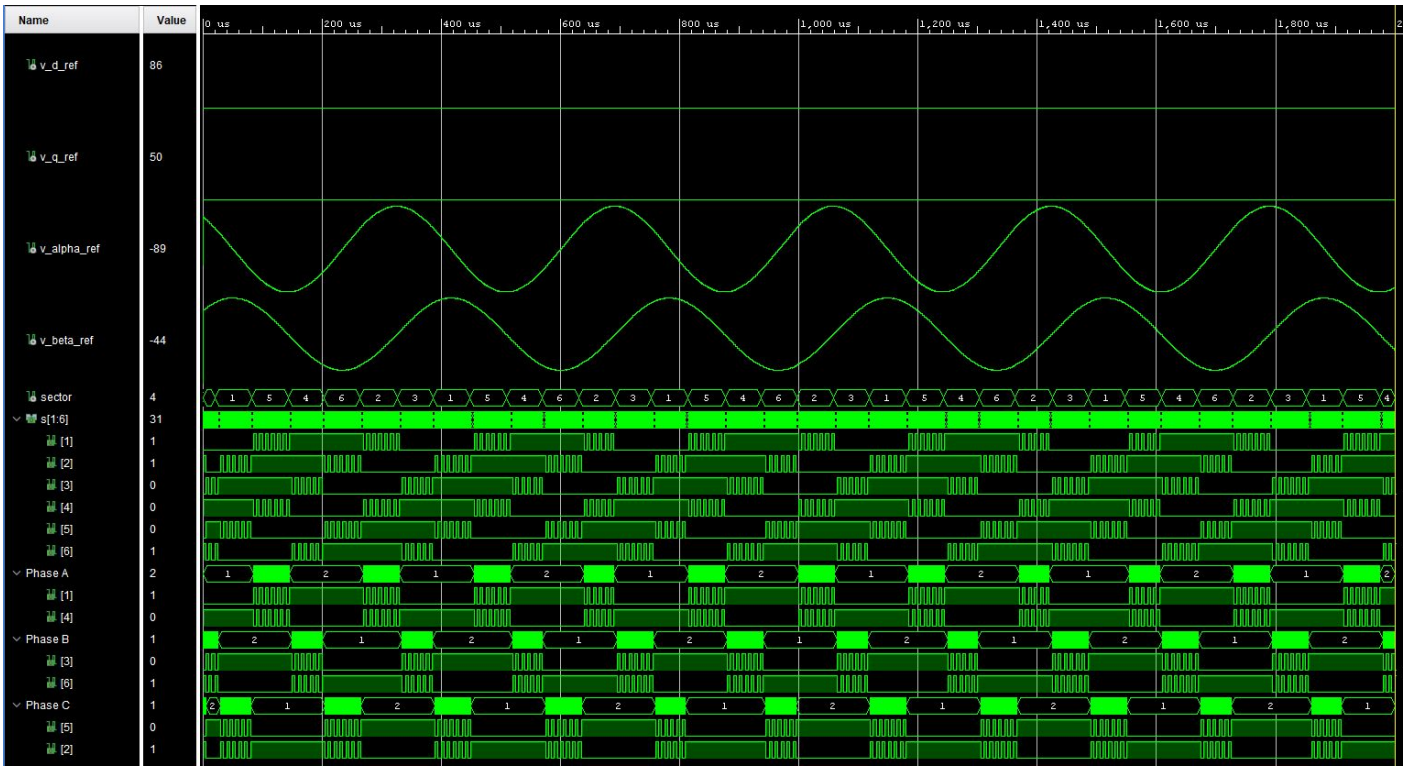


Figure 9.2 *Inverse Park transformation and SVPWM waveforms*

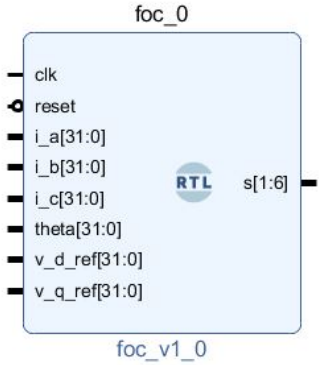


Figure 9.3 *FOC representation as a block*

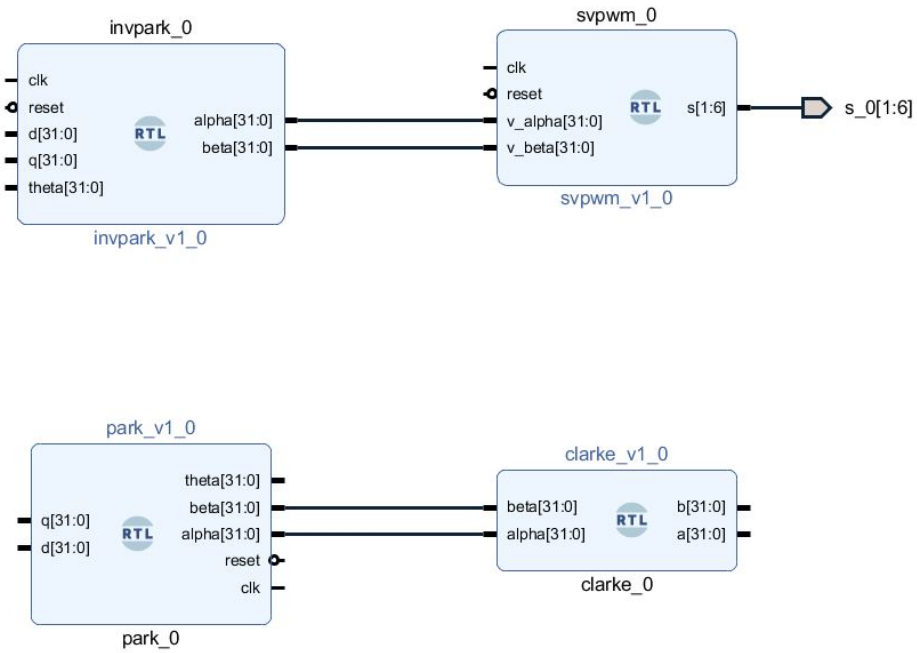


Figure 9.4 *FOC block diagram*

Part IV

Ending

10 Conclusion

The thesis *Implementation and verification of a hardware-based controller for a three-phase induction motor on an FPGA* collects the result of a period of documentation and research in the field of AC electric motors, especially regarding their control methods. The three most common methods are included: V/Hz scaling control, FOC and DTC, and a brief overview of FPGAs and induction motors is also done. At the practical development of the work, the hardware description is made for two of the methods: one for scalar and another for vectorial control. These two methods have been developed, implemented, simulated and successfully verified with the VHDL language in Vivado. In the case of scalar control, it has been possible to get it to work in a real three-phase induction motor in the laboratory.

From all of the work developed, one can deduce that control methods are much more important than even the construction and the quality of the motor. Depending on the complexity of the control system used, the motor has a higher performance and a better response. Of the two cases implemented in VHDL, it emerges that vector control is much more accurate and appropriate for the vast majority of engines, while scalar control, a more simple method, can be an effective alternative in other applications, especially those where the engine operates under familiar conditions: when torque and speed tend to be more constant.

Implementing controls in an FPGA simplifies the system at a hardware level and allows scaling and extending it much more easily than if it was made with real hardware. A notable advantage is the fact that an FPGA directly controls signals instead of bits, so that unlike a microcontroller, the delays are much smaller.

However, the control methods have some limitations at a practical level: if the purpose was to go beyond the simulation of the vector control method so to controlling a real inverter and motor, it would be necessary to adapt the code in order to implement PI controllers and position and an angular velocity estimator in software. It would also be necessary to use the Analog-to-digital Converter (ADC) on the board together with high-precision current sensors. This proposal of content extension is contemplated in the next chapter.

As a general conclusion, I consider the purposes of the thesis achieved and I want to show my personal satisfaction of having been able to enjoy, both in Tallinn and in Manresa, the possibility of studying and working on the topics included in the research, with two supervisors, David and Aleksander, with whom I have learned a lot.

11 Future work

The versatility and power of FPGAs help to quickly extend and improve HDL designs. Some of the proposed improvements are described in the following paragraphs.

Regarding the VFD scalar control, an extension of the project could be to modify the controller from an open-loop to a closed-loop version, monitoring either the current on the motor's phases or coupling an encoder at the shaft. If these variables can be controlled, then the amplitude of the sine signals (the effective torque of the motor) would be unlinked from the frequency and could be managed by other components. This would suppose an improvement in performance and especially the motor would become reliable at different loads and lower speeds.

When it comes to the vector FOC, the next step would be to adapt the VHDL model to a version that could run in a real FPGA and control a three-phase motor. For this purpose, the load of work is greater: an estimator block would have to be designed to generate an estimation of angular speed and magnetic field orientation according to the motor parameters, and written in software (e.g. C) on a microcontroller. In addition, the current of at least two phases would have to me monitored by the FPGA and converted from an analog into a digital signal.

Bibliography

[1] K Vinoth Kumar, Prawin Angel Michael, Joseph P John, and S Suresh Kumar. Simulation and comparison of spwm and svpwm control for three phase inverter. ARPN Journal of Engineering and Applied Sciences, 5(7):61–74, 2010. 12

[2] Vaughn Betz and Jonathan Rose. Vpr: A new packing, placement and routing tool for fpga research. In International Workshop on Field Programmable Logic and Applications, pages 213–222. Springer, 1997.

[3] Steven Trimberger, Dean Carberry, Anders Johnson, and Jennifer Wong. A time-multiplexed fpga. In Field-Programmable Custom Computing Machines, 1997. Proceedings., the 5th Annual IEEE Symposium on, pages 22–28. IEEE, 1997.

[4] Conrad Brunner. Efficient electric motor systems. In Presentation Slides, Motor Summit 2014, 2014.

[5] Rakesh Parekh. Ac induction motor fundamentals. Microchip Technology Inc, 2003.

[6] Hisao Kubota and Kouki Matsuse. Speed sensorless field-oriented control of induction motor with rotor resistance adaptation. IEEE Transactions on Industry Applications, 30(5):1219–1224, 1994.

[7] Georgios Papafotiou, Tobias Geyer, and Manfred Morari. Optimal direct torque control of three-phase symmetric induction motors. In Decision and Control, 2004. CDC. 43rd IEEE Conference on, volume 2, pages 1860–1865. IEEE, 2004.

[8] Douglas E Ott and Thomas J Wilderotter. A designer's guide to VHDL synthesis. Springer, 2013.

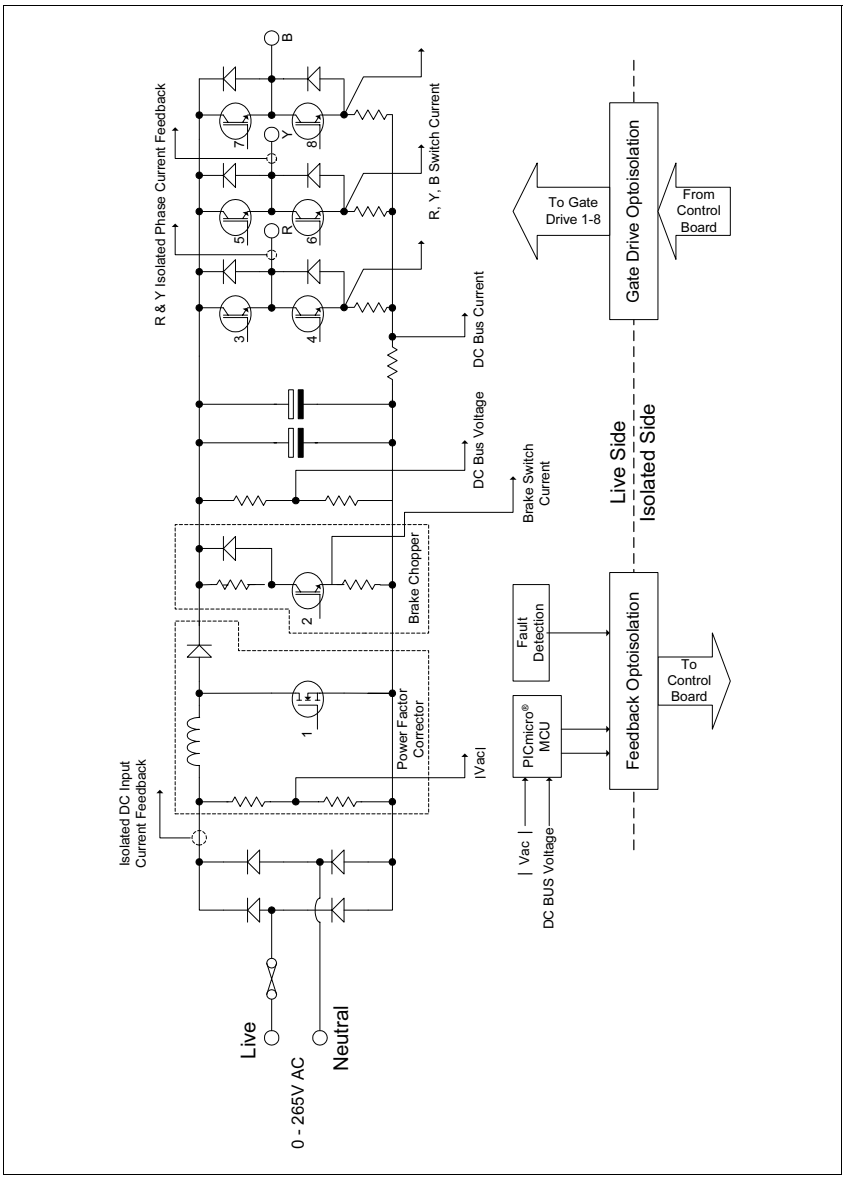
[9] Gopal B T Venu. Comparison Between Direct and Indirect Field Oriented Control of IM. Bangalore University, 2017.

Appendices

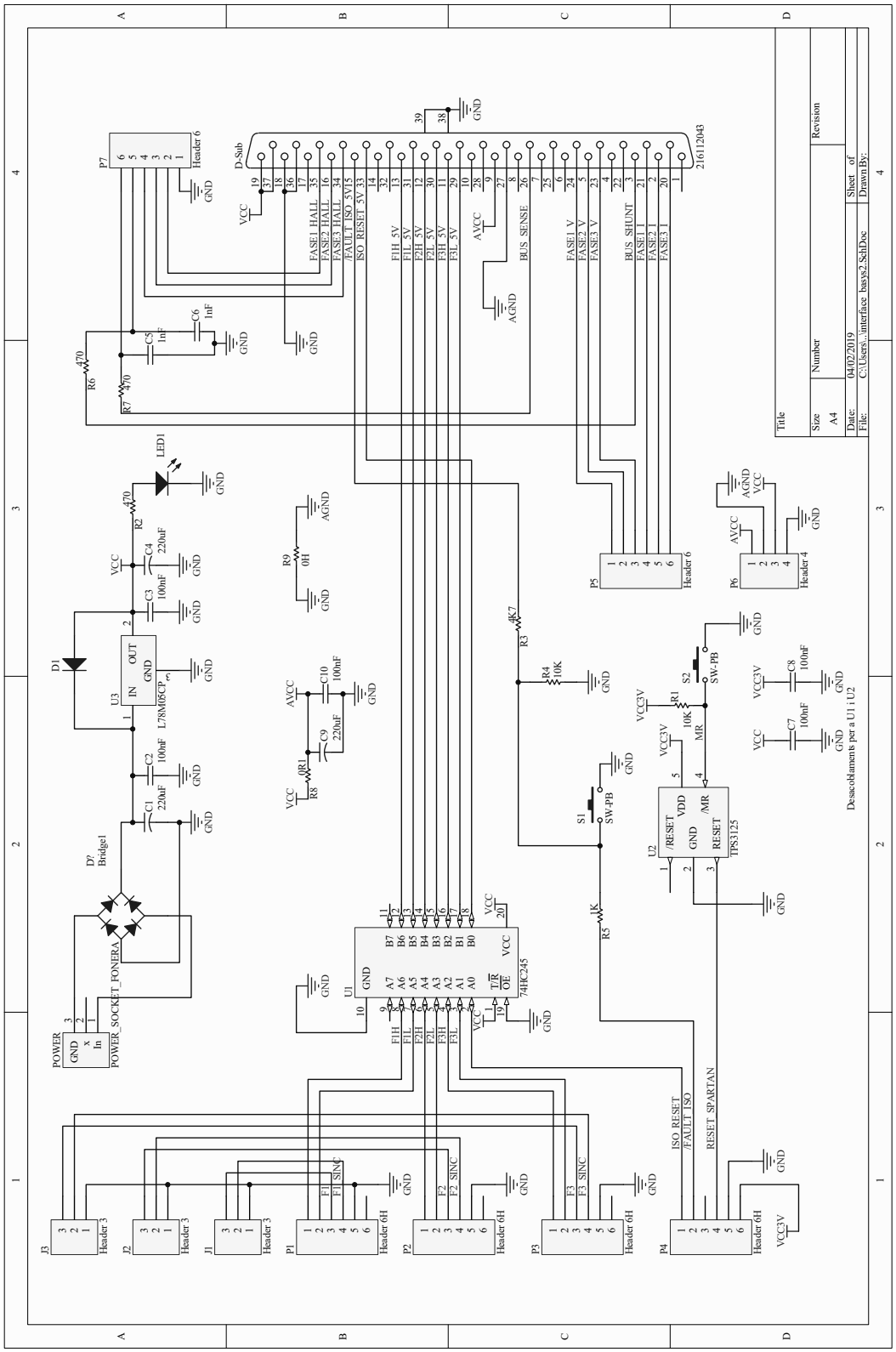
A MC1H 3-Phase Power Module Block Diagram

dsPICDEM™ MC1H 3-Phase High Voltage Power Module

FIGURE 1-2: MC1H 3-PHASE HIGH VOLTAGE POWER MODULE BLOCK DIAGRAM



B Voltage Adapter PCB Scheme



Author: David Soler