ROYAL INSTITUTE
OF TECHNOLOGY

Case study of a Rapid Control Prototyping system based on Xilinx System Generator

Fallstudie av ett Rapid Control Prototyping-
system baserat på Xilinx System Generator

MARIA BORGSTRÖM

Master's Thesis at KTH/ICT and ABB AB

Supervisors:
Johnny Öberg - KTH/ICT
Héctor Zelaya de la Parra - ABB AB, Roger Mellander - ABB AB
Examiner:
Ingo Sander - KTH/ICT

KTH/TRITA/ICT/EX-2010:30

Abstract

Time, money and quality are three important parameters for competitive industry companies and Rapid Control Prototyping is a concept that is used for improving the design process of for example a mechatronic system. It implies working with models in a high level program that automatically generates low level code. This saves time consuming code writing and by working with models a part in an algorithm can easily be changed or replaced without affecting other parts.

The Rapid Control Prototyping tool Xilinx System Generator that runs from Simulink is here investigated. Implementation of the Field Oriented Control algorithm for controlling a motor serves as main subject for the investigation. A library with ready-to-use blocks is created for having the possibility to implement other control algorithms in the future in a fast, graphical, intuitive and user friendly way.

It is shown the advantage of using an FPGA with its parallelism and re-programmable characteristics when implementing a motor control algorithm. It provides a high bandwidth and therefore a possibility to control several motors with one FPGA. By programming the FPGA with a Rapid Control Prototyping tool like Xilinx System Generator the opportunity to in an easy way change different parts becomes obvious. To use Model Based Design and Rapid Control Prototyping concepts extensive code writing is avoided and by that for example syntax errors. The gap between the software engineer and the hardware engineer reduces and the possibility to work in both of the domains is given.

Keywords: Rapid Control Prototyping, Xilinx System Generator, Field Oriented Control

Sammanfattning

Fallstudie av ett Rapid Control Prototyping-system baserat på Xilinx System Generator

Tid, pengar och kvalitet är tre viktiga parametrar för vinstinriktade företag och Rapid Control Prototyping är ett koncept som används för att förbättra designprocessen av till exempel ett mekatroniksystem. Rapid Control Prototyping innebär att jobba med modeller i ett högnivåspråk som automatiskt genererar lågnivåkod. Detta besparar tidskrävande kodskrivning och genom att arbeta med modeller kan en del i en algoritm lätt ändras eller bytas ut utan att påverka andra delar.

Xilinx System Generator, som körs ifrån Simulink, utvärderas här och Rapid Control Prototyping används som ett verktyg för att implementera fältorienterad motorkontroll. Ett bibliotek med block klara att använda skapas för att ha möjligheten att implementera andra kontrollalgoritmer i framtiden på ett snabbt, grafiskt, intuitivt och användarvänligt sätt.

Fördelen med att använda en FPGA, med dess parallellism och återprogrammerbarhet, påvisas genom implementering av en motorkontrollalgoritm. Den ger en hög bandbredd och därigenom möjligheten att kontrollera flera motorer med en FPGA. Genom att programmera FPGA:n med Rapid Control Prototyping-verktyg som Xilinx System Generator blir möjligheten att på ett lätt sätt byta olika delar uppenbar. Genom att använda modellbaserad design och Rapid Control Prototyping-konceptet undviks omfattande kodskrivning och genom det syntaxfel. Gapet mellan mjukvaru- och hårdvaruingenjörer minskas och möjligheten att arbeta i de båda domänerna ges.

Nyckelord: Rapid Control Prototyping, Xilinx System Generator, Fältorienterad kontroll

Acknowledgments

This thesis work was performed at the Department of Automation Technologies (AT) at ABB Corporate Research (CRC) in Västerås from September 2009 to January 2010.

I would like to thank my supervisors **Héctor Zelaya de la Parra** and **Roger Mellander** at ABB with their winner spirit and their great support and encouragement. Thanks to Héctor my dream of doing my master thesis at ABB came true and during the way Roger was always there giving me a hand, stealing *or giving* me a pen when I needed it.

A special thank goes to my family **Eva, Olle** and **Markus Borgström** and **Oscar Lucas Villalón** for all their love and faith in me and their never-ending encouragement and patience. I love all of you ♡

I would also like to thank **Maria Nilsson** for always being there as a friend and for lending me her precious red car JAN - without *"Janne"* it would have been impossible to go to work every day.

A big hug and *tusen tack* to all of my friends on CRC for being there and for giving the just mentioned precious little red car a push when it was needed during the coldest winter days.

It has been a pleasure!

Maria Borgström
January 2009

Contents

List of Figures

Chapter 1

Introduction

Rapid Control Prototyping is something that the industry nowadays uses for improving their benefit. By using a high level graphical programming tool the development process goes much easier and faster than with conventional low level code programming. The graphical high level program contributes by facilitating the development process, the testing and increases the possibilities to change some parts in the system in an easy way.

1.1 Purpose

The purpose is to investigate the possibility of using the high level program Xilinx System Generator as a Rapid Control Prototyping tool. The advantage with implementing motor control using an FPGA will be shown since its parallelism offers implementation of several motor axes. The fact that it is possible to work in different regions, the FPGA domain and the motor control domain, without extensive knowledge of hardware programming is also a showed advantage.

1.2 Scope

A motor control algorithm is implemented using Xilinx System Generator for examination of its features and performance. The possibilities and advantages by working with an FPGA, Model Based Design and Rapid Control Prototyping is also showed.

The evaluation described is based upon testing XSG as a Rapid Control Prototyping tool by implementing Field Oriented Control (FOC) for a motor. The FOC algorithm is chosen but could have been substituted with any other motor control algorithm.

This work is not about how to optimize for examples the regulators in the motor control algorithm. The quality of the automatically generated code is not investigated, the main purpose is to demonstrate the benefits of implementing the motor control algorithms in an FPGA with a high level program by using the MBD

and RCP concepts. The result of occupied area on the FPGA shows out to be good and more than sufficient for the purpose and therefore a deeper investigation is not done.

1.3 Definitions

The used abbreviations are written out the first time they are mentioned and can be found in table 1.1.

1.4 Structure

The structure of this report is as follows:

Chapter 1 - *Introduction* - describes the purpose and scope for this work as well as terms, abbreviations and acronyms used.
Chapter 2 - *Problem Description* - describes the background of this project, related and previous work as well as the problem description in a compact form as the problem statement.
Chapter 3 - *Conceptual Background* - starts by a brief introduction about FPGAs and then the used concepts are described.
Chapter 4 - *Theoretical Background* - contributes with the background theory like the Field Oriented Control algorithm, Space Vector Pulse Width Modulation and coordinate transformations.
Chapter 5 - *Development Environment* - contains information about the used software and hardware.
Chapter 6 - *Solution* - describes as the name reveals the solution and the implementation of the problem statement.
Chapter 7 - *Results* - presents the achieved results.
Chapter 8 - *Discussion* - conducts a discussion considering expectations, problems and achieved results.
Chapter 9 - *Conclusions* - contains the conclusions that can be drawn from the work that is done.
Chapter 10 - *Future Work* - refers to the continuing work that can be done from this work.

In the end a reference list with bibliography is found that specifies source material and further reading.

ADC	**A**nalog to **D**igital **C**onverter
BRAM	**B**lock **R**andom **A**ccess **M**emory
FOC	**F**ield **O**riented **C**ontrol
FPGA	**F**ield **P**rogrammable **G**ate **A**rray
HDL	**H**ardware **D**escription **L**anguage
HIL	**H**ardware **I**n the **L**oop
MBD	**M**odel **B**ased **D**esign
PWM	**P**ulse **W**idth **M**odulation
RCP	**R**apid **C**ontrol **P**rototyping
RFOC	**R**otor **F**lux **O**riented **C**ontrol
SPI	**S**erial **P**eripheral **I**nterface
SPWM	**S**inusoid **P**ulse **W**idth **M**odulation
SVPWM	**S**pace **V**ector **P**ulse **W**idth **M**odulation
VHDL	**V**HSIC **H**ardware **D**escription **L**anguage
VHSIC	**V**ery-**H**igh-**S**peed **I**ntegrated **C**ircuit

Table 1.1. Abbreviations.

Chapter 2

Problem Description

2.1 Background

In all industry fields within processes of research and development time efficiency and flexibility are of great importance. The possibility to implement, test and develop algorithms in a time saving way is in focus nowadays and especially a concept called Rapid Control Prototyping can be brought out. In the area of electrical motors and their control algorithms the RCP shows in the way of implementing and developing the algorithms like models with a high level program and implement it into hardware, this instead of using time consuming low level programming systems.

As a new mechatronic laboratory is under construction at ABB Corporate Research in Västerås the renewal of the premises considers also renewal of the development processes. For the motor control part different systems of Rapid Control Prototyping are under evaluation and it is here where this project enters.

2.2 Problem definition

The main purpose of this project is to evaluate the high level program Xilinx System Generator (XSG), see section 5.1.2, as a part of the Rapid Control Prototyping concept. The evaluation is done by implementing the Field Oriented Control (FOC) algorithm, explained in section 4.1, on an FPGA for controlling a small motor. As the aim of Rapid Control Prototyping is fast and easy development, implementation and testing the FOC model is divided and developed as smaller parts in form of easy to handle compact sub blocks. These blocks will constitute library from where the different sub blocks can be easy accessed and used in new designs and future implementations.

2.2.1 Problem statement

- Create a library with the different parts of the FOC algorithm.
- Have a motor running with the FOC algorithm on the FPGA.
- Investigate XSG as a Rapid Control Prototyping tool.
  - Time and area estimation.
- Describe the advantages with the FPGA for this application.

2.3 Related work

To the best of the author's knowledge no previous implementations or evaluations with the FOC algorithm can be found using Xilinx System Generator. Therefore a comparison with the same motor control algorithm, the Field Oriented Control, but with different hardware and software is accomplished for having some idea of XSG's performance.

Another motor control algorithm, Rotor Flux Oriented Control (RFOC), has been implemented with Xilinx System Generator in [5]. The approach is the same as in this work - implementation of the motor control algorithm with Xilinx blocks where low level code is generated and tested through co-simulation. Although the motor control algorithm is different and the motor is modeled and simulated some comparisons can be made. In [5] the ML402 Development Platform is employed, instead of the ML507 that is used here, but overall it is more or less similar and comparable.

2.4 Previous work

The Field Oriented Control algorithm with the Rapid Control Prototyping concept has earlier been implemented at ABB with LabVIEW [3], [4] and a small comparison analysis with this work can be carried out from those ones. It is concluded that is relatively easy to start programming with LabVIEW without knowing any Hardware Description Languages (HDL). A disadvantage is the long compilation time, about 5 to 35 minutes.

Chapter 3

Conceptual Background

3.1 Field Programmable Gate Array

A Field Programmable Gate Array (FPGA) is a silicon device that contains logic. It is constructed of cells called Configurable Logic Block (CBL), each configurable logic block contains more or less a Look Up Table (LUT), a Flip-Flop and a multiplexer. In-between the CBLs there are interconnections and at the borders input and output cells. Figure 3.1 shows how the FPGA is built up. An FPGA is normally programmed with a Hardware Description Language (HDL) like VHDL or Verilog. An FPGA can be re-programmable and several tasks can be executed at the same time, in other words parallel programming can be applied to it.

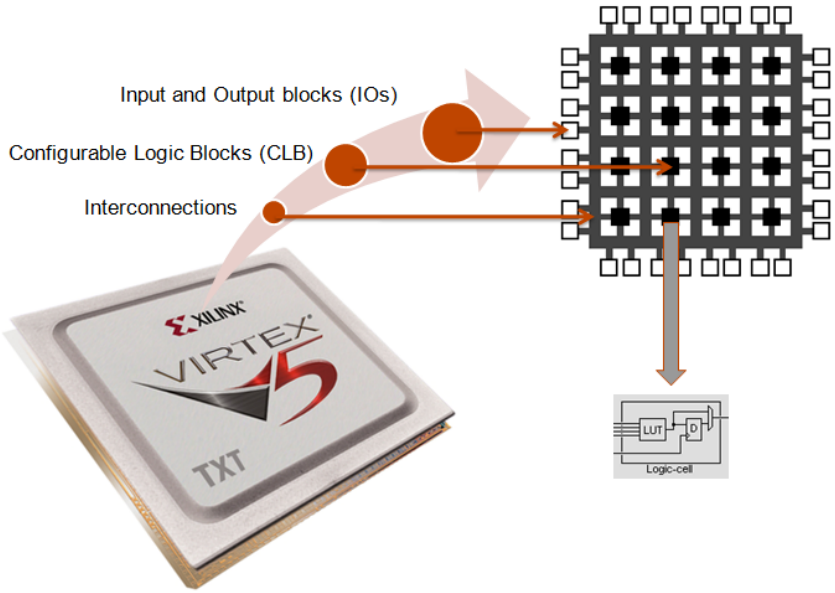

Figure 3.1. An FPGA.

3.2 Model Based Design

Model Based Design (MBD) is worth using when implementing and designing complex systems. The main key is as the name reveals models. Model Based Design means models from where applicable specifications can be drawn out, that the model then can be simulated, validated and tested. From this also automated code generation is an important point. The MBD can be summarized as shown in figure 3.2.

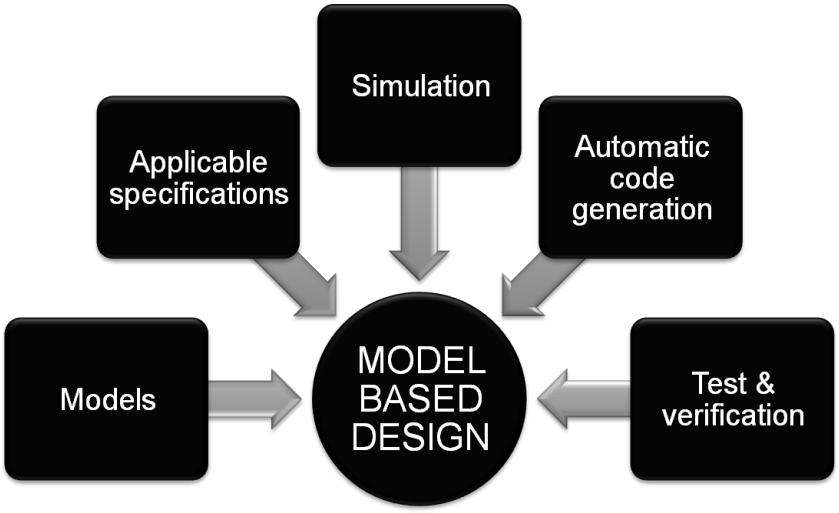

Figure 3.2. The model based design concept.

3.3 Rapid Control Prototyping

When going more towards hardware the MBD concept often is referred as the Rapid Control Prototyping concept. Rapid Control Prototyping is frequency used in industry since last decade because it allows testing and development of rather complex systems in real time and on the actual plant in a fast and easy way. The two most important key points in Rapid Control Prototyping are:

- Visual high-level programming.
- Automated low level code generation.

Automatic code generation permits an abstract model which is implemented by a visual high-level program. The quality of the code is an important issue but is normally hard to measure in a direct and non-subjective way.

The difference of implementing a design on an FPGA by writing the low level code by hand and modeling the design with a high level tool that generates the code is often shown in time, flexibility and knowledge. Figure 3.3 shows how to implement a counter in VHDL code and how to implement it with a high level program as a block.

By using a Rapid Control Prototyping tool the gap between hardware engineers and software engineers reduces. It is possible to simulate, test and verify control algorithms and put it onto hardware without being an expert in a HDL like VHDL or Verilog.

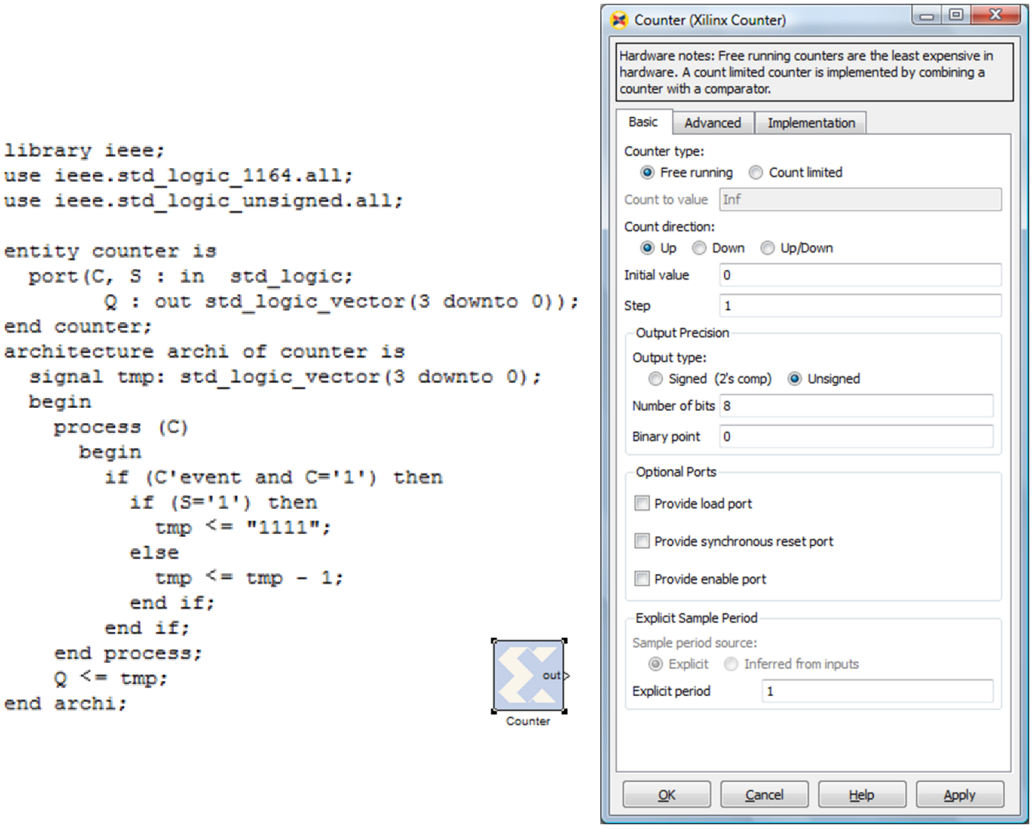

Figure 3.3. To the left a counter written in VHDL code and to the right a block with its corresponding dialog window.

3.4 Hardware in the loop

Hardware in the loop (HIL), or FPGA in the loop, is a concept that as revealed by the name uses the hardware in the simulation loop, see figure 3.4. This leads to easy testing and the possibility to see how the actual plant is behaving in hardware. By having the stimuli in software on the PC, implement a part of the loop in hardware and then receive the response from hardware back in the software, a good indication of the design's performance is given.

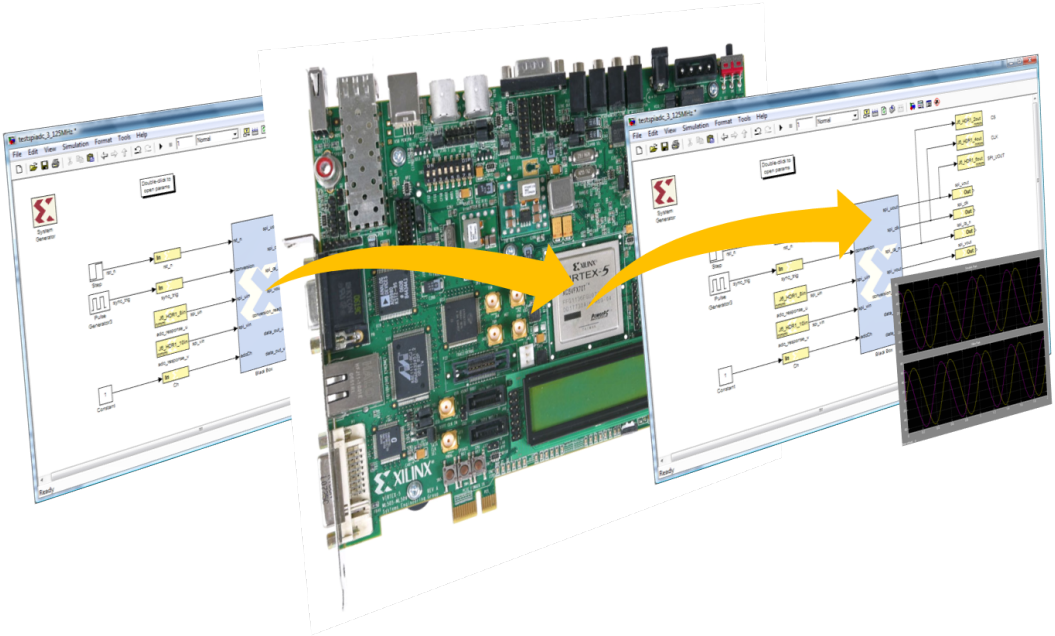

Figure 3.4. Hardware in the loop.

Chapter 4

Theoretical Background

4.1 Field Oriented Control

Field Oriented Control (FOC) is an algorithm that provides low torque ripple and smooth motion control at low speeds but also efficient performance at high speeds. The FOC algorithm, see figure 4.1, is controlling the currents in the dq-reference-frame of the rotor although the currents are measured in the stator frame which implies that a transformation has to be done. The algorithm starts by measuring the currents I_a and I_b in two of the three phases to the motor. The third current, I_c, is calculated by using Kirchhoff's current law that says that the sum of the currents in a joint should be zero which in this case implies $I_a + I_b + I_c = 0$. A rotation is done by the Clarke- and Park Transforms and then the currents are manipulated by two PI controllers and a feed forward controller and rotated back with the inverses of the Park- and Clark Transformations to the abc-frame. The voltage vectors V_a, V_b and V_c are then used to calculate the PWM duty cycles.

4.2 Coordinate transformations

In the FOC the PI controllers manipulates the currents in the rotating dq-frame, the currents from the motor are measured in the abc-frame and to change in between these two different coordinate systems the $\alpha\beta$-frame is used. How the different coordinate systems relates is visually shown in figure 4.2 and how to change in-between them is explained in the following sections.

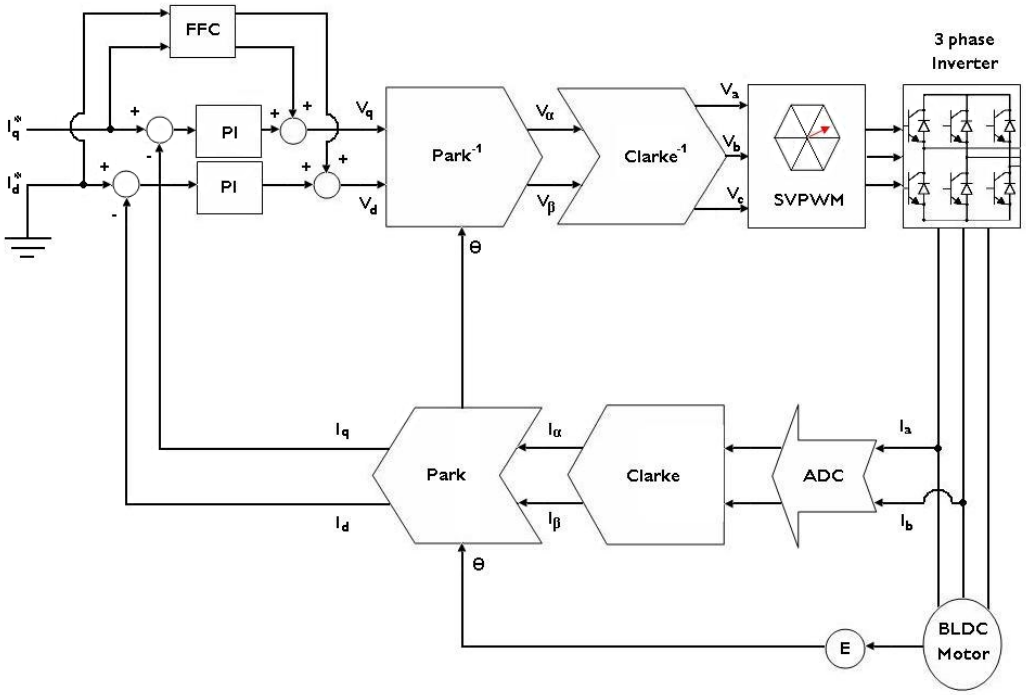

Figure 4.1. Field Oriented Control block diagram. *FFC - Feed Forward Control, SVPWM - Space Vector Pulse Width Modulation, E - Encoder.*

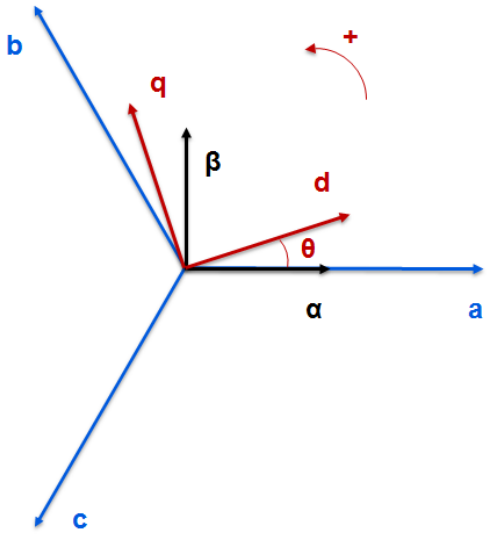

Figure 4.2. The different coordinate systems.

4.2.1 The Clarke Transformation

The first coordinate transformation that appears in the FOC algorithm after the ADC is called the Clarke Transformation. It shifts from a three axis system, the *abc*-frame, into a two axis system, the $\alpha\beta$-frame, both frames referenced to the stator. The transform is applicable both for the currents and the voltages in the phase windings but normally, as shown in figure 4.3, the currents are used. The transformation is realized by equation 4.1.

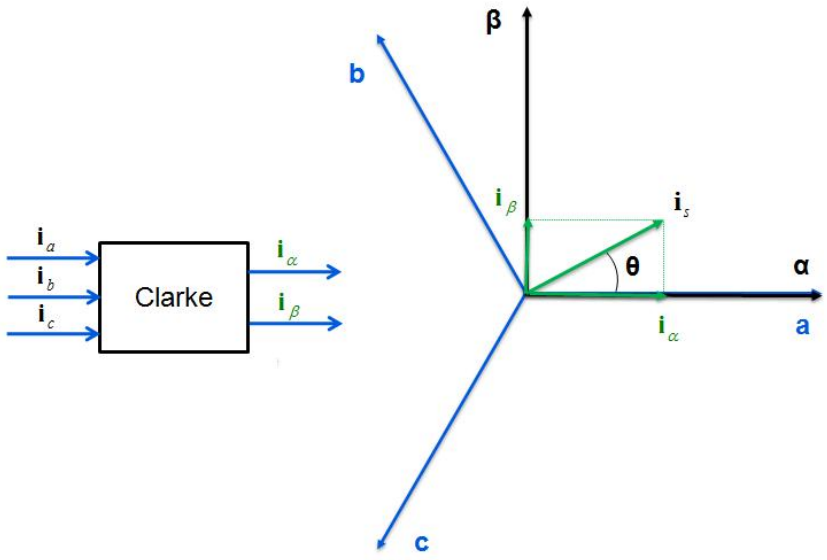

Figure 4.3. The Clarke Transformation.

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

4.2.2 The Inverse Clarke Transformation

The inverse Clarke Transformation is used just before the SVPWM block in the FOC algorithm and it transforms back from the $\alpha\beta$-frame to the *abc*-frame, see figure 4.4, with equation 4.2.

$$\begin{cases} V_a = V_\alpha \\ V_b = -\frac{1}{2} V_\alpha + \frac{\sqrt{3}}{2} V_\beta \\ V_c = -\frac{1}{2} V_\alpha - \frac{\sqrt{3}}{2} V_\beta \end{cases} \tag{4.2}$$

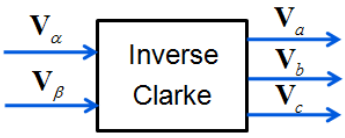

Figure 4.4. Inverse Clarke Transformation.

4.2.3 The Park Transformation

When having the stator current represented on a two axis system, the $\alpha\beta$-frame, the next step is to transform it into a new two axis system that is rotating with the motor flux, which is done by the Park Transformation as illustrated in figure 4.5. The angle θ defines the position of the rotor flux. The rotation is done by equation 4.3.

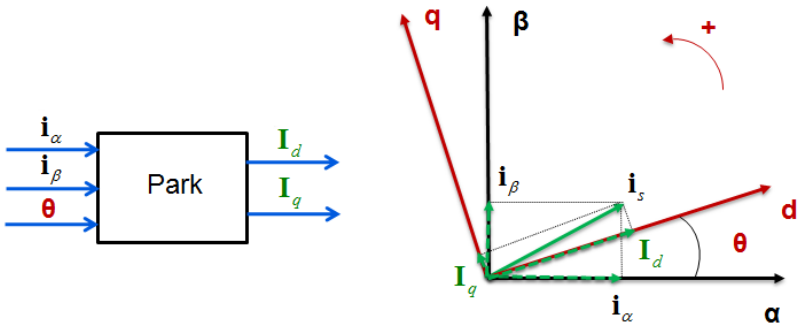

Figure 4.5. The Park Transformation.

$$\begin{cases} I_d = i_\alpha \cos\theta + i_\beta \sin\theta \\ I_q = -i_\alpha \sin\theta + i_\beta \cos\theta \end{cases} \tag{4.3}$$

4.2.4 The Inverse Park Transformation

The Inverse Park Transform transforms from the two axis rotating dq-frame to the two axis stationary $\alpha\beta$-frame as seen in figure 4.6. The rotation uses equation 4.4.

$$\begin{cases} V_\alpha = V_d \cos\theta - V_q \sin\theta \\ V_\beta = V_d \sin\theta + V_q \cos\theta \end{cases} \tag{4.4}$$

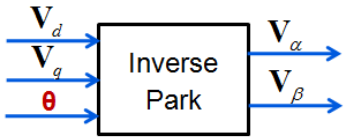

Figure 4.6. The Inverse Park Transformation.

4.3 Pulse Width Modulation

Pulse Width Modulation is used to generate a three phase voltage with the possibility to vary the frequency as well as the voltage. There are a lot of different methods that can be used and a big issue is to reduce the harmonics in the created outputs. Conventional pulse width modulation uses a comparator that compares a reference signal with a triangle wave signal. If the reference signal is a sinusoid the method is called Sinusoid Pulse Width Modulation (SPWM), see figure 4.7. SPWM can be used for one phase as well as for three phases. A drawback with SPWM is that it creates a lot of harmonics and therefore it is more common to use Space Vector Pulse Width Modulation (SVPWM) for controlling motors etc.

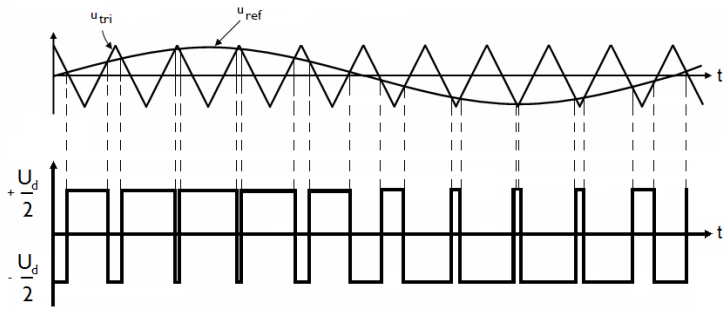

Figure 4.7. Sinusoid Pulse Width Modulation.

4.3.1 Space Vector Pulse Width Modulation

Space Vector Pulse Width Modulation is a widely used technique and is the most common for creating the voltage references generated by the Field Oriented Control. SVPWM generates less harmonic content than other techniques as for example SPWM. The three phase inverter, see figure 4.8, has $2^3 = 8$ different setups as shown in figure 4.9.

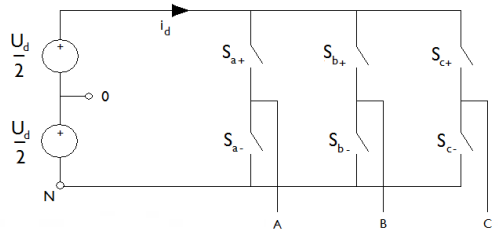

Figure 4.8. Simplified three phase inverter.

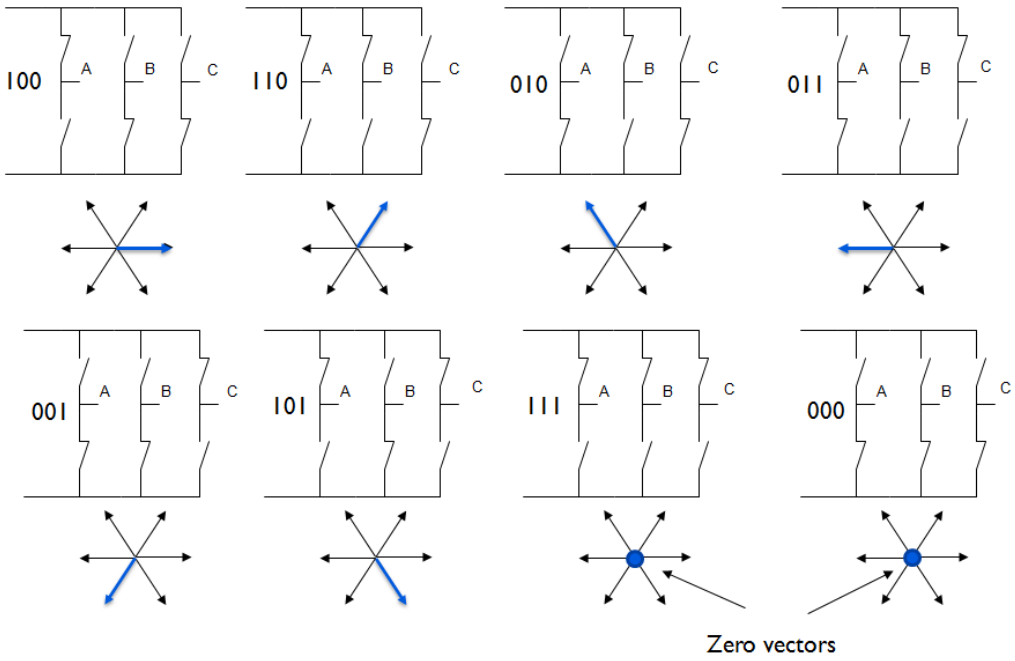

Figure 4.9. Switching states for the three phase inverter.

Duty cycle calculation

From the Inverse Clarke transform comes the voltage reference vector as an input to the SVPWM. The reference vector will land in its appropriate sector based on the signs of the three reference voltages. An illustration is shown in figure 4.10 where the reference voltage vector is placed in sector one and the corresponding vectors in the $\mathbf{V}_4$ and $\mathbf{V}_6$ directions. The vectors $\mathbf{V}_0$ and $\mathbf{V}_7$ are called zero vectors due to that the phase to phase voltages of these two vectors are zero.

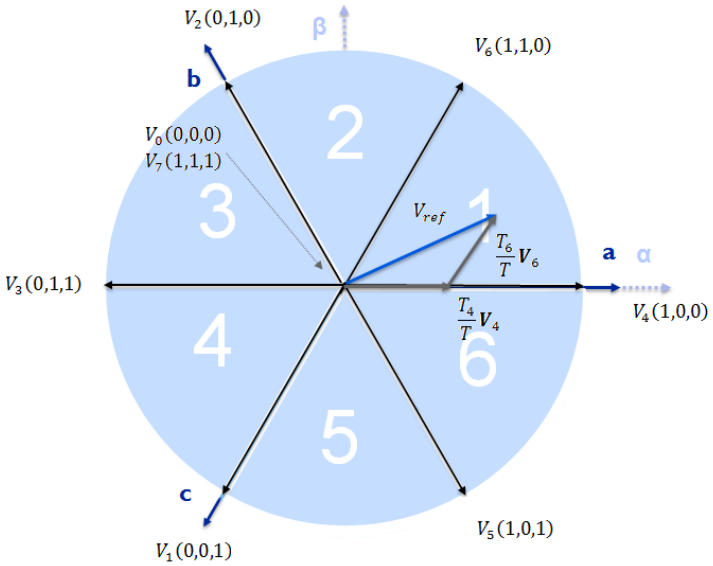

Figure 4.10. The reference voltage vector placed in sector one.

The desired stator voltage is created due to an appropriate switching pattern of the space vectors. To produce the voltage vector in the example the two adjacent voltage vectors and the two zero vectors are applied during a certain amount of time as shown in equation 4.5 where T_{PWM} is the switching period. The generated PWM waveform for the three phases is illustrated in 4.11. As can be seen the generated PWM waveform is symmetric or center aligned as this implementation minimizes the harmonic contents by reducing the switching frequency. The duty cycles for the remaining sectors are calculated in analogous way as described in the above example.

$$\mathbf{V}_{ref} = \frac{T_0\mathbf{V}_0 + T_4\mathbf{V}_4 + T_6\mathbf{V}_6 + T_7\mathbf{V}_7}{T_{PWM}} \tag{4.5}$$

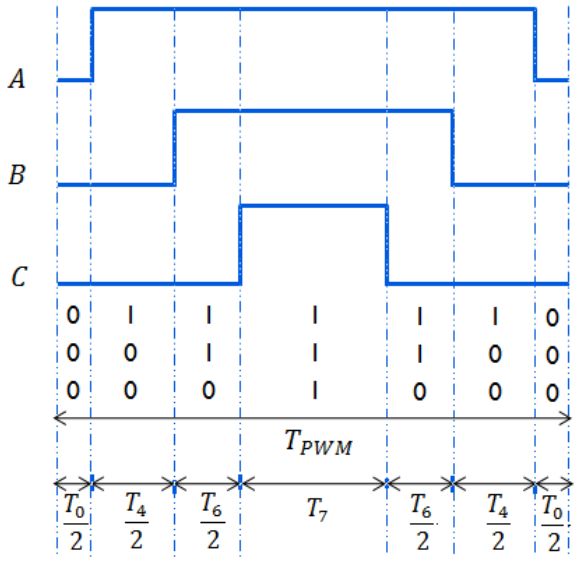

Figure 4.11. The generated PWM waveform for sector one.

Chapter 5

Development Environment

5.1 Software

5.1.1 Simulink®

Simulink® from The Mathworks™ that extends MATLAB, is a high level model-based design and simulation program for embedded and dynamic systems. With its graphical environment and block libraries it allows implementation, simulation and testing in an intuitive way from very simple systems to even more complex systems. The MATLAB version used in this project is R2009a.

5.1.2 Xilinx System Generator

System Generator is a tool that extends Simulink for implementing algorithms in a high level way that automatically can be implemented into an FPGA. The version used in this work is 11.3. Figure 5.1 shows an overview of Xilinx System Generator where the different parts are explained below:

1. **The System Generator block** - This token always has to be in a System Generator design. From the compilation options a bitstream or HDL Netlist can be created and the target for co-simulation is set here. Clocking options like FPGA clock and Simulink system period are also set from this block.

2. **The Simulink Library Browser** - The Xilinx Blockset can be reached from here. All blocks like for example delays, adders, registers etc. are here to be chosen.

3. **Gateway In Block** - The border between the Simulink blocks and the Xilinx blocks. In the Gateway In block the sample time, the quantization by the number of bits and the binary point are set.

4. **Gateway Out Block** - Is the outer border from Xilinx blocks to Simulink blocks. It converts Xilinx fixed point inputs into outputs of Simulink type like

double, float or fixed point. Specifications about I/O pin mapping can also be realized in this block.

5. **Resource Estimator Block** - This block estimates the slices, the embedded multipliers, flip-flops, BRAMs etc used in the design.

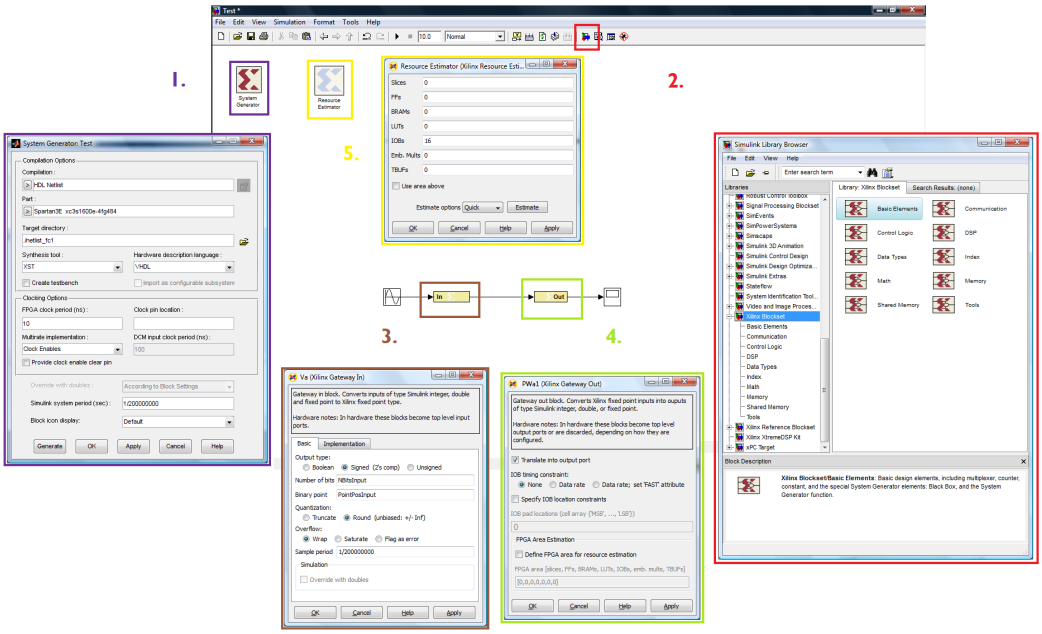

Figure 5.1. Xilinx System Generator overview.

An embedded design can be created with the different blocks in the Xilinx Blockset. Some special blocks are worth mention for their possibility to implement blocks created of already written functions in for example VHDL or MATLABs *m*-code. A brief introduction of two of these blocks is found here below and more information can be found in [1].

Figure 5.2. The Black Box block.

The Black Box block

The Black Box block, see figure 5.2, allows importing and implementing a function written in VHDL or Verilog into the design. When putting the Black Box block into the design you are asked to chose your HDL file as *.*v* or *.*vhd*. Some special additions has to be done in the code before using it. For example each clock name and clock enable name must contain the substring *clk* and *ce* respectively. When importing the HDL file a configuration .*m*-file is created that is used in the Black Box. It is good to know that the clock and clock enable ports are not visible on the black box block icon.

The MCode Block

The MCode Block, see figure 5.3, passes the input values of a MALTAB function for evaluation in Xilinx fixed-point type. The inputs and outputs of the function becomes the input and output ports of the block. The *m*-code that the block uses is translated in a straightforward way into equivalent behavioral VHDL/Verilog when hardware is generated. Some design rules has to be followed when writing the *m*-code, for example all block inputs and outputs must be of Xilinx fixed-point type and all blocks must at least have one output port.

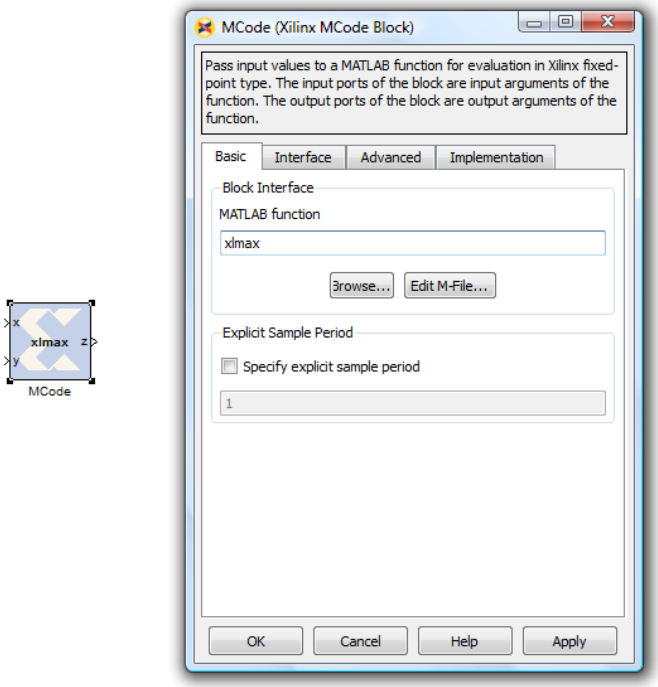

Figure 5.3. The MCode block with its dialog window.

Figure 5.4 shows a summary of the three different implementation approaches considered - With pure blocks (adders, multipliers, registers etc), with a *m*-code function as a MCode block and as a Black Box created of VHDL code.

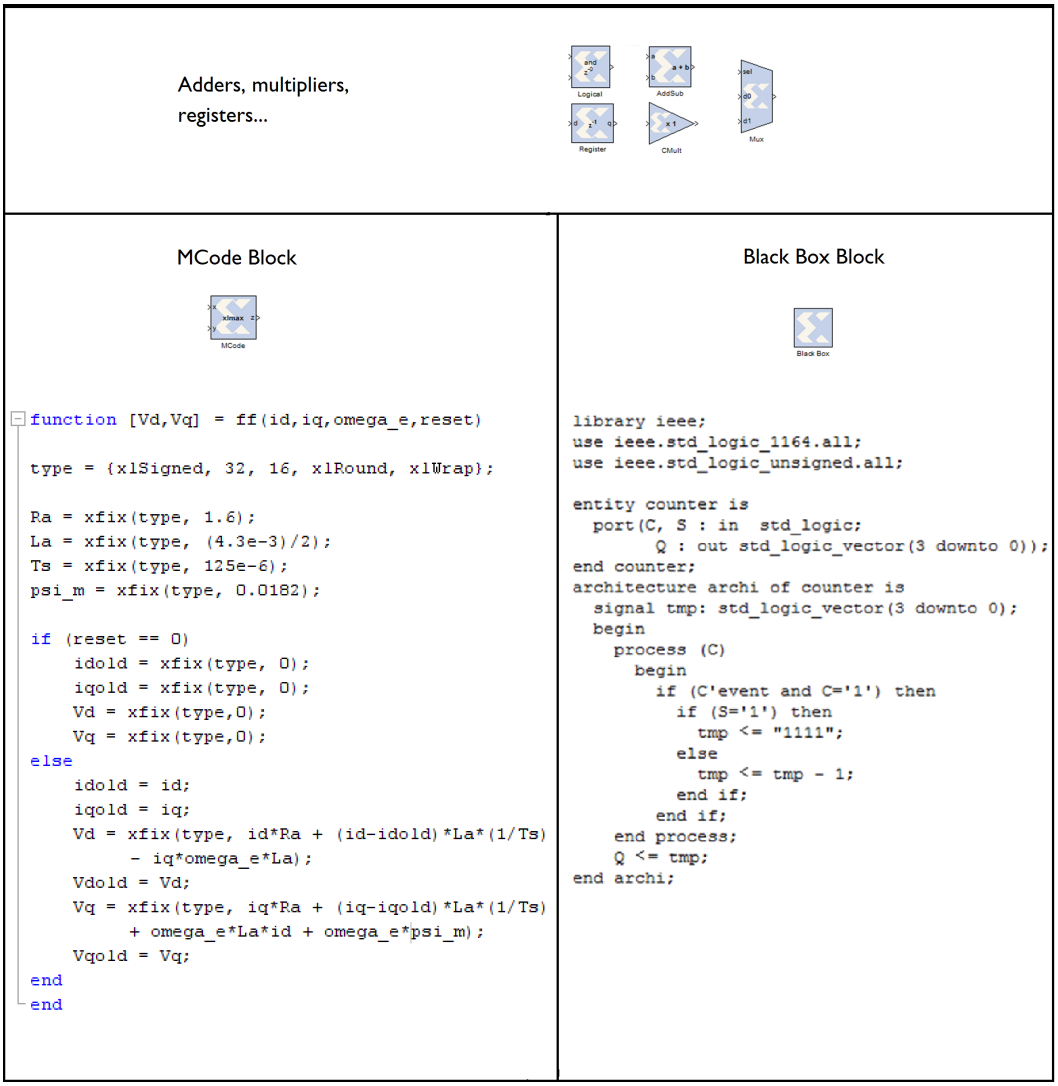

Figure 5.4. Summary of implementation approaches.

5.2 Hardware

5.2.1 Xilinx ML507 Evaluation Platform

The ML507 Evaluation Platform used in this project, see figure 5.5, contains an FPGA from the Virtex-5 family named XC5VFX70T-1FFG1136. The board contains a lot of different features like for example Flash PROMs, expansion header with 32 single-ended I/O, 14 spare I/Os shared with buttons and LEDs, power, JTAG chain expansion capability and a 16-character x 2-line LCD display [2]. Information about the board's Virtex-5 FPGA is shown in table 5.1.

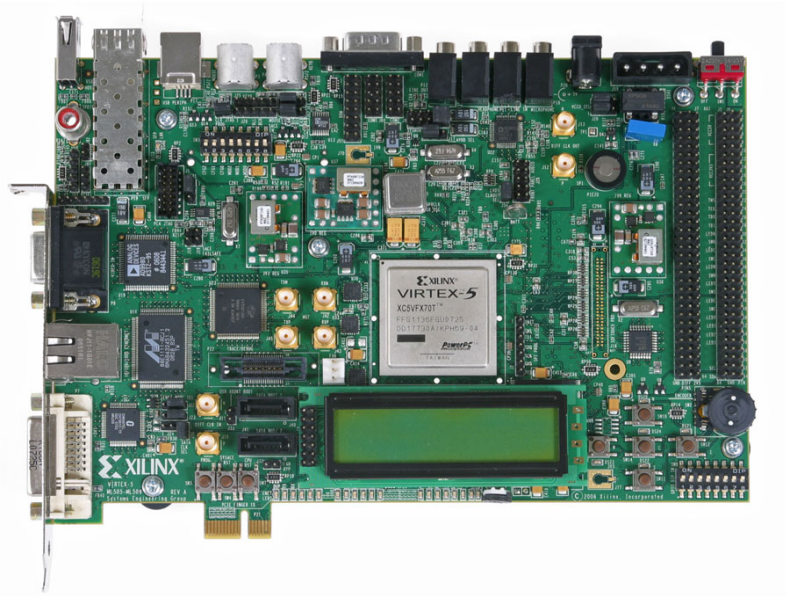

Figure 5.5. Xilinx ML507 Evaluation Platform.

Slices	11 200
Logic Cells	71 680
LB Flip-Flops	44 800
Total Block RAM	5 328 kbits
Digital Clock Managers	12

Table 5.1. FPGA Virtex 5 XC5VFX70T-1FFG1136.

5.2.2 Power Board

A special designed Power Board is used containing a full three-phase inverter bridge and two 12-bit ADCs, see figure 5.6.

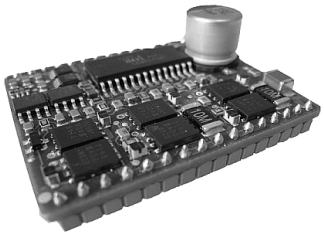

Figure 5.6. Single drive inverter board.

5.2.3 Brushless DC Motor

A small brushless DC motor, figure 5.7, with a relative encoder (see section 5.2.4) is used for testing the FOC algorithm. The motor parameters are specified in table 5.2.

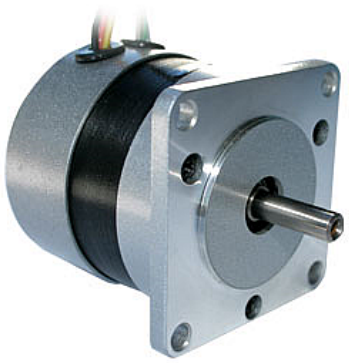

Figure 5.7. BLDC motor 57BLS01.

5.2.4 Encoder

The encoder is of relative type which means that it shows the relative angle and not the absolute. The connection to the resolver is made through a flat cable and the connections are shown in figure 5.8.

Supply voltage	36 V
Rated Speed	4000 rpm
Max peak current	6.8 A
Cont. torque at rated rpm	0.15 Nm
Resistance	1.6 Ω
Inductance line to line	4.3 mH
Torque constant	0.063 Nm/A
Back EMF constant	6.6 V/krpm
Time constant	2.7 ms

Table 5.2. Motor parameters for 57BLS01.

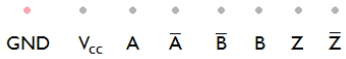

Figure 5.8. The flat cable connections of the encoder.

5.2.5 Experimental Setup

The experimental setup used for testing the hardware is shown in figure 5.9. A PC (referred as number 1 in the figure) is connected to the ML507 board (number 4) by a USB platform cable (number 2). Cables with crimp contacts are then connecting the power board (number 3) and the motor (number 5) with the ML507 board. An adjustable power source (number 7) is also used in the setup for complementing the power board's 5.0 V and 3.3 V output. For testing an oscilloscope (number 6) is used. The oscilloscope is a Yokogawa DL9710L Digital oscilloscope, 5 Gs/s, 1 GHz.

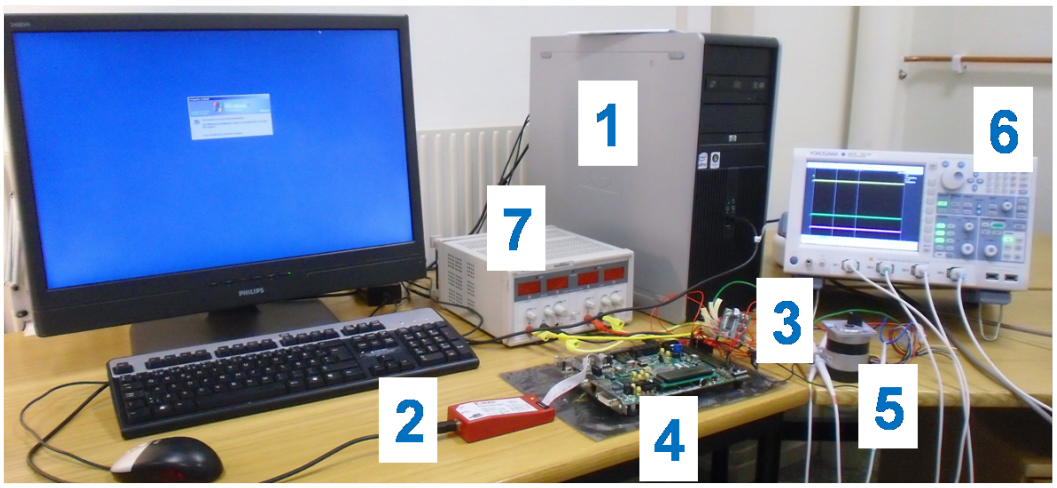

Figure 5.9. The hardware setup in the project.

Chapter 6

Solution

6.1 Implementation of the different parts of the FOC algorithm in Xilinx System Generator

For evaluating and trying the different available implementations explained in Section 5.1.2 the different parts of the Field Oriented Control algorithm are implemented with these different approaches. Some parts are implemented purely by blocks like for example adders, multipliers and registers, some parts by implementing m-code in a MCode Block and the rest with a Black Box approach. Figure 6.1 shows which parts are created in what way. Why the different blocks were implemented in just that way is merely a random choice.

To assure a correct behavior of the different parts in the FOC algorithm every sub block is created and tested in an appropriate test bench where the outputs from the discrete time Xilinx blocks are compared with the outputs from the continuous time Simulink model. An oscilloscope is also used for validation.

The obtained results are presented in chapter 7.

6.1.1 Implementation of the coordinate transformations

A test bench is built with Simulink blocks for comparison with the corresponding Xilinx blocks, figure 6.2 and 6.3 shows the implementation of the Park Transform. By putting both the models together and let them have the same stimuli, in this case a sine wave and a saw tooth wave corresponding to the angle θ, a test bench is created, see figure 6.4. The output from the test bench is shown in figure 6.5 where the similarity of the both models can be observed. The lower plot in figure 6.5 shows the quantization error that occurs because of that the Simulink blocks are of continuous time and the Xilinx blocks of discrete time. Why the error is growing with time is due to some phase delay and that the look up tables (LUTs) in the Xilinx blocks differs from the Simulink block's implementation for calculating cosine and sine of the angle θ.

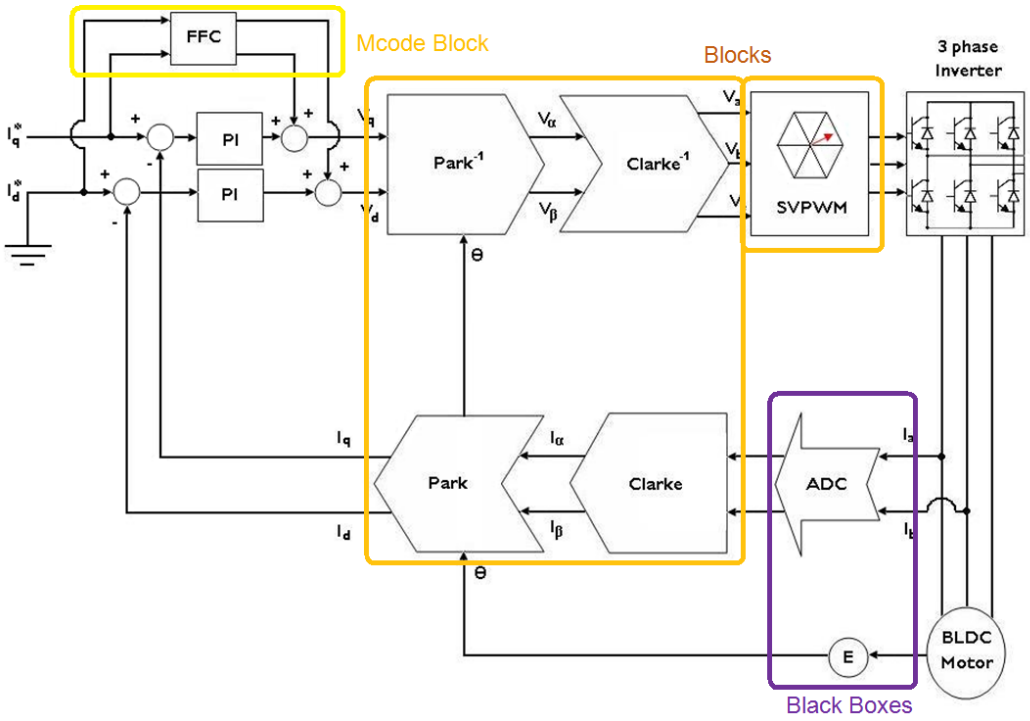

Figure 6.1. How the different parts were implemented.

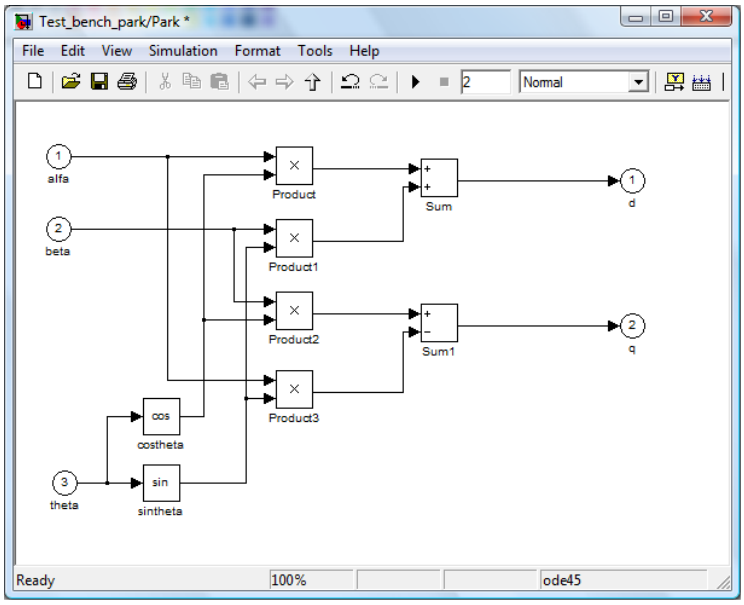

Figure 6.2. The Park Transform with Simulink blocks.

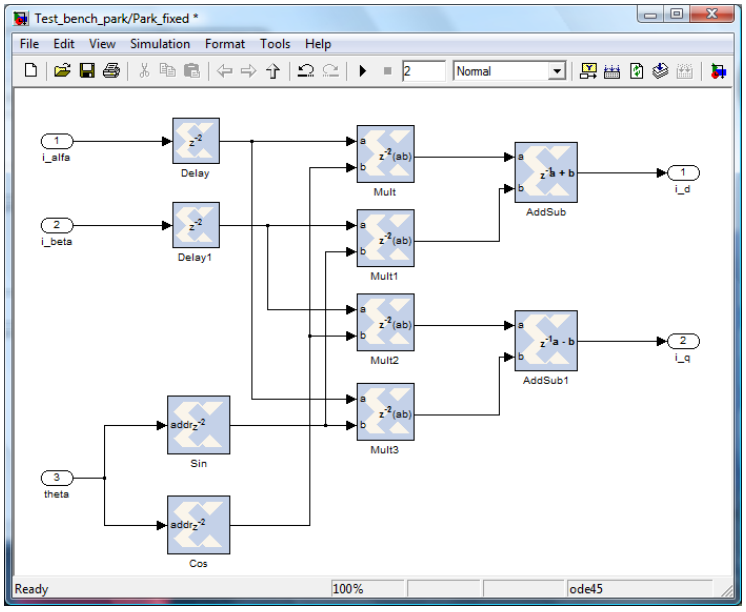

Figure 6.3. The Park transform with Xilinx blocks.

The other coordinate transformations, The Inverse Park, The Clarke and the Inverse Clarke are created and tested in the same way.

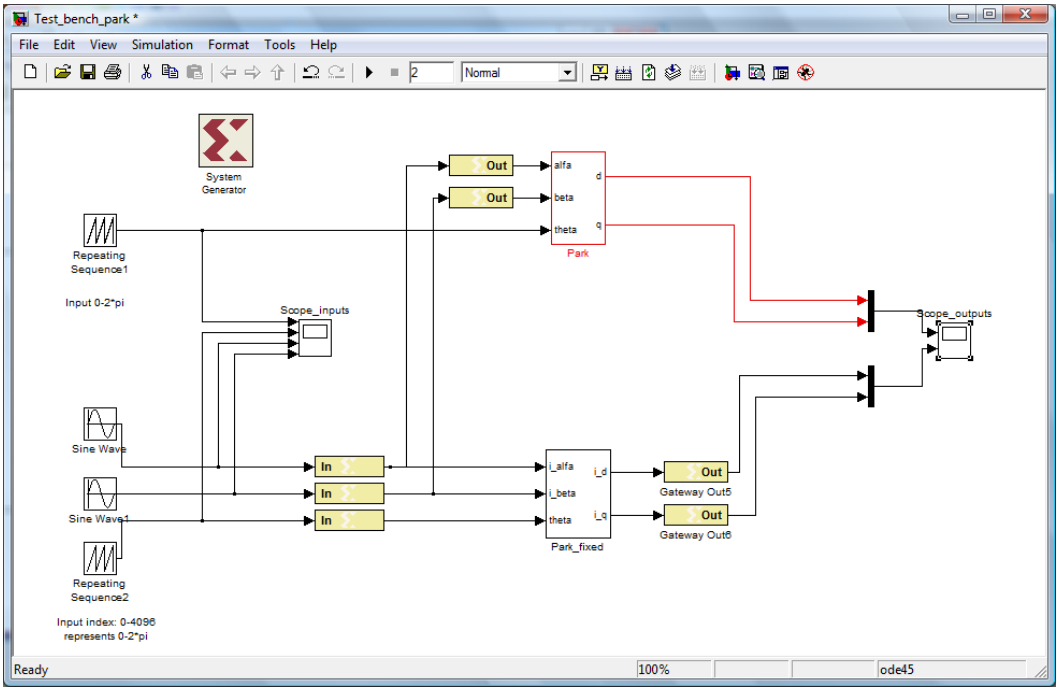

Figure 6.4. Test Bench for the Park transformation.

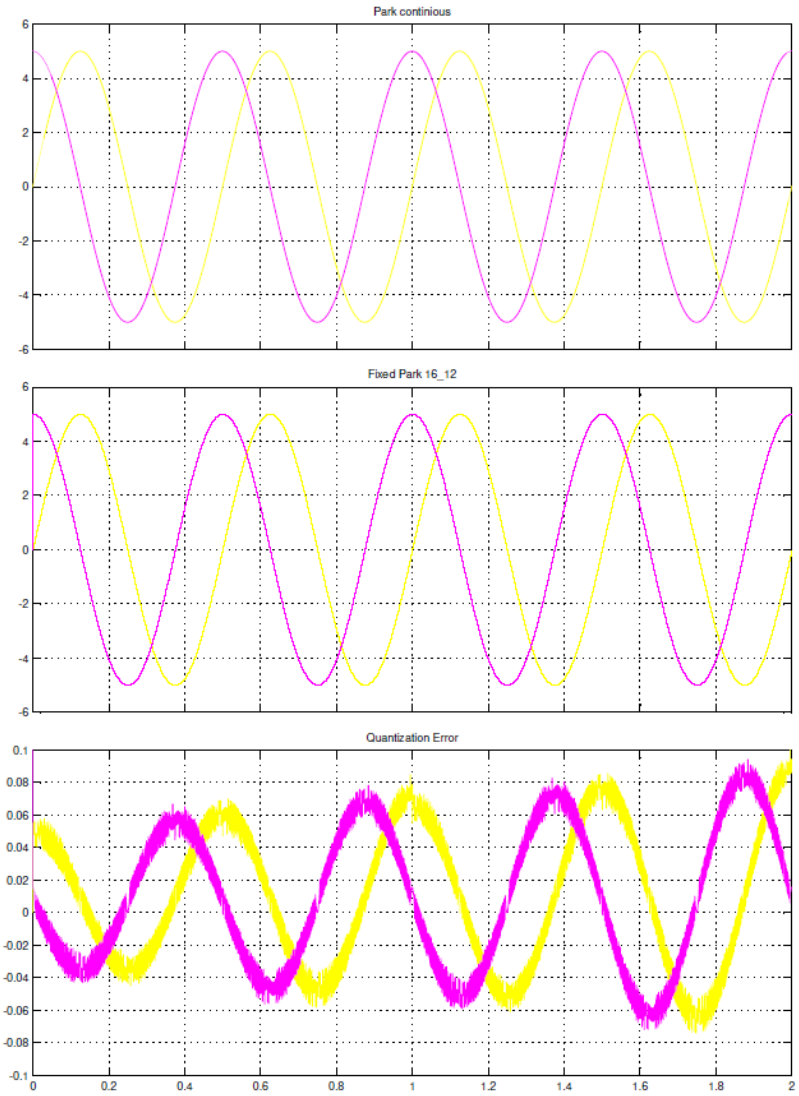

Figure 6.5. Output from the Park test bench. *Upper plot - Simulink (continuous), middle plot - Xilinx (Fixed point 16_13), lower plot - quantization error.*

6.1.2 Implementation of the PI regulators

The PI regulators for controlling the d and q currents are implemented with Xilinx blocks according to the characteristic discrete time equation for PI controllers, see equation 6.1. Here e is the error between the actual value and the desired value, T is the sample time, K_p the proportional gain and T_i the time constant for the integral term.

$$u(n) = K_p e(n) + \frac{K_p T}{T_i} \sum_{k=0}^{n} e(k) \tag{6.1}$$

In the implementation a saturation block is created and added for avoiding integral wind-up, see figure 6.6.

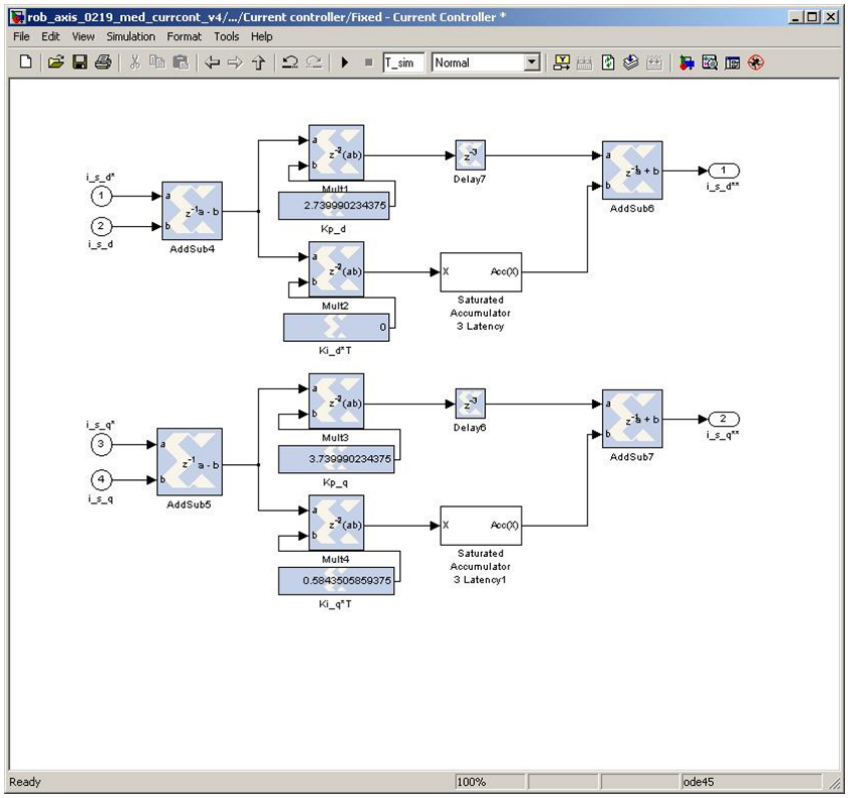

Figure 6.6. PI current controllers for the d and q currents.

6.1.3 Implementation of SPI ADC

The SPI ADC is implemented as a black box described in section 5.1.2. The functionality of the SPI ADC is written in VHDL code with some amendments for

correct implementation in the black box. When the black box has created a configuration m-file it has to be reviewed and complemented. The implementation is shown in figure 6.7. Special input and output blocks are created and mapped with its desired I/Os on the I/O header seen to the right on the ML507 board, shown in figure 5.5. This header contains 32 single-ended signal connections to the FPGA I/Os and permits the connector to carry high-speed signals set to 2.5 or 3.3 V.

The correct functionality of the block is validated through co-simulation via a JTAG connection, known as the concept Hardware In the Loop (HIL) or FPGA in the loop, explained in section 3.4. A co-simulation block is generated from the System Generator block and by that the operation of the SPI ADC is validated directly on hardware by an oscilloscope, see figure 6.8.

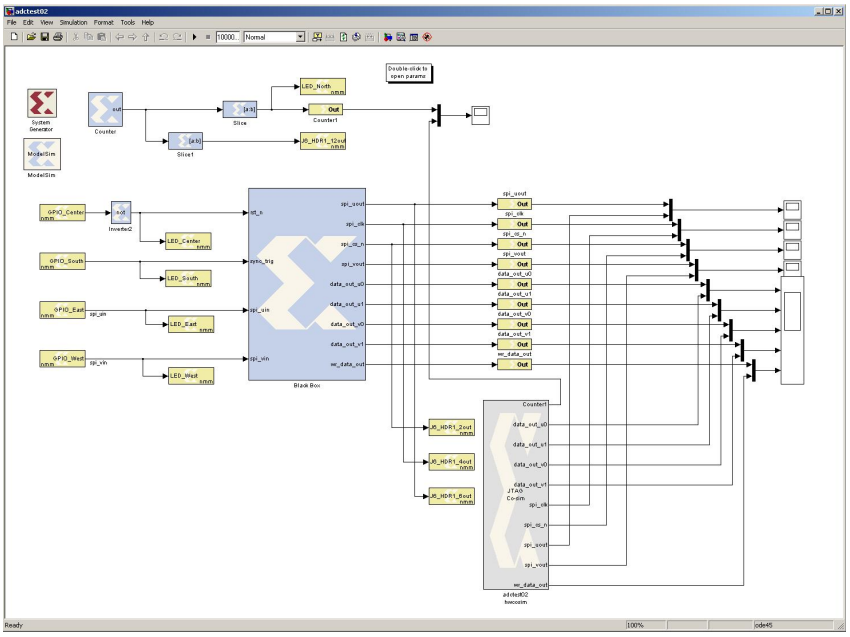

Figure 6.7. The SPI ADC created as a black box and the corresponding co-simulation block.

The output from the 12-bit ADC is measured by the scopes in Simulink and confirmed by equation 6.2. ADC_out is the output from the ADC represented by 12 bits, the resistors R_7 and R_8, the voltages V_{bus} and V_A are shown in figure 6.9.

$$\begin{cases} BUS\_SENSE = V_{bus} \cdot \frac{1}{R_7+R_8} \cdot R_8 \\ \\ ADC\_out = BUS\_SENSE \cdot \frac{4096}{V_A} \end{cases} \tag{6.2}$$

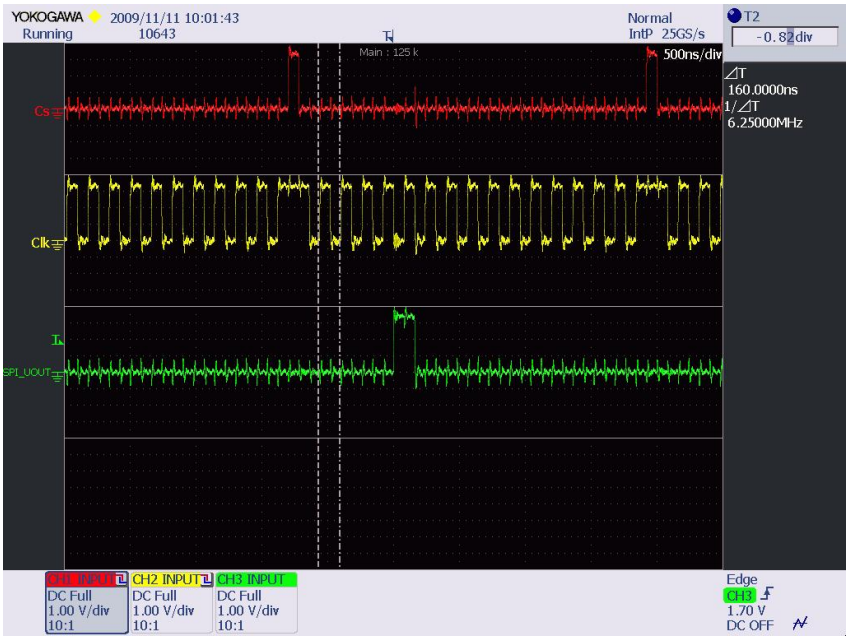

Figure 6.8. The output from the SPI shown on the oscilloscope during co-simulation. *Top graph - Chip select, middle graph - SPI clock, bottom graph - SPI out.*

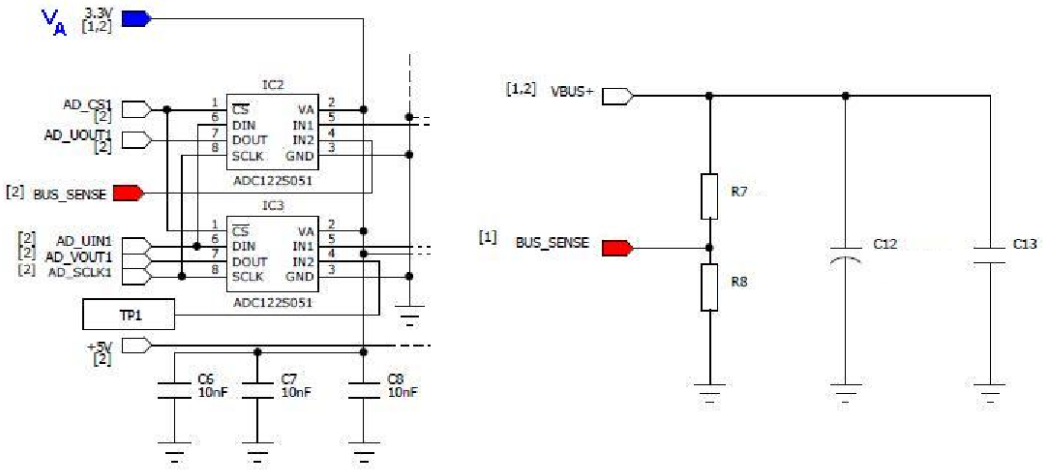

Figure 6.9. Schematic figure of the ADCs.

6.1.4 Implementation of Encoder

The encoder mounted on the motor is a relative quad encoder and its reading is implemented with a black box. First the VHDL code is written with the functionality for the encoder and then implemented in the black box, see figure 6.10. The outputs from the encoder enters via the I/O header as done with the SPI ADC in section 6.1.3. The black box is tested in hardware with a co-simulation block.

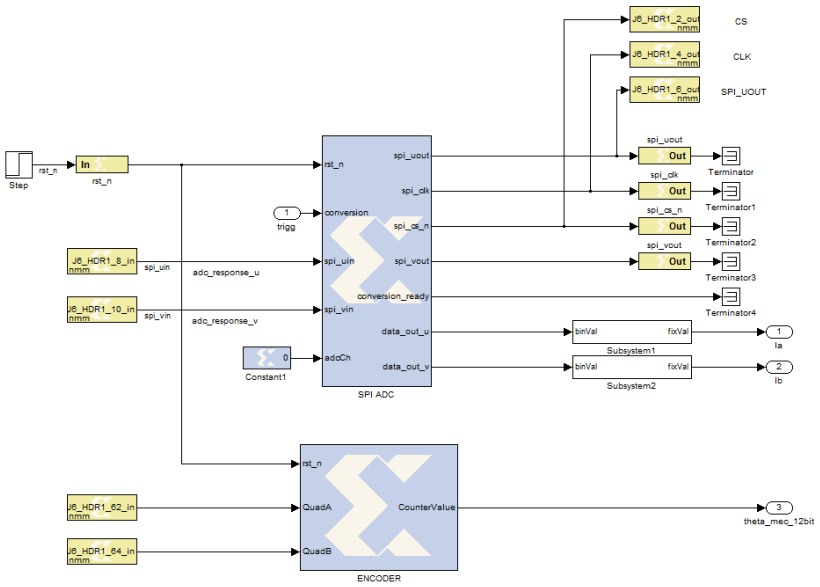

Figure 6.10. The lower block is the Black Box for the encoder.

6.1.5 Implementation of Feed forward control

The feed forward control is implemented in parallel with the PI controllers, the equations are shown in equation 6.3 where ψ_m is the linkage flux, L the inductance and R the resistance. The feed forward control is implemented both as a MCode Block and with separate building blocks for easier validation, see figure 6.11, 6.12 and 6.13.

$$\begin{cases} V_d = Ri_d + L\frac{d}{dt}i_d - Li_q\frac{d}{dt}\theta \\ V_q = Ri_q + L\frac{d}{dt}i_q + Li_d\frac{d}{dt}\theta + \psi_e\frac{d}{dt}\theta \end{cases} \tag{6.3}$$

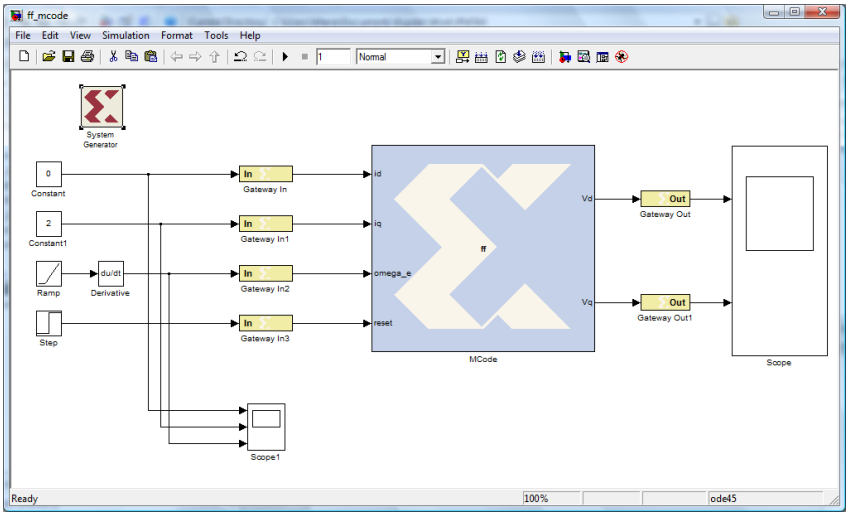

Figure 6.11. Feed forward control implemented as a MCode Block.

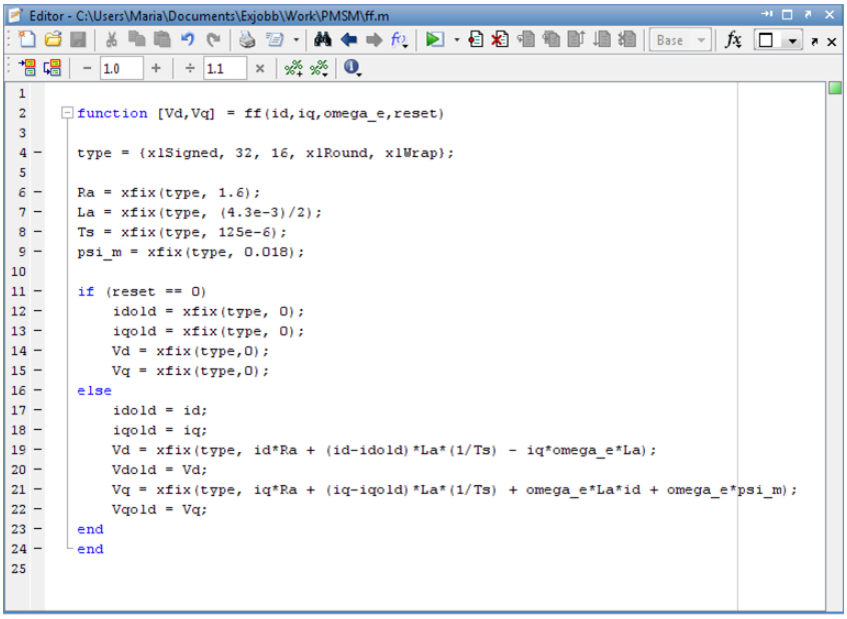

Figure 6.12. *m*-code for implementation of feed forward in a MCode Block.

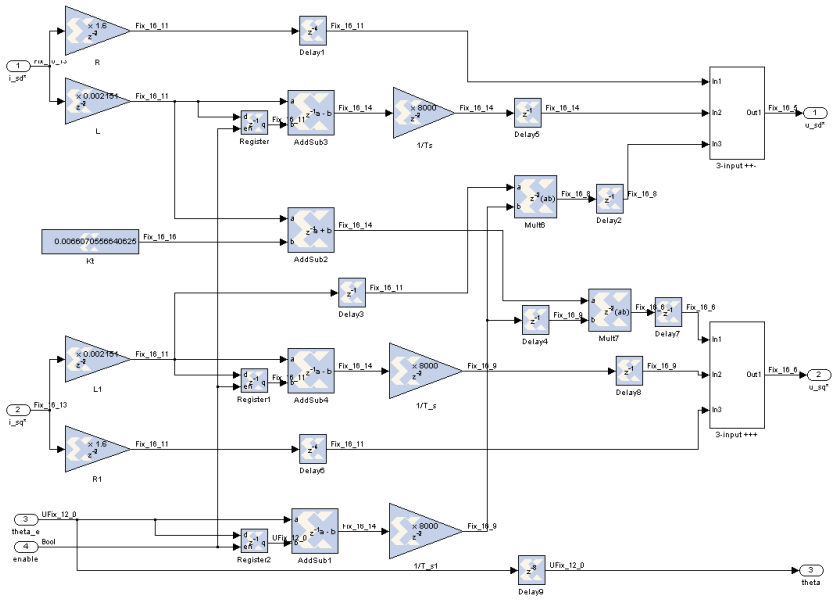

Figure 6.13. Feed forward control implemented with blocks.

6.1.6 Implementation of the FOC algorithm

When all the separate blocks in the FOC algorithm are constructed and tested they are used for implementing the actual Field Oriented Control algorithm, as described in section 4.1. The ready and created sub blocks are used for constituting a library for further use. The FOC design after putting everything together with all the sub blocks is shown in figure 6.14.

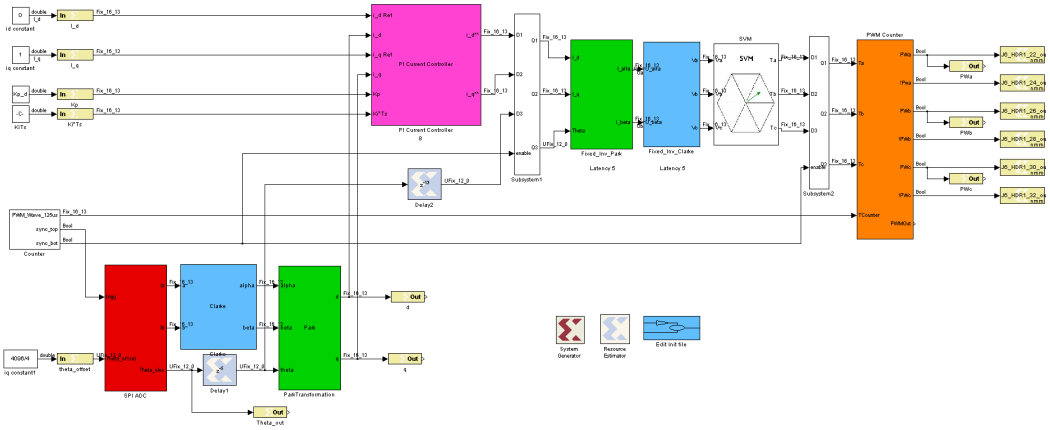

Figure 6.14. Implementation of the Field Oriented Control with Xilinx blocks.

The main issue with putting everything together is the timing analysis and the normalization. Area estimation is also performed for the whole design by using the Estimator block.

6.1.7 Xilinx blocks vs. MCode

To see the difference in utilized space on the FPGA between implementing a design, at one hand with blocks and on the other hand with a MCode Block, resource estimation is done for the two different approaches. The Clarke transform is chosen as a test object because of its not to complicated composition and two test benches are created, see figure 6.15 for the blocks and figure 6.16 for the *m*-code.

To implement the Clarke transformation in a MCode Block a MATLAB *m*-function is written. The bit representation has to be declared in the code and here the same representation is used as within the block design.

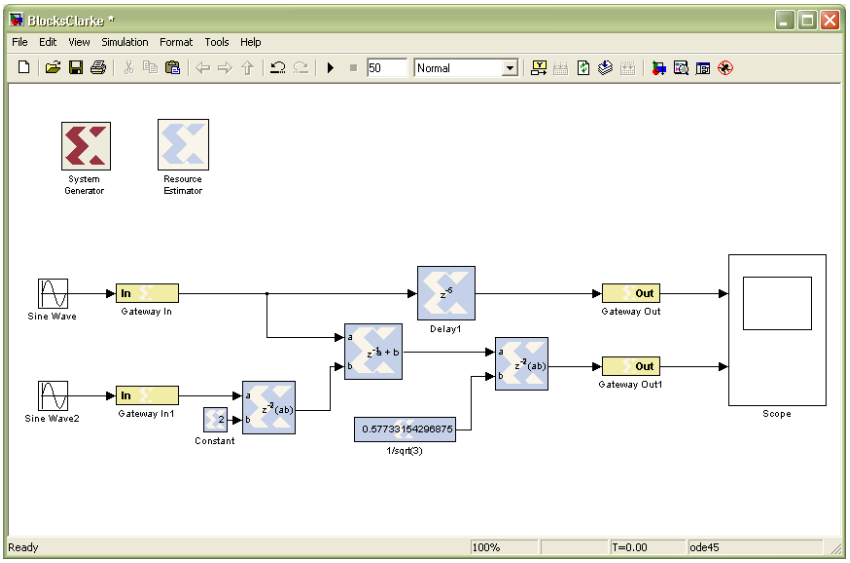

Figure 6.15. Test bench for area estimation of the Clarke transformation.

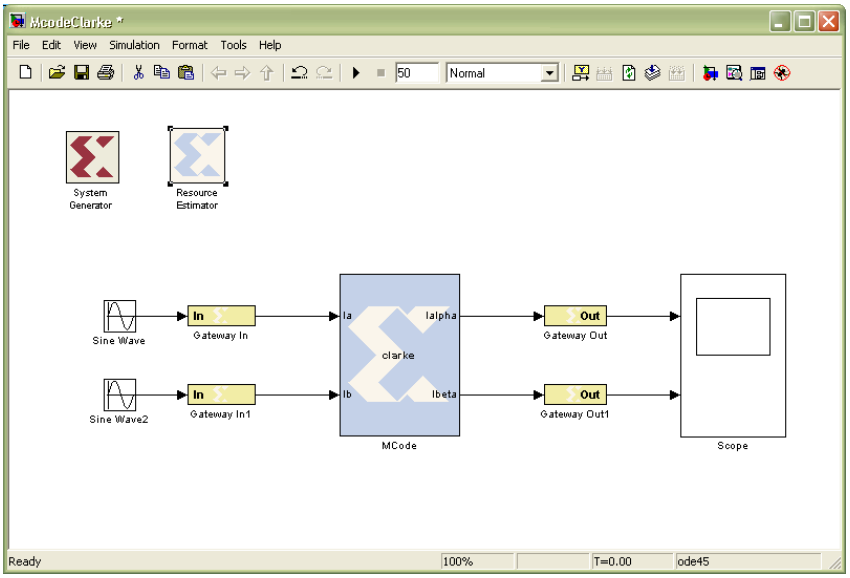

Figure 6.16. Test bench for area estimation of the Clarke transformation implemented with the MCode Block.

Chapter 7

Results

7.1 FOC algorithm

The result after synthesizing the whole Field Oriented Control algorithm including ADCs and encoder is shown in figure 7.1.

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	2,521	44,800	5%
Number of Slice LUTs	2,023	44,800	4%
Number used as logic	1,236	44,800	2%
Number used as Memory	743	13,120	5%
Number of occupied Slices	935	11,200	8%
Number of BlockRAM/FIFO	8	148	5%

Figure 7.1. Result after synthesizing the FOC algorithm.

Here it can be seen that 935 slices are used of the FPGA's 11200 slices which implies around 8% of its capacity. The total amount of clock cycles are about 50, which with an FPGA with a clock period of 5 ns implies a frequency of 4 MHz, see figure 7.2. In this project the maximum speed is not used because of other hardware limitations in for example the ADCs, therefore the design is clocked with a frequency of 8 kHz.

The possibility with the parallelism provided by the FPGA and by that its broad bandwidth makes it favorable to be used for implementing control of several motor axes.

In comparison with the previous mentioned paper [5] the total number of used slices was around 5600, which is a lot greater than the value obtained here. The total control loop in [5] takes 126 clock cycles which also is greater than the here obtained value. It has to be pointed out that the control algorithm used in [5] is not the FOC but it gives an indication that the obtained values are good.

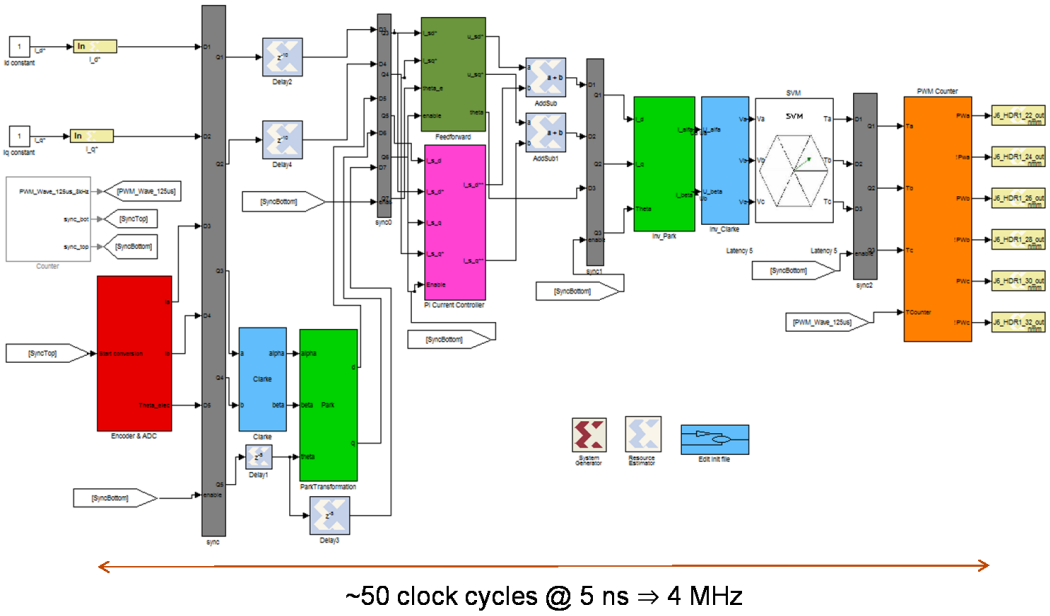

Figure 7.2. Demonstration of clock cycles.

7.2 Normal blocks vs. the MCode Block

The area estimation result for implementing the Clarke transformation by blocks like adders and multipliers versus implementing it with a MCode Block as described in section 6.1.7 is shown in figure 7.3.

The result shows that an implementation with blocks costs 33 slices while the implementation with a MCode block costs 10 slices. It is worth pointing out that this is just estimation.

7.3 Creating a library

A library is created and the constituting different sub blocks are shown in figure 7.4.

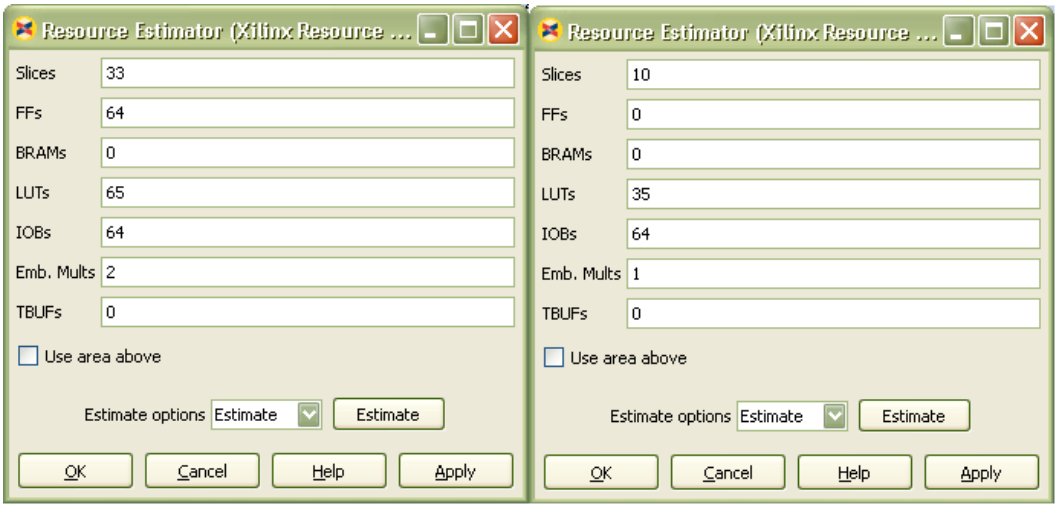

Figure 7.3. Area estimation result for Clarke transformation implemented by blocks to the left and with a MCode Block to the right.

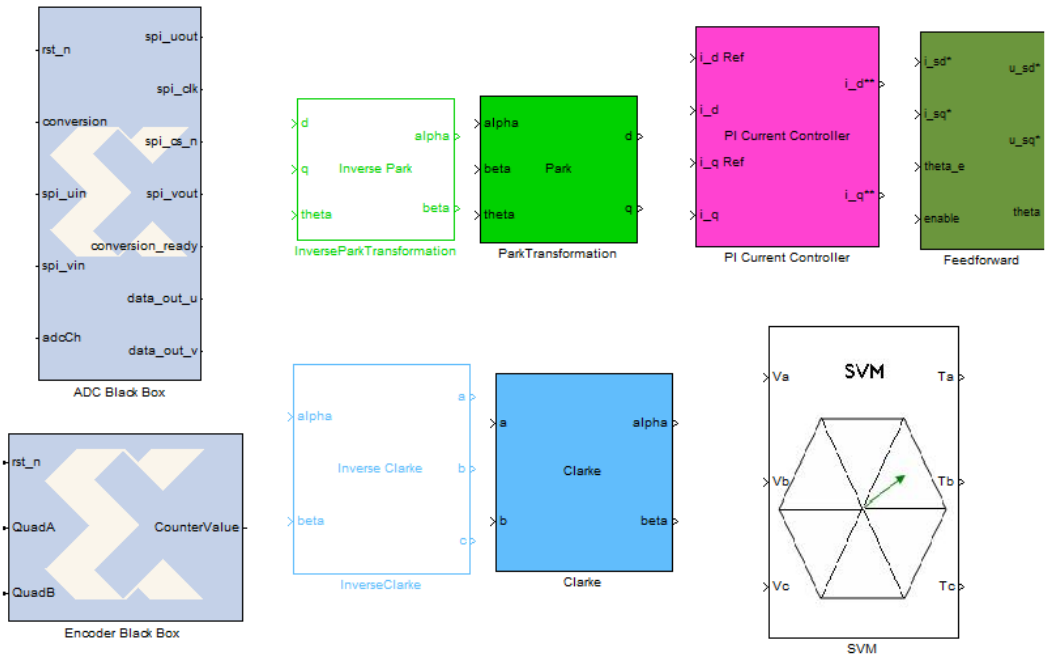

Figure 7.4. The different blocks created for the library.

Chapter 8

Discussion

8.1 HDL and RCP

The resource estimation that is done is just an approximation and why the MCode Block occupies just one third of the area occupied of the blocks is hard to say. This is a problem with automated code generation, the problem of not knowing how things are implemented in hardware. Especially if you are dedicated and experienced with low level programming and you know exactly how to implement it yourself with HDL code. It is worth mention that Rapid Control Prototyping and tools like this have a big advantage because you can work with both hardware and software even though you are not an expert in none of the two areas. What happens and how the design is implemented in hardware are maybe not of to much interest if you are gaining time, flexibility and get as good or quite similar results as with HDL coding.

8.2 Xilinx System Generator and LabVIEW

Xilinx System Generator can in comparison with for example LabVIEW be seen as a more flexible program. It has a lot of features and options but with that comes a higher complexity. Sometimes the program requires input and knowledge on too low level for being a high level program. When now having a library created the problem of that reduces in the future, at least for this type of implementation.

8.3 Lots of parameters

An issue that occurred during the project was that several parameters had to be set in each block and by clicking on each block for changing much time was wasted. The problem was solved by creating a *m*-file declaring variables that were set in the blocks.

8.4 Different time domains

When simulating the models some problems with the different time domains took place. This occurred because of that the FPGA has a corresponding time period in the region of nanoseconds and the motor in milliseconds. Synchronization had to be done and after working, testing and understanding the occurring problems it was not an issue anymore. This is a different type of complexity that programs like XSG has to solve for having the possibility to work in the two different regions at the same time with the same program.

Chapter 9

Conclusions

9.1 Runs from Simulink

Xilinx System Generator is a rather good program for Rapid Control Prototyping. The learning phase for the program is quite intense with all the features, options and choices in the blocks but when that threshold is reached the program has a lot of possibilities. The graphical interface is pleasant because it runs in the familiar Simulink environment. By comparing this with LabVIEW [3] it seems that XSG is harder to start with but when you are working with it and get used to it the flexibility is higher.

9.2 Easy implementation

The time saved by working with these type of high level programs is a real advantage. Also the fact that you do not have to be a hardware expert for implementing control algorithms in an FPGA gives a lot of possibilities. Of course one of the drawbacks by programming with blocks in general is that the code is not optimized and debugging in a design with a lot of blocks is relatively time consuming and an error is sometimes very hard to find. With normal written code an error can be seen more easily because it is not hidden in a block. This is although always a subjective thing because it depends on for example your proper experience.

9.3 Parallelism

The parallelism in an FPGA is a great advantage for algorithms and problems that can be implemented and solved in parallel. The parallelism and the speed of the FPGA lead to a higher bandwidth and that is why an FPGA is suitable for motor control with several axes like in for example robotic and mechatronic systems. The advantage of programming tasks in parallel instead of in series is that a change or replacement of a task can be done in an easier way without affecting other tasks.

Chapter 10

Future work

Possible extensions and future work is always an interesting part in this type of investigation projects. A natural continuation is to optimize the blocks and algorithm by area and time. The next step would be implementation of several motors. Possible extensions would be implementing other control algorithms and modulation techniques. In general:

- Optimize controllers and blocks.
- Optimize speed and area.
- Implement dead-time bands for SVPWM.
- Implement other modulation techniques.
- Implement interfaces to different measurement systems.
- Generic approach to motor control.
- Interfaces to an external control other than Simulink.
- Further implementation with several motors.
- Comparison between XSG generated code and hand written code respecting.
  - Time of implementation.
  - Area and speed.
- Implement other control algorithms.

Bibliography

[1] Xilinx system generator user guide. Website, December 2009. http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/sysgen_user.pdf.

[2] Xilinx ug347 ml505/ml506/ml507 evaluation platform user guide. Website, October 2009. http://www.xilinx.com/support/documentation/boards_and_kits/ug347.pdf.

[3] Magnus Fredrixon. Evaluation of a system for rapid control prototyping in the field of robot control. Technical report, ABB Corporate Research, Sweden, 2009.

[4] Eike Husing. Rapid control prototyping with labview fpga. Technical Report SECRC/AT/TR-2009/040, ABB Corporate Research, Sweden, 2009.

[5] Stéphane Simard, Rachid Beguenane, and Jean-Gabriel Mailloux. Performance evaluation of rotor flux-oriented control on fpga for advanced ac drives. *Journal of Robotics and Mechatronics*, 21(1):113–120, 2009. http://www.fujipress.jp/finder/xslt.php?mode=present&inputfile=ROBOT002100010014.xml.