TECHNISCHE UNIVERSITÄT MÜNCHEN

Lehrstuhl für Elektrische Antriebssysteme und Leistungselektronik

FPGA-Based Nonlinear Model Predictive Control of Electric Drives

Saeid Saeidi

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor–Ingenieurs

genehmigten Dissertation.

Vorsitzender:		Univ.–Prof. Dr.–Ing. Hans-Georg Herzog
Prüfer der Dissertation:	1.	Univ.–Prof. Dr.–Ing. Ralph Kennel
	2.	Prof. Dr. Eric Monmasson Université de Cergy-Pontoise/Frankreich

Die Dissertation wurde am 12.01.2015 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 17.05.2015 angenommen.

Abstract

The thesis focuses on the development of MPC algorithms for FPGA-based control of AC electric drives and power electronics. Two algorithms are investigated: Continuous Set Nonlinear Model Predictive Control (CS-NMPC) and Model Predictive Control with Imposed Target Dynamic (MPC-ITD). CS-NMPC relies on modulation whereas MPC-ITD allows to directly control power electronic converters. The proposed algorithms enable the use of multi-parametric nonlinear optimization and improve dynamic as well as steady state performance of the controller.

For practical implementation of algorithms, an FPGA-based platform is proposed which allows to expedite development and verification of control algorithms. Experimental results demonstrate high performance of the control schemes and confirm theoretical investigations.

Kurzzusammenfassung

Die vorliegende Arbeit befasst sich mit der Entwicklung von modellprädiktiven Algorithmen für mittels FPGAs geregelte Drehstromantriebe und Leistungselektronik. Hierbei werden zwei Algorithmen untersucht: die kontinuierliche nichtlineare modellprädiktive Regelung (CS-NMPC) und die modellprädiktive Regelung mit Vorgabe der Zieldynamik (MPC-ITD). Während CS-NMPC ein Modulationsverfahren benutzt, erlaubt MPC-ITD eine direkte Ansteuerung der leistungselektronischen Schaltelemente. Die vorgeschlagenen Algorithmen ermöglichen die Behandlung nichtlinearer Modelle und führen zu einer verbesserten Regelgüte im dynamischen sowie stationären Betrieb.

Zur praktischen Implementierung der Algorithmen wurde zudem eine FPGA-Plattform entwickelt, mithilfe derer Steueralgorithmen zeiteffizient entworfen und verifiziert werden können. Experimentelle Ergebnisse belegen weitreichenden Möglichkeiten der Regelungsverfahren und bestätigen die theoretischen Untersuchungen.

Acknowledgements

Completing this doctoral thesis could not be possible without the help and support of many people to whom I would like to express my deepest gratitude.

Above all, I do appreciate support and contributions of my supervisor, Professor Ralph Kennel. He gave me this tremendous opportunity to do my doctoral research at the institute for Electrical Drive Systems and Power Electronics. During the time I have been working at the institute, he has been of great help to me.

I would like also to acknowledge support of Professor Hans-Georg Herzog in helping me to evaluate my contributions. I am grateful for fruitful discussions with him and the time which he devoted to reviewing my reports.

I am thankful for the financial support of DAAD. Furthermore, I am grateful to the supportive staff of DAAD, the TUM International Office as well as the Graduate School.

I would like to thank my colleagues at the institute for Electrical Drive Systems and Power Electronics for their kindness, friendship as well as their permanent enthusiasm for the scientific discussions. In particular I would like to thank Mr. Julien Cordier and Mr. Daniel Glose, with whom I had great time working in the same office.

I am grateful to Mr. Schuster and Mr. Kopetschny for their assistance in the laboratory. Furthermore, I would appreciate coordination supports of Mrs. Daniela Dietmaier and Mrs. Julia Menz.

Last, but by no means least, I thank my family for their love and patience.

Contents

List of Figures

Abbreviations

MPC	**M**odel **P**redictive **C**ontrol
NMPC	**N**onlinear **M**odel **P**redictive **C**ontrol
FS-MPC	**F**inite **S**et **M**odel **P**redictive **C**ontrol
ITD	**I**mposed **T**arget **D**ynamic
MIMO	**M**ultiple **I**nput **M**ultiple **O**utput (system)
SISO	**S**ingle **I**nput **S**ingle **O**utput (system)
LTI	**L**inear **T**ime **I**nvariant (system)
PID	**P**roportional **I**ntegral **D**erivative (controller)
SMC	**S**liding **M**ode **C**ontrol
GPC	**G**eneralized **P**redictive **C**ontrol
AC	**A**ternative **C**urrent
DC	**D**irect **C**urrent
PWM	**P**ulse **W**idth **M**odulation
SVM	**S**pace **V**ector **M**odulation
IM	**I**nduction **M**otor
PMSM	**P**ermanent **M**agnet **S**ynchronous **M**otor
BLDC	**B**rush **L**ess **DC** (motor)
CPU	**C**entral **P**rocessing **U**nit
RAM	**R**andom **A**ccess **M**emory
DSP	**D**igital **S**ignal **P**rocessor
PCI	**P**eripheral **C**omponent **I**nterconnect
PCIe	**P**eripheral **C**omponent **I**nterconnect **E**xpress
UDP	**U**ser **D**atagram **P**rotocol
FPGA	**F**ield **P**rogramable **G**ate **A**rrays
VHDL	**V**ery high speed integrated circuit **H**ardware **D**escription **L**anguage

RTL	**R**egister **T**ransfer **L**evel
TTL	**T**ransistor **T**ransistor **L**ogic
ADC	**A**nalogue **D**igital **C**onverter
DAC	**D**igital **A**nalogue **C**onverter
IGBT	**I**solated **G**ate **B**ipolar **T**ransistor

Symbols

$\boldsymbol{v}_s$, $\boldsymbol{v}_r$	stator and rotor voltage vectors	V
$\boldsymbol{i}_s$, $\boldsymbol{i}_r$	stator and rotor current vectors	A
$\boldsymbol{\psi}_s$ $\boldsymbol{\psi}_r$	stator and rotor flux vectors	Wb
θ_s, θ_r	stator and rotor flux angles	rad
T_e	electromagnetic torque	Nm
T_l	load torque	Nm
J_m	moment of inertia	$kg \cdot m^2$
P	number of pole pairs	$-$
I	current magnitude	A
i_α, i_β	current components in $\alpha\beta$ coordinate	A
i_d, i_q	current components in dq coordinate	A
v_α, v_β	voltage components in $\alpha\beta$ coordinate	V
v_d, v_q	voltage components in dq coordinate	V
V_{DC}	DC link voltage	V
T_s	sampling time	s
τ_s, τ_r	stator and rotor time constants	s
τ_m	mechanical time constant	s
L_s, l_s	stator inductance	H
L_m, l_m	mutual inductance	H
L_r, l_r	rotor inductance	H
R_s, r_s	stator resistance	Ohm
R_r, r_r	rotor resistance	Ohm
ω_r, ω_e	mechanical and electrical angular frequency	$rads^{-1}$
J	cost function	$-$

To my parents

Part I

Background

Chapter 1

Introduction

Electrical Machines and Energy Systems

Compared to the traditional DC motors which were widely applied for producing mechanical energy in automation industry, transportation and etc, AC electrical motors such as Induction Motors (IM) and Permanent Magnet Synchronous Motors (PMSM) have higher efficiency and power/size ratio. AC motors do not require mechanical commutations. Elimination of mechanical commutation leads to a simple physical construction and robust operation. Therefore AC motors need less maintenance and show higher reliability.

However, mathematical modeling of AC motors is nonlinear Multiple-Input Multiple-Output (MIMO). It makes control and optimization more challenging. Furthermore, for practical implementation of control algorithms, more computational performance is necessary. These challenges rise interests of industrial as well as academic research communities to investigate new ideas and methods in order to find solutions delivering higher performance, improving reliability at the same being simple and easier to be realized.

Advances In Power Electronic Components

Power electronic converters, such as DC-AC inverters, provide high performance and controllable electrical energy. By invention of power transistors, it is possible to control flow of electrical power with higher dynamic, better efficiency and at lower cost. Using power electronic components for feeding electrical motors, however, increases number of controlled parameters and optimization considerations.

Microelectronic and Digital Computation

Power electronic itself could not be so beneficial for the electrical drive technology without microelectronic based digital computational devices. Digital components, such as microcontrollers and programmable logic devices are the essential parts in any digital control systems. Advances in the microelectronic technology allow to realize high performance control strategies.

In absence of digital computers, analog devices had been utilized for implementation of the control system. In theory analog devices are faster and show higher bandwidth. However, they are restricted in dealing with the complexity of control algorithms. Aging and change of parameters are further drawbacks of the analog controllers.

In almost every modern electrical drive, there is a computation core that takes care of a diversity of tasks in real-time. Control of dynamic phenomenas, is the most important part which makes the drive fulfill technological requirements. Digitalizing of the control system has many advantages such as more intelligent human interface, easiness of fault recording and diagnostics. To keep up with the demand of having more reliable and flexible systems, advanced control algorithms are needed to fulfill tasks which sometimes are contradictory, in an optimal way. Model Predictive Control (MPC) is perhaps the best example in this regard. It is established on digital measurement and discrete control. Furthermore, MPC enables to deal with multi-objective optimization problems.

Filed Programmable Gate Arrays

In conventional microcontrollers, the hardware architecture is fixed. Computational tasks are first converted into elementary instructions which can be sequentially executed by the Central Processing Unit (CPU). In contrast to that, Field Programmable Gate Arrays (FPGA) are hardware configurable and the computational architecture can be designed according to any specific applications. Adaptation of the architecture to the algorithm, makes it possible to improve computational performance at the same clock frequency. Increasing logic density and clock frequency of FPGAs allow to realize many high performance digital systems at lower cost. Nevertheless, main challenges of utilizing FPGAs, are the design complexity and longer development time.

Computer Aided Design Tools

Due to rapid change of technologies and high competition between manufactures, innovation and time to market are of significant importance for being successful. Computer

Aided Design (CAD) tools expedite design and development of electromechanical systems. They are widely used to improve quality of products and shorten the development time.

For a control system, the computer simulation simplifies early correction of errors and reduces the design iteration time. A popular simulation program which is widely used in academic as well as industrial communities, is MATLAB/Simulink. The complexity of control algorithms is significantly decreased by using the graphical design environment of Simulink.

Expanding application of electrical drives and rapidly advancing drive related technologies require a comprehensive investigation of various fields for solving practical problems.

1.1 AC Electrical Motors

Electrical motors are clean, controllable and efficient source of mechanical energy. Thanks to the elimination of mechanical commutation which is the main drawback of conventional DC motors, AC motors present a better operation performance with higher power ratio and require less maintenance.

Modeling of Electrical Motors

Mathematical modeling of electrical drives is essential for designing the control system. Over the last century several methods have been developed for simplification of modeling of electrical drives.

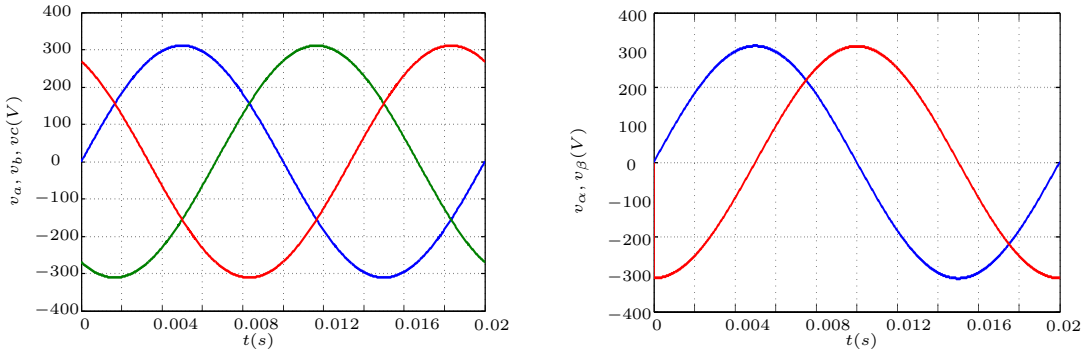

FIGURE 1.1: A three-phase and equivalent two-phase voltage source

In 3-phase electrical motors stator windings are placed in 120 degree from each other making a symmetrical 3-phase system. From physical and construction point of view,

3-phase systems have several advantages. However, since only two independent components are sufficient to represent a two-dimensional space vector, it is more convenient to transform 3-phase variables into a 2-phase equivalent model [4]. Clark transformation allows to represent space vectors in a 2-phase system. In this dissertation it is relied on the 2-phase equivalent model of electrical motors (see figure 1.2).

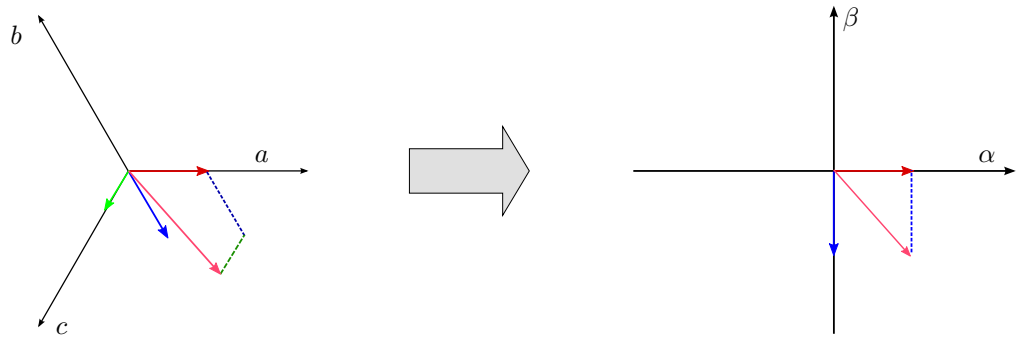

FIGURE 1.2: Transformation from 3-phase to 2-phase system

Coordinate Transformation

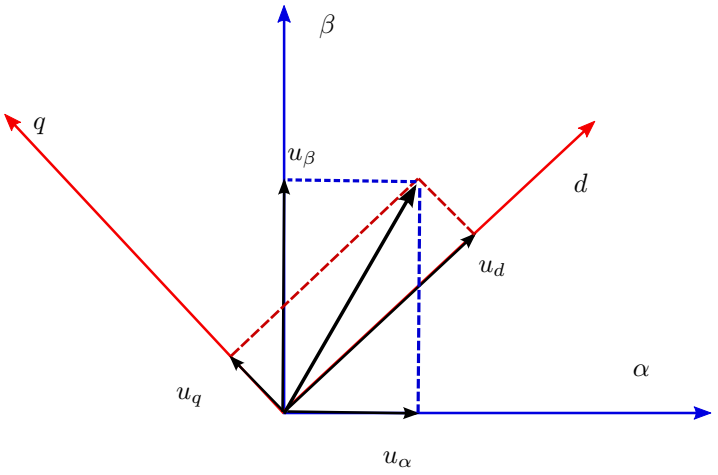

FIGURE 1.3: Space vector representation of Park transformation

The electromagnetic torque is created by interacting rotor and stator flux rotating around the air-gap. A revolutionary contribution to the modeling and control of AC electrical motors is done by the concept of coordinate transformation usually referred to as Park transformation [5]. It enables to change the reference frame from the stator to a rotating reference frame in which electromechanical parameters of AC motors can be represented as DC components. It simplifies the complicated model of AC machines similar to the conventional DC machines.

As figure 1.3 shows, to transform the vector components into the new coordinate system Park transformation is applied by means of the following equations:

$$u_d = u_\alpha cos(\theta) + u_\beta sin(\theta) \tag{1.1}$$

$$u_q = u_\alpha sin(\theta) + u_\beta cos(\theta) \tag{1.2}$$

The so-called Field Oriented Control (FOC) is established on the idea of coordinate transformation [6]. Furthermore, analysis of other control strategies is, more often, done in the synchronous reference frame. It is not only important for the simplification and better understanding of the operation principles of AC drives but also it is a suitable tool for design of the controller and optimization [7, 8].

1.2 Electronic Based Power Converters

The semiconductor technology is base of power electronic devices which are applied for power conversion ($AC \longleftrightarrow DC$) or in general to control the flow of electrical energy. The simple element widely used in rectifiers is diode. It is a switch which can be controlled by polarity of the voltage between its two terminals. When the voltage polarity is positive, it can be considered as a short circuit whereas for negative voltage it is open circuit. Therefore it can be used for rectification of AC current [9].

Another basic power electronic component is transistor. A transistor can conduct electrical current in one direction while it is turned on by the gate signal and its terminal voltage is larger than a minimum value.

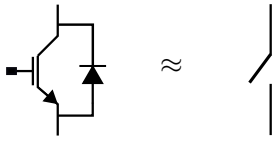

FIGURE 1.4: A transistor switch with anti-parallel diode

These two components are basic elements in most power electronic devices. Figure 1.5 shows a 2-level Voltage Source Inverter (VSI) which produces controllable (variable frequency and magnitude) 3-phase voltage, out of a constant DC source. The inverter is connected to a 3-phase sinusoidal voltage source, through a diode bridge rectifier. The diodes are turned on and off automatically by the potential between their terminals. This operation principle allows rectifying the input currents. To smooth the output voltage of the rectifier a capacitive energy storage is utilized. The DC voltage of the capacitor is converted into a 3-phase controllable voltage by the VSI.

There are several types of transistors differing from their physical construction. Isolated Gate Bipolar Transistors (IGBT) are capable of switching at higher current and voltage, satisfying a wide range of applications from low power to megawatt power range.

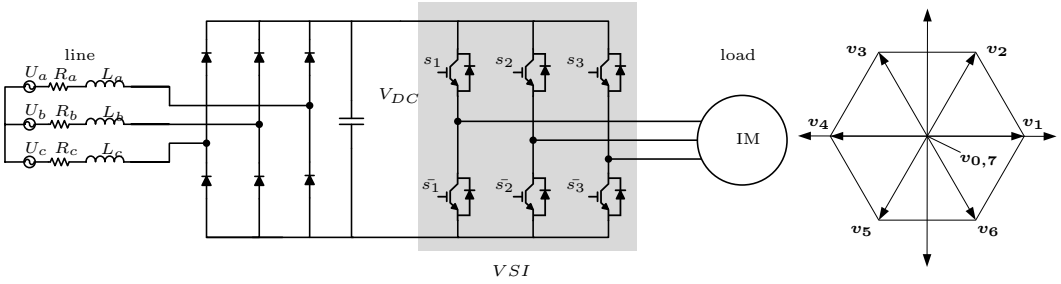

FIGURE 1.5: Schematic of a typical three phase 2-level VSI.

1.3 Control Strategies for Electrical Drives and Power Electronics

The controller is the most important part of the electrical drives. Without proper control, these are not capable of fulfilling useful tasks. Dynamic performance and energy efficiency are major quality factors for the control system of electrical drives. Optimality criteria may vary depending on the operation conditions [8]. Over the last three decades several control schemes have been investigated for the electrical drives [6]. Nevertheless, there are mainly two well known control methods which have received higher industrial acceptance:

1. Field Oriented Control (FOC)
2. Direct Torque Control (DTC)

FOC is established on linear control theory whereas DTC can be considered as a nonlinear controller. However, main difference between them is that FOC relies on modulation to control the inverter [10, 11] while DTC utilizes a direct switching strategy [12].

Field Oriented Control

Before the era of digital controllers, dominating type of electrical motors widely applied for demanding applications, used to be so-called *DC* motors. The reason is the simplicity of the control design for this type of electrical drives. An important feature of *DC* motors

making the control system straightforward is that in these motors the rotor magnetic field and the stator current vector are mechanically kept in a fixed angle. It leads to decoupling of the field and torque producing current. Therefore, the stator field and electromagnetic torque can be controlled separately. To establish the same model for AC electrical drives, the field orientated control (FOC) concept has been developed.

FOC relies on a mathematical transformation of flux and stator current vectors into a new coordinate system rotating at the same speed as the rotor field. Since in steady state operation stator current vector and rotor field are rotating at the same frequency, these components can be treated as *DC* values and controlled just like in conventional *DC* motors. As this transformation must be repeated at each sampling time, it requires higher computational performance compared to the rather simple control system of the DC motors. Nevertheless this is not a limit any more since digital controllers are capable of doing such computations at reasonable cost today.

Cascade FOC

The cascade control concept relies on the physical notion of the electrical drives in which the control parameters have different time constants. Electrical parameters such as current and flux, have relatively smaller time constants compared to mechanical variables such as speed and position. In cascade control the controller is designed in separate cascaded loops [13].

The complexity of the mathematical model of the induction motor drive including the inverter makes the design of the current controllers more challenging. The current components can be controlled in both stationary reference frame or in rotor flux oriented coordinate system [14]. Nevertheless, implementation of linear controllers shows better performance in rotor flux oriented reference frame since current components become *DC* variables at steady state condition. Another advantage of coordinate transformation is reduction of the computational complexity.

Figure 1.6 shows the structure of a cascade control system of an induction motor drive. FOC technique in cascade structure is widely applied for industry as a standard control strategy. For design of the controllers, many researches have been carried out over the last 3 decades. Most studied controller type is PID. However there are also other strategies investigated for realization of the current controllers. Performance of the controller has been evaluated from many aspects. In [15] a state feedback with feed forward technique is proposed to improve the performance.

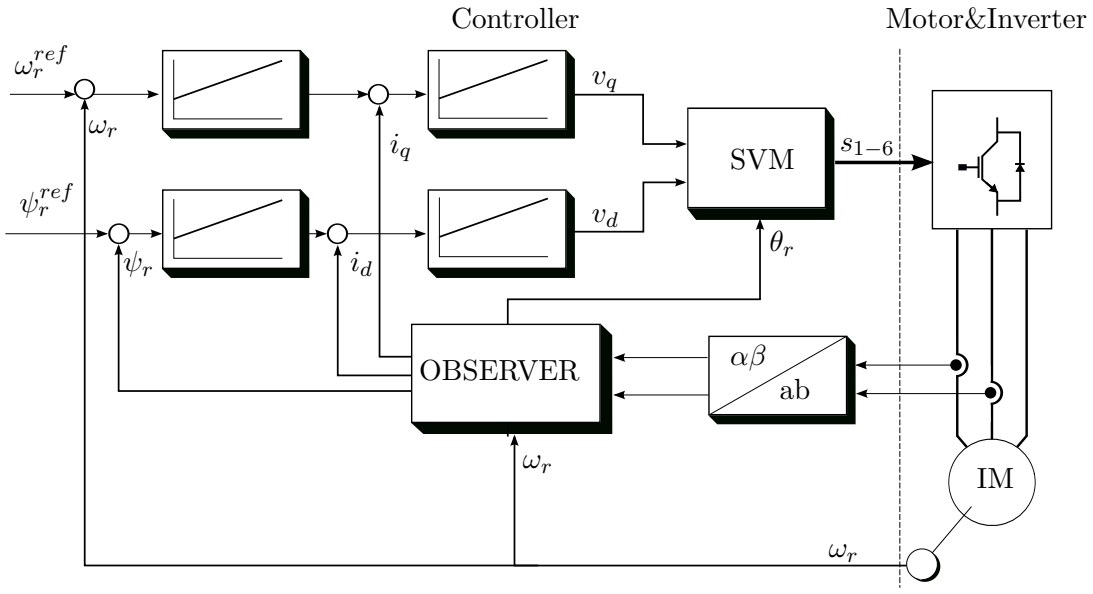

FIGURE 1.6: Cascade structure of FOC for an induction motor

Another important feature of the current controller is robustness to parameter variation and change of the operation point. Although electrical drives have relatively well understood mathematical model, it is difficult to estimate exact parameters. Thus parameter adaptation and robustness are important to be studied [16].

Direct Torque Control

In FOC, flux and torque of IM are controlled through respective stator current components. To control stator current vector, the controller produces a reference value for the stator voltage vector. The desired voltage vector is applied through the inverter by means of a PWM technique [17]. Direct Torque Control (DTC) is a conceptually different method that controls the flux and torque directly by changing the switching state of the inverter [12, 18]. Schematic of a typical DTC for an IM is shown on 1.7.

The word *Direct* refers to control of the inverter without PWM. In this respect, direct torque control methods that rely on PWM to generate the voltage vector [19], in fact, can not be considered as DTC strategy.

Although the DTC concept was initially proposed for control of induction motors fed by 2-level inverters, it can be extended for control of other electrical variables such as active (reactive) power or for other converter topologies [20].

In DTC, controlled variables are kept within *hysteresis bands* by choosing a proper switching pattern of the inverter. Due to that, the switching frequency is variable depending mainly on the hysteresis bands and operation points. Since in DTC the

sampling frequency is much higher than the average switching frequency, the inverter time constant is shorter than the PWM-based methods. In addition, the hysteresis controller is faster than linear controllers which mostly are applied in FOC. Both these characteristics, lead to higher dynamic performance of DTC [21].

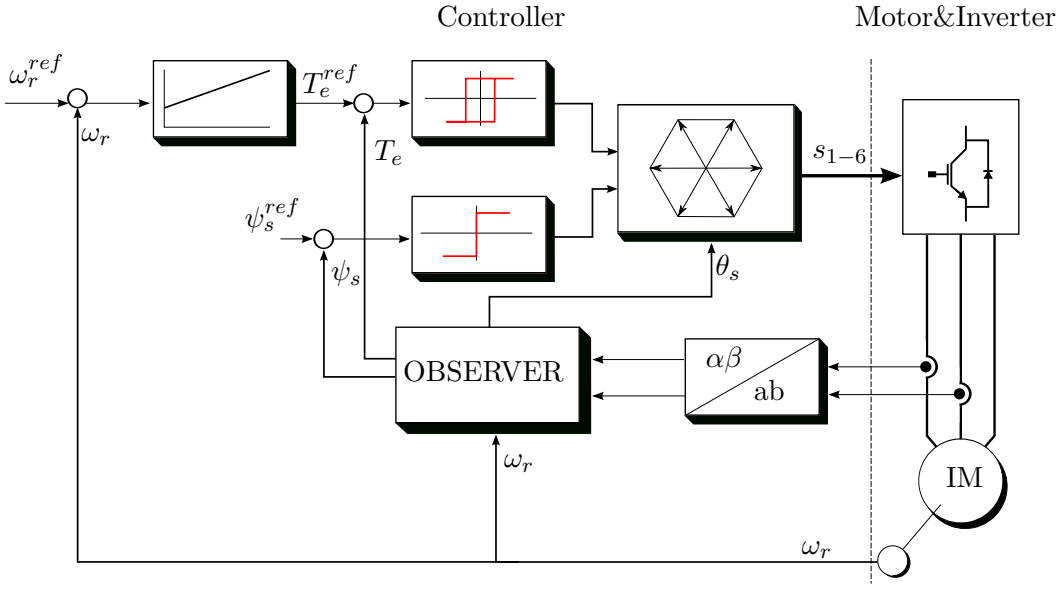

FIGURE 1.7: Cascade structure of DTC for an induction motor

In DTC, a switching takes place when flux or torque errors exceed the hysteresis bands. It allows to control the switching frequency dynamically by changing the hysteresis bands. Therefore DTC is superior to PWM-based methods for higher power drives.

Nevertheless, due to variable switching frequency in DTC, the torque and current ripples are distributed in frequency spectrum and cause acoustic noises. It is the major issue of DTC that excited many research efforts.

Another shortcoming of DTC is so-called dead-lock when no feasible switching possibility exists to keep the controlled variables within their respective bands [22]. It happens while the controller is intended to reach the maximum available torque mainly in the speed weakening region.

The DTC technique, in vast majority of cases, is studied for 2-level inverters. For more complex topologies, such as multilevel inverters, using simple hysteresis bands is not sufficient to solve optimization problem and extract a Look-Up Table (LUT) for switching [23].

1.4 Contributions of this Thesis

This thesis investigates FPGA-based algorithms for nonlinear model predictive control of AC electrical drives and power electronics. Two control strategies are proposed: Continuous Set Nonlinear Model Predictive Control (CS-NMPC) and Model Predictive Control with Imposed Target Dynamics (MPC-ITD). Moreover, issues associated with practical implementation of these algorithms on the FPGA, are discussed.

1.4.1 FPGA-Based Development Platform

In fact, early applications of FPGAs used to be mostly as glue logics on PCB boards to interconnect digital components such as memories, buses, network drivers and processors together. By increasing clock frequency and logic density of FPGAs and developing computer aided design tools, there are increasing interests in using them for computationally intensive applications.

FPGAs represent a freely configurable logic system being able to adapt to any computational tasks. Advancing in FPGA technology increases logic density and speed of configurable logic blocks of FPGAs on the one hand, and enables to integrate various logic components on a single chip, on the other. For instance, many modern FPGA chips, in addition to configurable blocks, have embedded hardcore processors, transducers, multipliers and etc that can flexibly communicate with each other or with the rest of configurable logics, through the FPGA interconnect fabric. All these factors allow to realize an extraordinary computational engine.

In addition to high computational performance of FPGAs, thanks to parallelism and more point-to-point connections, they can potentially improve reliability and safety of the design. There are many demanding industrial applications including drive control systems, in which high performance FPGAs can be beneficial. Nevertheless, main reason that many industries still prefer to utilize DSPs, is that development time of FPGAs is considerably longer. In particular for drive control applications, which is the topic of this thesis, FPGA-based development challenges can be divided into two stages: 1- Developing appropriate control algorithms to efficiently make use of FPGA logic resources. 2- Creation of respective FPGA architecture, debugging and functionality verification.

In this thesis, an FPGA-based platform is introduced for developing, verification and instrumentation of control algorithms. At the design stage, this platform enables to carry out Model-In-the-Loop (MIL) simulation. Thanks to high speed interface of the developed platform and its capability for extending data storage capacity using on board memory blocks, it can be considered as a high performance co-processing engine for

expedition of high precision simulation of power electronic and drive models. For instrumentation it is provided with a high performance host interface that allows data playback at high sampling rate up to the FPGA main clock frequency.

1.4.2 Continuous Set Nonlinear Model Predictive Control

While many researches have shown that the same algorithms can be implemented on FPGAs and obtain shorter execution time at lower cost, some algorithms are basically more suitable for DSP implementation. It might be, of course, because of the fact that DSP technology is more developed and more optimization techniques are available. Pipelining and parallelization are two elementary optimization techniques which can promote performance of the FPGA models. Nevertheless many computational tasks are described in a way, that they can benefit neither of pipelining nor of parallelism, since most of them are basically developed for processors which are able to fulfill tasks only sequentially. Thus the algorithm should be properly adapted to the FPGA architecture to obtain high computational performance.

Model Predictive Control is a computation based control strategy suitable for power electronic and drive applications. The numerical solution which we proposed for nonlinear model predictive control of inverter fed electrical drives, is well suited for FPGA-based implementation. The iteration based structure of the algorithm, allows to utilize the pipelining technique for efficient usage of logic elements on the one hand and reducing execution time on the other.

1.4.3 Model Predictive Control With Imposed Target Dynamic (MPC-ITD)

Another control strategy being appropriate for FPGA-based realization which is proposed in this thesis, is so called Model Predictive Control with Imposed Target Dynamic (MPC-ITD). MPC-ITD is an extension of already known idea of Synergetic control incorporated into Model Predictive Control. Integrity of Synergetic principles in describing dynamic phenomenas and self-organization, is utilized here to tackle problems of cost function design and optimization of the MPC algorithm. Unlike CS-NMPC which relies on a modulation technique to control the power electronic converters, MPC-ITD is investigated as a direct control method without a modulator. Indeed these two algorithms address different technological demands.

While CS-NMPC improves robustness and accuracy of control for tracking reference values, MPC-ITD enables to reduce switching frequencies and output ripples. The performance of the proposed algorithms is validated through experimental tests.

Chapter 2

Induction Motor

2.1 Introduction

Simple construction, low cost and high reliability of induction motors are the key factors making them the workhorse of electric power industry. In induction motors there are electrical circuits on both rotor and stator. Because of different rotational speeds of the magnetic field and the rotor, an induced voltage is produced in the rotor electrical circuit. The induced potential then causes electrical currents in the rotor windings. Interacting between rotor currents and stator flux, an electromagnetic rotational torque is produced that tends to rotate the rotor in the same direction as the air-gap flux. When the rotational speeds of the rotor and air-gap flux are equal, no current will be induced in rotor windings and consequently no torque will be produced. If the rotor speed is higher than the rotational speed of the air-gap flux, the produced torque has negative direction and the machine works in generator mode.

2.2 Mathematical Model of Induction Motor

Cross section of an IM is depicted on figure 2.1. For establishing mathematical model of IM the following simplifications are considered:

- Winding of all phases have the same parameters such as inductance and resistance.
- Magnetic fields of the motor phases have a sinusoidal distribution.
- Saturation of the magnetic circuits is not considered.

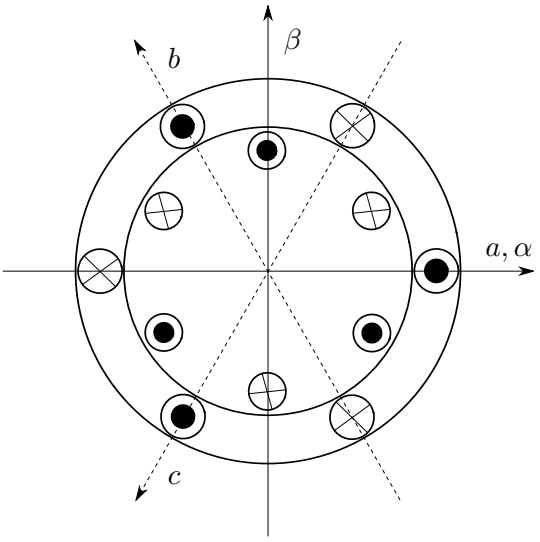

FIGURE 2.1: Cross section of an induction motor

For modeling of the dynamic phenomenas in 3-phase IM, more often the 2-phase equivalent model in which 2 windings are placed in 90 degree position from each other, is considered [13].

2.2.1 Differential Equations of Induction Motor

An induction motor consists of two separate electrical circuits on the rotor and stator. These two circuits are coupled together by the air-gap magnetic field. In so-called squirrel cage induction motor, rotor circuits are short-cut without connection to an external supply. The rotor currents are induced by the magnetic field produced by the stator windings [24].

Voltage equations of IM, in stator reference frame, are as follows:

$$\boldsymbol{v}_s = R_s \cdot \boldsymbol{i}_s + \frac{d\boldsymbol{\psi}_s}{dt} \tag{2.1}$$

$$\boldsymbol{v}_r = 0 = R_r \cdot \ \boldsymbol{i_r} + \frac{\boldsymbol{\psi}_r}{dt} - j \cdot \ p\omega_r \cdot \ \boldsymbol{\psi}_r \tag{2.2}$$

$$\boldsymbol{\psi}_s = L_s \boldsymbol{i_s} + L_m \boldsymbol{i}_r \tag{2.3}$$

$$\boldsymbol{\psi}_r = L_r \boldsymbol{i_r} + L_m \boldsymbol{i}_s \tag{2.4}$$

$$T_e = \frac{3}{2} P(\boldsymbol{\psi}_s \times \boldsymbol{\psi}_r) \tag{2.5}$$

$$\frac{d\omega_r}{dt} = \frac{1}{J_m}(T_e - T_l) \tag{2.6}$$

where

$$j = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{2.7}$$

2.2.1.1 State Equations

$$\frac{d\boldsymbol{\psi}_s}{dt} = \boldsymbol{v}_s - R_s \cdot \ \boldsymbol{i}_s \tag{2.8}$$

$$\frac{d\boldsymbol{\psi}_r}{dt} = -\frac{R_r}{L_m} \cdot \boldsymbol{\psi}_s + \frac{R_r}{L_m} \cdot \boldsymbol{i}_s - j \cdot \ P\omega_r \cdot \ \boldsymbol{\psi}_r \tag{2.9}$$

$$T_e = \frac{3}{2} P(\boldsymbol{\psi}_s \times \boldsymbol{\psi}_r) \tag{2.10}$$

Performing simple manipulations the equations can also be represented as followings:

$$\frac{d\boldsymbol{\psi_s}}{dt} = \boldsymbol{v}_s - R_s \cdot \ \boldsymbol{i}_s \tag{2.11}$$

$$\frac{d\boldsymbol{i}_s}{dt} = -\frac{R_e}{L_e}\boldsymbol{i}_s + \frac{K_r A_r}{L_e}\boldsymbol{\psi}_r - \frac{K_r P\omega_r}{L_e} j\boldsymbol{\psi}_r + \frac{\boldsymbol{v}_s}{L_e} \tag{2.12}$$

$$T_e = \frac{3}{2} P(\boldsymbol{\psi_s} \times \boldsymbol{\psi}_r) \tag{2.13}$$

$$K_r = \frac{L_m}{L_r}; L_e = L_s - \frac{L_m^2}{L_r}; R_e = R_s + R_r \cdot K_r^2; A_r = \frac{R_r}{L_r} \tag{2.14}$$

These equations can be used to build a simulation model of IM.

In fact many of dynamic phenomenas are not considered in the fundamental model [25, 26]. For instance saturation of iron core and anisotropies of stator and rotor cores caused by slots are not modeled. Nevertheless this simplified model is sufficient to investigate in the control system of IM.

2.3 Control of Induction Motor

Induction motors are widely applied for adjustable speed drives. Several control schemes with different level of complexity and performance have been investigated.

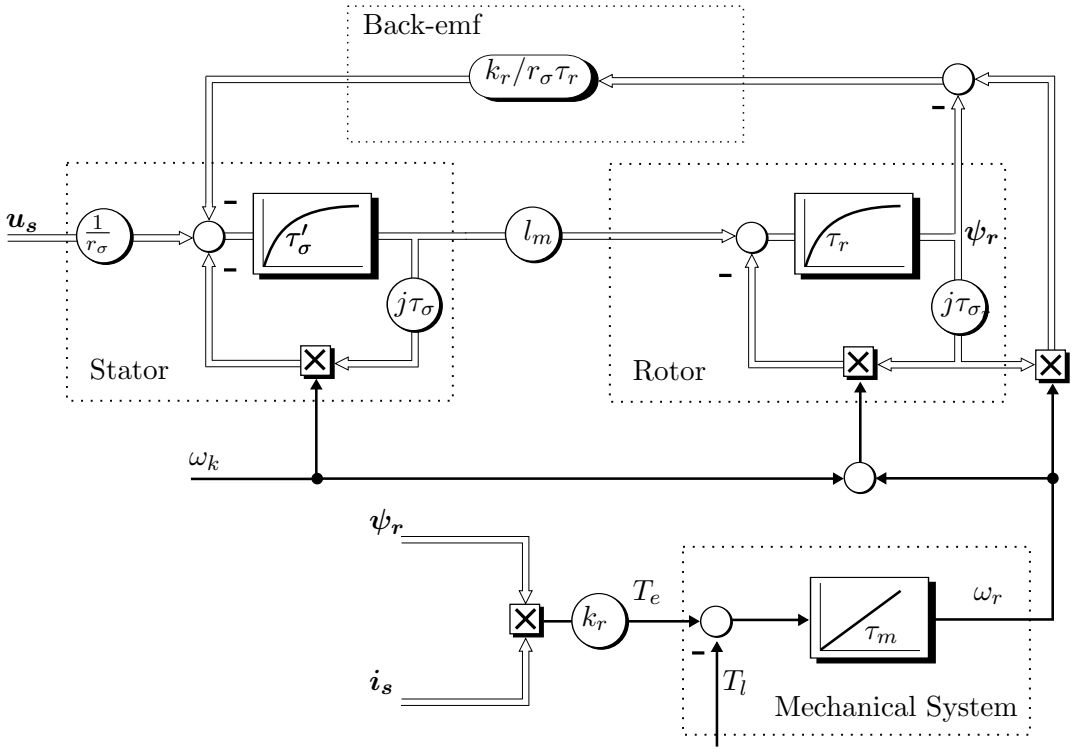

FIGURE 2.2: Signal flow of an induction motor in an arbitrary reference frame

2.3.1 Scalar Control

In addition to simple construction, induction motors have the advantage that with a simple control system soft startup and robust operation can be ensured. The main reason is that rotor and stator are coupled via the magnetic field. At the startup there exists no rotor flux and it is induced just by the stator windings. Initial position of the rotor is not necessary to know for producing electromagnetic torque. While a more complex vector control scheme shows better dynamic performance for the torque control, the so-called scalar control is more simple and robust to parameter variations and can be implemented without an encoder [27].

At the steady state, speed of the motor is close to the stator frequency. Therefore it is possible to control the rotor speed approximately by changing the stator frequency. In scalar control method air-gap flux is assumed to be proportional to the magnitude of the stator voltage divided by its frequency. To obtain nearly constant flux linkage, the ratio of stator voltage amplitude to its electrical frequency (V/F) should be kept constant.

The difference between frequency of the rotor and air-gap flux is so-called slip frequency. In steady state condition, electromagnetic torque is almost proportional to the slip frequency. Due to that load variations change the rotor speed. In open-loop scalar control this effect can be compensated by estimating the slip frequency. However when higher accuracy is required a closed-loop speed control strategy has to be adopted [28].

To maintain high efficiency operation of the motor and minimize losses while working at various mechanical speed, the air-gap flux must be regulated respectively [29, 30].

2.3.2 Field Oriented Control

Induction machine is a multidimensional dynamic system [24]. In rotor field oriented coordinates, stator current vector can be split into field generating and torque producing components. A proper control of current components allows to regulate rotor flux and electromagnetic torque separately like in conventional DC motors [31]. This strategy is so-called Field Oriented Control (FOC). Coordinate transformation and high performance current control require more computational complexity than scalar control method. Thanks to the advances in digital computing technology, FOC became one of the industry standard control strategies [32–34].

Flux and torque producing current components have slight nonlinear inter-coupling which must be considered to obtain a high performance torque control [13, 35].

2.3.3 Direct Torque Control

A control strategy, quite different from FOC, is so-called Direct Torque Control (DTC). In FOC a PWM technique is applied to control the output voltage of the inverter [17]. In contrast to FOC, in DTC, the electromagnetic torque is regulated via a direct control of the inverter switches [12]. In DTC two hysteresis controllers are implemented to control the torque and stator flux. In addition to the more simple structure of DTC, it provides a faster torque response [36]. However, DTC suffers from high torque ripples and variable switching frequencies.

2.3.4 Model Predictive Control

Advancing in digital computational devices enables to realize more complex control methods. An attractive concept which has emerged as a promising control strategy for induction motor drives, is Model Predictive Control (MPC). Predictive control conveys a wide definition and in practice there are many different MPC strategies. Generally, any control technique that includes future state of the plant in calculation of the control law can be considered as predictive control.

IM drives have a very broad application area. Due to that there are many different requirements for the control strategy depending on any specific applications. A primary issue which was investigated for predictive control of IM was reducing current distortion

for inverter fed drives in high power range. Due to the thermal problem of power electronic devices, switching frequencies can not exceed few hundred *Hz*. For such operation conditions the conventional PWM technique fails to maintain dynamic performance and to generate optimal switching time for power switches. To deal with these problems, a trajectory tracking dead-beat controller is proposed in [37]. Dead-beat control is a simple use of the prediction strategy for calculating the control law. It has been investigated for both linear and some class of nonlinear systems [38]. In [39] a dead-beat algorithm is employed to improve performance of DTC method for an induction motor.

Another contribution of model based prediction is to compensate computation time delay [40]. In most of microcontrollers, execution time of the control algorithm is equal to one sampling time. A prediction strategy enables to compensate the computation time.

In [41] current controllers in FOC are replaced with Generalized Predictive Control (GPC) to improve dynamic performance. Main contribution of this work is that it shows feasibility of more complex predictive algorithms with respect to computational performance. GPC strategy provides a systematic approach to designing the cost function and dealing with optimization problems. However, it is restricted to linear systems.

2.3.5 Flux Observer

Stator currents of induction motors can be measured by means of current sensors. However, the control algorithm requires more information about the state parameters such as stator and rotor fluxes or mechanical speed of the rotor. Due to difficulty of placing a flux sensor in the air-gap, almost always it is relied on an observer to estimate the flux [42].

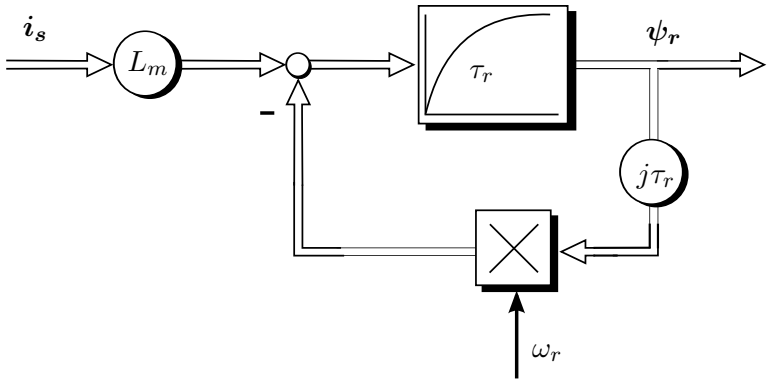

FIGURE 2.3: Signal flow of the flux observer for an induction motor [1]

Based on the mathematical model of IM, the flux observer can be designed. Various types of flux observers have been published for induction motors [43]. They differ depending

on the parameters and variables needed for the design of the observer. The voltage model observer has the advantage of not requiring electromechanical speed. However, due to integrator drift it offers low accuracy. An observer with higher performance is the current model observer [1]. Complex signal flow of the observer is depicted on figure 2.3. For the current model observer, the rotor resistance plays a significant roll. Parameter errors can appear due to inaccurate identification or changing operation points. The Model Reference Adaptive System (MRAS) is a well known scheme for dealing with the parameter variation [44].

2.3.6 Sensorless Control

Rotational speed of the rotor in IM is necessary for designing the flux observer. Furthermore, it is required as a feedback signal for the speed control loop. It is possible to measure the speed by means of a position or speed sensor. Using a sensor is, however, additional cost. Moreover, due to wiring and sensor interfaces, it reduces reliability of the drive system and needs more frequent maintenance. Modern electrical drives are provided with a digital signal processor being able to perform millions of mathematical operations per second. It rises interests in using numerical algorithms for estimation of the position and speed relying on the measured terminal currents and voltages. This technology is known as "sensorless" control. In fact the word *sensorless* dose not mean that there is no sensor in the drive at all, since almost in all schemes, current sensors are used to measure the stator currents.

Sensorless algorithms can be categorized basically into two main groups: 1- *Fundamental model-based* algorithms 2- *Anisotropy-based* techniques.

2.3.6.1 Fundamental Model Based Sensorless Control

The so called *Fundamental model-based* methods utilize the fundamental motor model and torque and flux producing stator current harmonics to estimate the rotor speed. Fundamental model based methods have good behavior at higher speed, however, they show limitations at lower speed and standstill. There are several schemes developed for model-based speed estimation of IM [43]. They can be distinguished with respect the to parameter dependency, required dynamic performance and computational complexity. Designing a closed-loop observer seems to have better performance and accuracy and can handle some sort of parameter deviations. The advantage of MRAS is that it can be also applied to nonlinear systems. IM has a well known and parametrized structure. Furthermore stator currents can be easily measured. Relying on this information, it is possible to design a MRAS scheme for speed estimation [45].

In the MRAS technique for speed estimation, at first it is treated as a slow changing parameter. Rotor flux can be estimated both by the voltage model without using mechanical speed and by the current model with mechanical speed as a parameter. Comparing estimated flux of both models produces an error which can be utilized for estimation of the speed. Thanks to the intuitiveness of parameter tuning of PI controllers, in most publications it is adopted for adaptation of the speed in variable model of MRAS. Nevertheless, to eliminate parameter tuning of the PI controller, a sliding mode controller can be utilized [46]. Figure 2.4 shows the structure of a typical MRAS estimator. Unlike conventional closed-loop observers such as Kalman filter or Luenberger observer [47], which compare output of the adaptive model directly with the plant output, in MRAS the adaptive model is compared with a reference model of the plant with defined parameters. It offers more flexibility to utilize all available knowledge about the system in design of the observer.

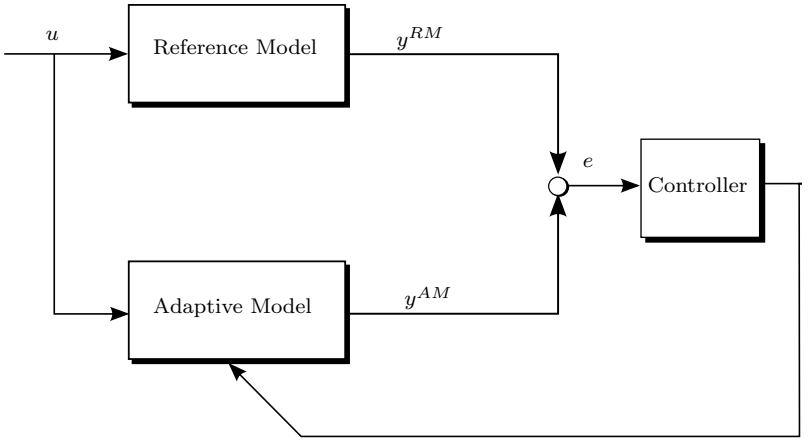

FIGURE 2.4: Structure of the MRAS estimation strategy

2.3.6.2 Anisotropy Based Sensorless Control

The major deficiency of *Fundamental model-based* sensorless methods, is that they need precise information about motor parameters which may vary during operation due to change of temperature or operation points. Another drawback of *Fundamental model-based* methods is the integration drift. It is basically more challenging at low speed region. *Anisotropy-based* sensorless algorithms have been widely investigated to deal with these issues. These extract speed and position dependent anisotropy image from measured stator current signal.

For electrical drives fed by power electronic inverters, stator voltages include high frequency components generated by switching. In addition high frequency voltage signals can be injected. Hence in the stator currents high frequency components appear which

are modulated through the anisotropies and carry useful information for estimation of the rotor position [48, 49].

Anisotropy-based methods received an intensive attention over the last decade [50, 51]. In any electrical motors there are several sources of anisotropies. For the *anisotropy-based* rotor position (speed) estimation, information of the rotor dependent saliences must be filtered. In drives with one dominating rotor dependent anisotropy, such as synchronous motors, it is straightforward to extract the anisotropy image from the stator currents. What makes practical implementation of the *anisotropy-based* sensorless methods for IM challenging is that rotor dependent anisotropies are small and therefore, it is difficult to distinguish them from measurement noises. Another challenge is the existence of multiple anisotropies. It requires precise information about their magnitude and frequency in order to separate them properly [52].

Numerous publications on *anisotropy-based* sensorless control and diversity of applications show that the anisotropies existing in almost all drives can be utilized for the position and speed estimation.

Chapter 3

Digital Computational Systems

3.1 Introduction

Digital Computer is, definitely, one of the greatest inventions of the human history. Over recent decades, digital computers have played major rolls in almost all area of our rapidly changing society. By appearance of the first programmable computers being able to perform user defined tasks, their computational performance as well as application area have been constantly advancing.

In everyday life we work with hundreds of computers. Architecture of computers and peripheral components may be different depending on applications. In this thesis two types of digital computers are distinguished: Hardware programmable and software programmable computers. For the hardware programmable computers, the computational task is defined as a hardware model implementable on the target device whereas for the software programmable computers tasks should be first interpreted into instructions executable by the computer.

This part makes a comparative evaluation of the hardware programmable versus the software programmable computers for the control system of electrical drives.

3.2 Digital Signal Processors (DSP)

A DSP is a complete computer that provides most required hardware peripherals embedded on a single chip. Integrating many peripherals such as memory blocks, memory controllers and etc, would lead to reduction of the system cost and improvement of the

computational performance. Further distinction of DSPs is the higher number of the basic instructions which the processor is capable to perform [53].

A big area of applications for DSPs is control of electrical drives. The main requirement for these applications is real-time operation. It demands performing all computations within a predefined discrete time. The strategy for meeting this requirement, in a cost effective way, is reducing the computational load of the CPU and assigning many of the tasks to the peripherals. One of the newest DSP chips is shown on figure 3.1. It is customized for motor drive control applications. What is noticeable about it, is its floating-point unit making it powerful for complex mathematical calculations.

As it is illustrated on the figure, the DSP has many peripherals which are embedded on the same chip and can perform tasks without involvement of the processor. It reduces the computational load of the processor to execute the application program faster. Such a compact computing system is intended to reduce the cost and increase the performance.

3.2.1 Central Processing Unit (CPU)

The CPU is responsible for controlling the program flow and executing the instructions. A typical CPU performs arithmetic, Boolean logic, multiplication and shift operations [53, 54].

3.2.2 On-Chip Memory Blocks

Each typical computer has a CPU which is capable of performing some elementary instructions. However, without a set of commands that tells the computer what to do, it is unable of doing anything. Those commands must be stored in the digital memory and sent to the processor to be fulfilled. The speed of the computer is not only determined by its clock frequency but also by the performance of transmitting commands from the memory. To increase the computational performance of DSPs, the on-chip memory is embedded closed to the accumulator unit that can communicate with a very high bandwidth. On-chip memory is the most significant part of a DSP integrated on the same chip and provides high communication speed for transferring data with the CPU. The processor uses the address bus to access the on-chip memory blocks. Due to limited storage capacity of the integrated memory blocks, efficient utilization is an important optimization issue for DSP programming [55, 56].

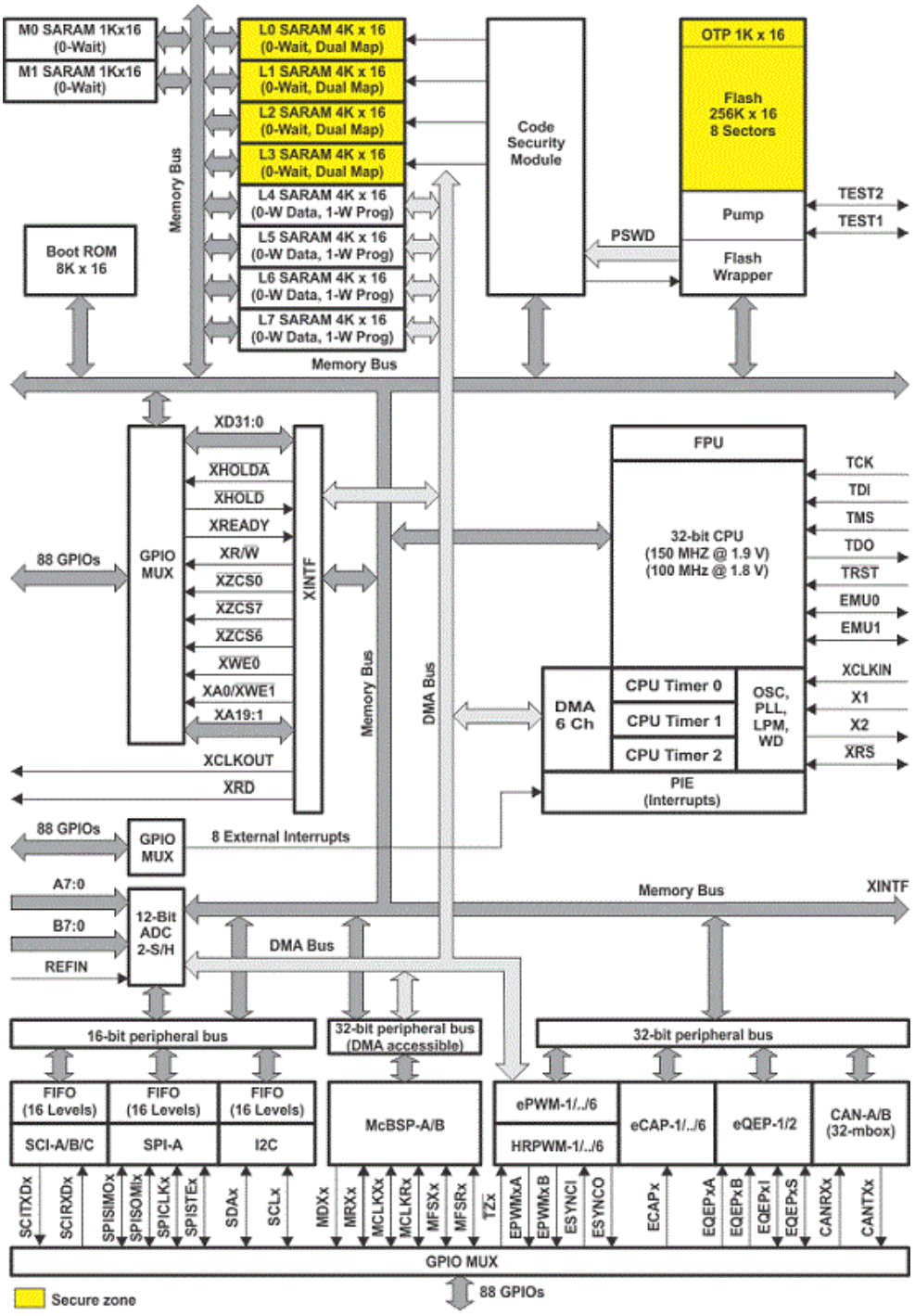

FIGURE 3.1: Functional block diagram of TMS320F28335 (source: www.ti.com)

3.2.3 Application Software

The application software is the soul of the DSP. It is a collection of commands that as a whole makes the control algorithm. Thanks to the high computational performance of DSPs and automatic code generation tools, nowadays many application programs can be developed much easier and in shorter time [57, 58].

3.2.4 Real-Time Signal Processing

In a computer, some tasks must be performed corresponding to events or at specific instants of time. For that purpose an interrupt signal is sent to the CPU. However, many interrupt signals can be generated simultaneously requiring the CPU to execute associated tasks. The interrupt manager decides about the priority of interrupts which must be fulfilled.

Modern DSPs are capable of performing billions of tasks per second. Timing schedule for all these tasks are managed by a real-time kernel [59]. By increasing number of interrupts with short periods, it is more and more difficult for a single processor to catch all of the interrupt events. In such cases a multi-core system can improve real-time performance and resolve interrupt handling problems [60].

The reliability of DSPs is the most important issue in particular for real-time operations. All of aforementioned parts of a DSP can fail during operation. Analyzing failures that can happen during operation of a processor, one thing is rather obvious: Most of failures are because of the centralized processing structure. In other words, when only one processor is performing all of tasks and all data should pass the same data gate, it is very difficult to detect errors.

3.3 Field Programmable Gate Array (FPGA)

FPGAs are hardware reconfigurable devices enabling the designer to flexibly customize the hardware architecture to create the desired embedded system. Parallelism of FPGAs and distributed on-chip logic resources increase the data through and enhance the computational performance.

Although, conventional DSPs are provided with application specific logics and peripherals, FPGAs provide more flexibility in terms of hardware programmability and can address a broader application area [61].

FPGAs consist of configurable logic blocks that can be interconnected to realize various designs [62, 63]. The FPGA technology, in particular over the last decade, has been rapidly advancing. Modern FPGAs, in addition to the configurable logic blocks and programmable interconnect fabric, include embedded DSP blocks such as multipliers, which can be utilized in a distributed design to promote the computational performance.

Thanks to the unlimited reconfigurability of FPGAs, for low to medium volume productions, they provide a cheaper solution and faster time to market as compared to the Application Specific Integrated Circuits (ASIC). FPGA configuration takes less than a minute and they cost anywhere around a few dollars to a few thousands of dollars depending on logic resources and the performance [64].

Standard FPGA logic resources include:

- Configurable logic blocks
- Programmable interconnect fabric
- I/O blocks which are connected to the logic blocks through the routing interconnect and make off-chip connections.

FPGAs have gone beyond a simple array of uniform logic elements. They offer a diversity of integrated components that may be incorporated into a design, yielding a more efficient product. Examples of such embedded hard cores are DSP blocks [65]. To address wide range of system performance, energy efficiency and design complexity, several FPGA families are provided [66]. Figure 3.2 shows advances in a low cost FPGA family over the last decade. Although the FPGA is further developed from various aspects, the cost and power consumption are significantly decreased.

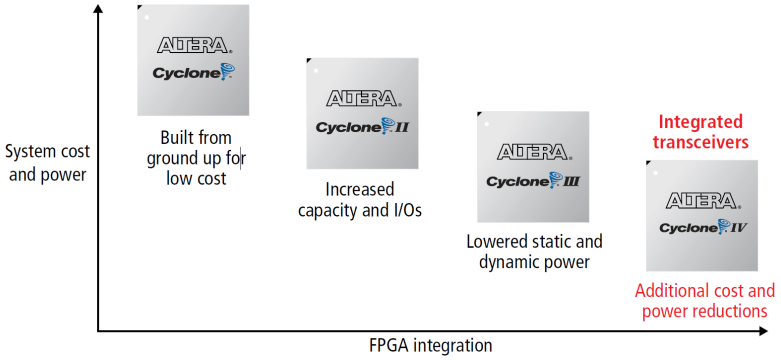

FIGURE 3.2: Cyclone FPGA family. Source: www.altera.com

3.3.1 Basic Logic Elements on FPGAs

Basically, configurable Logic Elements (LE) of FPGAs are combination of the silicon based transistors [67]. Due to the extremely high number of transistors, analysis of the FPGA circuits at transistor level is too complex for programming purposes. Instead of that, usually other units are provided in order to make an estimation of available logic resources and design performance. Depending on the FPGA technology, the architecture

of LE might be different [62]. For sake of simplicity we mainly investigate Cyclone FPGA family of Altera (see figure 3.2). LEs enable to realize combinational logic functions. They can be used also as glue logics for routing on-chip hard-cores [66].

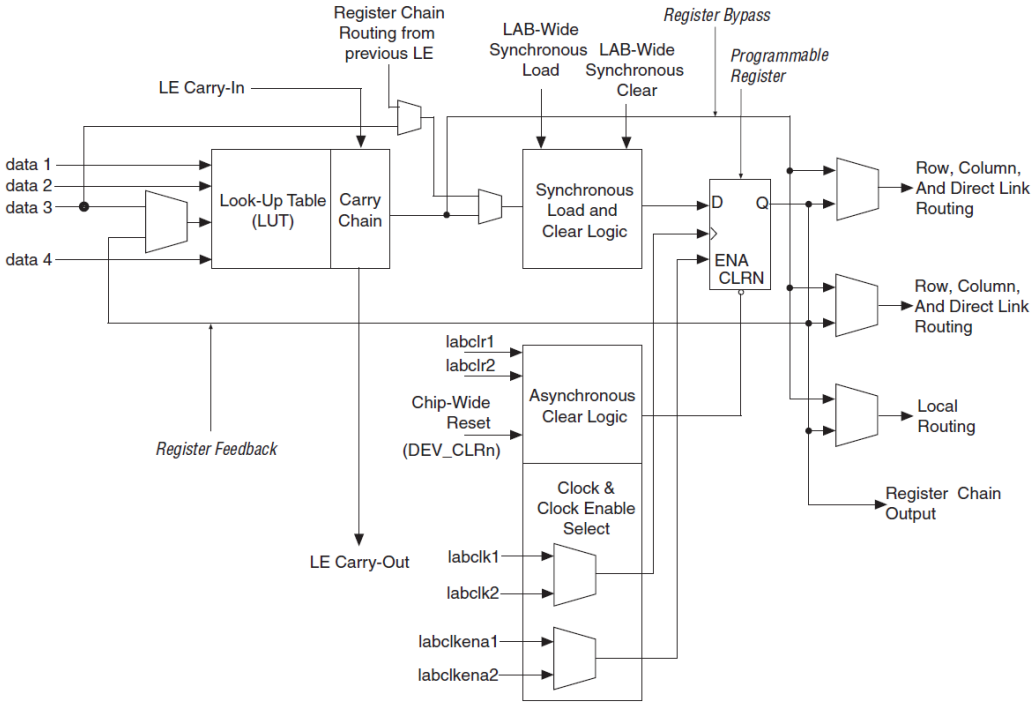

FIGURE 3.3: Logic Element of Cyclone IV FPGA chip, Source: www.altera.com

Logic Array Blocks (LAB) consist of several LEs as it is shown on figure 3.4. While the lab local interconnect can drive LEs of the same or neighboring LABs, memory block or I/Os, the multi track interconnects (row and column interconnects) are intended for routing of different LABs [66].

FPGA designs typically include thousands of LABs routed together by means of the interconnect fabrics. The FPGA compiler automatically places critical design paths such as clock signals, on faster interconnects to improve the design performance.

3.3.2 On-Chip Memory

Most of embedded computational systems require high bandwidth data storage with a high communication rate. Thanks to the hardware programmability of FPGAs, they can easily communicate with on-board data storage devices, however, for a more secure and higher transferring rate, distributed on-chip memory blocks are integrated on the FPGA. Embedded memory blocks can be utilized for realizing look up tables, storing design parameters or sampled data.

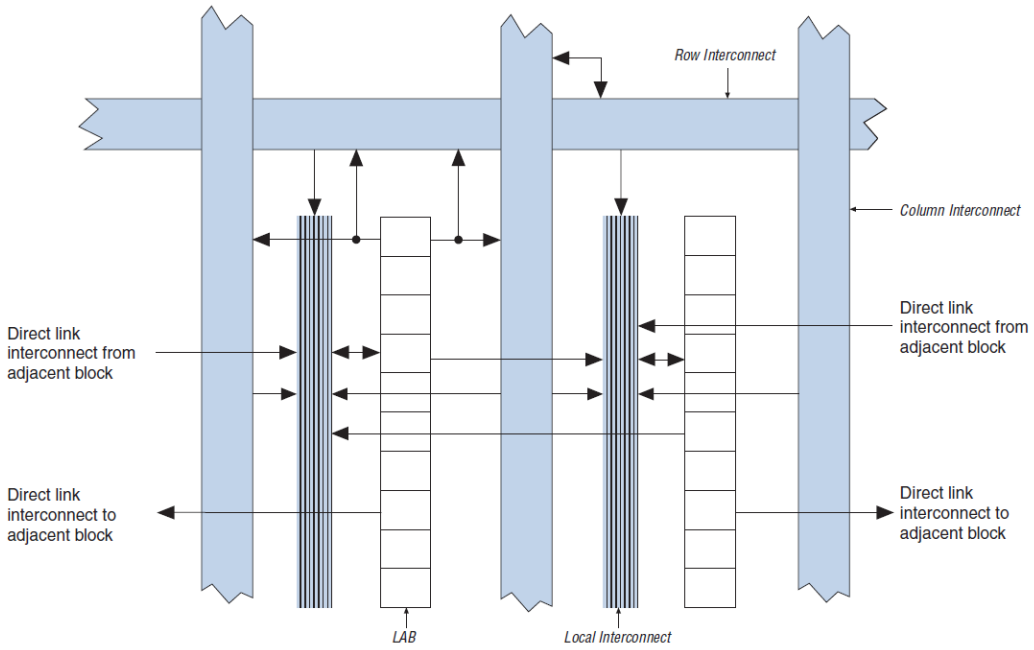

FIGURE 3.4: Interconnect Fabric of Cyclone IV FPGA chip, Source: www.altera.com

3.3.3 Hard-Core Functional Blocks

Hard-core functional blocks are implemented during the fabrication of the FPGAs. They are not freely programmable like logic elements, however, it is usually possible to configure and flexibly route them with the rest of the logic resources. Some examples of embedded hard-cores are:

- Transceivers
- Embedded multipliers
- Phase Lock Loop (PLL)

While the maximum clock frequency of the combinational logics are usually limited depending on number of logic elements and data paths, the hard-cores are optimized to operate at higher clock frequencies and lower energy consumption to satisfy demanding applications.

Figure 3.5 shows an embedded multiplier block of Cyclone IV. Input signals can be signed, unsigned or combination of both. By means of the multiplexers, it is possible to send input and output signals into the terminal registers or connect them without the registers. It allows to combine several multipliers for signals larger than 18 bits. Each register is provided with a clock, clock enable and asynchronous clear [66].

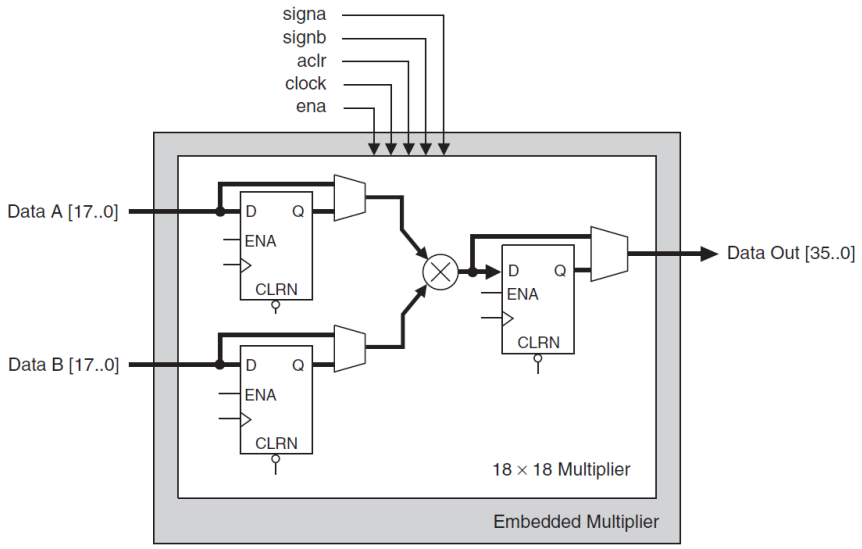

FIGURE 3.5: Embedded hard-core multiplier, Source: www.altera.com

3.3.4 Embedded Hard-Core Processor

The maturity of the software technology, which in many cases has grew up as a completely independent discipline, together with the flexibility of FPGAs for hardware programmability encourage many system design industries to use FPGA logics for creating a programmable computer architecture. Many open source design templates and programming tools are available for soft-core processors on FPGAs. The obvious advantage of FPGA implementation of the processor is customization of the hardware architecture. However, for the design of the soft-core processor, it requires considerably more logic resources compared to a hard-core processor with the same topology. Furthermore, performance of the hard-core processor is much higher.

SoC (System-on-Chip) is a definition for customized FPGA chip with an integrated hard-core processor. The processor is realized in hardware and can operate independently from other programmable logic elements. Nevertheless, by means of the available FPGA logic resources customized peripherals can be added to the processor to extend its functionality.

Figure 3.6 shows a SoC FPGA chip with two hard-core processors. It is intended to combine hardware and software flexibilities on a single chip maintaining high computational performance.

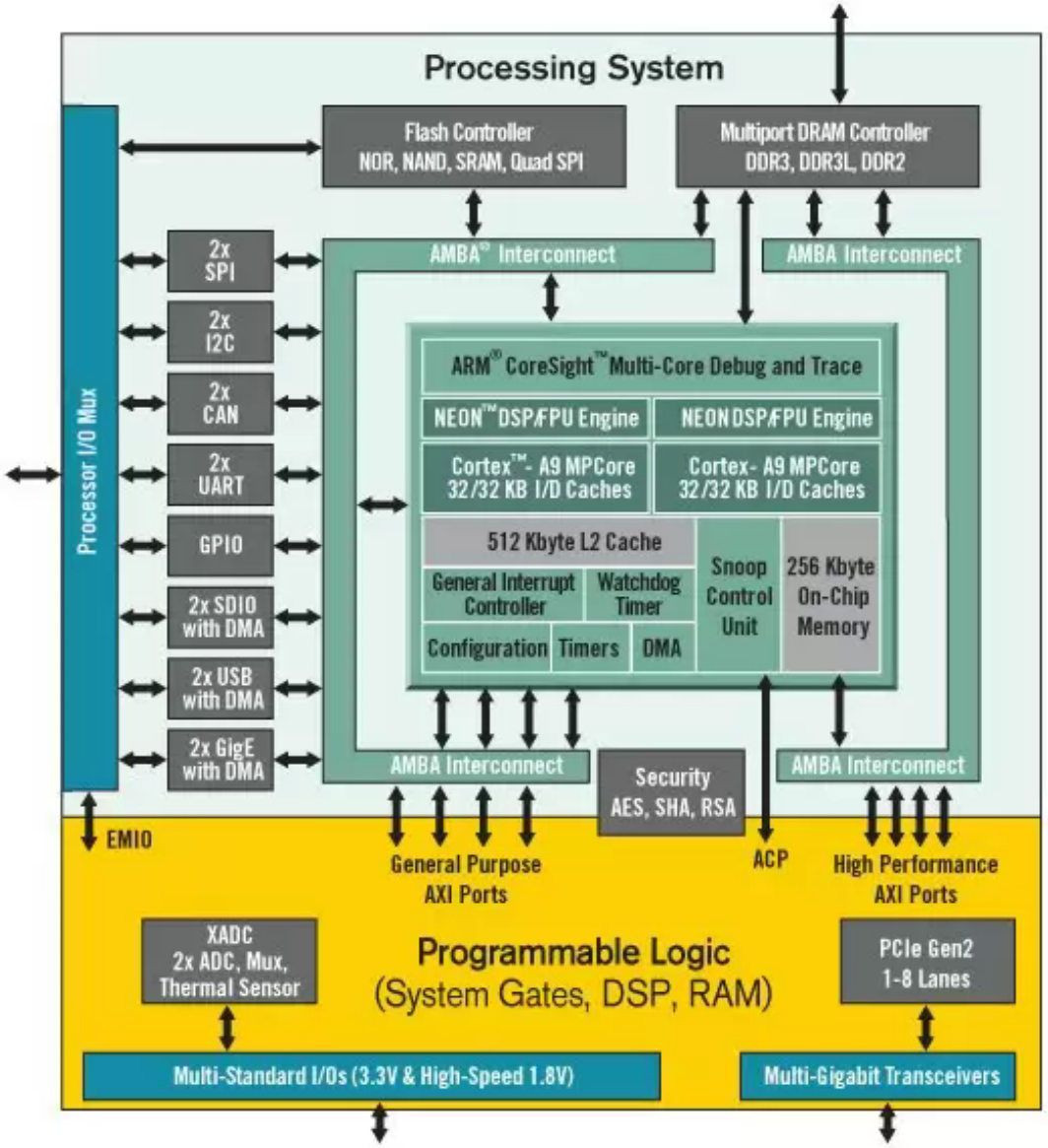

FIGURE 3.6: Functional block diagram of a SoC FPGA with integrated hard-core processor. Source : www.xilinx.com

3.3.5 FPGA Programming

Logic resources of the FPGA must be appropriately configured to enable it performing dedicated tasks. The FPGA program is, in fact, description of the hardware architecture. For FPGA programming, Hardware Description Languages can be used. Two main textual HDLs are VHDL (VHSIC Hardware Description Language) [68, 69] and Verilog [70]. These two languages are most popular industry standard HDLs [71]. The main feature of HDLs is their ability to efficiently describe hardware resources and configurations at different level of complexity.

By increasing the logic density, it becomes possible to implement more and more complex digital systems. To deal with the complexity of the FPGA programing, engineers prefer high level graphical environments such as Simulink with high capability of design automation [72] rather than textual hand coding. There are several possibilities of using Simulink or other graphical programming tools for developing and verification of FPGA configuration models and it is expected that number of such tools will be increasing. Currently, the two largest FPGA vendors, Xilinx and Altera, provide high level design tools integrated into Simulink to expedite FPGA-based development. In addition, Mathworks offers a more universal tool, so called *HDL coder*, which allows to generate HDL codes (VHDL or Verilog) from Simulink models [73].

Basic understanding of FPGA hardware structure and their principle of operation are essential for FPGA-based design. Nevertheless, aforementioned Computer Aided Design (CAD) tools enable to reduce the complexity of the FPGA programming.

3.3.6 Suitability of FPGAs for Drive Applications

The dominating digital computing devices for motor control applications are DSPs. Drive industry has already more than 30 years maturity of DSP applications. Moreover DSP vendors have also a long experience in customizing the microcontroller devices for demanding motor control applications. Despite these facts, using FPGAs has some advantages in terms of computational performance and design issues that make them attractive for research and industrial applications.

An important argument against using FPGAs is relatively longer development time. However, it is not a long term challenge, since there are always more and more skilled engineers in the area of FPGA development. In addition the computer-based design tools are always advancing which can be an effective way for dealing with the complexity of the FPGA programming. Such tools can play significant roles not only for code generation but also for verification and debugging of the code on the target FPGAs. Furthermore, the reconfigurability of FPGAs provides the same level of design convenience as the software in DSPs [74].

Despite the capability of the industry for integrating many digital and analog devices on a single chip, there are some performance criteria which are not cost effective and even not possible to obtain with state-of-the-art of the DSP technology [74].

Dynamic of the current control loop, in particular, for low power servo-drives are very important. Due to the high bandwidth, there are still many applications in which the conventional analog control devices are used. Digital microcontrollers usually provide

a lower bandwidth because of the computational time delay. By using FPGAs the digital hardware can be completely dedicated to the control algorithms. It leads to a significantly lower computation time. Therefore in many applications FPGAs can replace analog devices without compromising the performance.

Another application area for FPGAs is the Hardware-In-the-Loop (HIL) simulation. High computational performance of FPGAs can be beneficial for running real-time models. Because of the short sampling time of electrical drives and power electronics, a precision simulation is usually very time consuming. The parallel processing capability of FPGAs can be very helpful for running computationally intensive models [75–77].

In addition to all aforementioned applications for which FPGAs provide the only feasible solution, there are some other advantages of FPGA-based systems as alternatives versus conventional DSP systems [74].

3.3.6.1 Compactness and PCB Simplification

The hardware programmability of FPGAs allows to integrate many system parts on a single chip. It enables the developer to decrease number of on-board components and reduce the PCB size, still have an universal scalable PCB design.

3.3.6.2 Multiple Real-Time Tasks

In a drive control system, there are several tasks which must be performed in real-time. In DSP-based control systems usually only one CPU is responsible for multiple tasks. To divide the computational power between several tasks, interrupt controllers are used. By increasing number of interrupts, it is more likely that some interrupt events occur simultaneously and the CPU is not able to react on all of them. Furthermore, interrupt latencies can not be precisely controlled due to the software complexity. In contrast to processors, in FPGAs for any tasks independent hardware resources can be dedicated. Parallel processing of FPGAs is well suited for reducing the programming complexity and enhancing reliability of a digital system to handle multiple real-time tasks.

3.3.6.3 Reliability of FPGAs for Motor Control

For many applications, such as aerospace and automotive industries, reliability is an essential factor and the proper functionality of the system in the whole operation conditions must be ensured.

An important source of failures in digital control systems is the application software. In programmable devices the distributed architecture allows the designer to isolate different parts and in this way the errors have only localized impacts. Furthermore, self diagnostic and error correction can be implemented much easier on FPGAs.

3.3.6.4 Scalability of FPGA-Based Design

Data width plays an important role in computational performance. Using FPGAs, it is possible to vary data width of arithmetics as well as bus system of on-chip communication to exploit logic resources more effectively.

3.4 Conclusion

The DSP technology has three decades maturity for the control system of electrical motors and power electronics. Nevertheless, there are many control algorithms which are too complex to be realized on DSPs or at least it is not cost effective. Increasing logic density and programming tools rise interests in FPGA as an alternative.

Hardware programmability of FPGAs allows to fully dedicate the digital system to the control algorithm. Despite the maturity of the DSP technology, diversity of motor control applications makes it difficult to have an universal DSP meeting all requirements. A solution is using FPGAs to have a configurable design.

Further advantage of FPGAs is the scalability. As is shown on figure 3.1 hardware architecture of the DSP can not be changed. The floating-point unit is provided to ensure the compatibility for larger area of applications. It on the one hand causes wasting of so much logic resources which are not useful for many other applications. On the other hand, several parts of the DSP, for instance the integrated AD converters, may not be sufficient and need to be upgraded. In this respect an FPGA-based design has superiority as it represents a fully scalable solution. It is of great importance for the drive control with relatively long life cycle.

Part II

FPGA-Based Real-Time System

Chapter 4

FPGA-Based Computation

4.1 Introduction

The early generations of FPGAs were only for a narrow application area such as very fast and custom digital systems. Chip cost and limited logic resources were the main challenges for those FPGAs [78]. Not withstanding with limited amount of programmable logic resources, hardware implementation of digital systems provided better performance for very fast systems where CPU-based solutions could not meet the requirements in terms of time constraints [79].

Nowadays the FPGA technology has reached the point that can freely integrate hundreds of hard-cores, DSP blocks and millions of logic elements on a single chip. FPGA is in cross section of the hardware and software technology. FPGA programming in fact represents a physical model of the hardware. Therefore both knowledge of hardware and software is required for the FPGA-based design.

While CAD tools and design automation simplify programming of FPGAs, there are still many challenges and open questions. Above all is perhaps construction of system architecture for the efficient use of FPGA resources. Design environment has also a great impact on the effective use of logic resources and the development time. Another issue of FPGA-based applications is estimation of the computational power. In this chapter these topics are tried to be covered.

4.2 Programming of FPGAs

By increasing the density of logic elements of FPGAs, effective use of available resources becomes more and more challenging. The programming language and development environment have significant impacts on the effective usage of logic area on the one hand and shortening the design time on the other. To respond to the need for rapid prototyping of new FPGA-based systems, constantly new tools for programming and verification of FPGA configuration are introduced. Hardware Description Languages (HDL) are industry standard tools for describing models to be implemented on FPGAs.

There are two main textual HDLs: VHDL and Verilog [63]. Basically both of them have the same level of complexity. However, their programming styles are different. Therefore usually one of them only is being used in any designs depending on the conventions of the engineering groups. Aforementioned HDL programming tools provide high flexibility for describing low level hardware configurations. Nevertheless the drawbacks of low level HDL programming are difficulty of debugging, verification and low capability for dealing with the design complexity.

Although for programming of small FPGAs, HDLs are the effective tools representing high flexibility for describing the physical implementation, when dealing with complex algorithms for the target FPGA chips with millions of programmable logic blocks, they are very time consuming and there is a need for a programming language being capable of representing the hardware configuration at a higher level. To keep up with this demand, in addition to the HDLs, there are also system level configuration languages and code generation tools allowing to reduce the development time.

4.2.1 Very High Speed Integrated Circuit Hardware Description Language (VHDL)

An FPGA program is a configuration model of logic elements and routing between blocks. VHDL is a textual hardware programming language that allows to describe the hardware structure. Here a simple example of a VHDL program as well as its graphical view is provided (see figure 4.1).

```
library ieee;
use ieee.std_logic_1164.all;

entity ex_1 is

        port
        (       IN_A        : in std_logic;
                IN_B        : in std_logic;
```

```
                IN_C        : in std_logic;
                OUT_A       : out std_logic);

end entity;

architecture rtl of ex_1 is
begin
        OUT_A <= (IN_B or IN_C) and (IN_A and IN_B);
end rtl;
```

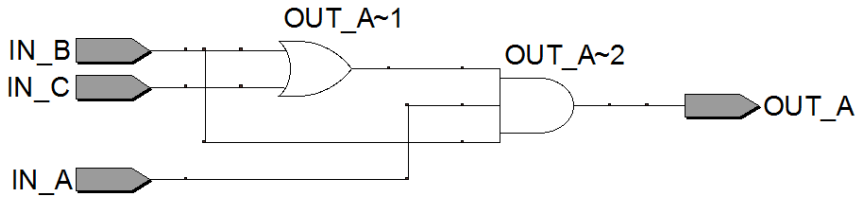

FIGURE 4.1: RTL (Register-Transfer Level) representation of the VHDL model

4.2.2 System Level Design (SLD)

SLD tools are intended to accelerate embedded development. They can be compared with high-level software programming languages. While VHDL is useful to develop hardware configuration and create new digital components, SLD tools deal with high level design automation and creating new design by interconnecting various IP (Intellectual Property) cores such as processors, memory and network controllers, communication interfaces and etc. These components are individually designed and optimized in VHDL or Verilog then can be routed within a larger design through SLD tools [80].

Number and diversity of the IP cores are continuously increasing. In addition to the complexity reduction and avoiding failures in large FPGA designs, SLD tools enable design reuse. An example of SLD tools is Qsys provided by Altera. For more information on this topic one can refer to the chapter 6 of the Quartus II handbook [81].

To address the FPGA-based design for the control engineering, there are some other SLD tools integrated into Simulink. These provide a convenient design environment for developing and verification of the control algorithms. Xilinx System Generator (XSG) and DSP Builder of Altera [82] are example of such SLDs.

4.2.3 Code Generation Tools

Code generation tools, generate the HDL codes from the conventional software models. A very popular developing environment for control engineers is Simulink. Mathworks provides the so-called *HDL coder* which enables to convert the Simulink model into VHDL or verilog codes. It has some advantage in comparison to the other SLD tools: It shows higher performance for simulation of models since before generation of HDL, the whole simulation process is in the Simulink environment. Furthermore the generated HDL code is open to the designer and can be combined with the rest of the hand coded HDL models. And last but not least, the benefit of *HDL coder* is that it is more or less universal and can be applied for all FPGA families while XSG and DSP Builder are only for the Xilinx and Altera FPGAs respectively. Therefore, for our investigation we utilize it for generation of VHDL codes beside hand coding.

4.3 Computational Performance of FPGAs

Dominating computational devices for motor control and power electronic applications are Digital Signal Processors (DSPs). However, interests in FPGA-based systems are rapidly increasing. Numerous publications on FPGAs for industrial control applications show their outstanding computational performance[83, 84].

Major distinctions of FPGA-based systems compared to DSPs, leading to higher computational performance, are the parallelism of FPGA operation principle and the possibility to flexibly dedicate hardware architecture to the computational tasks. An FPGA chip contains hardware blocks which can be freely programmed to construct the desired embedded system [78].

FPGAs can be used for implementation of soft-core processors with custom instructions and specific peripherals. Even though by customizing the hardware architecture, computational performance of soft-core processors can be increased [85, 86], they have relatively low performance compared to the hard-core processors due to the considerably lower clock frequency.

Focus of this thesis is mainly on the FPGA-based and hardware implementation of control algorithms without involvement of a hard- or softcore processor.

4.3.1 Enhancing Computational Performance and Optimizations

Following optimization techniques are mainly applied to FPGA-based designs:

1. Parallel processing
2. Pipelining
3. Resource sharing

FPGA-based designs should be optimized regarding different aspects: Computational speed and logic area. Pipelining and parallel processing are related to the speed optimization while resource sharing techniques address the area optimization. Suitability of algorithms has significant impact on speed of the design as well as logic usage optimizations. Suitability of algorithms will be discussed in further chapters.

4.3.1.1 Parallel Computing

A significant advantage of FPGAs is distributed parallel processing versus centralized sequential processing of the conventional CPU-based systems. Here the VHDL code for two parallel multiplications is provided:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity signed_multiply is


        port
        (
                clk         : in std_logic;
                a               : in signed (15 downto 0);
                b               : in signed (15 downto 0);
                c               : in signed (15 downto 0);
                d               : in signed (15 downto 0);
                result_ab   : out signed (31 downto 0);
                result_cd   : out signed (31 downto 0)
        );

end entity;

architecture rtl of signed_multiply is


begin
```

```
process_clk: process(clk) is
begin
                if rising_edge(clk) then
                        result_ab <= a * b;
                        result_cd <= c * d;
                end if;

end process;

end rtl;
```

Behavior of this model at RTL (Register Transfer Level) is shown on figure 4.2. The outputs are simultaneously updated when a rising clock edge takes place.

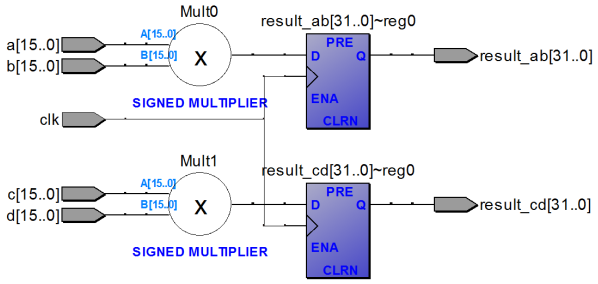

FIGURE 4.2: RTL view of the dual multiplier

4.3.1.2 Pipeline Processing

Pipelining is a common method to enhance data throughput of digital circuits. In a pipelined design, tasks are broken down into several parts through register stages in order to enable the system to be triggered at higher clock frequencies.

The maximum delay of the combinatorial logics between two adjacent registers must be less than one clock period. At rising edge of the clock, the current value of the input signal of the register is delivered to the next pipeline stage. Pipelining reduces the complexity of the timing analysis as well. Below a pipelined VHDL design and corresponding RTL model are provided.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity pipeline_design is

        port
           (clk          : in std_logic;
```

```
                a                : in signed (15 downto 0);
                b                : in signed (15 downto 0);
                c                : in signed (15 downto 0);
                d                : in signed (15 downto 0);
                result_sum   : out signed (31  downto 0));
end entity;

architecture rtl of pipeline_design is

signal result_ab : signed (31 downto 0);
signal result_cd : signed (31 downto 0);

begin

process(clk) is
begin
        if(clk'event and clk = '1') then
                result_ab <= a*b;
                result_cd <= c*d;
                result_sum <= result_ab + result_cd;
        end if;
end process;
end rtl;
```

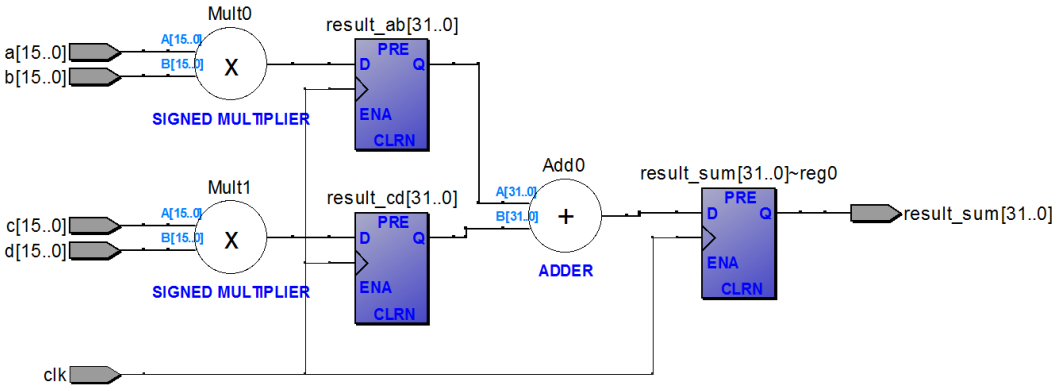

FIGURE 4.3: RTL view of pipelined arithmetic operations

4.3.1.3 Resource Sharing

Computational power of FPGAs is mainly limited by number of available on-chip logic blocks. In software based systems, execution time and number of instructions has almost a linear proportional relationship. For FPGAs this is more complex [87]. To utilize logic resources of FPGA efficiently, more often it is needed to share them between various data

paths. Here VHDL code of a shared multiplier is provided . Figure 4.4 illustrates RTL model of the shared multiplier. The input multiplexers enable to share the multiplier core between two data paths. According to the state of the multiplexers, the output is selected.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity shared_multiply is

        generic
        (
                DATA_WIDTH : natural := 32
        );

        port
        (       clk             : in std_logic;
                a_1         : in signed ((DATA_WIDTH-1) downto 0);
                a_2         : in signed ((DATA_WIDTH-1) downto 0);
                b_1         : in signed ((DATA_WIDTH-1) downto 0);
                b_2         : in signed ((DATA_WIDTH-1) downto 0);
                result_1    : out signed ((2*DATA_WIDTH-1) downto 0);
                result_2    : out signed ((2*DATA_WIDTH-1) downto 0)
        );

end entity;

architecture rtl of shared_multiply is

signal sw       : std_logic := '0';
signal sw_d     : std_logic;
signal result   :signed ((2*DATA_WIDTH-1) downto 0);


begin

process (clk) is

variable a : signed ((DATA_WIDTH-1) downto 0);
variable b : signed ((DATA_WIDTH-1) downto 0);

begin

if(clk'event and clk = '1') then
                sw <= not sw;
                if(sw = '1') then
                        a := a_1;
                        b := b_1;
                else
                        a := a_2;
                        b := b_2;
```

```
                end if;

                result <= a*b;
                sw_d   <= sw;

                if(sw_d = '1') then
                        result_1 <= result;
                else
                        result_2 <= result;
                end if;


end if;

end process;
end rtl;
```

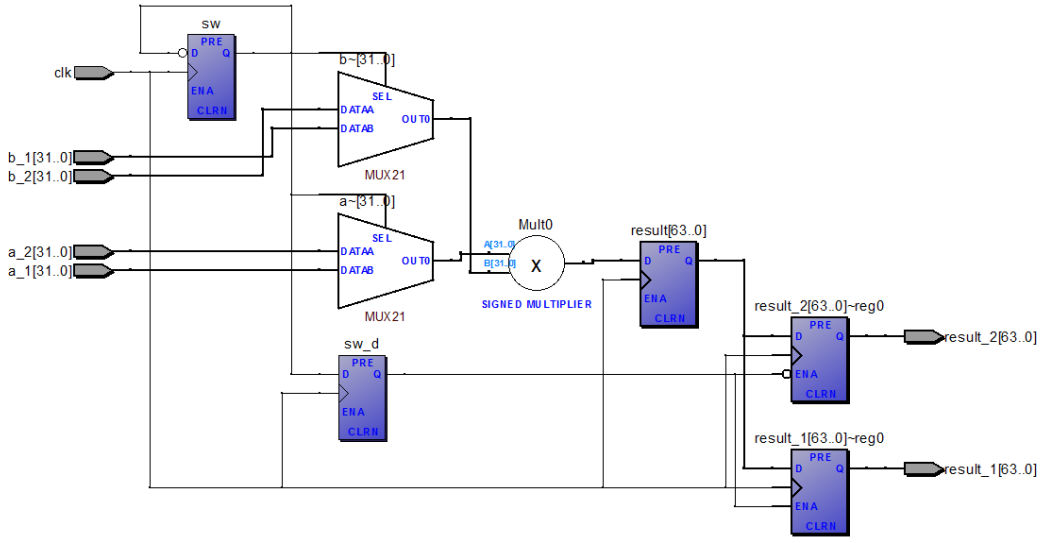

FIGURE 4.4: Sharing an embedded multiplier between two data paths

4.3.2 Design and Optimizations in MATLAB/Simulink

Graphical programming of MATLAB/Simulink has received a wide acceptance in particular as a rapid control prototyping tool [57, 88, 89]. By advancing in code generation, in many applications generated code from Simulink is used to program target control devices not only for prototyping but also at production stage. In addition to C code generation possibility that mainly addresses DSP-based target control hardware, Simulink is also provided with the *HDL coder* enabling to generate VHDL or Verilog models for implementation on FPGAs. Since graphical programming of Simulink provides a good

overview of different signal paths and subsystems, it is well suited for representation of HDL entities. Furthermore, Graphical User Interface (GUI) of MATLAB is an intuitive tool for parametrization of models as illustrated on figure 4.5.

To enable the *HDL coder* to generate VHDL code of a given Simulink model, it must be converted into fixed-point data type since the current version dose not support floating-point data type (2012b) [90]. In Simulink environment, a unit delay represents a register and can be used for synchronization and pipelining. Figure 4.7 shows pipelined arithmetics in Simulink. For convenience the reset, clock and enable signals are hidden in Simulink models. However these signals will be included in the generated VHDL code. Resource sharing in Simulink is demonstrated on figure 4.8.

To declare data type conversion and pipelining for hardware implementation of complex mathematical functions, Simulink model of the Park transformation is provided in appendix A.

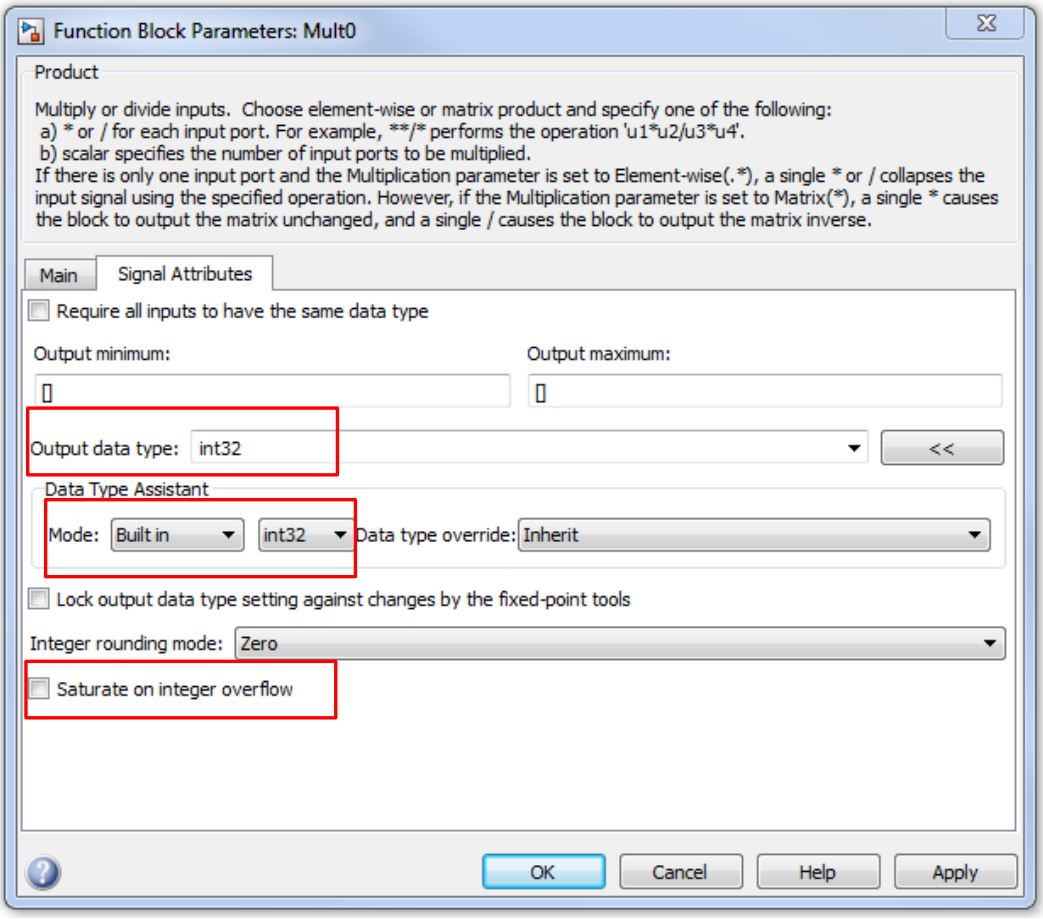

FIGURE 4.5: GUI of the multiplier block in Simulink

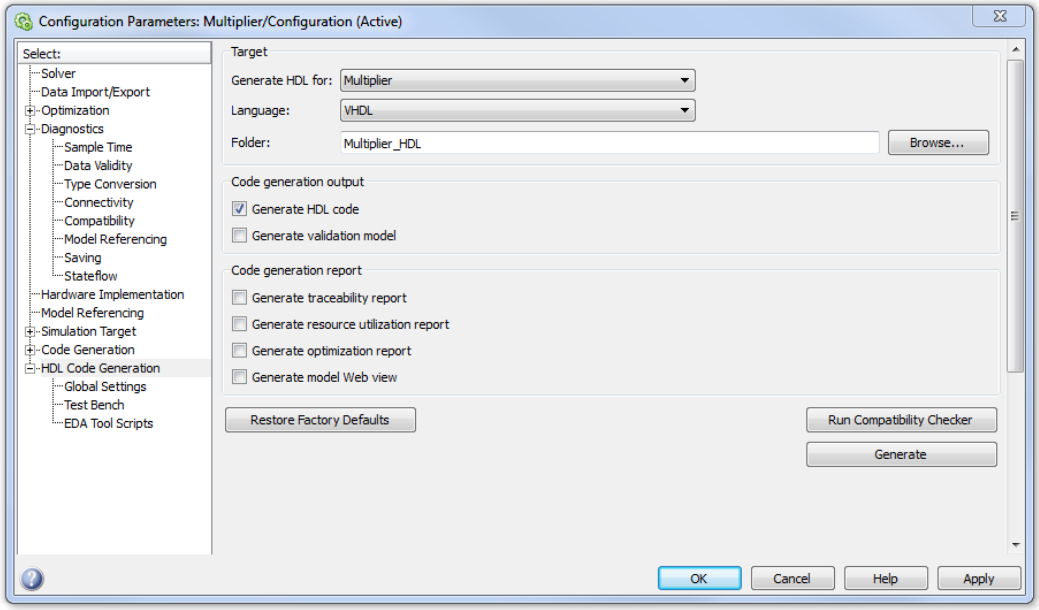

FIGURE 4.6: GUI of the *HDL coder*

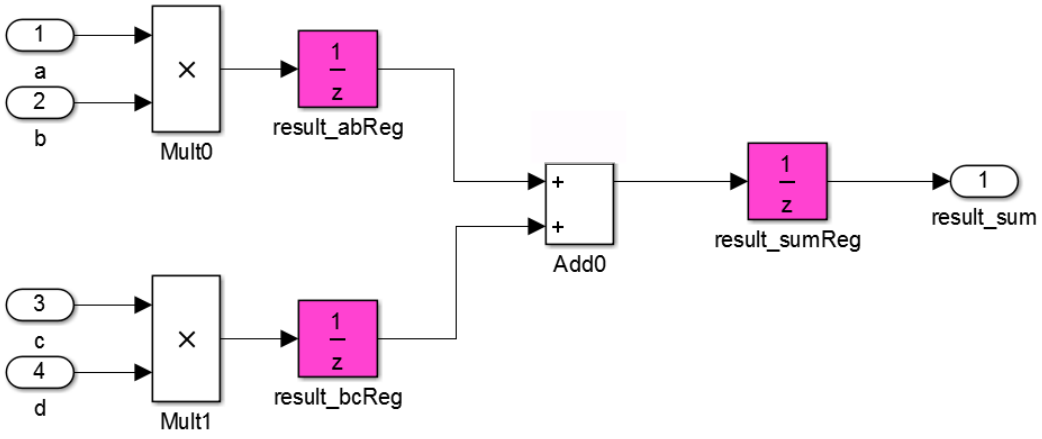

FIGURE 4.7: A pipelined design in Simulink

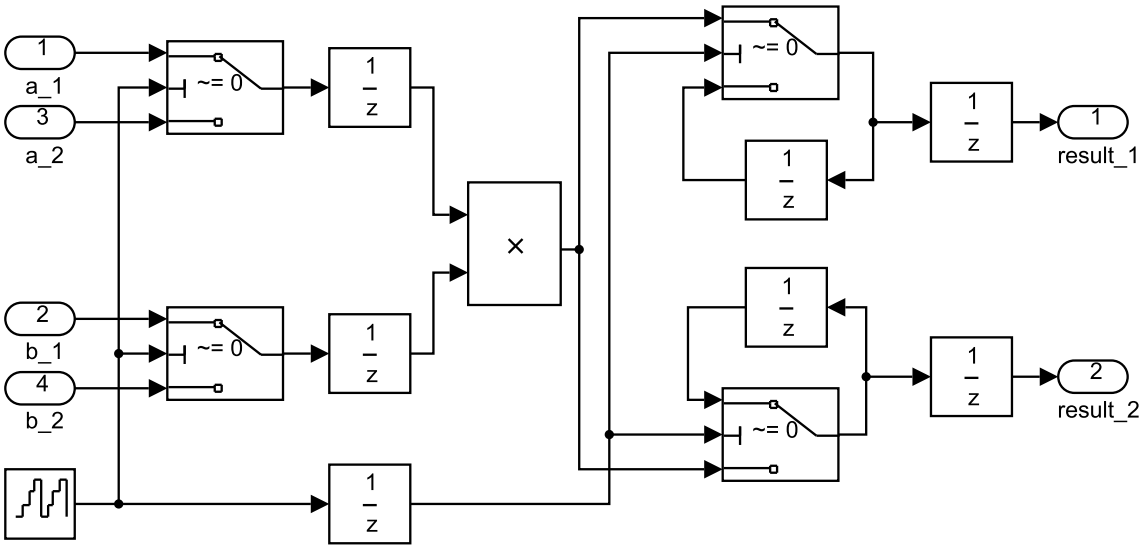

FIGURE 4.8: A shared multiplier in Simulink

4.4 FPGA-Based Design Methodology

FPGAs have emerged as solutions for many computationally intensive algorithms. There are several typical issues which must be considered for every FPGA-based design [74, 91]. The primary challenge for FPGA implementation of computational systems is determining whether FPGA is a right choice compared to other alternatives. In terms of computational performance, FPGAs are superior for most of applications with fixed computational tasks. The main reason is that FPGAs allow to dedicate the hardware architecture to the algorithms.

Depending on the design complexity different methodologies can be applied. The significant aims of a design methodology are minimizing the development time and utilizing hardware resources more efficiently [92]. Importance of these two issues may differ depending on the application. For mass production, the hardware cost plays a more important role while for low quantity production and prototyping, the development time has a bigger impact on the end product cost. The typical design process is described on figure 4.9.

4.4.1 Selection of Target FPGA Chip

The diversity of FPGA applications implies customizing logic resources to meet specific requirements [93]. Hard cores are the major elements for enhancing FPGA performance during manufacturing. Some FPGA chips are customized for use in the applications with high signal processing and arithmetic operations. These FPGA chips are provided with high number of embedded multipliers and DSP blocks.

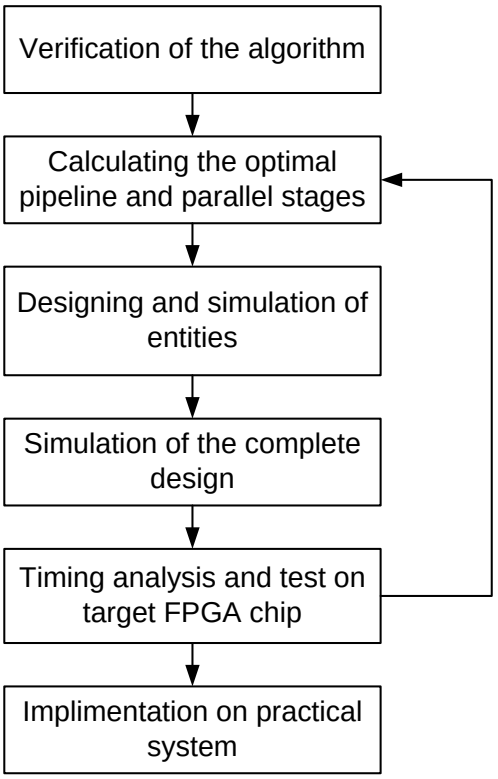

FIGURE 4.9: The FPGA-based design flow

4.4.2 Soft-Core IP

As the density of FPGAs increases, the challenge for the design reuse and the capability of synthesis at system level become more important. Considering a single chip with millions of programmable logic elements, the main design challenge is system level integration and a structural design reuse rather than performing low level optimization. By using parameterizable IP cores, the development time can be dramatically decreased.

Since in this thesis, we mainly rely on MATLAB/Simulink HDL code generation tool, instead of conventional IP cores provided for FPGAs, we create our own Simulink library to enable design reuse and expedite model development process. One example of the library block frequently used for the implementation of the control algorithms is the Park transformation. Its Simulink model optimized for VHDL code generation, is provided in appendix A.

4.4.3 Simulation and Verification Tools

Functionality validation of algorithms is possible in Simulink environment. For test of HDL models, Mathworks provides a tool, so-called HDL Co-Simulation. to ensure timing requirement of the design on target FPGA, TimeQuest Analyzer available within Quartus II [81] can be used.

For a more realistic test of HDL codes on the target FPGA we propose an FPGA-based hardware platform that allows Model-In-the-Loop simulation.

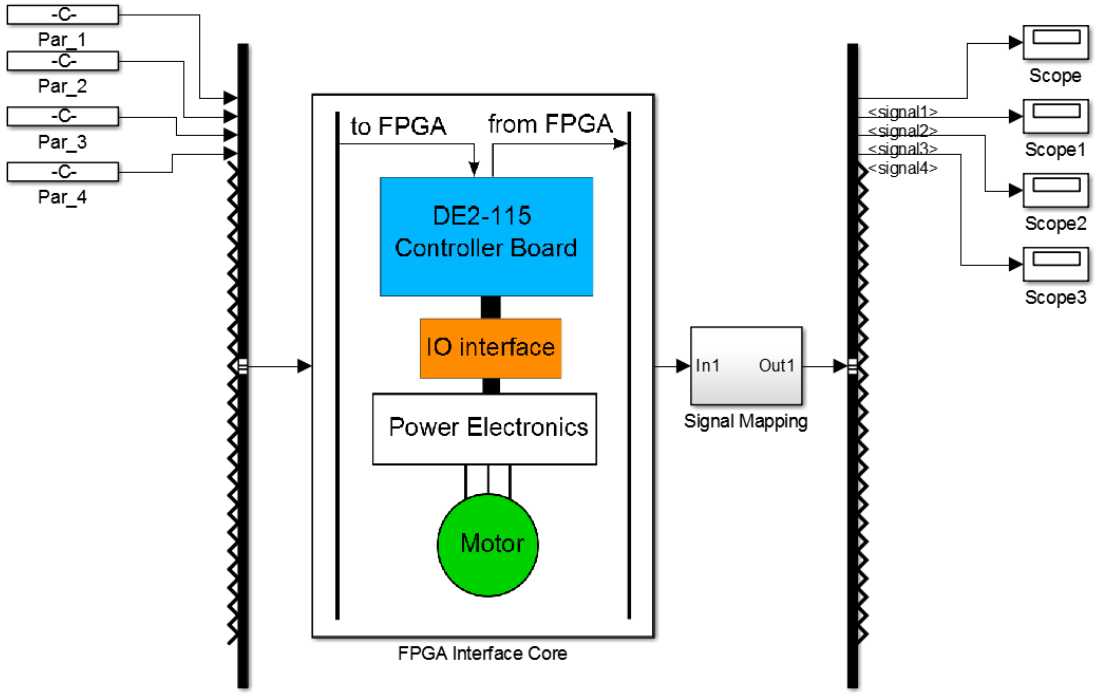

FIGURE 4.10: The Ethernet-based Simulink interface with the controller board for design verification and instrumentations

4.4.4 Experimental Tests and Instrumentations

Experimental test is the most important part of any control algorithms. Signal visualization and GUI (Graphical User Interface) in Simulink environment can be also exploited for instrumentation. In order to transfer experimental data to/from Simulink, we propose an Ethernet-based serial communication between MATLAB/Simulink environment and the FPGA control board. All data are sampled with an arbitrary sampling frequency up to 50 MHz and sent to Simulink. From Simulink also user commands and parameters can be sent to the control board during run time.

Since the personal computer is not capable of performing real-time tasks, data is stored on memory blocks available on the control board and transfered as soon as the PC is ready to accept the new packet. In this way, all samples are safely transmitted to Simulink for data playback and further processing purposes. A Screenshot of the Simulink model for FPGA-based Model-In-the-Loop and instrumentation is shown on figure 4.10. It will be further described in chapter 6 and 7.

4.5 Conclusion

High logic density and programmability of FPGAs represent extraordinarily high computational performance.

Thanks to parallelism and hardware implementation of all tasks, FPGAs are able to manage precise timing requirements which are necessary for real-time systems.

There are various tools available for development and verification of FPGA-based designs. MATLAB/Simulink is a graphical programming environment which enables to deal with complexity of FPGA-based development and allows to generate HDL codes from Simulink models.

For experimental test and instrumentation, the Ethernet-based communication interface, enabling to transfer manipulated parameters to the FPGA chip and sampled signals from the FPGA, is proposed. It provides state-of-the-art of instrumentation capabilities for FPGA based development at much lower cost. Thanks to high sampling rate of the FPGA, it can practically replace a commercial laboratory oscilloscope as well.

Chapter 5

FPGA Implementation of Electric Drive Models

5.1 Introduction

Hardware adaptability of FPGAs makes them powerful for real-time implementation of computationally intensive models with very short sampling time. The hybrid notion of modern electrical drives, which include both discrete and continuous parts, requires very short integration steps to properly model dynamic behavior of power electronics [94]. FPGA-based implementation of the electrical drive models allows to reduce the simulation time. Furthermore, it is useful for real-time Model-In-the-Loop tests of the FPGA-based control algorithms.

5.2 Data Type Conversion

Hardware reconfigurability of FPGAs allows to finely customize computational architecture and to efficiently utilize logic resources. Most important optimization parameters for mathematical computation on digital devices are data type and bit-width. Higher number of bits leads to larger logic area and consequently lower performance. Because of the significance of these parameters, HDLs support various data types and flexible choose of bit-width.

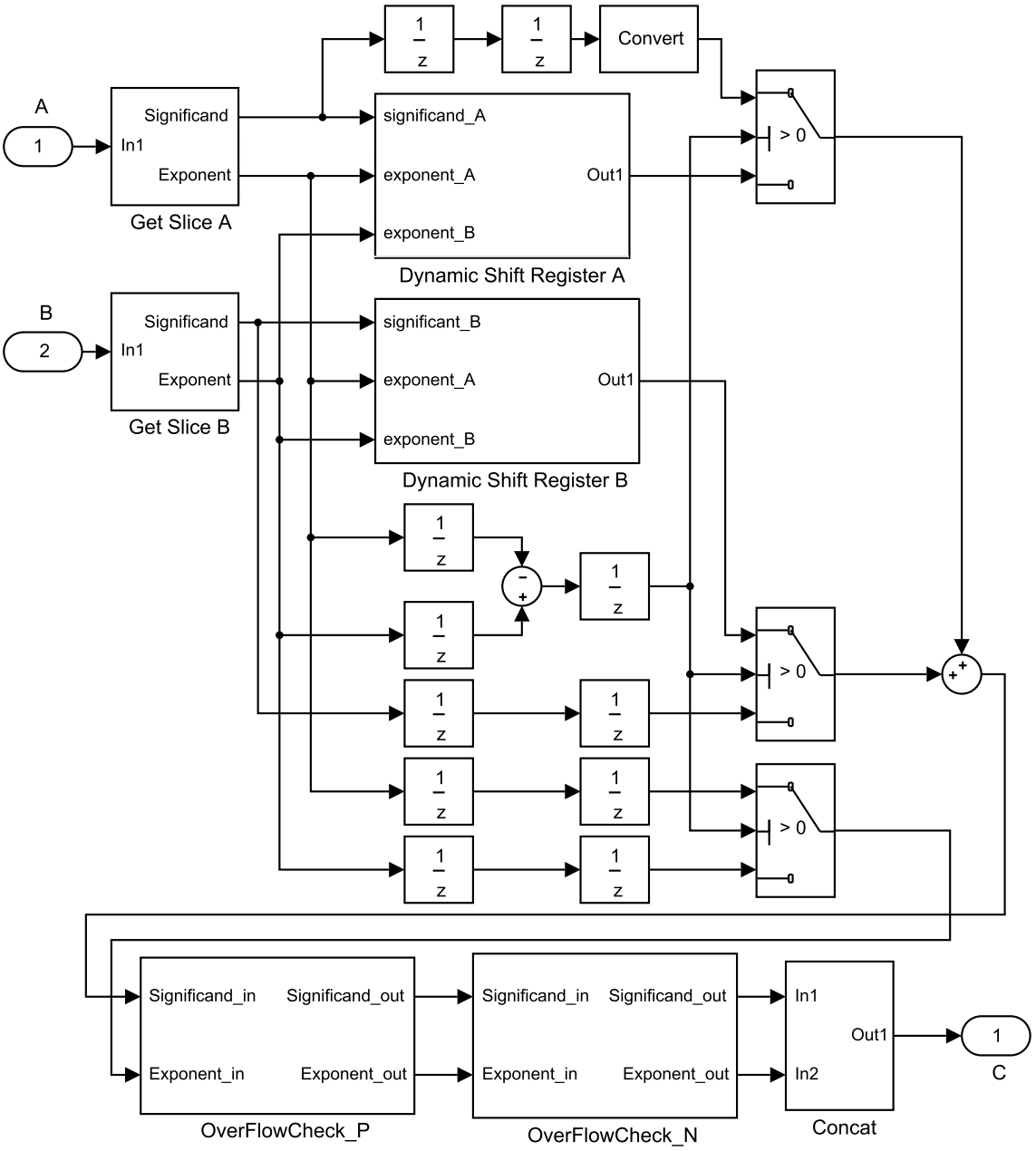

FIGURE 5.1: Hardware model of floating point addition in Simulink

5.2.1 Floating Point Data Type

Floating-point provides higher flexibility and programming convenience. In contrast to fixed-point, in floating point data type the scaling factor is dynamically set for each value in order to exploit highest possible precision. It is at the cost of higher computational burden. Floating-point arithmetics need more hardware resources [95]. In most of the modern digital processors the floating-point unit is intended to provide the programming flexibility. Although existing FPGA chips are mainly facilitated with embedded

hardware multipliers for fixed-point data type, in the future they will be more and more floating-point oriented to meet the requirements of the demanding applications [96–98].

Floating point implementation increases portability and scalability of the design [99]. FPGAs are often combined with processor in a hardware-software co-design. Since most of processors are facilitated with floating point units, a floating point FPGA simplifies the communication [100].

5.2.1.1 Performance of Floating Point Arithmetic on FPGAs

Hardware model of the floating-point adder is depicted on figure 5.1. This model is optimized so that it can be converted into VHDL code by means of the *HDL coder* tool. As the model shows, implementation of the floating-point operations needs extra logic resources. It leads to larger design and higher clock latencies compared to the fixed-point. Pipelining technique can improve performance of the floating-point blocks. However, depending on the FPGA architecture it is not always possible to achieve best routing possibility to maintain the demanding performance.

Hardened floating-point arithmetics which are well optimized during manufacturing, can provide higher performance compared to a soft implementation using configurable logic blocks of the FPGA. At the moment a few families of FPGAs are provided with hardcore floating-point arithmetics. Because of the importance it is anticipated that in the future more FPGA chips will be facilitated by embedded standard floating-point arithmetics.

5.2.2 Fixed Point Data Type

Fixed-point data type is a way to represent fractional values usually in base two [101]. The reason is that digital computation is established on base 2 and multiplication and division by two can be simply realized using shift registers.

A value in fixed point must be necessarily an integer. Therefore for representation of the fractional part, a scaling factor is required. Precision of fixed-point numbers depends on the bit-width and correct choice of the scaling factor. For conversion of signals with a known range of variation, the best possible scaling factor can be easily calculated. Due to the limit of available bit-width for processing and storage of data, usually losing some information is unavoidable. Nevertheless, such precision is, in most cases, acceptable. Thanks to the simple hardware model and higher performance, more often fixed-point data type is preferred, unless floating-point hardened arithmetic blocks are available.

5.3 FPGA-Based Model of Induction Motor

Capability of FPGAs for parallel processing and their flexibility to be dedicated to various hardware topologies make them suitable for implementation of computationally complex models with high sampling frequencies in real-time.

While control algorithms are implemented on FPGA, an FPGA-based model of electrical drives is necessary for the Model-In-the-Loop simulation.

5.3.1 Generalized Procedure for FPGA-Base Real-Time Modeling

Assume that the dynamic plant is described with the following set of nonlinear differential equations:

$$\dot{x} = A(t) \cdot x + b(t) \cdot u \tag{5.1}$$

where $\dot{x} = [\frac{dx_1}{dt} \ \frac{dx_2}{dt} \ ... \ \frac{dx_n}{dt}]^T$ is the derivative of the state variable vector, $x = [x_1 \ x_2 \ ... \ x_n]^T$ vector of state variables, $A(t)$ the state matrix, $b(t)$ the input matrix, and u the input vector.

Using the forward Euler technique difference equations can be derived:

$$\dot{x} = \frac{x(k+1) - x(k)}{T_s} \tag{5.2}$$

$$x(k+1) = x(k) + T_s \cdot A(k) \cdot x(k) + T_s \cdot b(k) \cdot u(k) \tag{5.3}$$

Representation of the system of difference equations can be further simplified as:

$$x(k+1) = x(k) + \tilde{A}(k) \cdot x(k) + \tilde{b}(k) \cdot u \tag{5.4}$$

The block diagram of 5.4 is shown also on figure 5.2.

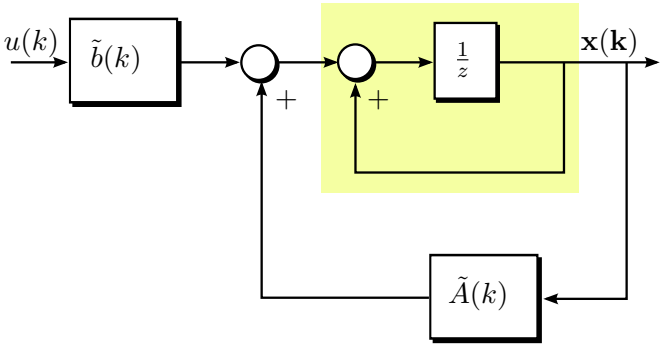

FIGURE 5.2: Hardware implementation of the difference equations

The objective of the FPGA implementation is to achieve very short integration step. While we rely on pipelining to improve the performance, the central part of the model should be realized only in one register stage to ensure the shortest integration path.

The term $\tilde{A}(k) \cdot x(k)$ represents back emf of the motor. Since it has a relatively lower dynamic compared to VSI, this part can be implemented with a longer sampling time.

To meet the timing constraint without reducing the clock frequency, it is necessary to pipeline the computational model. Pipelining causes the computational delay which is negligible.

5.3.2 Inverter Model for Control

One of main contributions of FPGA implementation is that the simulation can be realized in real-time even for a very short sampling time. The integration step of the model is smaller than time constant of the power electronic switches. It allows including most of switching transient effects and pulse width modulation as well as the switching dead times. To avoid short circuit between upper and lower transistors, dead-time is added to the gate signals and during this time both transistors are turned off (see figure 5.3). It means no active voltage vector is applied. Depending on the direction of the current i, one of anti-parallel diodes conducts and determines the output voltage.

5.4 Conclusion

In this chapter the focus was on the application of FPGAs for real-time modeling of electrical drives systems. Extremely high computational performance of FPGAs gained

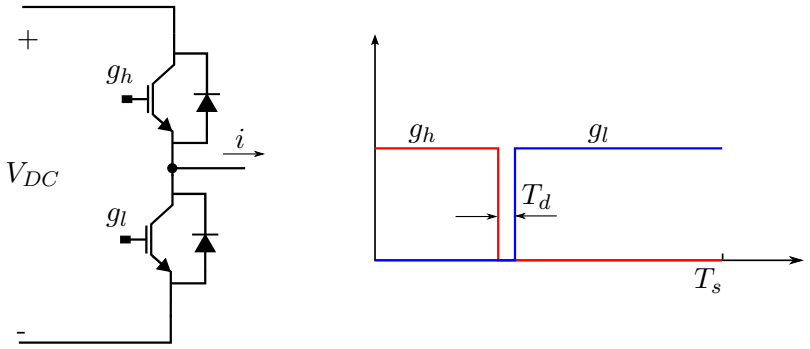

FIGURE 5.3: One leg of VSI switches.

from the parallelism and distributed architecture, enables to obtain a very short sampling time and ensure more realistic simulation.

The data type has a significant impact on the computational performance of the hardware model. Floating-point data type reduces the complexity of the implementation since no effort is needed for conversion and scaling. However, the floating-point arithmetics require more logic resources. The further drawback of floating-point operations is the higher number of clock latencies.

Even though available logic resources on FPGAs allow to realize all arithmetics in floating point, for the implementation of the mathematical model of electrical drives, the fixed-point data type shows better performance and requires less CLB and DSP blocks.

A generalized approach for real-time modeling of electrical drives is investigated. It enables to build the hardware model from the standard differential equations, maintaining short integration steps.

Chapter 6

FPGA-Based Hardware-In-the-Loop Simulation

6.1 Introduction

During and after production of control devices, several tests must be done to verify correctness of both hardware and software functionalities. Majority of such tests are not reproducible or it is not cost effective to perform them on real drives. Hardware-In-the-Loop (HIL) systems are hardware emulators that replace real plants in most of operation and failure conditions [102].

Depending on the Device Under Test (DUT), different specifications are needed for the HIL system [103–105]. Electrical drives consist of several parts: control unit, power electronics, electrical motor and mechanical load. For verification of the drive under particular load condition, the HIL must replace the mechanical load, whereas to confirm operation of the power electronic part, the electrical motor must be emulated [102]. Here the focus is on the signal level HIL mainly for verification of the control unit.

6.2 Hardware Architecture

As shown on figure 6.1, the real-time model can be used for the Model-In-the-Loop (MIL) simulation. In MIL scheme the controller and the drive model are implemented on the same FPGA. However, unlike the offline simulation, it is in real-time operation and enables to verify control algorithms on the target FPGA.

Distinction of HIL is that it enables to test the control algorithm as well as the hardware including signal conditioning interfaces (see figure 6.2).

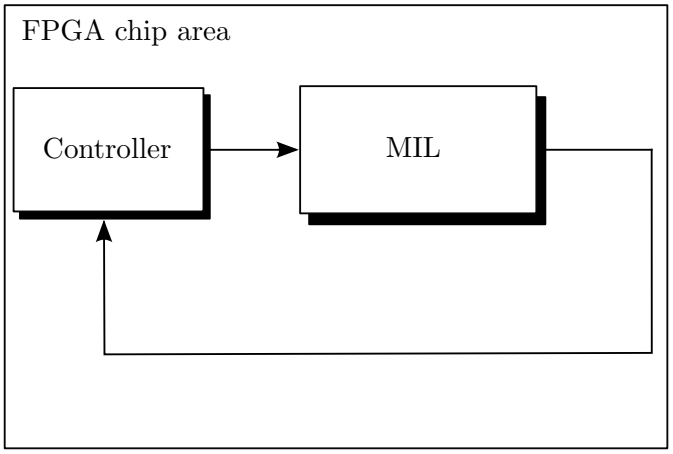

FIGURE 6.1: FPGA-based MIL system

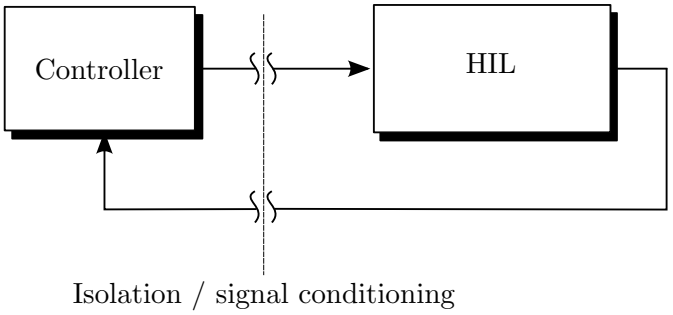

FIGURE 6.2: The HIL structure

Since we are going to realize a HIL simulator at signal level, some signal conditioning analog and digital components are required to connect the HIL with the controller. The simplified schematic of the HIL board is represented on figure 6.3. DAC interfaces are used for generating voltage and current measurement signals.

The Ethernet interface is intended on the board to realize high speed communication of the HIL with the host PC. The high speed Ethernet communication allows instrumentation and on-line parameterization.

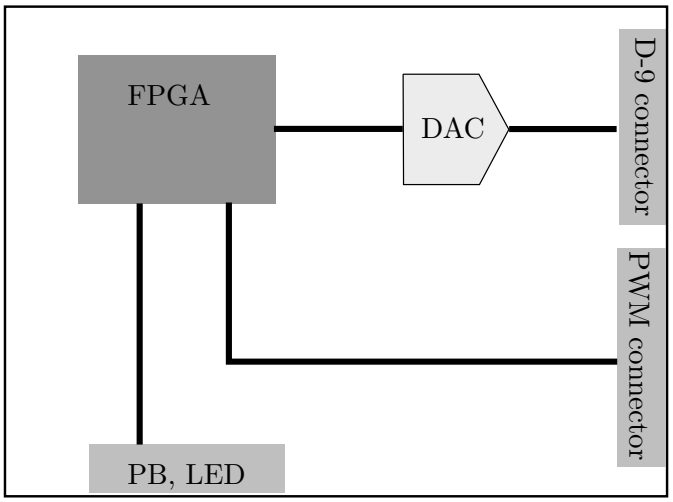

FIGURE 6.3: HIL schematic

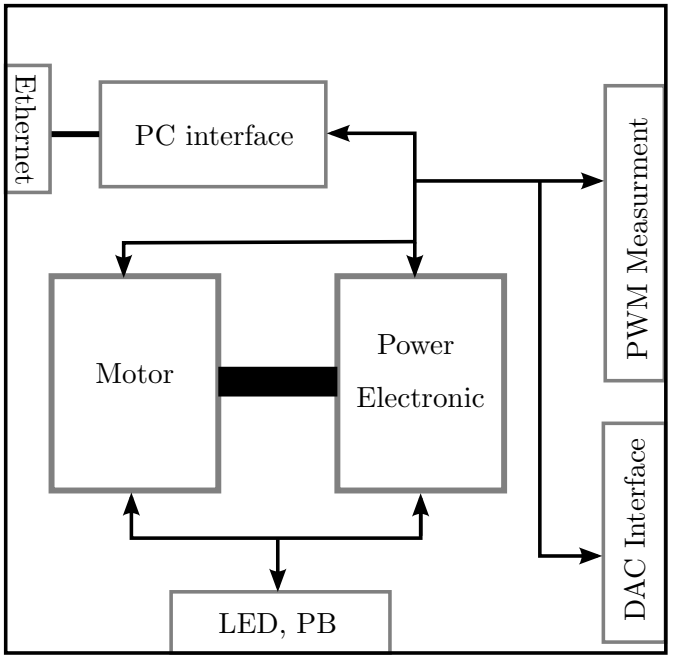

FIGURE 6.4: Hardware structure of the FPGA model

6.2.1 FPGA Model

Figure 6.4 shows top entities of the FPGA model of the proposed HIL system. The high speed communication within FPGA, is mainly done through the high bandwidth point-to-point connections. However, for the lower speed communication, the bus system is used to save logic resources. The real-time model of the motor and the power electronic converter is developed in Simulink and VHDL codes are obtained by means of *HDL coder*, whereas the DAC interface and Ethernet controller are developed directly in VHDL.

6.3 Host PC Interface

A very important part of a Rapid Control Prototyping (RCP) platform, is the user interface. We rely on an ordinary Personal Computer (PC) with Windows and MATLAB/Simulink. On the host PC, Simulink is used to visualize and analyze the results or set parameters of the HIL model running on the FPGA. The system has several parts as shown on figure 6.5.

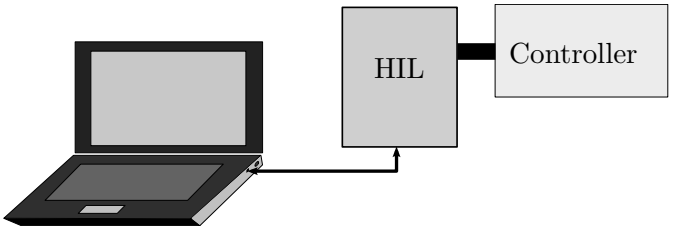

FIGURE 6.5: HIL structure

6.3.1 Ethernet Communication Interface

For communication between the host computer and the HIL, a serial interface is designed that can sample and store data within FPGA and send it to the PC. One of the advantages of using FPGA, is its flexibility to implement various communication standards. The Ethernet is a high performance serial interface frequently used for networking. There exist several Ethernet protocols. Here we use UDP communication protocol which is also supported by MATLAB.

The bandwidth of the Ethernet is sufficient to transfer data with MATLAB on-line for tuning the parameters and observation of the slow changing variables. However for higher sampling rate for instance oversampling up to the FPGA main clock frequency, it is not sufficient. To ensure that no data will be lost due to the lower communication rate, we utilize on-chip and on-board memory blocks to store the sampled data. The Ethernet controller then transfer samples as soon as the PC is ready to accept a new data packet. Therefore no data is missing while sampling with very short intervals.

6.3.2 MATLAB Interface

MATLAB/Simulink is widely used for model-based design and graphical programming. Since the control algorithms are developed in Simulink, it makes the experimental verification of algorithms easier while it is also used for instrumentation and analysis of experimental data. Bringing the experimental results in the Simulink environment enables to employ MATLAB tools for further analysis.

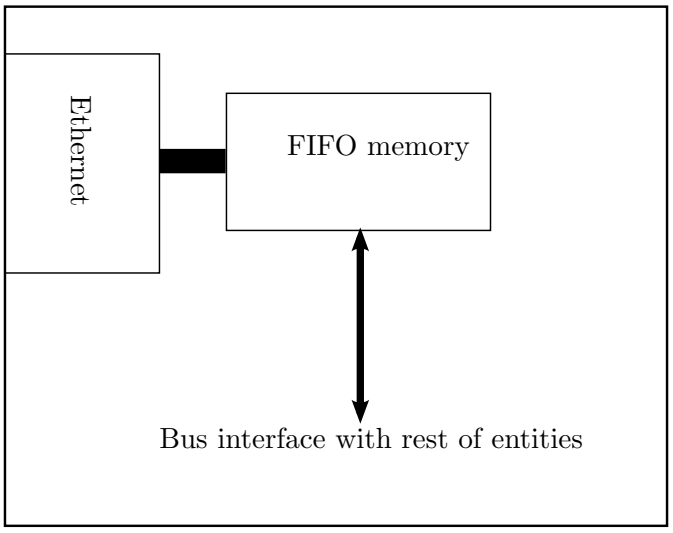

FIGURE 6.6: Ethernet controller in the FPGA

6.4 Conclusion

In this chapter the full FPGA-based HIL system is introduced. It enables to emulate behavior of power electronics, electrical motors and measurement boards, at signal level. The proposed platform is simple, cost effective and flexible. It can be utilized for education and research on power electronic as well as drive control applications.

The high computational performance and unlimited hardware reconfigurability of FPGAs, make them interesting for HIL applications. Thanks to the hardware programmability various standard or custom communication interfaces can be realized. It allows to implement high performance analog-digital conversion and signal conditioning circuits, which are essential parts of any HIL simulators. To realize a robust data playback on the host PC even at extremely high sampling rate up to the FPGA main clock frequency, on-board as well as on-chip memory blocks are utilized. It underlines further advantage of using FPGAs for control and HIL applications, which is the high performance memory interface.

Chapter 7

Experimental Setup

7.1 Introduction

A very powerful way for early evaluation of control algorithms is the computer-based simulation. As investigated in the previous chapters, simulation can be extended to real-time HIL models which can verify not only control algorithms but also the control hardware. Simulation-based validation is, however, limited to the correctness of the mathematical model of the physical plant. Since all phenomenas in a real plant can not be modeled, an experimental setup is also needed for a comprehensive evaluation of the algorithm as well as the control hardware. This chapter introduces the experimental setup which will be used for the practical implementation of the algorithms being proposed in the next part of this thesis.

A platform for research and prototyping must be facilitated with high computational power so that it dose not require so much code optimization effort. Due to the high number of applications for industrial digital control and the demand for prototyping new methods, there are many commercial solutions which offer a user friendly developing environment and expedite development and verification of control algorithms. High cost is, of course, one of the problems which make them not affordable for many education and research institutes. Commercial systems usually do not provide state-of-the-art in terms of digital hardwares. The reason is their long development time. Due to that, it takes longer time for industrial rapid prototyping solutions to keep up with the latest computational technology. Because of these reasons some time is devoted to the design and realization of the custom rapid prototyping platform for the current research. In this chapter the high performance FPGA-based experimental setup is discussed which can be programed either directly using standard HDL languages (to ensure portability of the HDL models) or using the graphical programming environment of MATLAB/Simulink.

Since the first commercial FPGAs, their logic density and operation clock frequency have been significantly increased. At the same time, the price of FPGAs has been decreasing. Although the first generation of FPGAs have addressed very specific applications only, now they can overcome conventional DSPs even in many cost sensitive applications such as motor control. Furthermore, hardware programmability of FPGAs allows to gain very high computational performance by dedicating the hardware architecture to the algorithm. Therefore for many applications with high computational complexity increasing attention is paid to the FPGAs.

A simplified schematic of the experimental setup is depicted on figure 7.1.

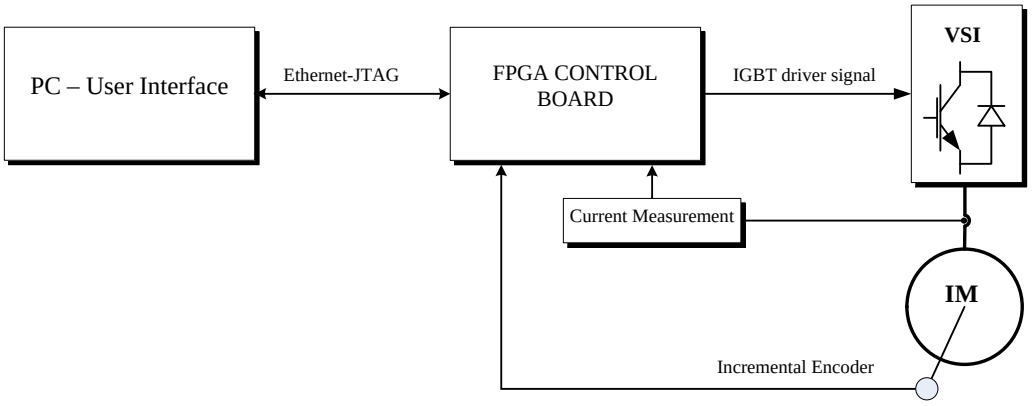

FIGURE 7.1: Block diagram of the experimental setup

7.2 Controller Board

Computation engine on the control board is the Cyclone IV Altera FPGA chip [66]. For research and education many development boards with all required components for building desired digital systems are available. There are tens of development boards from Altera with the same I/O standard and connectors. Therefore, relying on the development board enables to create a universal and upgradeable platform. In the previous chapters we introduced the FPGA-based digital control system developed with various functionalities for Hardware-In-the-Loop test and real-time simulation. It is relied on the same board to design the controller. The controller board is connected to the measurement, encoder and power stage boards through the interface board. The main board is DE2-115 accommodating the Cyclone IV FPGA chip [2]. Here we just describe the interface board designed and manufactured at our lab in particular for this project.

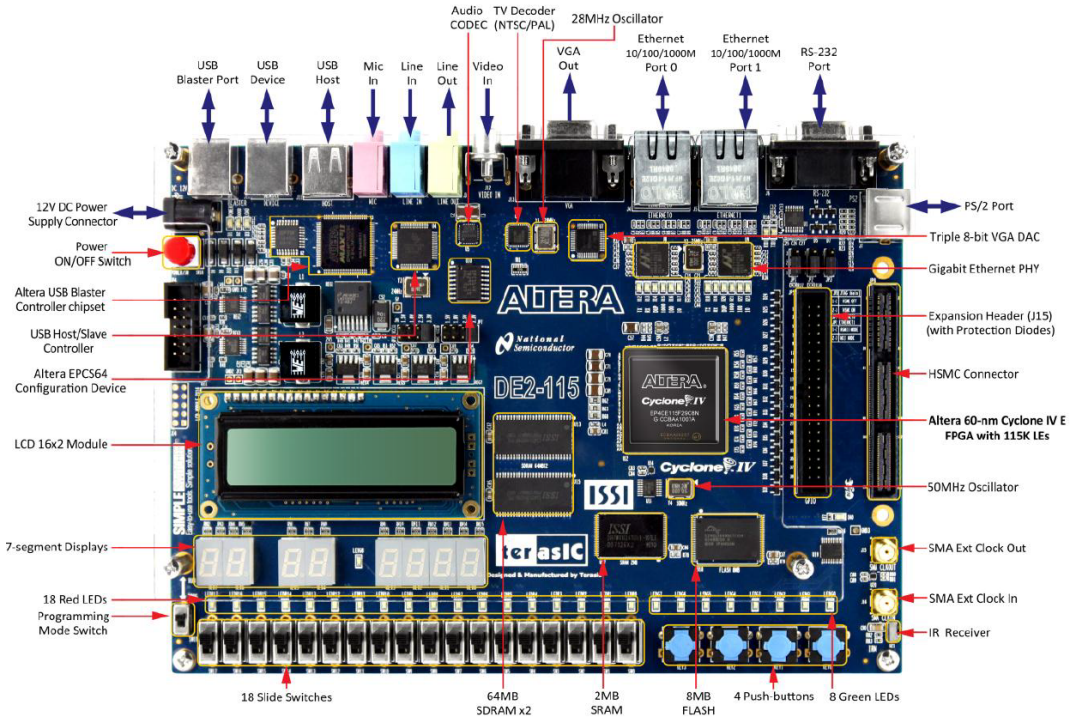

FIGURE 7.2: DE2-115 FPGA development board [2]

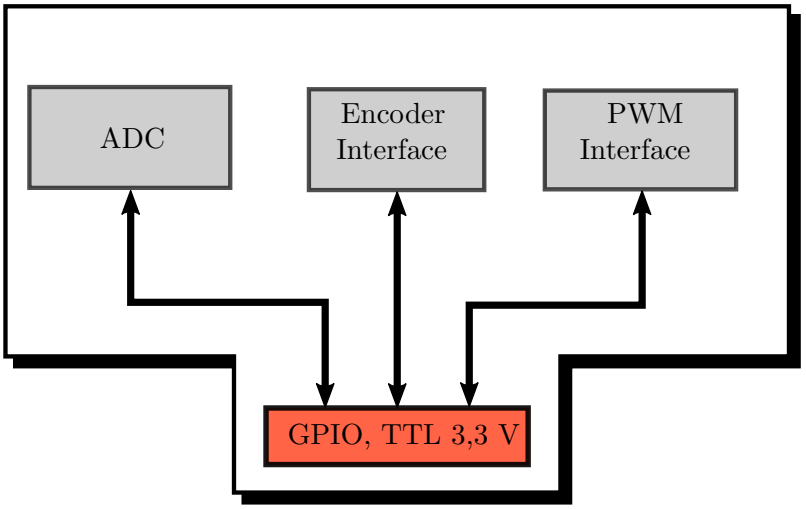

FIGURE 7.3: I/O board for the FPGA main board designed to connect to the 40-pin expansion header

7.2.1 Analog-Digital (AD) Converter

The voltage and current measurement boards provide analog signals which must be converted into digital. Since these signals are utilized as feedbacks in the control loop, dynamic performance and bit-resolution of the AD converters have a big impact on the performance of the controller. An AD converter of Texas Instrument is chosen [106]

providing high bit-resolution and sampling rate. It is sufficient to perform oversampling. As it is demonstrated in [107], oversampling can significantly improve quality of the current measurement. The main features of the AD converter are listed here:

- 2.7-V to 5.5-V Analog Supply, Low Power: 15.5 mW
- 1 (MHz) Sampling Rate
- Excellent DC Performance
- 16-bit Resolution
- Excellent AC Performance at $f = 10\ (kHz)$
- Built-In Conversion Clock (CCLK)
- Unipolar Input Range: $0\ (V)$ *to* V_{REF}

7.2.1.1 SPI Communication

High performance serial communication capability of the FPGA, allows to connect many peripherals with various communication protocols. A well known peripheral interface for point-to-point data transmission is SPI (Serial Peripheral Interface). A hardware configuration program is developed which allows to configure the AD chip and read the converted data via SPI. The VHDL code is provided in the appendix B.

7.2.2 Encoder Interface

The speed and position of the motor can be measured by different types of sensors depending on the accuracy and dynamic required for the control system [108]. A well known position sensor is optical encoder that converts the position information into bit stream signals. Since the output signals of the encoder are differential, they should be first converted into single ended TTL signals. For the encoder interface the differential driver is implemented on the board [109].

7.2.2.1 Speed and Position Calculation

The encoder provides 3 signals: A, B and Z. In each rotor revolution, 1024 pulses of A and B signals are produced, which are 90 degree out of phase. Combination of A and B gives a two-bit signal describing state of the encoder. In positive direction the state increments as enumerated on the figure 7.4, while in negative direction, it decrements.

Therefore the direction of the rotation can be simply detected. On the line Z, the encoder generates one pulse per mechanical revolution of the rotor, so that an initial position can be defined for the encoder.

To decode these signals into speed and position information, duration of the encoder pulses should be measured. A Simulink model is developed which can be directly converted into VHDL code by means of the *HDL coder*. It includes also a spike filter to eliminate the environment noises. In addition, the speed signal is further smoothed using a digital low-pass filter.

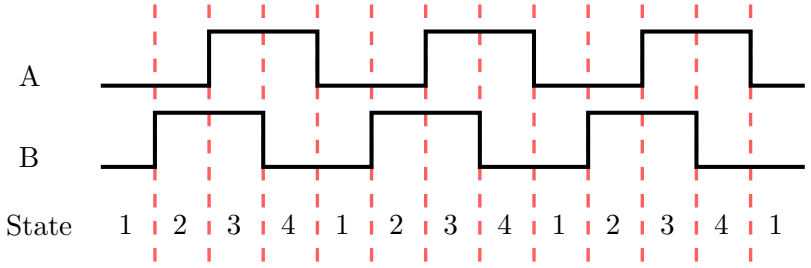

FIGURE 7.4: Signals of an optical encoder

7.3 Measurement Board

The measurement board accommodates current and voltage sensors which are needed for measuring currents of the inverter as well as the DC link voltage as it is indicated on figure 7.5.

7.3.1 Current Sensors

The quality of current sensors directly affects the controller performance. Currents can be measured by means of Hall effect sensors. Hall effect sensors provide high bandwidth satisfying the requirements of the current control loop.

7.3.1.1 Hall Effect Sensor

The output current of the Hall sensor is proportional to the measured current. The input and output currents are physically isolated from each other as they are just connected through the magnetic field. Due to that it does not need additional isolation and the

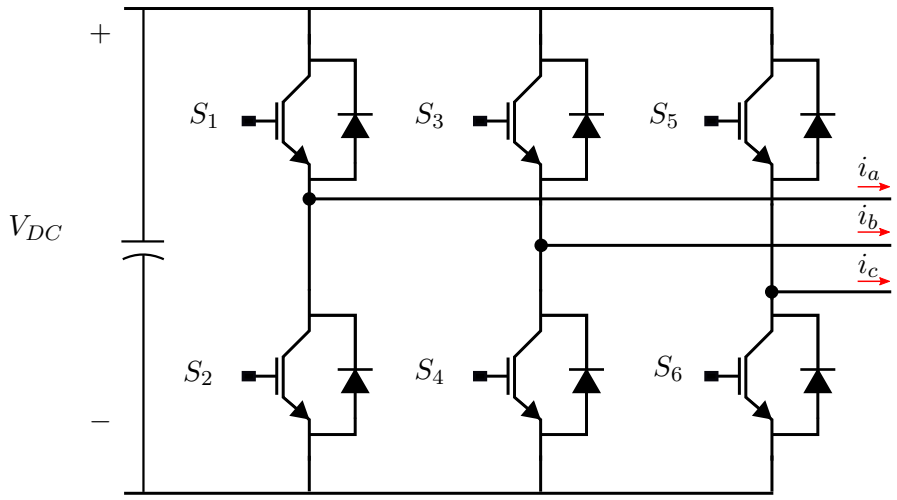

FIGURE 7.5: Current measurement for the VSI

output signal can be directly connected to the control board. According to [110] hall effect sensors have the following characteristics making them appropriate for the current control loop:

- Wide frequency range
- Good overall accuracy
- Fast response time
- Low temperature drift
- Excellent linearity

7.3.2 DC-Link Voltage Sensor

The voltage signal can be measured by the resistance divider circuit. As shown on figure 7.6, the V_{out} is proportional to the input voltage at much lower level. However a proper isolation is needed to avoid possible damage of the control board. This can be done through the isolation amplifier which is demonstrated on figure 7.7. More technical information about the isolation amplifier can be found in its data sheet [111].

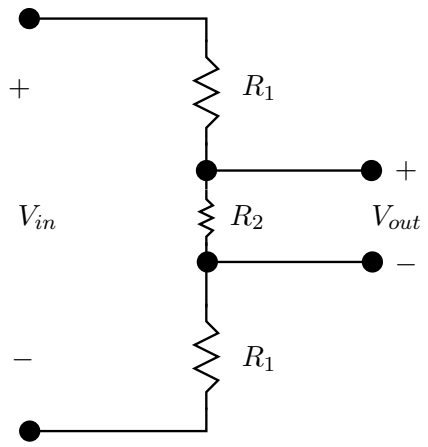

FIGURE 7.6: Resistive divider circuit for the voltage measurement

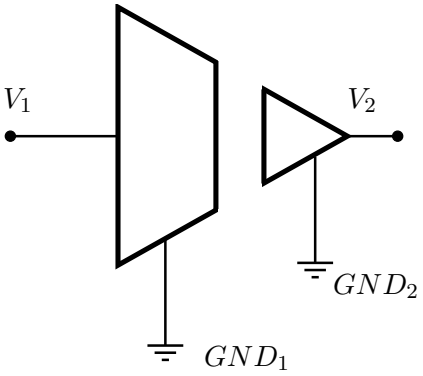

FIGURE 7.7: The isolation amplifier

7.4 Electrical Machines

Focus of the current research is mainly on the control of AC electrical motors. Two types of AC motors being studied here are Permanent Magnet Synchronous Motor (PMSM) and Induction Motor (IM). Experimental investigations of the proposed control strategies are done on the commercial electrical motors. On the laboratory setup two motors are coupled together so that one of them can be used to produce mechanical load torque. The load drive is controlled by the commercial inverter whereas the test motor is controlled by the proposed FPGA-based controller. Parameters of the electrical motors are provided in appendix F.

7.5 The Power Stage

The power stage is connected to the grid trough a controllable three-phase transformer.

Voltage Source Inverter

Figure 7.8 depicts the three-phase VSI consisting of the IGBT switches and diodes. To switch off (on) each IGBT, it needs the gate control signal which is generated by the control board.

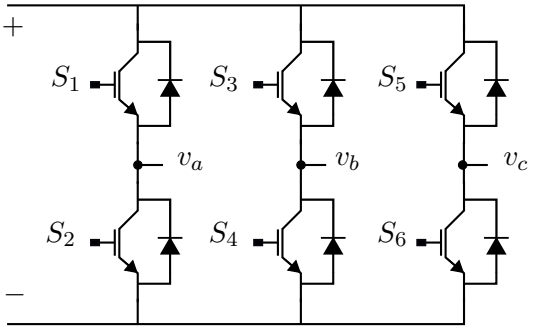

FIGURE 7.8: The 3-phase voltage source inverter

Intelligent Power Module (IPM)

The 2-level voltage source inverter is assembled using the IPM manufactured by Mitsubishi. The six IGBT switches are integrated into one pack. The gate drivers are embedded making a compact solution [3].

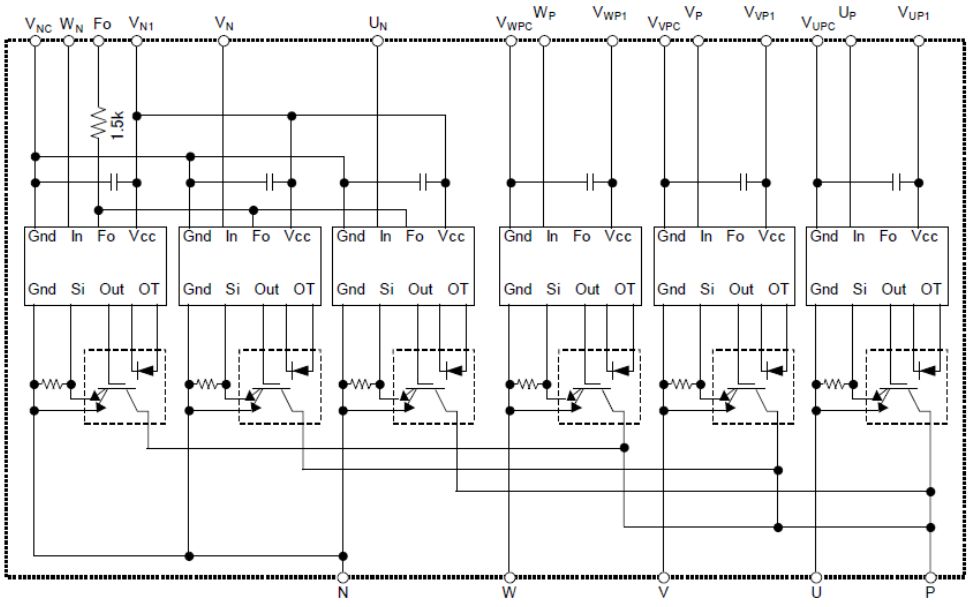

FIGURE 7.9: The schematic of the IPM (Source [3])

Isolation Board

The outputs of the controller board are gate signals for the power transistors. These signals should be connected to the IPM through a proper electrical isolation to protect the control board from high voltages. The isolation is realized using optocouplers as shown on figure 7.10.

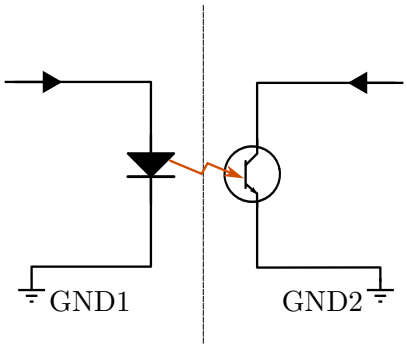

FIGURE 7.10: Isolation of gate signals through the optocoupler

Over-Current Protection

The over-current protection can easily be realized inside the FPGA. The function of the over-current protection is to switch off all of IGBTs when an over-current fault happens. The fault might occur due to the possible errors in the algorithm or a damage to the motor.

In addition the IPM provides the self-protection unit which switches off the faulty transistor and generates a fault signal for the control board. The fault signal can also be used to detect failures and switch off all transistors.

7.6 Conclusion

For verification of the control algorithms the experimental setup is designed based on the FPGA-based rapid prototyping platform. It is a cost effective setup providing high computational performance. Its main features are as follows:

- High computational performance
- Suitable for hand coding or using code generation tools

- It is provided with Ethernet-based host PC interface for visualizing and analyzing experimental results

Commercial solutions require high hardware and software costs which might be not affordable for the research institutions. On the other hand, the technology of digital computational devices is rapidly evolving and due to the long development time, commercial rapid prototyping systems usually are not facilitated with the state-of-the-art of the computational technology. In the proposed scheme each individual component can be upgraded to improve the performance of the setup, if needed.

A Simulink user interface is developed allowing to set the parameters of the control board and visualize the variables with high sampling frequency up to $50MHz$. These features make the experimental setup appropriate for the practical investigations.

Part III

Nonlinear Model Predictive Control

Chapter 8

Model Predictive Control

8.1 Introduction

Model Predictive Control (MPC) is an emerging control strategy established on the digital and optimal control theory [112]. The discrete notion of MPC makes it appropriate for implementation on digital controllers. Several algorithms for model predictive control have been developed [113, 114]. The concept of MPC, as its name suggests, is a model-based design strategy which is the special case of the optimal control techniques developed in 1960 and later [115]. MPC offers a systematic solution for the control problem of MIMO plants with some sort of nonlinearities such as constraints on the inputs and state variables.

In principle, any control method that optimizes the control action taking into account the predicted behavior of the controlled plant, can be considered as predictive control. For Linear Time Invariant (LTI) systems, there is a classical definition for MPC with the quadratic performance index which can be analytically solved:

$$J(k) = \sum_{i=0}^{H} \big(x(k+i)Qx(k+i) + u(k+i)Ru(k+i)\big) \tag{8.1}$$

where $u(k)$ and $x(k)$ are vector of inputs and state variables respectively, i time index, H prediction horizon, Q and R matrices of weighting factors.

Figure 8.1 shows the simplified view of the MPC controller. $J(k)$ is a scalar function determining the dynamic and steady state performance of the controller. $y(k)$ is the

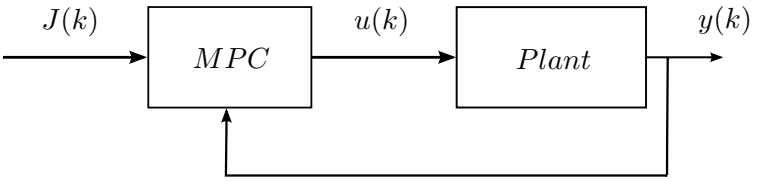

FIGURE 8.1: Abstract structure of MPC

vector of measurable outputs. Hence the MPC block includes an observer to estimate vector of the state variables $x(k)$.

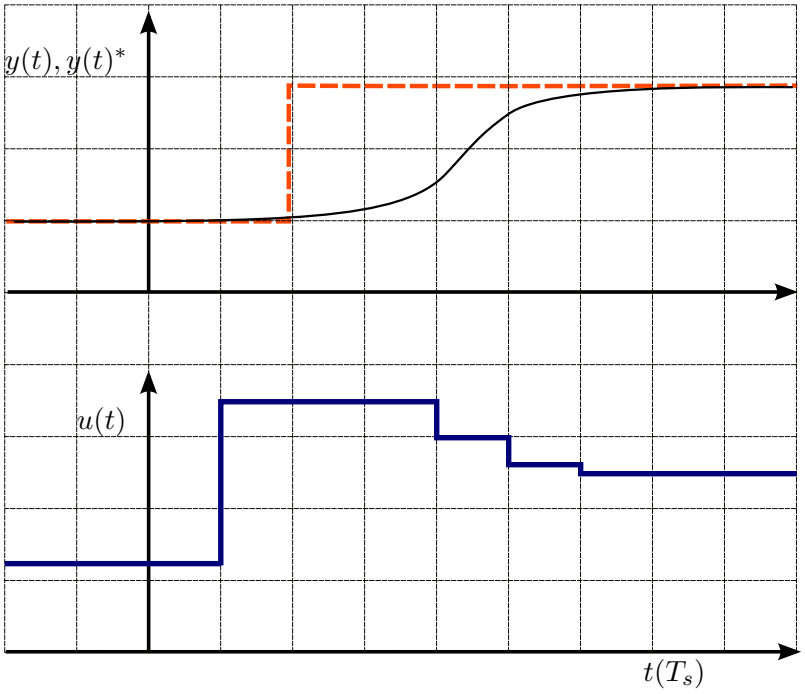

FIGURE 8.2: An example of continuous set MPC performance

The MPC has the following basic advantages:

- It provides simple principles for the design of the controller.
- The flexible cost function can easily include vast majority of the control objectives
- MPC enables to compensate the dead-times of different notions
- It is well suited to the MIMO systems
- MPC can effectively compensate the measurable disturbances

For some sort of nonlinearities, for instance the existence of constraints over the control actions and state variables, systematic approaches for solving the optimization problem

have been developed [114–116]. However these methods suffer, mainly, from the computational complexity and are not realizable in real-time. Computational burden is the main challenge for the practical implementation of most MPC algorithms.

8.2 Model Predictive Control for Power Electronic and Drives

The first publications on MPC for the electrical drive applications, are [117–119]. In these works, the predictive control principles ware investigated for the direct control of power converters, mainly to minimize the switching frequencies and losses.

Among the direct control strategies, DTC can be also categorized as a type of predictive control [120]. In DTC there is no well formulated predictive model for choosing the switching states, however, it relies on some sort of prediction of the drive behavior. Absence of the internal model for DTC restricts optimization capability of the controller compared to the MPC methods.

For quite a long time, there have been no major publications on MPC for electrical drive applications because of two reasons: MPC potential contributions were not broadly known to the drive industry and research community. The second reason was high computational burden of the optimization algorithms restricting them for the practical implementations. Investigation of MPC is refreshed by the review and reconsideration of MPC opportunities done in [120].

By advancing in power electronics and digital controllers, MPC became more attractive as a modern control strategy for the drive applications. There are some interesting contributions that practically realized MPC strategies to improve dynamic and static performance of the current control [41, 121]. Unlike the initial publications on MPC for electrical drives, in these works, a PWM-based approach is investigated to improve the dynamic performance of the controller rather than minimizing the switching losses. It shows that the drive industry may gain more from MPC than was expected.

Despite advances in the digital microcontroller technology, due to the computational complexity, classical MPC is still not feasible to be realized on industrial drive systems. Even with offline calculation capability of the Explicit Model Predictive Control, it still requires a huge computational engine and is not interesting for drive systems [122, 123]. The computational complexity is a major drawback of any digital controller since it implies additional hardware cost and reduces the reliability. Nevertheless it is not

the only challenge for the conventional MPC strategies. The basic MPC methods are established on linear models and just extension of them enables to include constraints. It degrades accuracy of the control for nonlinear systems such as electrical drives and power electronics.

Over the last decade, intensive efforts were devoted to another group of MPC strategies relying on discrete notion of power electronic devices. These methods can be considered as extensions to the direct control strategies initialized during 80's [12, 117, 124]. Two main concepts for direct control of converters can be distinguished: One approach is the so-called Finite Set Model Predictive Control (FS-MPC) proposed in [125]. Another MPC technique is the Model Predictive Direct Torque Control (MPDTC) [126].

FS-MPC and MPDTC share the idea of enumeration of the feasible switching states. Nevertheless they have a very important difference which is including the hysteresis band in MPDTC. It enables MPDTC to perform long horizon prediction and obtain outstanding results in particular for medium-voltage drives [127]. It shows the importance of combination of the MPC strategy with other control concepts to insert pre-optimization assumptions. Some recent publications on FS-MPC also investigate including calculated trajectories to decrease the computational effort of FS-MPC for longer prediction horizon [128].

8.3 Finite-Set Model Predictive Control

The simple concept of FS-MPC and its capability to handle constraints and multiple variable control problems, have encouraged many researchers to investigate it for vast majority of applications [129]. High dynamic of power electronic devices compared to the time constant of electrical motors allows to consider them as ideal switches (see figure 8.3) and neglect the switching transient.

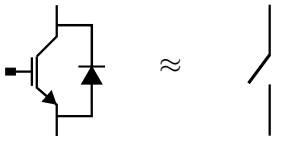

FIGURE 8.3: Power transistor as an ideal switch

In the three phase 2-level VSI, combination of states of the power electronic switches leads to 8 output voltage vectors. Assume that the switching state can be changed only at the sampling instants, behavior of the system over the next sampling interval can be predicted through the knowledge of its current state, disturbances and the new state

of the inverter switches. In FS-MPC the cost function is calculated for any change of switching state, so that optimal switching can be attained by minimization of the cost function.

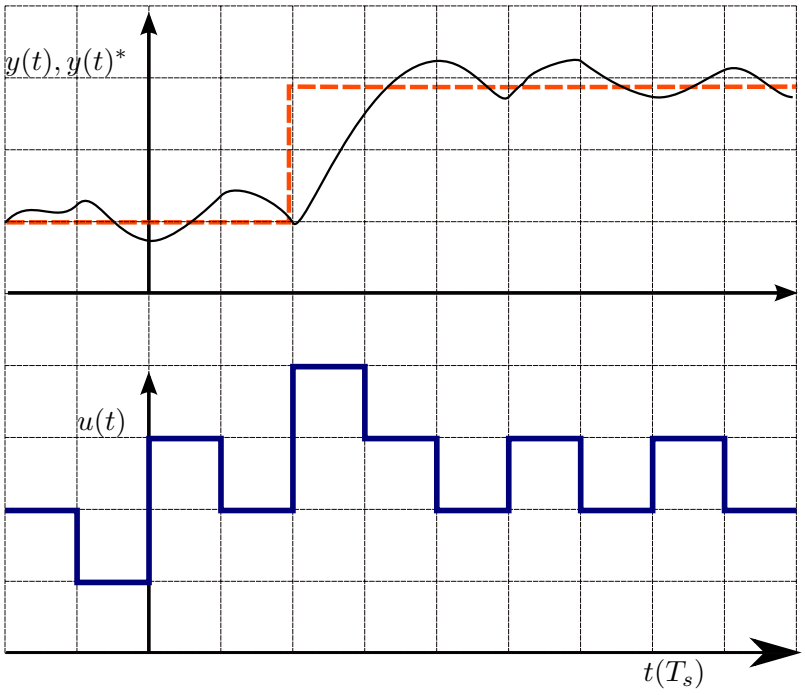

FIGURE 8.4: FS-MPC for discrete systems

8.3.1 Cost Function

The performance index in FS-MPC, as considered in most publications, is a scalar mathematical function reflecting all control objectives with respective weighting factors [129]. It defines the performance criterion for the drive control system.

$$J(k) = |x_1^p - x_1^{ref}| + |x_2^p - x_2^{ref}|...|x_n^p - x_n^{ref}| \tag{8.2}$$

8.3.2 Prediction Model

Minimizing the cost function requires a mathematical model of the plant being able to predict its future behavior. Using the mathematical model enables to include available information in the optimization process. In particular for the drive and power electronic systems, a mathematical model can be developed by means of the differential equations. Using Euler approximation, prediction model can be extracted from differential equations of the plant:

$$\frac{dx}{dt} \approx \frac{x_{k+1} - x_k}{T_s} \tag{8.3}$$

8.3.3 Constraints in FS-MPC

Constraints exist almost in all practical systems. The important advantage of FS-MPC is its flexibility to include different terms in the cost function.

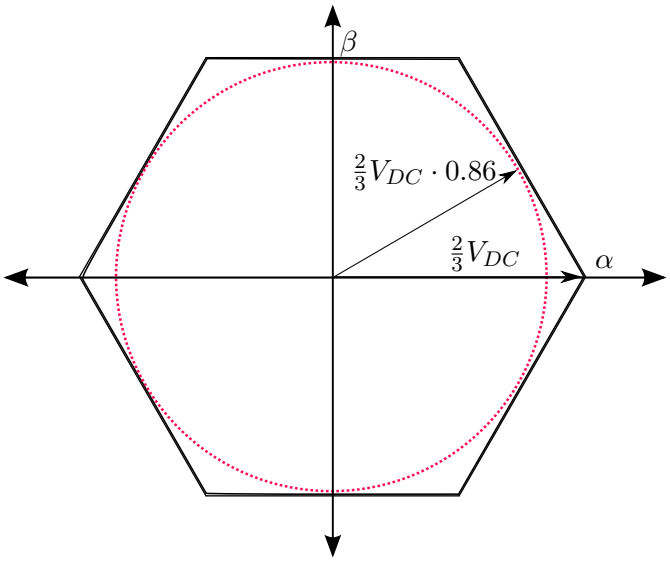

FIGURE 8.5: Voltage constraints of the voltage source inverter

Only voltage vectors which are inside the hexagon shown on figure 8.5 can be generated by the inverter. For a sinusoidal output voltage, the magnitude of the voltage is limited to $\frac{2}{3}V_{DC} \cdot 0.86$. It leads to the following constraint:

$$\sqrt{u_\alpha^2 + u_\beta^2} \leq \frac{2}{3}V_{DC} \cdot 0.86 \tag{8.4}$$

For operation on the overmodulation region the whole hexagon area must be considered [17]. In FS-MPC the voltage constraint is inherently included since only feasible voltage vectors are taken into account in the optimization.

In addition to the limits of the control, there are also constraints referring to the state variables such as currents, fluxes and the torque. To avoid saturation as well as over current problems, magnitude of the stator currents must be kept within the allowed band.

$$I_{max} \leq I_{lim} \tag{8.5}$$

$$T_{max} \leq T_{lim} \tag{8.6}$$

To maintain the constraints over the state variables, penalty terms can be imposed into the cost function to exclude all control actions violating the constraint conditions [129].

$$g_c = \begin{cases} 0 & |x^p| < x_{max} \\ \infty & \text{else} \end{cases} \tag{8.7}$$

8.4 FS-MPC for Induction Motor

Depending on the switching states of the power electronic switches, the voltage source inverter feeding the induction motor is capable of producing a finite number of voltage vectors. In the direct control methods such as DTC [12] and FS-MPC, [125], the output variables can be controlled by the appropriate selection of the switching states. In DTC, the switching states are selected according to the lookup table whereas in FS-MPC the performance index is used to calculate the optimal voltage vector. The cost function enables to include secondary control goals [130].

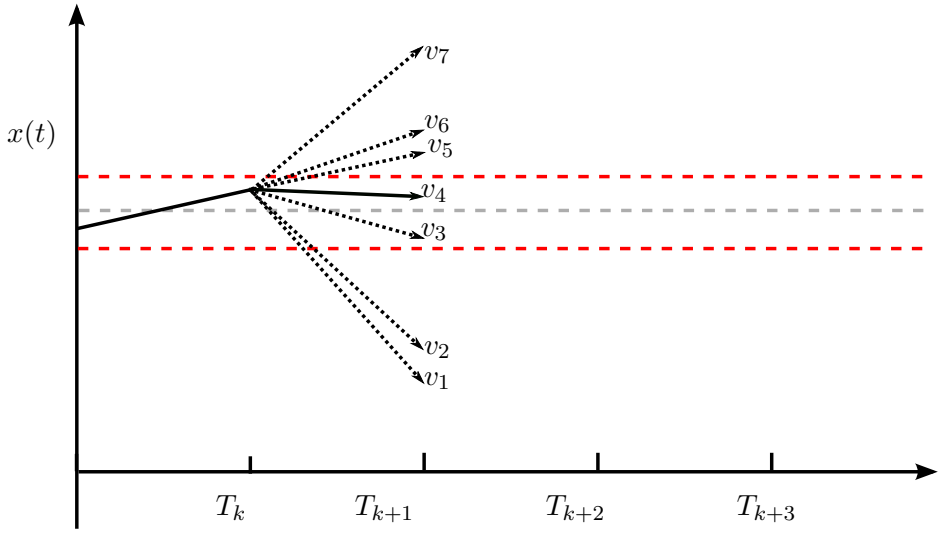

FIGURE 8.6: Cost function optimization in FS-MPC: a graphical representation

As figure 8.6 shows, depending on state of the system, each voltage vector may lead to different slope on output variables which in case of the induction motor are the flux and electromagnetic torque. For simplicity here only one variable is illustrated.

8.4.1 Design of the Controller

8.4.1.1 Predictive Model of Induction Motor

Dynamic phenomena in IM can be described using the differential equations 8.8 - 8.12. v_s and v_r are voltage vectors of the stator and rotor respectively, i_s and i_r stator and rotor current vectors, ψ_s and ψ_r stator and rotor flux vectors, ω_r mechanical speed, R_s and R_r stator and rotor resistances, L_s and L_r stator and rotor inductances, L_m mutual inductance, T_e the electromagnetic torque, p the number of pole pairs.

$$\boldsymbol{v_s} = R_s \cdot \ i_s + \frac{d\psi_s}{dt} \tag{8.8}$$

$$v_r = 0 = R_r \cdot \ i_r + \frac{\psi_r}{dt} - j \cdot \ p\omega_r \cdot \ \psi_r \tag{8.9}$$

$$\psi_s = L_s \cdot \ i_s + L_m \cdot \ i_r \tag{8.10}$$

$$\psi_r = L_r \cdot \ i_r + L_m \cdot \ i_s \tag{8.11}$$

$$T_e = \frac{3}{2} P(\psi_s \times i_s) \tag{8.12}$$

where

$$j = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{8.13}$$

Using the forward Euler approximation the difference equations are obtained:

$$\boldsymbol{i}_s(k+1) = \boldsymbol{i}_s(k) + T_s \cdot (-\tfrac{R_e}{L_e}\boldsymbol{i}_s(k) + \tfrac{K_r A_r}{L_e}\boldsymbol{\psi}_r(k) - \tfrac{K_r p\omega_r(k)}{L_e} \cdot j\boldsymbol{\psi}_r(k) + \tfrac{\boldsymbol{v}_s(k+1)}{L_e}) \tag{8.14}$$

$$\boldsymbol{\psi}_s(k+1) = \boldsymbol{\psi}(k) + T_s \cdot (\boldsymbol{v}_s(k+1) - R_s \cdot \boldsymbol{i}_s(k+1)) \tag{8.15}$$

$$T_e(k+1) = \frac{3}{2} P \cdot (\boldsymbol{\psi}_s(k+1) \times \boldsymbol{i}_s(k+1)) \tag{8.16}$$

where T_s is the sampling time. Equations 8.15 - 8.16 are used to predict the future behavior of the controlled variables ($\boldsymbol{i}_s$, $\boldsymbol{\psi}_s$ and T_e) for the given control action. As mechanical time constant is considerably larger than electrical, for torque control loop mechanical speed can be treated like a slowly varying parameter:

$$\frac{d\omega_r}{dt} \approx 0 \tag{8.17}$$

8.4.1.2 The Cost Function

Primary objective of the torque and flux controller in IM, is tracking respective reference values. It can be reflected in the following cost function:

$$J = |\psi_s^{ref} - \psi_s^p| + G_T \cdot |T_e^{ref} - T_e^p| + g_I \tag{8.18}$$

where G_T is the gain factor of the torque, ψ_s^{ref}, ψ_s^p reference and predicted value of the flux, T_e^{ref}, T_e^p reference and predicted value of the torque and g_I the discontinuous function imposed into the cost function for realizing the current limit:

$$g_I = \begin{cases} 0 & |{I_s}^p| < I_s^{max} \\ \infty & \text{else} \end{cases} \tag{8.19}$$

8.4.2 FPGA-Based Implementation of FS-MPC

The computation time has a big impact on the performance of the controller. Reducing the computation time makes it possible to have an immediate action after measurement and calculation of the control.

Hardware programmability of the FPGA allows to fully dedicate it to the control algorithm. It leads to significant reduction of the execution time. Pipelining and parallelism are the two main techniques which can be effectively used to improve the computational performance of the FPGA. Thus important part of any FPGA-based design, is the hardware architecture. In fact, iteration based structure of FS-MPC makes it suitable for the FPGA implementation. Figure 8.7 shows the functional block diagram of FS-MPC on the FPGA.

The input clock trigger launches the execution. System clock frequency is $100(MHz)$. In the first block, all possible voltage vectors are generated by the *state counter* which restarts each time the clock trigger appears. The clock latency for this block is one.

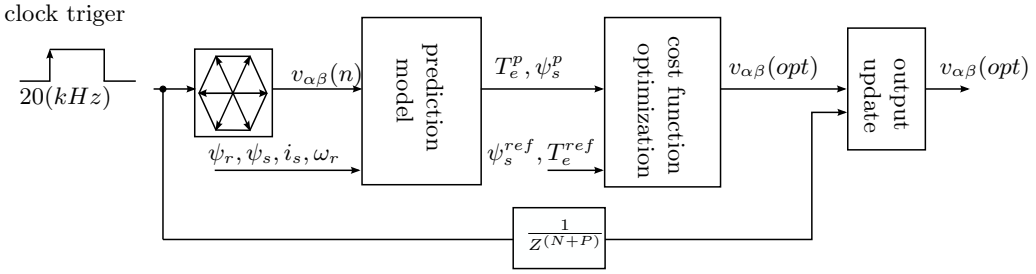

FIGURE 8.7: Cost function optimization in FS-MPC

The *prediction model* calculates the torque and flux for each voltage vector. Discrete model of IM is utilized for the prediction. This block has 6 clock latencies. The number of the clock latencies is associated with the pipeline stages.

In the cost function optimization block, the performance index is minimized and the respective voltage vector is selected as the optimal value ($v_{\alpha\beta}(opt)$). Clock latencies for these block are 18. Relatively higher number of clock latencies is due to CORDIC algorithm for calculation of predicted flux magnitude. However, in this block only a few embedded multipliers are utilized.

In the *output update* block, new voltage vectors which must be applied over the next sampling interval, is updated each time the trigger edge appears. Number of clock latencies for the *output update* block is 2.

All simultaneous operations are done in parallel to minimize the total clock latencies. Since prediction model and the cost function for all voltage vectors are identical, the design is pipelined so that it can be efficiently reused for all voltage vectors.

To synchronize the *voltage update* with the last calculation, this block is fed by the execution trigger delayed by $(N + P)$ clocks. N corresponds to the total number of voltage vectors and P number of pipeline stages. The total pipeline stages are 27. It leads to 34 clock latencies. Since system clock is $100(MHz)$, computation time of FS-MPC is only $340(ns)$. Such calculation time can be safely neglected without compensation.

The proposed architecture for FS-MPC algorithms, can be applied for different types of electrical drives as well as many converter topologies. It needs just to modify the prediction model and the cost function accordingly.

8.4.3 Experimental Results

To demonstrate tracking performance of FS-MPC in practice, experimental tests are carried out for the induction motor fed by the 2-level voltage source inverter.

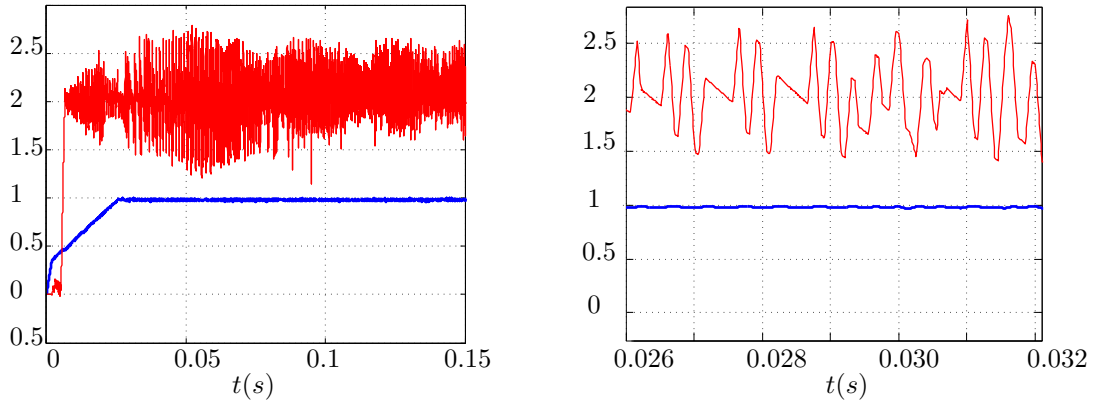

FIGURE 8.8: Experimental results: performance of FS-MPC for torque and flux control. The blue curve is the flux (Wb) whereas the red one corresponds to the torque (Nm).

Figure 8.8 shows the dynamic as well as steady state performance of FS-MPC for tracking of reference values. The blue curve corresponds to the flux whereas the red curve is the electromagnetic torque. Although FS-MPC enables to control mean value of the torque and flux at the desiered level, it suffers from high torque ripples. This problem is widely mentioned for FS-MPC in many publications [131].

High ripples are due to the fact that the controller is intended to change the switching state only at the sampling instants in contrast to the PWM-based methods wherein switching time is precisely calculated within the sampling interval. Nevertheless, FS-MPC has many advantages. It is inherently nonlinear and flexible to include many control goals. Further benefit of FS-MPC, as shown here, is its simple structure for implementation on the FPGA maintaining high computational performance.

8.5 Discussion

FS-MPC provides a very intuitive way for the modeling of the power electronics and its flexible cost function allows to include also secondary objectives. Since there are other control strategies that rely on the discrete nature of the controlled system, it makes sense to carry out a comparison. One of the prior works is Variable Structure (VS) or so-called Sliding Mode Control (SMC). In SMC, the control is changed in accordance

to the output error. Poor steady state performance is the major drawback of SMC. However, the controller is usually robust to the parameter deviation and in many cases the model parameters are not needed at all. It makes SMC a universal solution in many applications. In SMC, the concept of order reduction and stable invariant manifolds are the basic philosophy of the controller design. It is assumed that number of control parameters is equal to the number of variable structures. It is due to the very general optimization criteria defined in SMC which is mainly asymptotic stability of the closed loop system. As figure 8.9 shows, for the *sliding motion* on the hyperplanes, in the 2-level VSI there are more than one discrete control available at the same time. FS-MPC is obviously superior to SMC in utilizing all these degrees of freedom.

In fact, the model independent property of SMC can be also a restriction which prevents to include the available information of the plant in design of the controller. This is of great importance in particular for the power electronic and drive systems for which a mathematical model is in most cases available.

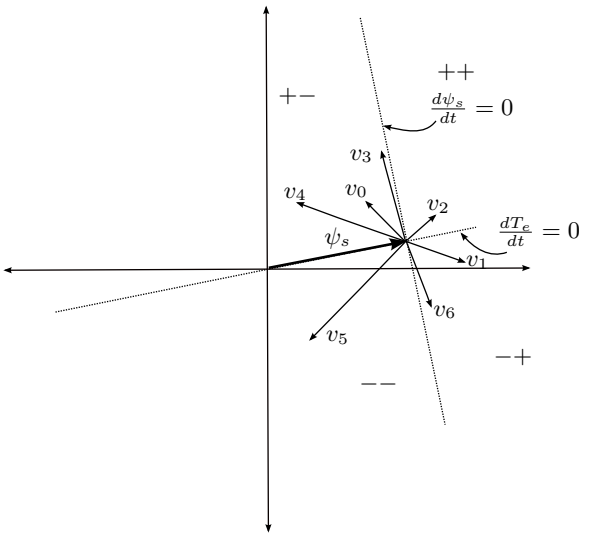

FIGURE 8.9: Sliding Model Control for a VSI

FS-MPC has the capability of utilizing additional degrees of freedom which are available in many power electronic systems to improve the dynamic and steady state performance.

Chapter 9

Nonlinear Model Predictive Control

9.1 Introduction

FS-MPC investigated in the previous chapter, inherently includes constraints and discrete nature of the power electronic as well as nonlinearities of the drive model. In FS-MPC, the optimization is performed for a set of feasible controls associated with the switching states of power electronics. Flexibility of FS-MPC in choosing the cost function, makes it applicable for various power electronic and drives systems. It has a relatively simple optimization strategy that allows to include several control objectives in the cost function. The main limit of FS-MPC is that the computational complexity rises exponentially by increasing the prediction horizon and with only one step prediction it suffers from high ripples of the output variables. Higher ripples at steady state operation are due to the fact that switching time is only at sampling instants in contrast to the SVM-based methods that can precisely calculate the switching time with minimum ripples as is depicted on figure 9.1.

To reduce output ripples, optimal switching time must be applied to the inverter. Optimality of the duty cycles should be assured considering a combination of at least three switching states, as shown on figure 9.1. Using the modulator, the switching time can be separately optimized independent from the control algorithm. Since the sampling time is constant in FS-MPC, it requires a very long prediction horizon to attain the optimal switching time for a sequence of three vectors. For 3 step prediction, as it is illustrated on the figure, equal duty cycles cause higher torque ripples compared to the space vector modulation.

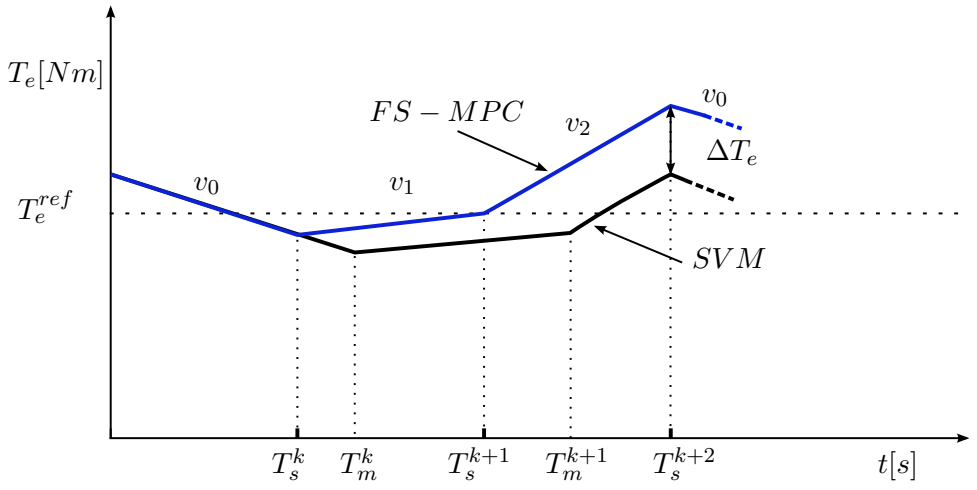

FIGURE 9.1: Performance of FS-MPC vs. SVM based control

To keep the advantages of FS-MPC such as flexibility, simplicity and high dynamic performance, an effective optimization algorithm is investigated which allows to calculate the optimal continuous voltage vector including all nonlinearities of the model. Furthermore, the proposed algorithm enables to maintain constraints in order to achieve the maximum performance of the inverter in all operation regions.

9.2 Nonlinear Optimization

Vast majority of practical systems are nonlinear. Many methods have been developed to deal with nonlinear and multi variable optimization challenges. Important features for a solution are the level of complexity and the implementability on digital computers, since these devices are dominating tools for performing numerical calculations.

Although many iterative techniques such as Newton-Raphson's, for solving mathematical problems, have been developed before the age of computers, they have received more and more attention after invention of digital computers. The reason is that iterative strategies usually consist of simple steps which must be repeated several times until a solution with a desiered accuracy can be found. This attribute makes such methods appropriate for implementation on digital computers.

Attempts to find a generalized strategy being valid for all nonlinear systems usually fail due to unpredictability of the global behavior. Effectiveness of solutions mainly depends on the structure of nonlinear models. Many engineering systems need to be studied in a limited operation range and optimization should be done only for bounded

parameters. Relying on such practical considerations leads to significant reduction of the computational complexity.

9.2.1 Region Elimination Optimization Method

Region elimination methods are effective in particular for constrained and bounded nonlinear optimization problems [132]. There are several region elimination techniques that differ depending on the region reduction ratio and the initialization strategy. In most optimization problems a function must be minimized over a predefined interval. A big advantage of the region reduction is that it does not imply excessive complexity such as calculation of derivatives or interpolation. Therefore, these algorithms are easily realizable.

Golden section method suggests the optimal interval reduction ratio for finding extremum point of an unimodal and one dimensional function over a bounded interval. Assume that $f(x)$ is to be minimized over the interval $[a, b]$. At each step x_a and x_b points are selected so that:

$$\frac{(b-a)}{(x_b-a)} = \frac{(x_b-a)}{(x_a-a)} = R \tag{9.1}$$

$$R = \frac{\sqrt{5}-1}{2} \tag{9.2}$$

where R is so-called *golden ratio*. At first iteration $f(x)$ must be calculated at two points while at each further iteration it needs to be calculated only once. *Golden section method* is optimized with respect to the reduction ratio and calculation burden in particular for unimodal functions.

Interval halving also known as bisection method relies on region reduction principle as well. Bisection method requires calculation of $f(x)$ for three points at first step and then two points per iteration. It has, however, a more simple strategy for choosing the intervals just by dividing the previous interval by 2 which can be easily done in digital computers using a shift register.

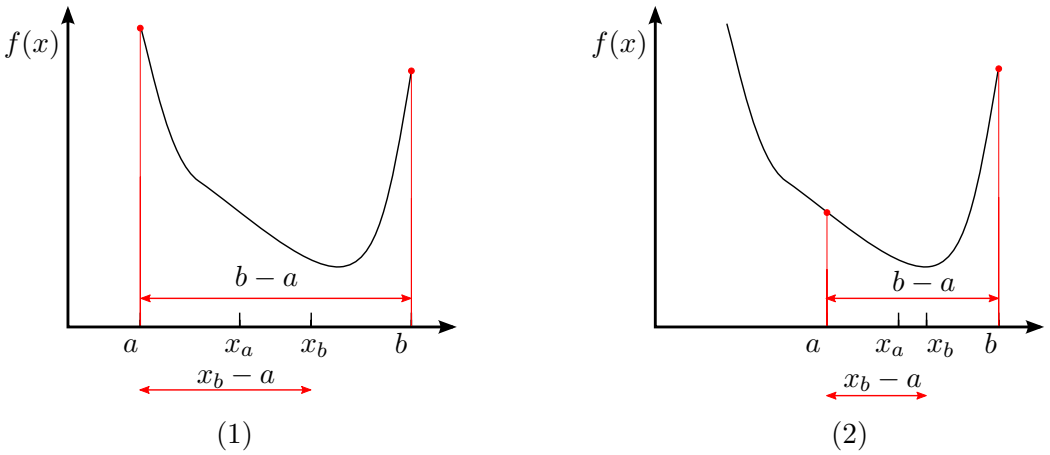

FIGURE 9.2: First and second iteration steps of the golden section method

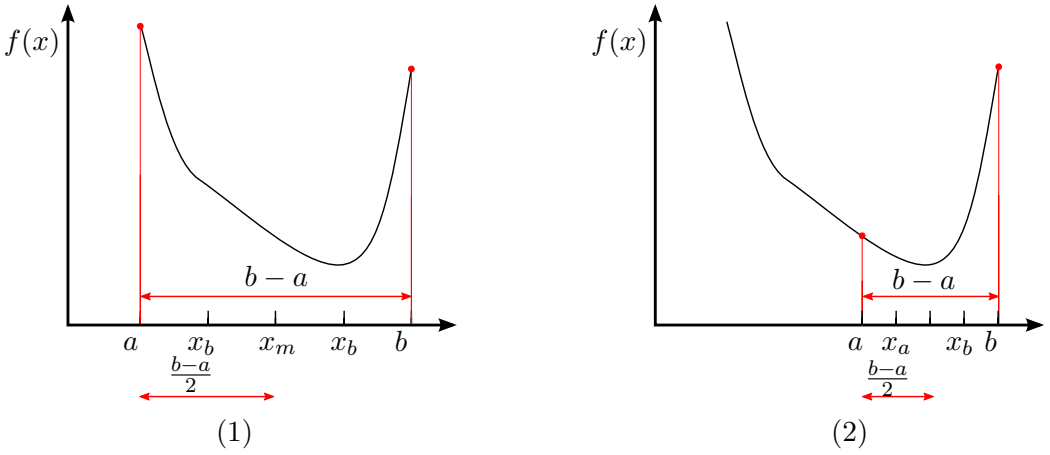

FIGURE 9.3: Two steps of the interval halving optimization

9.3 Multi-Variable Optimization

MPC often requires optimization of the cost function with respect to multiple parameters. The region reduction optimization technique investigated in the previous section can be extended to multi-variable optimization as well.

Figure 9.4 shows a simple MIMO system.

$$\frac{dx}{dt} = -\frac{1}{\tau_x}x + \frac{1}{\tau_x}u_x \tag{9.3}$$

$$\frac{dy}{dt} = -\frac{1}{\tau_y}y + \frac{1}{\tau_y}u_y \tag{9.4}$$

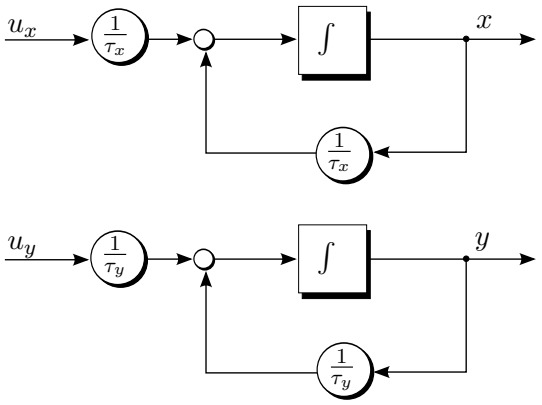

FIGURE 9.4: First order MIMO plant

Above equations represent a simplified current control loop of an electrical motor. The control variable is the stator voltage vector supposed to the following constraint:

$$u_x^2 + u_y^2 < U^2 \tag{9.5}$$

Consider one step prediction horizon, the following cost function can be derived to ensure reference value tracking:

$$J = |x^{ref} - x(k+1)| + |y^{ref} - y(k+1)| \tag{9.6}$$

where x^{ref}, y^{ref}, $x(k+1)$, $y(k+1)$ are reference and predicted values. By Euler approximation predictive model is obtained from initial differential equations of the plant:

$$x(k+1) = x(k) - \frac{T_s}{\tau_x}x(k) + \frac{T_s}{\tau_x}u_x(k) \tag{9.7}$$

$$y(k+1) = y(k) - \frac{T_s}{\tau_y}y(k) + \frac{T_s}{\tau_y}u_y(k) \tag{9.8}$$

T_s and k stand for sampling time and sampling instant respectively. Substituting initial system equations into the cost function leads to:

$$J = \left| x^{ref} - \left(x(k) - \frac{T_s}{\tau_x} x(k) + \frac{T_s}{\tau_x} u_x(k) \right) \right| + \left| y^{ref} - \left(y(k) - \frac{T_s}{\tau_y} y(k) + \frac{T_s}{\tau_y} u_y(k) \right) \right| \tag{9.9}$$

By optimization of this cost function at the time instant k, a control vector $[u_x \ u_y]^T$ is obtained which can be applied over the sampling interval $k+1$ to ensure tracking of the reference values (x_{ref}, y_{ref}) with the minimum error. The calculated control vector must fulfill the constraint condition as well.

In order to simplify inclusion of the constraint, parameters are redefined as:

$$u_x = R \cdot cos(\theta) \tag{9.10}$$

$$u_y = R \cdot sin(\theta) \tag{9.11}$$

where

$$-\sqrt{U^2} < R < \sqrt{U^2} \tag{9.12}$$

$$0 < \theta < \pi \tag{9.13}$$

The *halving interval* strategy is extended for multiple-variable by multiplexing all regions. Extension of optimization increases number of calculations exponentially:

$$N = 2^P \tag{9.14}$$

where P and N are number of parameters and calculations respectively. The system has 2 parameters. Therefore at each step the cost function must be calculated for 4 set of control variables. The flowchart of the optimization algorithm is depicted on figure 9.5.

Figure 9.6 shows performance of the closed-loop system for tracking of the reference values. The defined constraint is maintained throughout the control. Since the cost function has periodic behavior with respect to θ, the search region for it can be divided in more than two regions to improve resolution of the solution.

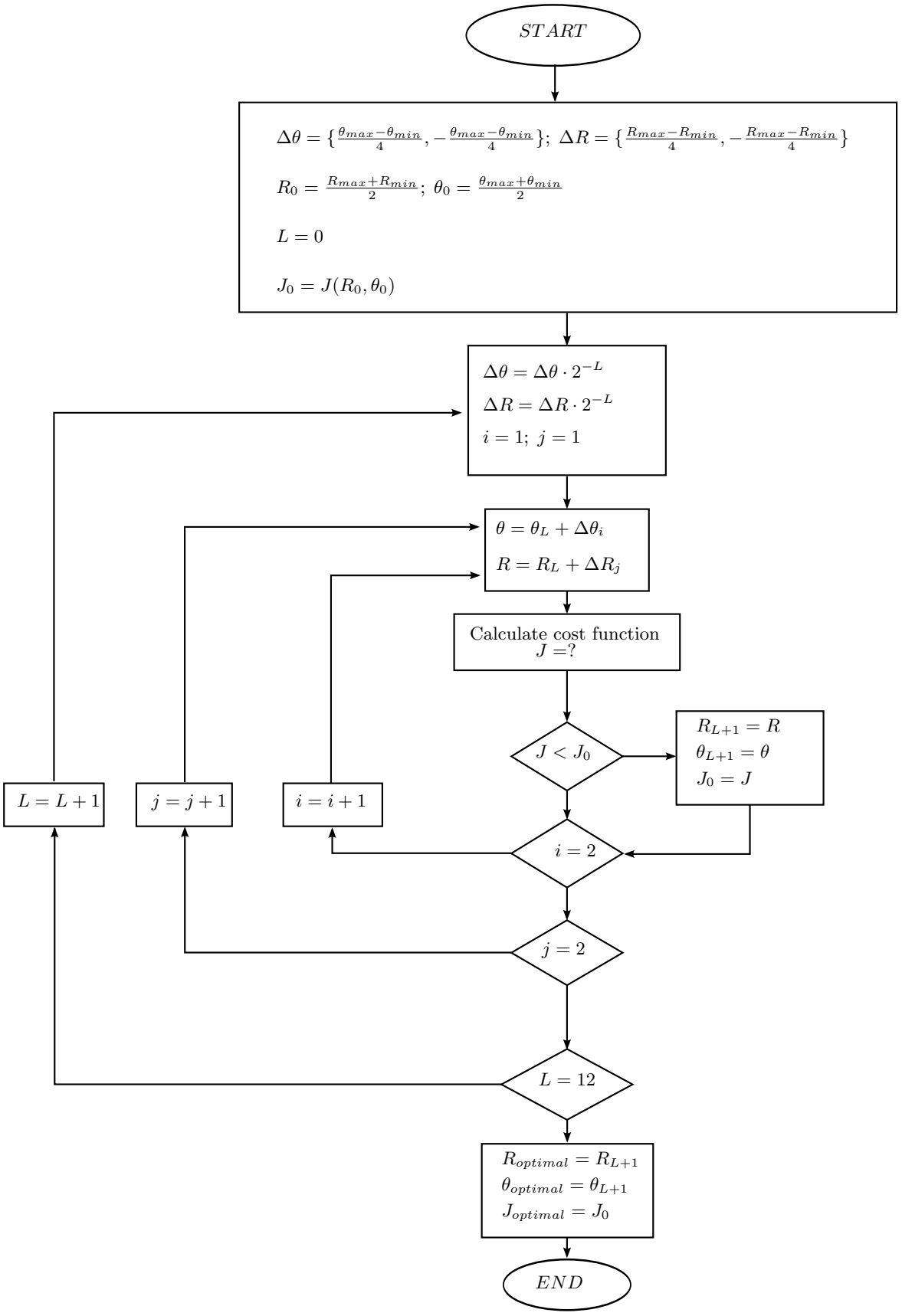

FIGURE 9.5: Multi-variable optimization algorithm

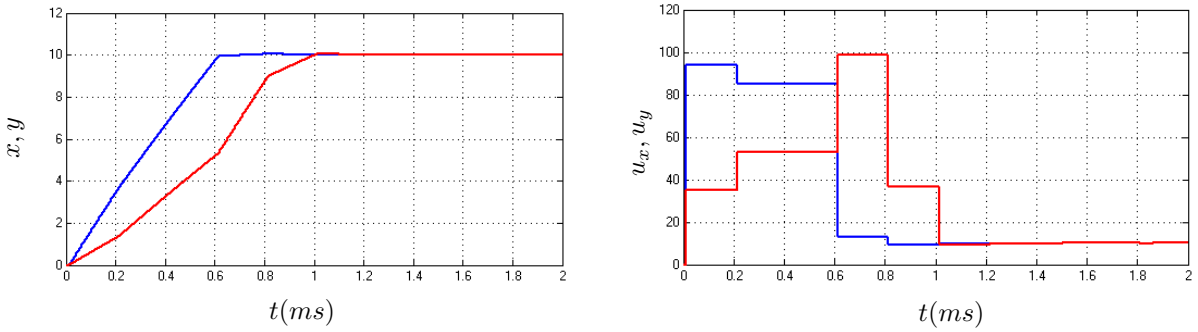

FIGURE 9.6: Performance of the control algorithm maintaining circle constraint

The example provided here can demonstrate a simplified current control loop of electrical drives. The imposed constraint, in fact, refers to voltage limit of the converter. In three phase electrical drive systems fed by VSI, feasible voltage vectors are limited inside the hexagon. To verify the effectiveness of the optimization algorithm for maintaining such type of dynamic constraints which can not be simply considered in initial search intervals, the same simulation is carried out considering the new constraint.

As it is depicted on figure 9.7, considering the whole hexagon, the maximum value of u_y is the same as for the circle, whereas the maximum value of u_x is increased to:

$$u_x^{max} = \frac{1}{cos(\pi/6)} \cdot R \tag{9.15}$$

All points which violate the constraint condition are excluded by giving a large cost function value.

Trajectory of the control signal is depicted on figure 9.9. The optimization algorithm is able to derive maximum available control within the hexagon to achieve the best possible performance. In the cost function a gain factor larger than 1 can be interpreted as a higher priority given to x whereas a gain factor lower than 1 means less importance for x compared to y. For this investigation, the gain factor is set to 2 and as expected the faster response for x is attained.

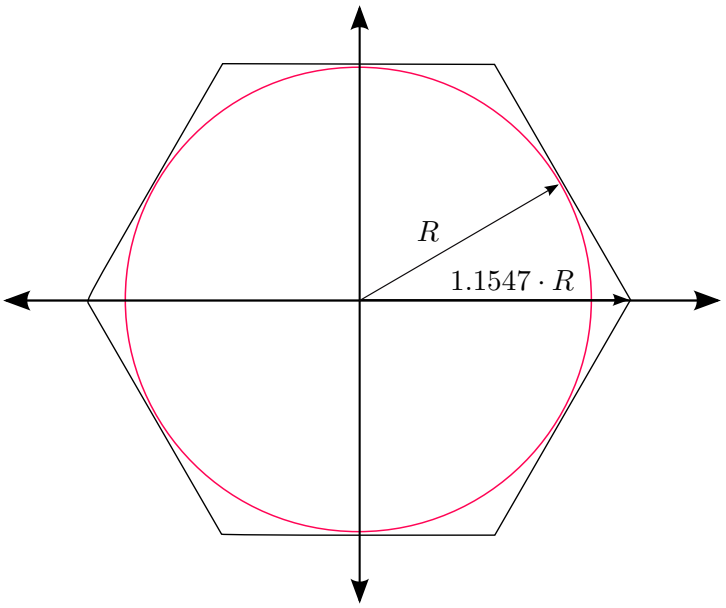

FIGURE 9.7: Constraint of control

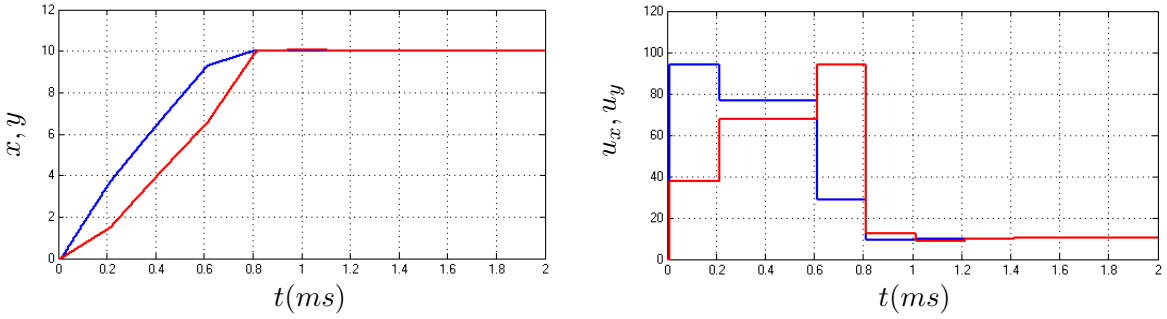

FIGURE 9.8: Dynamic behavior of controller maintaining hexagon constraint

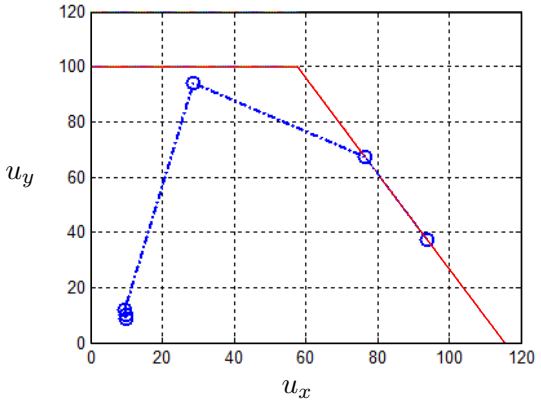

FIGURE 9.9: Control trajectory inside the hexagon

9.4 Discussion

An optimization algorithm based on the *region elimination* strategy is investigated. Major advantage of the proposed method is its applicability for a wide class of nonlinear optimization problems. For sake of simplicity, investigations are carried out on a linear system with input constraints. Nevertheless results are also valid for nonlinear systems.

Iterative structure of the region elimination optimization makes it suitable for FPGA implementation. In general many parts of the optimization algorithm is independent from the cost function. It allows to create a scalable FPGA model usable for different drive types. It is an important aspect since FPGA models require relatively longer development time.

Chapter 10

Nonlinear Model Predictive Control of Induction Motor

In this chapter Nonlinear Model Predictive Control of IM is investigated based on the optimization algorithm proposed in the previous chapter. Furthermore, practical issues associated with the FPGA implementation are discussed.

10.1 Structure of the Controller

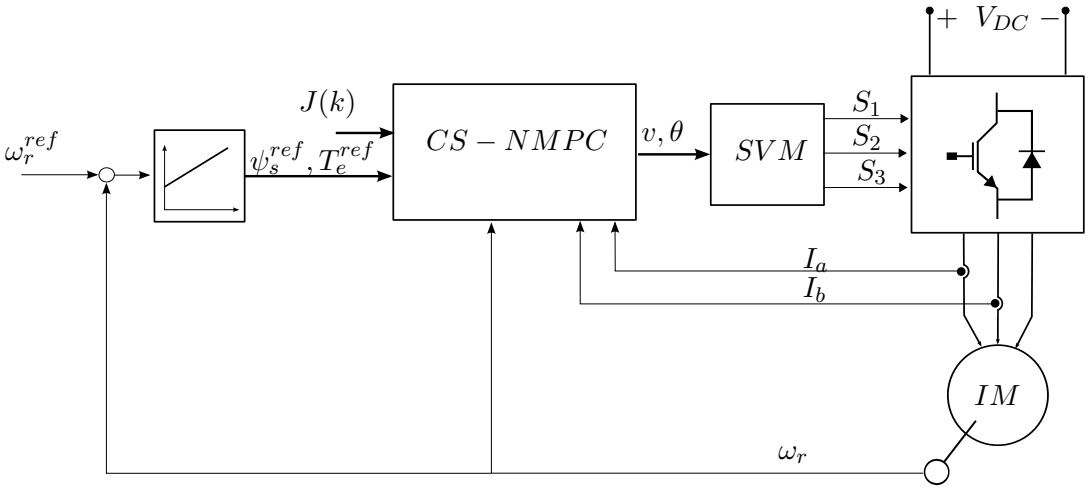

FIGURE 10.1: Block diagram of CS-NMPC

Structure of the proposed MPC strategy is depicted on figure 10.1. Since the controller solves the optimization problem for a continuous control set, we refer to it as Continuous Set Nonlinear Model Predictive Control (CS-NMPC). Although it shares the same cost

function with FS-MPC, the optimization algorithm enables to calculate a continuous voltage vector which can be applied through the modulator.

10.1.1 Cost Function

FS-MPC algorithm is already applied to diversity of drives types and power electronic topologies. The most significant feature of FS-MPC is its capability to include several control objectives simultaneously. For induction motor drive the primary control task is to keep the torque and stator flux at determined levels. In FS-MPC, the absolute errors are directly used in the cost function. In CS-NMPC it is relied on the same cost function:

$$J = |\psi_s^p - \psi_s^{ref}| + G_T|T_e^p - T_e^{ref}| \tag{10.1}$$

where J is the cost function to be minimized, G_T the design parameter, ψ_s^p, ψ_s^{ref}, T_e^p and T_e^{ref} reference and predicted values of the stator flux and electromagnetic torque. The gain factor G_T is the only design parameter. Compared to the cascade control, in which at least 4 parameters are required to be tuned, it represents a more simple design strategy by reducing the number of parameters.

10.1.1.1 Including Constraints

Various types of constraints can be included into the cost function simply by adding penalty terms [133]:

$$C_i = \begin{cases} \infty & \text{violating constraint} \\ 0 & \text{else} \end{cases} \tag{10.2}$$

This additional term enables to exclude all solutions that lead to violation of the constraint. It can maintain constraints of the state as well as control variables.

10.1.2 Optimization Algorithm

The task of the optimization algorithm is calculating a voltage vector which minimizes the cost function over the next sampling interval. In the previous chapter a numerical optimization method is investigated. To apply the same strategy, optimization task is first broken down into calculation of phase and magnitude of the optimal voltage vector

as shown on figure 10.2. All voltage vectors located within the hexagon are feasible and can be produced by the inverter.

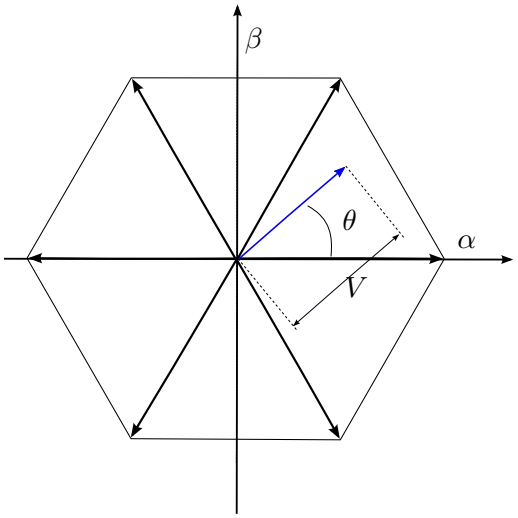

FIGURE 10.2: Space vector representation of the voltage vector to be calculated by CS-NMPC

The procedure proposed here for calculation of the optimal voltage vector with respect to the cost function is not so much depended on the form of the cost function. The calculation consists of finite number of iterations. At each iteration, it evaluates a set of voltage vectors and choose a vector with minimum cost function like what is done in FS-MPC. However, in CS-NMPC test vectors don't have necessarily direct physical interpretations.

After each iteration the new sub-optimal voltage vector is calculated and the search region is reduced around it. The algorithms is terminated when the search region is small enough. Due to the fact that at each step the search region is divided by 2, 12 iteration is sufficient to obtain 12-bit resolution.

10.2 Control in Overmodulation Region

The aim of the overmodulation is to extract maximum possible volt-time out of the voltage source inverter. Maximum available voltage which can be achieved by a linear controller, is the circular area within the hexagon. Since the inverter voltage is not distributed uniformly along the stator windings, any further increase of the output voltage leads to current distortion which must be compensated properly to decrease its influence on the current control loop [134]. However such compensation causes additional complexity for the controller.

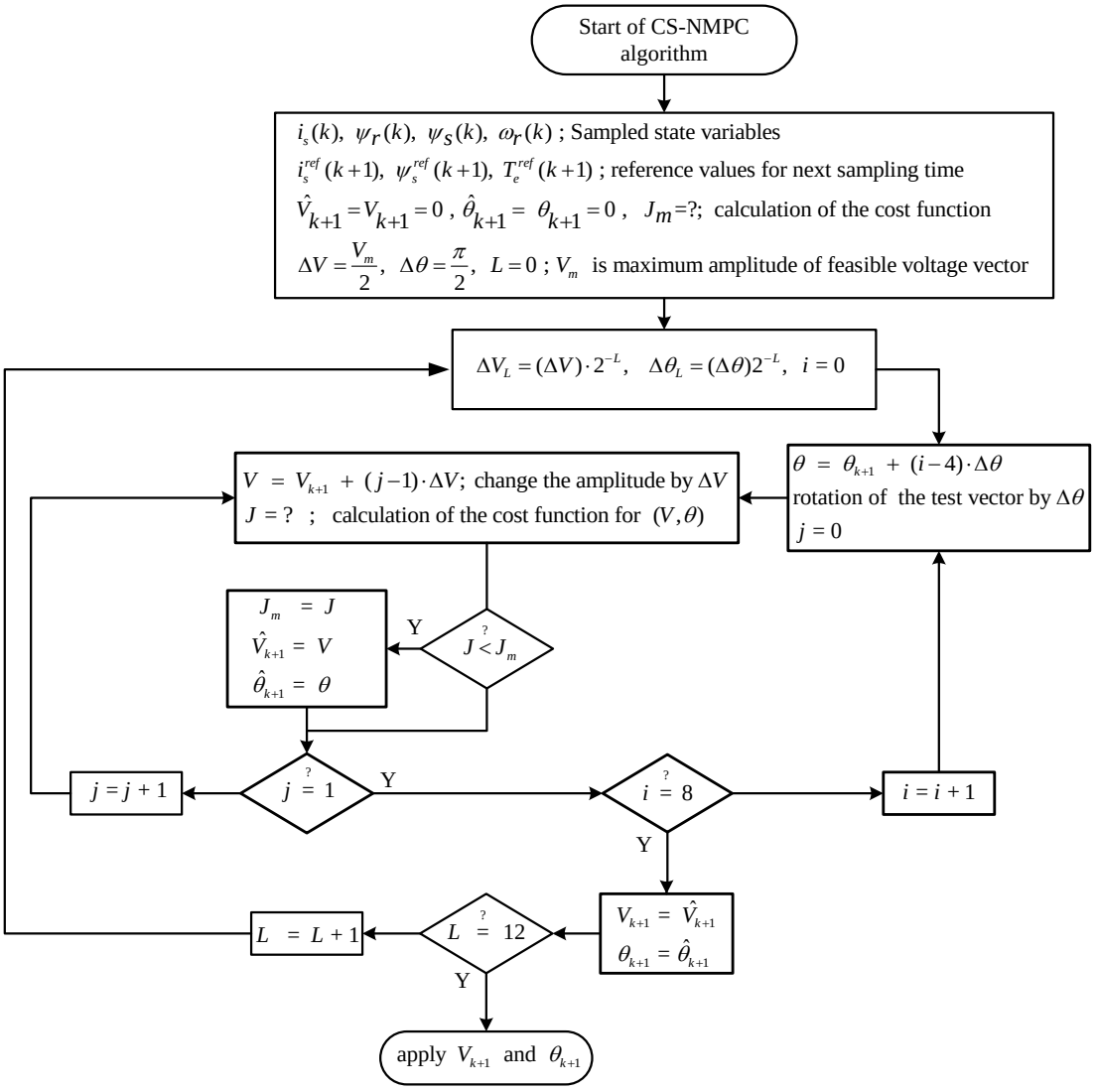

FIGURE 10.3: The flowchart of the CS-NMPC algorithm for the induction motor

Thanks to the high dynamic performance of the CS-NMPC there is no need for coordinate transformation. Furthermore it provides a direct torque and flux control which simplifies operation in overmodulation region. To maintain maximum available voltage vector, the voltage constraint should be extended to the whole hexagon area.

10.3 FPGA Implementation

Modularity of the CS-NMPC algorithm decreases complexity of the FPGA model. It can be separated into subcomponents so that functionality of each part can be individually verified. Furthermore, structure of the optimization algorithm is independent of the

cost function and drive type. Therefore, a generalized and scalable FPGA model can be created which is able to be parametrized and reproduced for various applications.

In previous chapters two main design environments are discussed: 1- Conventional HDL model development environment using low level hardware description languages such as VHDL 2- Model base design in the Simulink environment relying on the *HDL coder* tool of Mathworks, for code generation. At different stages of the thesis, CS-NMPC is implemented in both mentioned environments. The structure being discussed here is valid for VHDL hand codding as well as Simulink model. Figure 10.4 shows functional block diagram of the FPGA implementation.

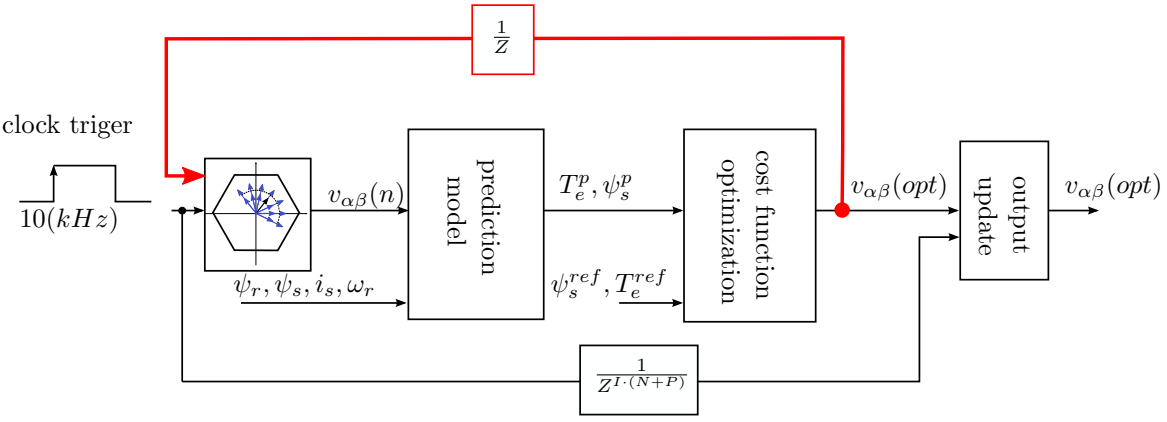

FIGURE 10.4: FPGA model of CS-NMPC for the induction motor

10.3.1 Computational Performance

The structure of CS-NMPC enables to effectively use pipelining technique and enhance computational performance of the algorithm. A further improvement of the computational performance is possible thanks to parallel processing of the FPGA.

The whole CS-NMPC model is developed in Simulink and through the *HDL coder* tool, corresponding VHDL code for FPGA configuration, is generated. Graphical programming of Simulink provides a clear overview of the algorithm and makes it very convenient to synthesize the model. More importantly such graphical overview is helpful for organizing pipeline stages. Figure 10.5 shows a screenshot of the Simulink model. Further information regarding subsystems is provided in appendix E.

Between each pair of adjacent pipeline registers, there should be as much combinatorial logic as the maximum time constraint dose not exceed one clock period. This time constraint can be roughly estimated with respect to the complexity of operations, signal paths and performance of the target FPGA. Therefore, complicated timing analysis for each model is not necessary.

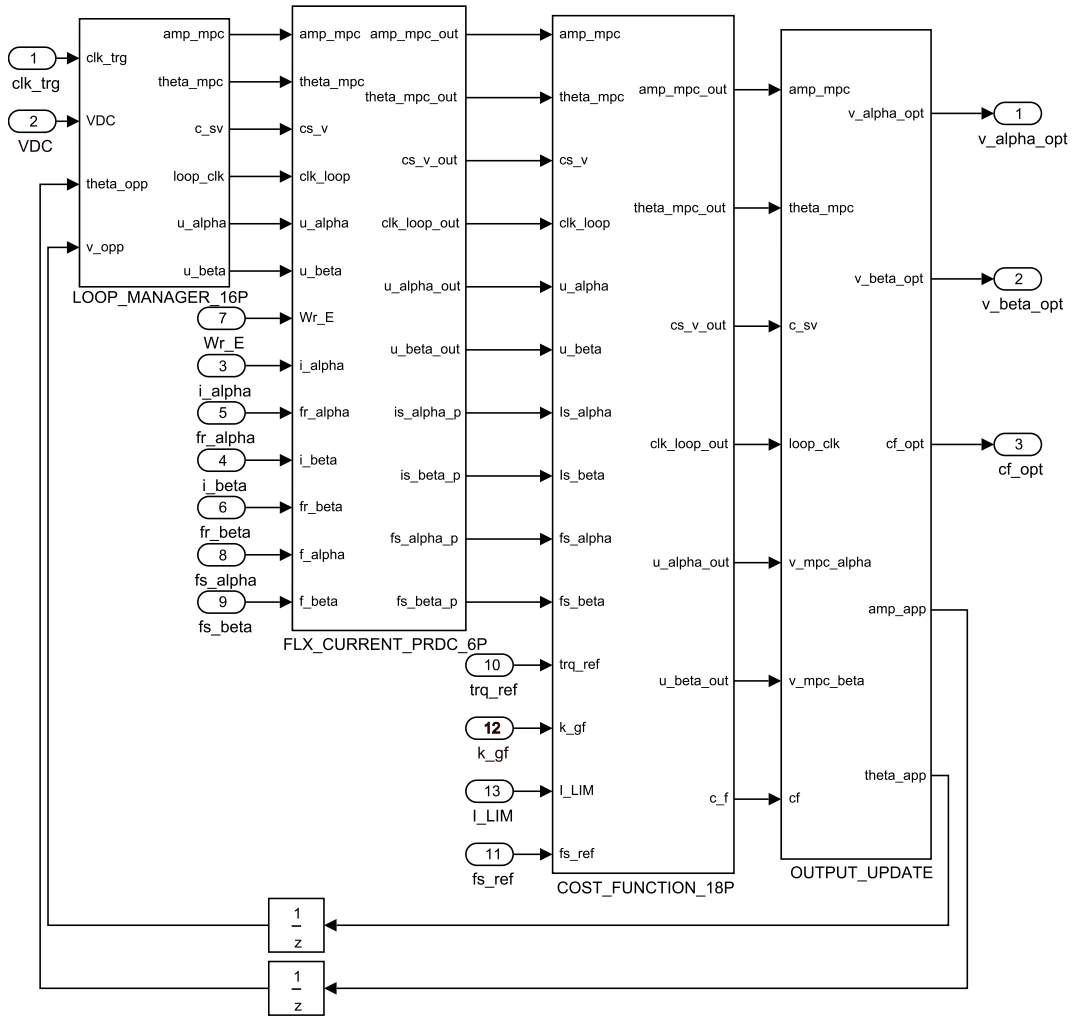

FIGURE 10.5: The Simulink model of CS-NMPC for the induction motor

Calculation of execution the time is straightforward just by counting pipeline stages, number of test vectors and iterations. The calculation time will be as follows:

$$t_c = \frac{I \cdot (N + P + 1)}{f_{clk}} \tag{10.3}$$

where t_c is execution time, I number of iteration, N number of voltage vectors, P pipeline stages and f_{clk} clock frequency of the FPGA.

As the Simulink model shows, one iteration consists of 42 pipeline stages and in each iteration, 14 voltage vectors are evaluated. For an acceptable accuracy, 12 iterations, corresponding to 12 bit resolution, are sufficient. Since the FPGA clock frequency is 100 MHz, computation time will be 6.72(μs).

10.4 Experimental Results

To practically demonstrate the dynamic performance of CS-NMPC for controlling electromagnetic torque and stator flux of the induction motor, it is realized on the experimental setup. Detail of the setup is provided in chapter 7. On the control board the Cyclone IV FPGA chip is intended to execute the control algorithm.

The induction motor fed by the voltage source inverter is used as test drive whereas the induction motor controlled through an industrial inverter is intended to provide the load torque.

For speed measurement, the incremental encoder is installed on the motor shaft. Stator currents are measured through the Hall sensors. A flux observer is designed to estimate rotor and stator fluxes. Formulation as well as FPGA implantation of the observer are provided in appendix D

10.4.1 Motor Startup

Figure 10.6 shows the dynamic performance of CS-NMPC for the induction motor, at startup. Controlled variables are torque and stator flux while the controller must maintain voltage and current constraints.

At startup stator flux is zero and the stator current must initialize the flux. Afterwards the electromagnetic torque can be produced. In the classical field oriented control, torque and flux are controlled in separate loops.

As figure 10.6 shows , with the only control parameter of CS-NMPC, dynamic behavior of the torque and stator flux can be managed. Choosing a larger G_T leads to a faster dynamic of the torque still maintaining the same constraints.

10.4.2 Decoupling of Flux and Torque Control

The stator flux must be properly controlled to maintain maximum performance and energy efficiency of the drive. Figures 10.7 and 10.8 demonstrate that CS-NMPC successfully decouples flux and torque control. In all operation regions, the performance of torque and flux control can reach physical limits. Such superior performance is achieved at lower complexity and design challenges.

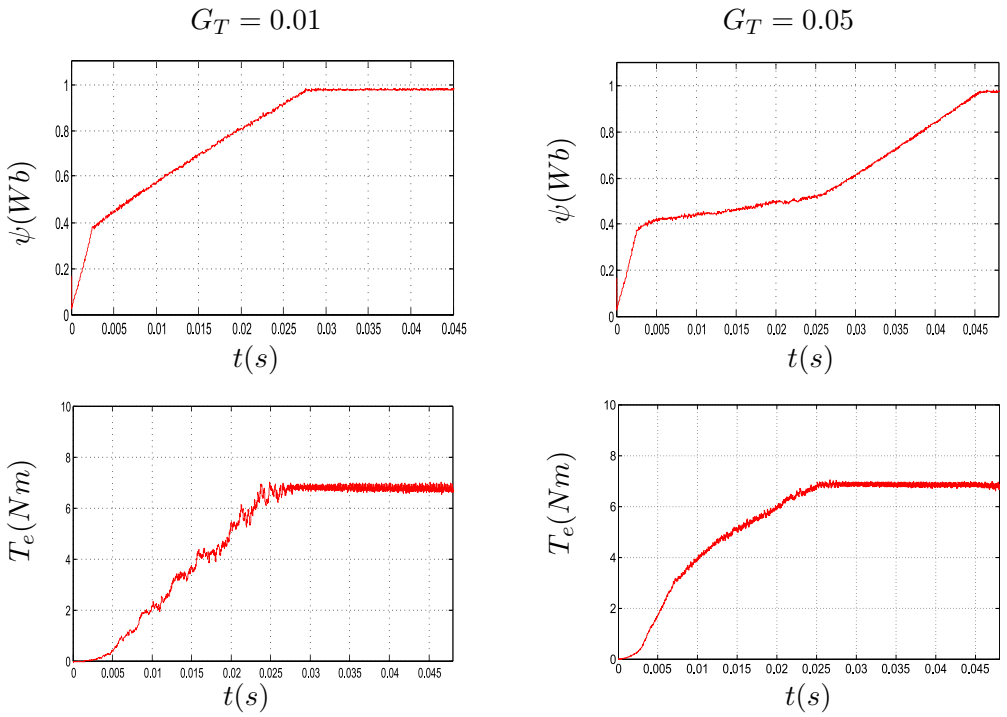

FIGURE 10.6: Dynamic performance of CS-NMPC for the induction motor at startup. $V_{DC} = 300(V)$ and $I_s^{max} = 15(A)$ for different gain factors G_T.

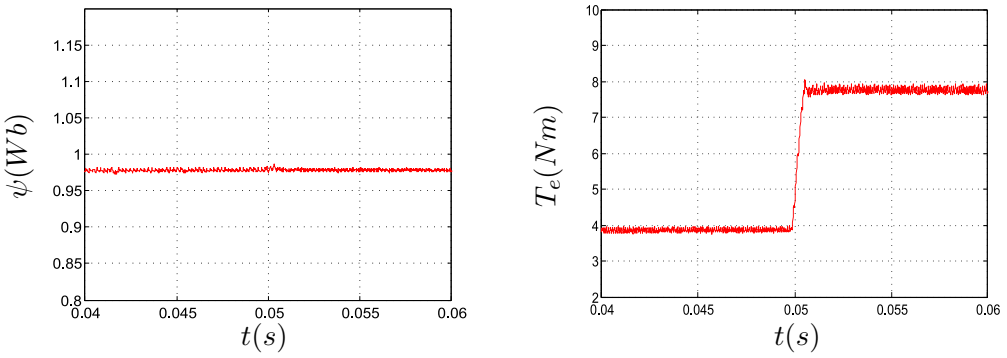

FIGURE 10.7: Dynamic performance of the torque control loop. $V_{DC} = 300(V)$ and $I_s^{max} = 15(A)$

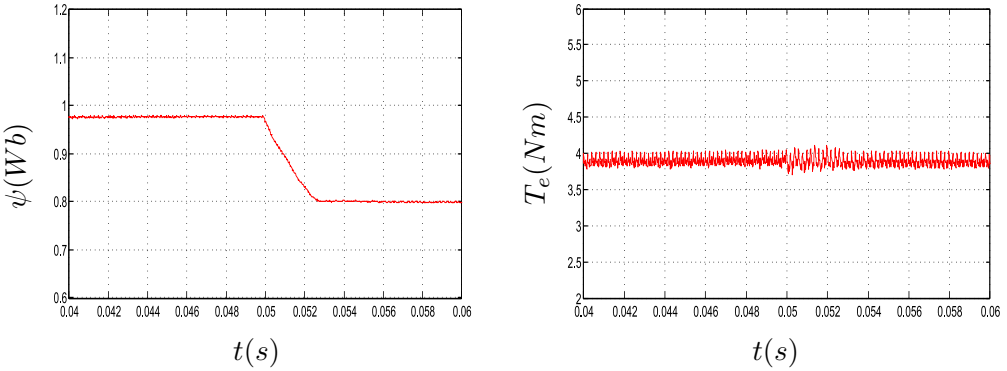

FIGURE 10.8: Dynamic performance of the flux control loop. $V_{DC} = 300(V)$ and $I_s^{max} = 15(A)$

10.5 Discussion

Control of induction motors is more challenging compared to other drive types. The reason is complexity of the mathematical model which is highly nonlinear. The conventional approach to control of induction motors is linearization of the motor model and applying linear control concepts. Such simplifications lead to lower performance and energy efficiency. The proposed control method enables to synthesize the controller without linearizing the model.

The cost function of CS-NMPC used for optimization is very flexible and many control tasks can be simultaneously included by imposing additional terms.

Structure of the optimization algorithm in CS-NMPC is similar to FS-MPC. The number of test vectors is increased to include those vectors which have no direct physical interpretations, in order to calculate a quasi continuous voltage vector. Therefore, the same hardware structure is also applicable for the FPGA implementation of the optimization algorithm.

Chapter 11

Model Predictive Control with Imposed Target Dynamic

11.1 Introduction

MPC algorithms being studied in this dissertation, mainly rely on one step prediction. The computational complexity is the main limit for real-time implementation of MPC algorithms for longer prediction horizons, especially for nonlinear systems.

However, computational complexity of MPC is not the only design challenge. Other issues, associated with MPC applications, are parametrization and definition of the performance criteria. For classical control methods such as PID, controller attributes have explicit relation with the design parameters. In MPC, performance of the controller can be manipulated through the gain factors. Nevertheless, an insightful approach for calculation of the gain factors, which can maintain a predefined dynamic behavior of the controlled plant, is still not available. Due to that design of the cost function is mainly based on empirical techniques.

In this chapter *order reduction* and concept of *invariant manifolds* are investigated as useful tools for dealing with the aforementioned challenges of MPC.

11.2 Computational Complexity of FS-MPC for Long Prediction Horizon

FS-MPC treats the hybrid plants as variable structure systems with finite states. Finite states considered in FS-MPC are naturally inherited from the discrete notion of power electronics. Nevertheless unlike pulse width modulation that allows only to control mean values within the time discrete interval, FS-MPC falls into the group of the direct control strategies and potentially can lead to higher dynamic performance and lower switching frequencies [125, 129].

Similar approach, prior to FS-MPC, is Sliding Mode Control (SMC) [135–137]. However FS-MPC has following advantages which make it superior to SMC:

- Chattering problem of controller outputs can be decreased.
- Dealing with higher degrees of freedom in choosing the finite control set is possible.
- Additional control objectives can be included into the cost function of FS-MPC.

There are some heuristic ways for dealing with the computational complexity of FS-MPC by applying pre-optimization to exclude some of the possible states and reduce the computational effort. In [128] a continuous-set model predictive control strategy with off-line optimization is applied to bound optimal control trajectories. For FS-MPC only voltage vectors are considered which are located near the pre-calculated trajectory. This is an interesting approach that allows to combine well-known continuous-set MPC with FS-MPC to reduce the computational complexity for longer prediction horizons. The main remark to this solution is that the MPC method applied for pre-optimization is still computationally complex and has many problems of practical implementation. Furthermore it degrades flexibility of FS-MPC in choosing the cost function since both control algorithms should have the same performance index for combination to make sense.

Another way for reducing the computational complexity of FS-MPC is investigated in [126], wherein a bounded control output is taken and the controller is allowed to switch only if it goes out of the bounded area in the next sampling time. This is, in fact, a predictive hysteresis control and consequently the output ripples is to be set by the designer or calculated from a supervisory control and the predictive algorithm reduces the number of switchings over the prediction horizon. In other words, it reduces the optimization effort at the cost of omitting the flexible cost function of FS-MPC and declining the number of objectives.

11.3 Synergetic Control

Synergetic theory is established by H. Haken to describe nonlinear behaviors in quantum physics. The aim of the synergetic theory is explaining nonlinear and stochastic behavior, self-organizing and pattern formation in highly nonlinear dynamics of different notions [138–141]. Th synergetic concept tries to abstract universal principle for the so-called self-organization in many natural systems. According to the synergetic theory moving from an unstable state to a new equilibrium condition happens through an enormous reduction of the system order. However, this transitional process is governed by only few so-called *order parameters*.

A. A. Kolesnikov applied the idea of the synergetic to formulate a nonlinear control method. To illustrate the definition of order parameter, Kolesnikov relies on the invariant manifolds by means of which he describes the lower order dynamics [142], [143] [144]. The synergetic control strategy proposed by Kolesnikov has been widely investigated for control of vast majority of practical systems [145–148].

In fact, invariant manifolds are well known in control science and there are also other approaches established on the similar concept. A more recent study of the invariant manifolds and the notion of order reduction for dealing with control of nonlinear dynamics is the Invariant and Immersion (I&I) technique proposed by A. Astolfi et. al. [149]. However, in their paper more effort is devoted to mathematical formulation of the I&I controller. Work of A. Kolesnikov, which is also prior to I&I, has tried to find a link in the modern physics and focuses on physical insight into the control theory.

A. A. Kolesnikov calls the controller based on the synergetic approach "Analytical Design of Aggregative Regulators" (ADAR) [141]. The word *analytical* refers to the systematic approach of the synergetic control in designing the controller whereas the *aggregation* reflects the order reduction property of it [141, 149, 150].

11.3.1 Time Optimal Control

An interesting problem in applied control, is a plant consisting of a double integrator. It can describe many practical systems. Here, the well known *time optimal* solution to this problem is investigated, mainly, to show *order reduction* property of the control law [144].

$$\dot{x}_1 = x_2 \tag{11.1}$$

$$\dot{x}_2 = u \tag{11.2}$$

where u is subject to the following constraint:

$$|u| < u_{max} \tag{11.3}$$

Optimal control of the system with respect to the dynamic performance is:

$$u(x_1, x_2) = -u_{max} sign\ \varphi(x_1, x_2) \tag{11.4}$$

$$\varphi(x_1, x_2) = x_1 + \frac{0.5}{u_{max}} x_2 |x_2| \tag{11.5}$$

The minimum-time solution represents a *switching* control law. In [151, 152] it is applied to realize Direct SPeed Control (DSPC) of an inverter fed induction motor. Figure 11.1 shows the phase-plane of the closed-loop system. Independent from initial points, all trajectories tend to the invariant $\varphi(x_1, x_2) = 0$ and then move along it to the origin.

The closed-loop dynamic can be divided into two intervals. The first interval is the second-order system described by the initial equations. The second one begins as the trajectory hits the manifold $\varphi(x_1, x_2) = 0$. After reaching this invariant, the order reduction takes place and from this point to the origin the closed-loop system can be described by the first-order dynamic as follows [144]:

$$\dot{x}_1 |\dot{x}_1| = -2 u_{max} x_1 \tag{11.6}$$

Primary observation of this example is that, optimal control law for a linear system is highly *nonlinear*. Further and more significant observation is the *order reduction* notion of the closed-loop system.

11.3.2 Optimal Reaching Trajectory

Time optimal solution to the double integrator requiems extremely fast switching of the controller which is practically not realizable. Declining the switching frequency leads to

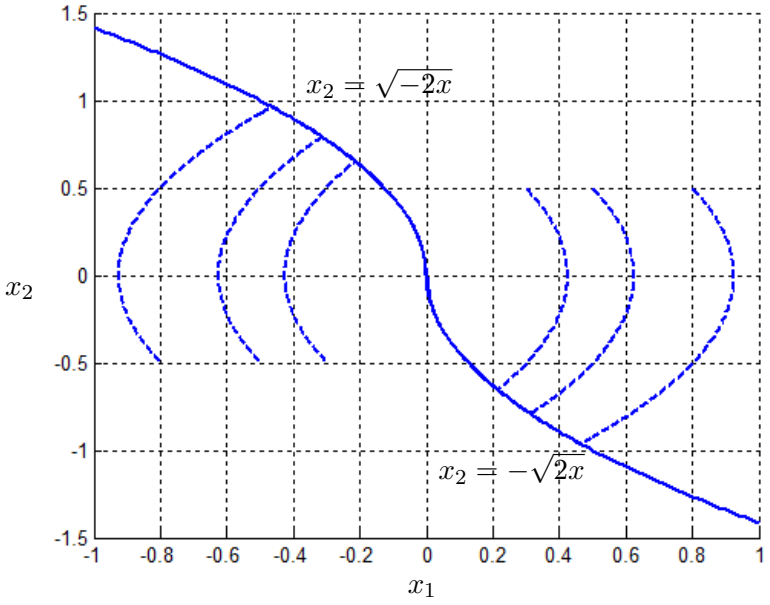

FIGURE 11.1: Phase-plane of the time optimal control trajectories for $u_{max} = 1$

undesired ripples of the controlled variables.

In fact a major part of the controller is definition of the lower order target dynamic. In [144], a generalized *evolution equation* is proposed, ensuring reaching to the lower order target system:

$$T \cdot \frac{d\varphi}{dt} + \varphi = 0 \tag{11.7}$$

$$T > 0 \tag{11.8}$$

where T is the design parameter. Significance of the new formulation is highlighting the order reduction attribute of the optimal solution. The *evolution equation* specifies behavior of the system only before approaching the optimal manifold $\varphi = 0$. Although here a first-order equation is considered, it should not be necessarily linear.

In this example, the design parameter can illustrate switching frequency of the controller. Solving the *evolution equation* leads to the new control law:

$$u = \left(-x_1 - \frac{0.5}{u_{max}} x_2 |x_2| - T \cdot x_2 \right) \cdot \frac{u_{max}}{T \cdot |x_2|} \tag{11.9}$$

$$u \leq u_{max} \tag{11.10}$$

As figure 11.2 shows, the obtained control law enables approaching the manifold $\varphi = 0$ through a smooth transition. The reaching trajectory is manageable by the design parameter.

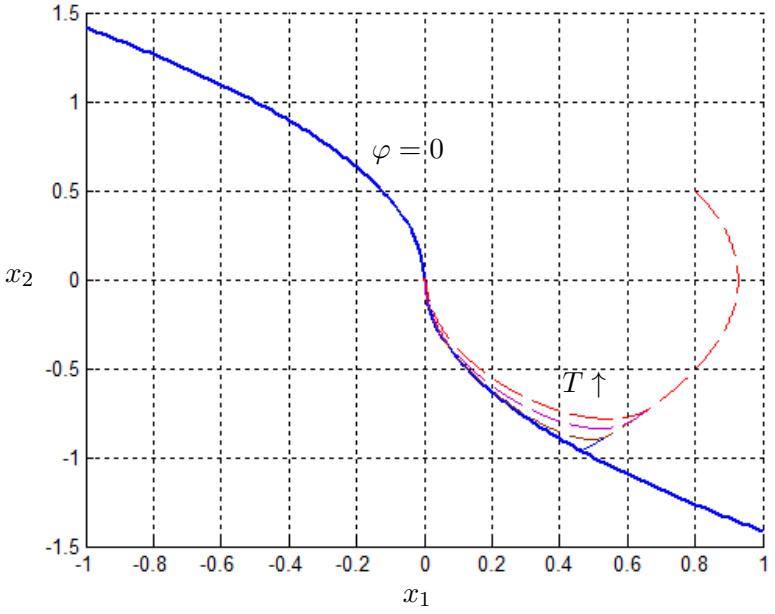

FIGURE 11.2: Phase-plane of the optimal control with various reaching dynamics

11.4 Model Predictive Control with Imposed Target Dynamic

The essential part of the synergetic control is defining a lower order target dynamic. As it is demonstrated through the example, the synergetic control has a strong link to the optimal control. The control law obtained by means of generalized *evolution equation*, ensures the order reduction and reaching the same invariant manifold as the time optimal solution.

Unlike the time optimal solution, the synergetic control law is in continuous form. However such behavior is not available for most drive systems because of switching notion of the power electronics and discretization of the controller. We extend the control scheme from the previous section to make it capable of evaluating time discrete control as well finite number of switching states to select the best possible reaching trajectory with respect to the design parameter.

11.4.1 Discrete Evolution Equation

By means of the forward Euler approximation method, the *evolution equation* 11.7 can be converted into the following predictive form:

$$T\frac{\varphi(k+1)-\varphi(k)}{T_s}+\varphi(k+1)=0 \tag{11.11}$$

where φ is the invariant manifold describing the reduced order target dynamic, T_s discrete time, k sampling instance, T design parameter.

Solution of 11.11 leads to a digital control law that reduces the tracking error with respect to the desiered dynamic performance defined by T, until the manifold $\varphi = 0$ is reached, afterwards behavior of the closed-loop system can be described by the lower order target dynamic.

11.4.2 Cost Function of MPC-ITD

Advantage of the derived control law from 11.7 as continuous or 11.11 as digital form, is that they are not necessarily limited to $-u_{max}$ and u_{max} and can take continuous values to ensure any desired reaching trajectories.

Since control actions in power electronics are bounded and only a finite control set is applicable, the calculated trajectory may not be feasible meaning that it dose not exist among natural trajectories of the system. To predict the optimal natural trajectory, the following cost function must be minimized with respect to the feasible control set:

$$J=\left|T\frac{\varphi(k+1)-\varphi(k)}{T_s}+\varphi(k+1)\right| \tag{11.12}$$

11.5 Conclusion

In this chapter formulation of MPC-ITD for discrete and hybrid systems is investigated. Basic design steps of MPC-ITD are definition of a lower order target dynamic that fulfills technical and technological requirements and afterwards calculation of the optimal trajectory which from any initial condition converges to the defined target dynamic. The discretized evolution equation is proposed that leads to the nonlinear predictive control law. Constraints of controlled variables as well as the control are inherently included in the proposed scheme.

Conventional algorithms of FS-MPC basically enumerate all combinations of the finite control set over the prediction horizon. It requires a huge computational burden which is, in most cases, not realizable or will not be cost effective for the drive applications. In the MPC-ITD scheme there is no need for long prediction horizon since optimality of the predefined trajectories can be ensured independently.

Even though the proposed scheme requires inclusion of the additional terms into the cost function, but it must be emphasized that MPC-ITD introduces an insightful strategy for deigning the cost function. It has two major benefits:

- MPC-ITD is a more insightful approach and gives the physical interpretations for calculation of the additional parameters. It simplifies the design process for control engineers.
- The proposed algorithm reduces the complexity of design and tuning of the controller. It is an important aspect in particular for MIMO systems.

Chapter 12

MPC-ITD for PMSM

12.1 Introduction

High efficiency and dynamic performance of Permanent Magnet Synchronous Motors (PMSM), make them an excellent alternative for demanding applications. In PMSMs, the magnetic field is produced by permanent magnets mounted on the rotor. Therefore iron losses are less than induction motors. Furthermore PMSMs provide high performance and torque density.

In contrast to DC motors, for PMSMs decoupling of stator currents and the magnetic field is done by power electronic devices and control algorithms. Control of PMSMs relies on similar concepts as for IM. Different control schemes have been published for PMSMs. Thanks to many advantages of the model predictive control, this has been widely investigated for PMSM as well.

Direct control of power electronic switches as an alternative to PWM-based control techniques has some advantages. A direct switching strategy can lead to higher dynamic performance since the PWM delay time is eliminated. Another interesting feature which can be gained by applying a direct switching strategy is that the average switching frequency can be significantly decreased without compromising the dynamic performance. It is important in particular for higher power range where switching frequencies are extremely limited.

MPC-ITD developed in the previous chapter is an extension of MPC that includes order reduction concept of the *synergetic control*. The new formulation suggests an insightful approach for design of the cost function. To demonstrate effectiveness of

the proposed control strategy, its application for a PMSM drive is investigated. To maintain direct control of the power electronics, feasible switching states are included in the optimization.

12.2 Mathematical Model of PMSM

For design of any model-based control strategy, a proper model of the controlled plant is crucial. Differential equations are used to derive the dynamic model.

12.2.1 2-Phase Equivalent Model of PMSM

The dynamic of a surface mounted PMSM ($L_d = L_q$) can be demonstrated by the 2-phase equivalent model.

$$\boldsymbol{u}_s = R_s \cdot \boldsymbol{i}_s + \frac{d\boldsymbol{\psi}_s}{dt} \tag{12.1}$$

where $\boldsymbol{u}_s = [u_{s\alpha}\ u_{s\beta}]^T$ and $\boldsymbol{i}_s = [i_{s\alpha}\ i_{s\beta}]^T$ are the stator voltage and current vectors, R_s stator resistance and $\boldsymbol{\psi}_s$ the stator flux linkage. In fact $\boldsymbol{\psi}_s$ represents the air-gap flux. In PMSM, the air-gap flux is produced by the stator windings and the permanent magnets mounted on the rotor:

$$\boldsymbol{\psi}_s = L_s \cdot \boldsymbol{i}_s + \boldsymbol{\psi}_r \tag{12.2}$$

The magnitude of the rotor flux is almost constant. Flux linkage of permanent magnets in stator reference frame can be described as:

$$\boldsymbol{\psi}_r = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \Psi_m \\ 0 \end{bmatrix} \tag{12.3}$$

where $\boldsymbol{\psi}_r = [\psi_{r\alpha}\ \psi_{r\beta}]^T$ is the rotor flux linkage in the stator coordinate, Ψ_m magnitude of the flux produced by the permanent magnets.

$$\frac{d\boldsymbol{i}_s}{dt} = \boldsymbol{u}_s \cdot \frac{1}{L_s} - \frac{R_s}{L_s} \cdot \boldsymbol{i}_s - \frac{1}{L_s} \cdot \frac{d\boldsymbol{\psi}_r}{dt} \tag{12.4}$$

$\frac{d\boldsymbol{\psi}_r}{dt}$ stands for the back emf and can be calculated as follows:

$$\frac{d\boldsymbol{\psi}_r}{dt} = j\omega_e \cdot \boldsymbol{\psi}_r \tag{12.5}$$

$$j = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{12.6}$$

where ω_e is the electrical speed of the rotor. Electromagnetic torque can be described as:

$$T_e = \frac{3}{2} P(\boldsymbol{\psi}_s \times \boldsymbol{i}_s) \tag{12.7}$$

where T_e is the electromagnetic torque, P number of pole pairs. Block diagram of PMSM is depicted on figure 12.1.

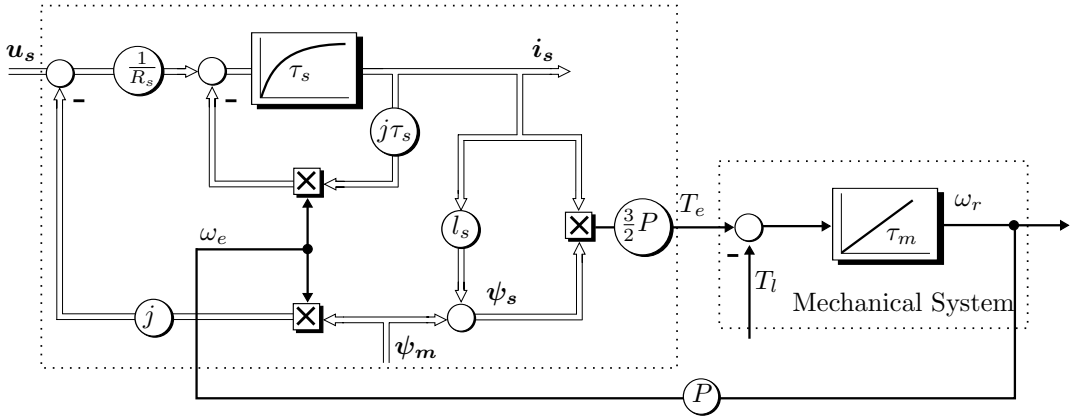

FIGURE 12.1: Signal flow of the PMSM model. $\tau_s = \frac{L_s}{R_s}$

12.3 FS-MPC for PMSM

Due to the simple and intuitive concept of FS-MPC it was investigated for vast majority of applications. To provide a means for comparison of the proposed control method, here we briefly study FS-MPC scheme for PMSM. For more information on this topic, one can refer to [153].

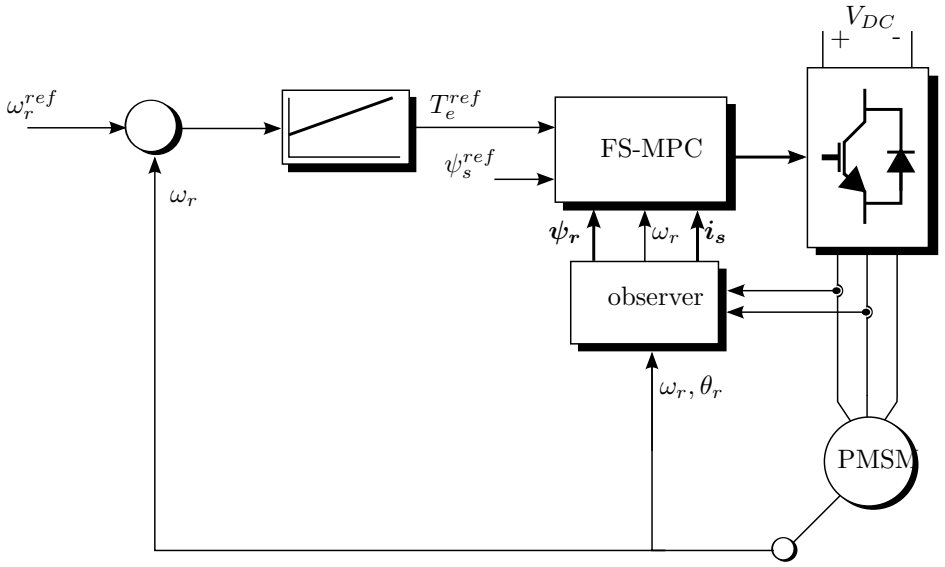

FIGURE 12.2: Block diagram model of FS-MPC

12.3.1 Predictive Model

Optimization of FS-MPC requires a discrete model of PMSM. The model can be developed based on the differential equations:

$$\boldsymbol{i}_s(k+1) = \boldsymbol{i}_s(k) + T_s \cdot \left(\boldsymbol{u}_s(k) \cdot \frac{1}{L_s} - \frac{R_s}{L_s} \cdot \boldsymbol{i}_s(k) - \frac{1}{L_s} \cdot j\omega_e \cdot \boldsymbol{\psi}_r(k) \right) \tag{12.8}$$

$$\boldsymbol{\psi}_s(k+1) = \boldsymbol{\psi}_s(k) + T_s \cdot (\boldsymbol{u}_s(k) - R_s \cdot \boldsymbol{i}_s(k)) \tag{12.9}$$

$$T_e(k+1) = \frac{3}{2} P \cdot (\boldsymbol{\psi}_s(k+1) \times \boldsymbol{i}_s(k+1)) \tag{12.10}$$

Accordingly the predictive model can be built in Simulink and converted into VHDL code by the *HDL coder* tool. The Simulink model for current prediction is depicted on figure 12.3.

12.3.2 Optimization Algorithm

In FS-MPC discrete notion of VSI is directly included in the optimization. Depending on the prediction horizon, at each sampling time a sequence of voltage vectors are calculated, however, only the first vector is applied [129]. The computational complexity of the FS-MPC algorithm exponentially increases for longer prediction horizon. Therefore only one step prediction is mostly considered [133].

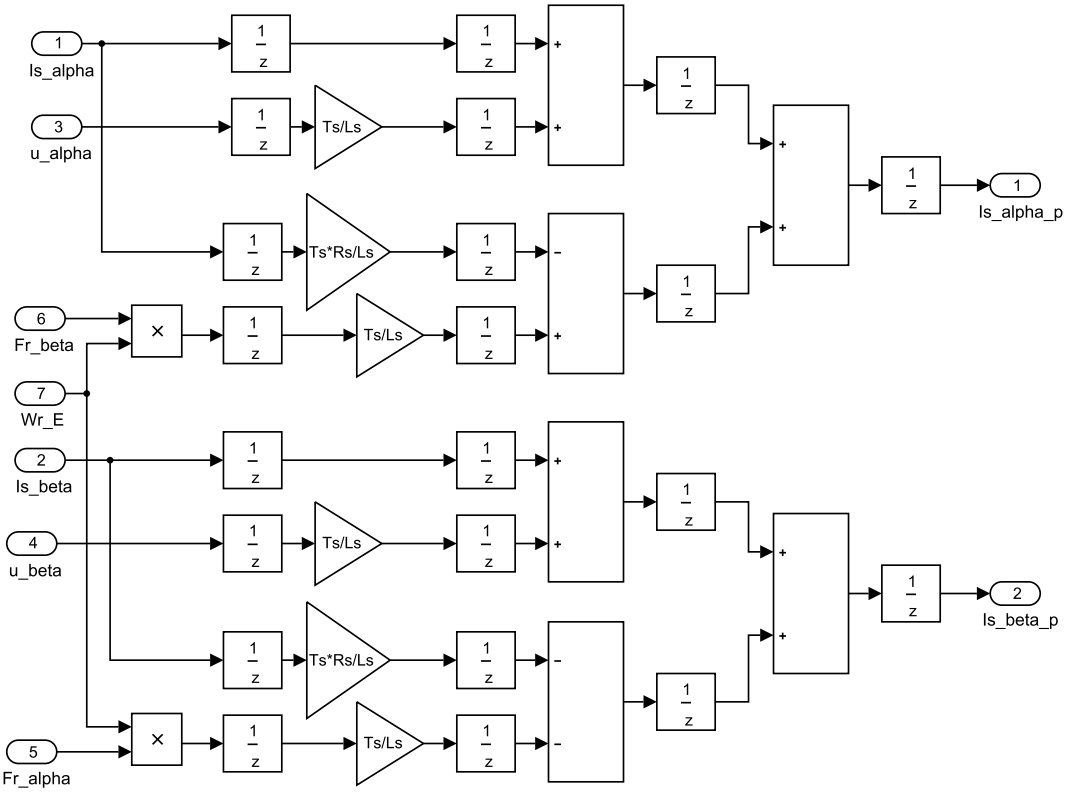

FIGURE 12.3: Simulink model for Current Prediction

In many scientific papers a processor-based computational system are utilized for implementation of the algorithm. Due to that the computational delay must be explicitly compensated to ensure proper operation of FS-MPC. Even though it might be not challenging to eliminate or reduce the effect of the delay, it leads to extra complexity of the algorithm. In our digital platform thanks to the computational performance of the FPGA, the computation time is only a fraction of the sampling interval. It allows to obtain nearly ideal results.

The proposed hardware model of FS-MPC for IM (see figure 8.7), which was demonstrated in chapter 8, is compatible to be used for FS-MPC of PMSM. It requires only modification of the prediction model and the cost function.

12.3.3 Model-In-the-Loop Simulation of FS-MPC

To evaluate FS-MPC algorithm, FPGA-based MIL simulation is carried out. In the MIL scheme the control algorithm as well as the drive model are implemented on the FPGA. Short integration step of the drive model on the FPGA, allows to realize a more precise simulation model including switching dead-time of the inverter.

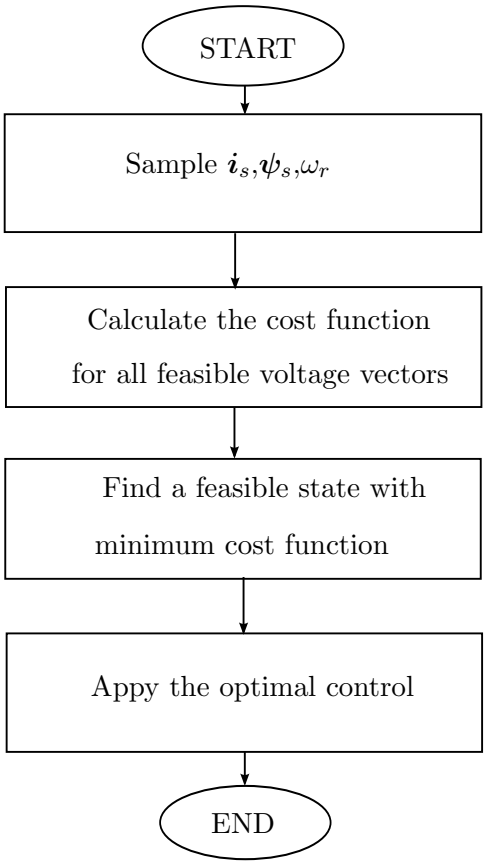

FIGURE 12.4: FS-MPC algorithm for PMSM

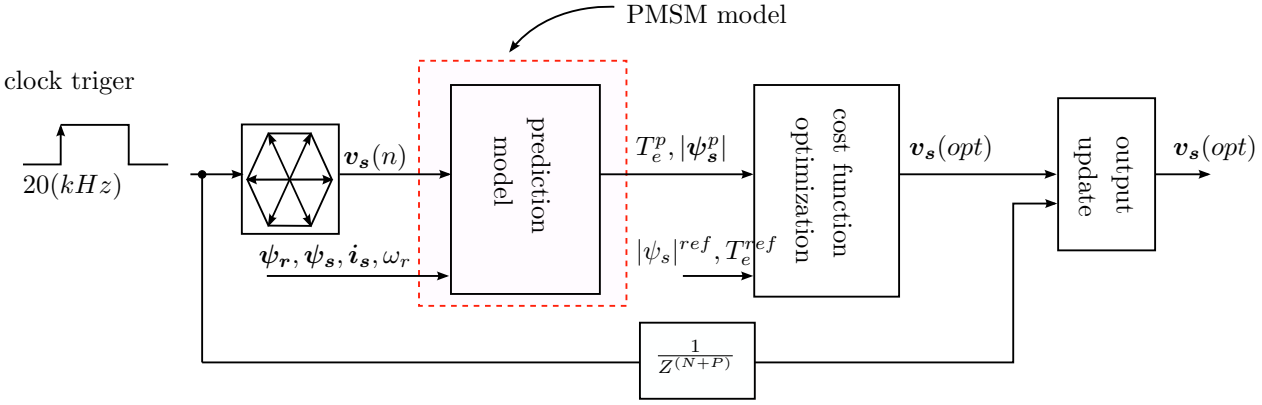

FIGURE 12.5: Hardware Model of FS-MPC for PMSM

Dynamic performance of FS-MPC for PMSM is shown on figure 12.6.

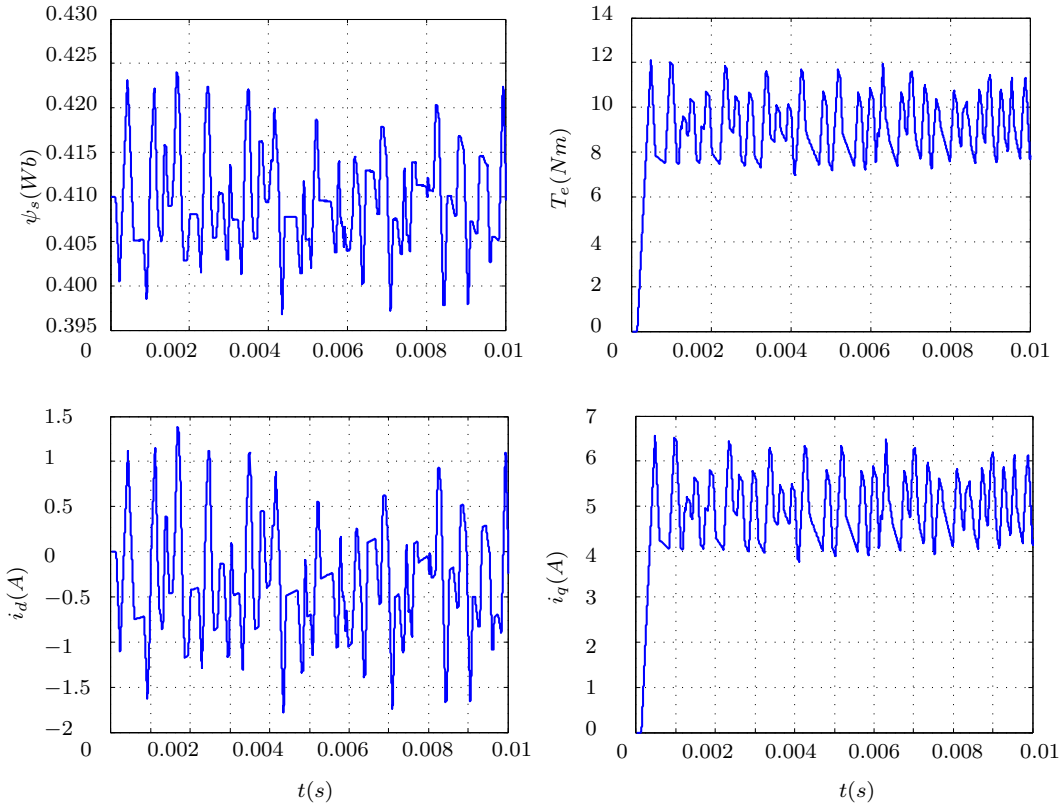

FIGURE 12.6: Dynamic performance of FS-MPC for PMSM, $G_T = 0.01$.

12.4 MPC-ITD for PMSM

As the first step of designing MPC-ITD for PMSM, the target dynamic needs to be defined. The control objective is keeping the electromagnetic torque and stator flux at the reference levels. In PMSM the air-gap flux is mainly produced by the permanent magnets mounted on the rotor. The stator windings are responsible only for producing torque component of the current i_q. However, in field weakening region, stator current should also contribute to the air-gap flux.

Invariant manifolds reflecting the control task can be described as follows:

$$\varphi_1 = T_e - T_e^{ref} \tag{12.11}$$

$$\varphi_2 = \psi_s - \psi_s^{ref} \tag{12.12}$$

where T_e, T_e^{ref}, ψ_s and ψ_s^{ref} are electromagnetic torque and flux linkage and their reference values.

Accordingly the *evolution equation* can be written as:

$$\sigma_1 \frac{dT_e}{dt} + T_e - T_e^{ref} = 0 \qquad \sigma_1 > 0 \tag{12.13}$$

$$\sigma_2 \frac{d\psi_s}{dt} + \psi_s - \psi_s^{ref} = 0 \qquad \sigma_2 > 0 \tag{12.14}$$

where T_e and ψ_s are the torque and flux respectively, σ_1 and σ_1 are design parameters. These differential equations describe a dynamic system with equilibrium point of $T_e = T_e^{ref}$, $\psi_s = \psi_s^{ref}$ and the design parameters are related to the stability and reaching time for falling into the equilibrium point. Based on the formulation introduced in the previous chapter the following predictive dynamic equation can be derived:

$$\left(\frac{\sigma_1}{T_s} + 1\right) \cdot T_e(k+1) = \frac{\sigma_1}{T_s} T_e(k) + T_e^{ref}(k+1) \tag{12.15}$$

$$\left(\frac{\sigma_2}{T_s} + 1\right) \cdot \psi_s(k+1) = \frac{\sigma_2}{T_s} \psi_s(k) + \psi_s^{ref}(k+1) \tag{12.16}$$

where T_s is the sampling time. The task of the controller is tracking this target dynamic by minimizing the following cost function:

$$J = \left|(\frac{\sigma_1}{T_s}+1)\cdot T_e(k+1) - T_e^{ref}(k+1) - \frac{\sigma_1}{T_s} T_e(k)\right| + \left|(\frac{\sigma_2}{T_s}+1)\cdot \psi_s(k+1) - \psi_s^{ref}(k+1) - \frac{\sigma_2}{T_s}\psi_s(k)\right| \tag{12.17}$$

Since the gain factor is omitted in the cost function, normalized values of the controlled variables must be used.

12.5 FPGA Implementation of MPC-ITD and Experimental Results

For experimental validation, MPC-ITD is implemented on the FPGA-based laboratory setup. The inverter is designed in the lab whereas the test motor is a commercial PMSM. Results of the torque control are shown on figure 12.7. As expected, compared to conventional FS-MPC, ripples of controlled variables are considerably reduced. It is interesting to notice that both ripples and switching frequencies are decreased at the same time. In FS-MPC only tracking errors are included in the cost function. In addition

to the dynamic error, MPC-ITD includes the predicted derivative of the error into the performance index. It allows to further optimize the reaching trajectories without the need for long prediction horizons.

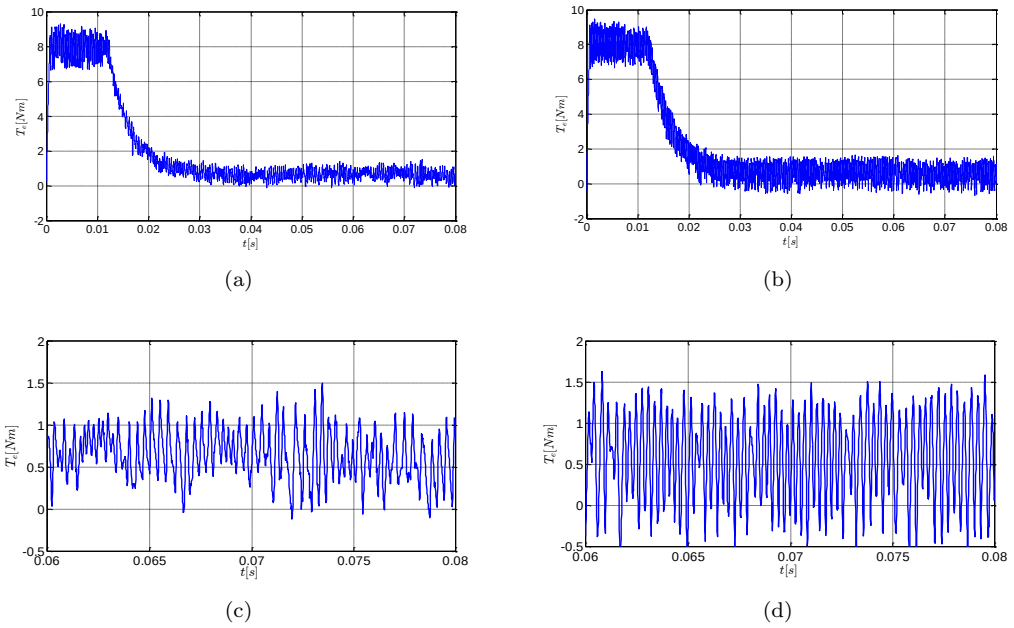

FIGURE 12.7: Measurement results of the electromagnetic torque control in transient and steady state operation. Rotor speed is $50(\frac{rad}{s})$, sampling frequency $20(KHz)$. (a) and (c) show the results of the proposed MPC-ITD while (b) and (d) represent conventional FS-MPC.

12.6 Conclusion

In this chapter the MPC-ITD algorithm is investigated for the PMSM drive. The imposed target dynamic enables to modify the cost function of FS-MPC. The new cost function gives an insight into gain factor calculation with respect to the defined performance criteria.

By defining a target dynamic the MPC algorithm is able to find natural trajectories with global optimality by only one step prediction horizon. It allows to reduce the switching frequencies of the VSI at the same time improve steady state performance of the torque and flux control.

Optimization algorithm of MPC-ITD has the same level of complexity as FS-MPC. It is easily realizable on FPGAs. Thanks to parallelism and pipelined implementation of the FPGA model, the computational performance is very high and the computation time can be neglected.

Chapter 13

Conclusion and Future Works

In the work presented here applications of nonlinear model predictive control for the electrical drives, with focus on FPGA-based implementation, have been investigated. Advancing in automation technology requires more and more controlled electrical drives. The bandwidth and accuracy of the control system must be improved to meet technical and technological requirements. In this respect new control algorithms and digital hardware are studied.

13.1 Summary of Contributions

13.1.1 FPGA-Based Design

One topic being investigated in this PhD research is FPGA-based design. By advancing in the hardware programmable devices, there are increasing attentions paid to FPGAs for high performance real-time computation. In this direction some contributions are made to deal with challenges of FPGA-based development in particular for electrical drive and power electronic applications.

13.1.1.1 FPGA-Based Rapid Prototyping

FPGA programming is in the cross section of the hardware and software engineering. HDL programs describe a physical structure of digital circuits which must be realized by means of configurable logic blocks available inside the FPGA chip. To ensure proper operation of the FPGA, it requires on the one hand hardware design considerations

to confirm stability and timing constraints and logical functionality test on the other. Due to parallelism and distributed architecture of FPGAs, a precise computer-based emulation of FPGAs is too complex and in reality the target FPGA chip has to be used for the design verification.

Thanks to the capability of FPGAs to adapt to algorithms and to perform parallel tasks, the computational performance of FPGA-based systems is higher than state-of-the-art of hard-core processors even using low cost FPGA devices at much lower clock frequencies. In addition, more point-to-point connections potentially can improve the reliability and safety of the FPGA-based systems.

Despite all these advantages, programming of FPGAs and design verification are main challenges for expanding their applications. In the present research, a hardware platform is proposed for rapid prototyping and functionality test of algorithms on the target FPGA. In the proposed scheme, data are sampled at each FPGA clock event and then sent to the Simulink model. Since FPGA is much faster than Simulink model, data is stored in on-chip and on-board memory blocks and then is sent to the host PC for playing back synchronized with the Simulink model. Unlike similar commercial solutions such as FPGA-In-the-Loop provided by Mathworks, our solution supports multi-clock design.

Indeed design environment and choose of HDLs for the FPGA programming plays a major role in the FPGA-based development time. Standard HDLs are VHDL and Verilog. These are textual hardware configuration languages. They represent high flexibility to access hardware resources and optimization of resource usage. However, for rapid prototyping, the graphical modeling tools are preferred. Model-based design tools are in particular well prepared for design automation and dealing with complexity and most importantly documentation of the FPGA-based projects.

Throughout this thesis, both VHDL hand coding and using *HDL coder* of Mathworks are considered. Almost all of hardware models are created using VHDL and in Simulink as well. While VHDL hand coding gives more flexibility and better FPGA resource optimization, using *HDL coder* has the following benefits:

- Fixed point data type and data type conversion can be easier handled.
- Simulink is more developed for dealing with the complexity and hierarchical design.
- Parametrization of models in Simulink is more flexible.
- Off-line simulation capability of Simulink allows to verify logical correctness and synchronization of entities, before implementing on the target FPGA device.

At the time *HDL coder* can be helpful mainly for rapid prototyping. For end implementations hand coding is more efficient with respect to the logic area usage and speed of the design.

13.1.2 Control Algorithm for FPGA Implementation

FPGAs increase the computational performance since the hardware model can be well adapted to any control algorithms. There are two main techniques for optimization and enhancing the computational performance of FPGAs: Pipelining and parallelism. To effectively make use of these strategies, control algorithms must be properly adapted to the FPGA as well. Thus, part of the thesis was devoted to developing of control algorithms being suitable for the FPGA implementation.

In this thesis two control methods for electric drives have been investigated. An iteration-based numerical solution is proposed for Continuous Set Model Predictive Control (CS-NMPC). It allows to utilize logic resources efficiently. Pipelining architecture of the FPGA-model of CS-NMPC enables to reduce the execution time. The total computation time is less than 10 μs which can be safely neglected.

Another control method, being investigated here, is so-called Model Predictive Control with Imposed Target Dynamic (MPC-ITD). While CS-NMPC calculates duty cycle of power electronic switches within a fixed modulation period, MPC-ITD directly outputs switching state of the inverter so that the modulator can be eliminated. Computation time is a crucial factor for any control strategy. It is even more significant for direct control of the inverter. Thanks to the FPGA-based implementation and well optimized hardware architecture, execution time of MPC-ITD is less than 1 μs and can be safely neglected considering the time constant of the drives.

13.1.2.1 Continuous-Set Nonlinear Model Predictive Control

The numerical solution developed for cost function optimization of CS-NMPC allows to include all nonlinearities and constraints of the model. Further control objectives can be included in the optimization process simply by imposing additional terms into the cost function.

For the same control tasks, compared to classical controllers such as PID, number of design parameters is reduced. For control of the flux and torque, CS-NMPC needs only one gain factor to be tuned.

CS-NMPC is implemented on the FPGA-based experimental platform to control torque and flux of a commercial IM. In addition to its high performance, CS-NMPC enables to decouple torque and flux control loops.

13.1.2.2 Model Predictive Control with Imposed Target Dynamic

For direct control of the inverter so-called FS-MPC method is developed over the recent years. The main idea of FS-MPC is to put finite state of the inverter directly into the cost function. At each sampling time switching state of the inverter is changed so that the cost function is minimized. Thanks to the flexibility of FS-MPC for including several goals into a single cost function, generality of the concept and intuitiveness of parameter tuning, it has received high acceptance from research communities.

Cost function design and calculation of the gain factors are the most reported challenges of FS-MPC. Cost function design is, in particular for relatively complex systems of significant importance, since FS-MPC provides no direct connection between the controller performance and the design parameters. This is the divergence of FS-MPC from traditional control methods such as PID. To deal with these issues the concept of *order reduction* is proposed.

The MPC-ITD provides a more insightful approach to the design of the cost function. To demonstrate performance of the proposed algorithm, experimental tests on the PMSM drive are carried out. For direct switching strategies, output ripples and the average switching frequency of the inverter are two main evaluation factors. In fact, from physic of motors these parameters are usually contradictory i.e. reducing switching frequencies leads to higher torque and flux ripples. Thanks to the improvement of switching time with respect to the defined reference target dynamic, MPC-ITD makes it possible to reduce switching frequencies and ripples at the same time.

The important feature of MPC-ITD compared to FS-MPC, is that instead of static output error it considers the tracking error with respect to the predefined target dynamic. It improves global optimality of the closed-loop system.

Even though additional parameters are included in the cost function, MPC-ITD provides physical interpretations which simplify the design and tuning of the controller.

13.2 Future Research Directions

The aim of doing research in engineering disciplines is mostly developing new methods to solve practical problems or improving existing solutions. Nevertheless through the research new challenges and motivations for further investigations are discovered. Such findings are, perhaps, as important as the obtained results.

13.2.1 Future Challenges for FPGA-Based Design

Opportunities and challenges of the FPGA applications are tightly related to each other. Considering rapidly increasing performance and logic resources offered by the FPGA technology, it is obviously a need for automatic code generation to reduce the development time. Existing HDL code generators have poor optimization capabilities and still for an effective usage of the logic resources, the designer must be involved in the low level design considerations. While application area of FPGAs is permanently expanding, developing FPGA-based design automation tools shall encourage more research interests.

Advances in nano-technology enable to integrate configurable logic elements, hard-cores and analog components in a single chip. It simplifies PCB design and improves reliability of embedded systems. Nevertheless design complexity is increased. Effective usage of the chip resources and rapid prototyping require respective computer aided design tools.

13.2.2 Future Perspective of Nonlinear Model Predictive Control

Results obtained in this work show effectiveness of the nonlinear model predictive control methods for electrical drives and power electronics. The proposed algorithms and the FPGA-based implementation guidelines can be extended to various drive types and power electronic systems.

Appendix A

CORDIC Algorithm for Park Transformation

A.1 CORDIC Algorithm

CORDICs are iteration-based algorithms for numerical calculation of trigonometric functions. The main feature of the CORDIC algorithms is the simple hardware implementation. Pipelining is an effective technique for enhancing the performance and data throughput of the iteration-based algorithms.

Simulink incorporates many graphical and textual programming tools. The so-called *MATLAB Function* block enables to integrate embedded MATLAB codes into the Simulink models. It supports *Fixed Point* as well as Bit-Wise operations. Furthermore it can be converted into VHDL by means of *HDL coder*. For more detail the reader can refer to the Simulink help.

A.1.1 MATLAB Code for Park Transformation

While graphical programming language of Simulink is appropriate for having a clear overview of the model, textual programming of MATLAB is more compact and portable. moreover it can be synthesized with various text editor tools.

As it can be seen from the provided code here, the CORDIC model of Park transformation consists of 16 pipeline stages. The MATLAB code is optimized so that it can be converted into VHDL by means of the *HDL coder* tool.

```
% Author: Saeid Saeidi
% EAL-TUM
% MATLAB Version: R2012b


function [d, q]= fcn(a,b,theta)

%Defining persistent variables
%In VHDL, persistent variables will appear as Signals

persistent i1
persistent i2
persistent i3
persistent i4
persistent i5
persistent i6
persistent i7
persistent i8
persistent i9
persistent i10
persistent i11
persistent i12
persistent i13
persistent i14
persistent i15
persistent i16
% %
persistent i1n
persistent i2n
persistent i3n
persistent i4n
persistent i5n
persistent i6n
persistent i7n
persistent i8n
persistent i9n
persistent i10n
persistent i11n
persistent i12n
persistent i13n
persistent i14n
persistent i15n
persistent i16n
```

```
% %
persistent a1
persistent a2
persistent a3
persistent a4
persistent a5
persistent a6
persistent a7
persistent a8
persistent a9
persistent a10
persistent a11
persistent a12
persistent a13
persistent a14
persistent a15
persistent a16
% %
persistent b1
persistent b2
persistent b3
persistent b4
persistent b5
persistent b6
persistent b7
persistent b8
persistent b9
persistent b10
persistent b11
persistent b12
persistent b13
persistent b14
persistent b15
persistent b16
% %
persistent beta1
persistent beta2
persistent beta3
persistent beta4
persistent beta5
persistent beta6
persistent beta7
persistent beta8
persistent beta9
```

```
persistent beta10
persistent beta11
persistent beta12
persistent beta13
persistent beta14
persistent beta15
% %
persistent a1n
persistent a2n
persistent a3n
persistent a4n
persistent a5n
persistent a6n
persistent a7n
persistent a8n
persistent a9n
persistent a10n
persistent a11n
persistent a12n
persistent a13n
persistent a14n
persistent a15n
persistent a16n
% %
persistent b1n
persistent b2n
persistent b3n
persistent b4n
persistent b5n
persistent b6n
persistent b7n
persistent b8n
persistent b9n
persistent b10n
persistent b11n
persistent b12n
persistent b13n
persistent b14n
persistent b15n
persistent b16n
% %
persistent beta1n
persistent beta2n
persistent beta3n
```

```
persistent beta4n
persistent beta5n
persistent beta6n
persistent beta7n
persistent beta8n
persistent beta9n
persistent beta10n
persistent beta11n
persistent beta12n
persistent beta13n
persistent beta14n
persistent beta15n
% %

%Initialization of Persistent Variables
%After generating VHDL this part will be located in the RESET

if isempty(a1)
    a1  = fi(0, numerictype(a), hdlfimath);
    a2  = fi(0, numerictype(a), hdlfimath);
    a3  = fi(0, numerictype(a), hdlfimath);
    a4  = fi(0, numerictype(a), hdlfimath);
    a5  = fi(0, numerictype(a), hdlfimath);
    a6  = fi(0, numerictype(a), hdlfimath);
    a7  = fi(0, numerictype(a), hdlfimath);
    a8  = fi(0, numerictype(a), hdlfimath);
    a9  = fi(0, numerictype(a), hdlfimath);
    a10 = fi(0, numerictype(a), hdlfimath);
    a11 = fi(0, numerictype(a), hdlfimath);
    a12 = fi(0, numerictype(a), hdlfimath);
    a13 = fi(0, numerictype(a), hdlfimath);
    a14 = fi(0, numerictype(a), hdlfimath);
    a15 = fi(0, numerictype(a), hdlfimath);
    a16 = fi(0, numerictype(a), hdlfimath);
    % %
    b1  = fi(0, numerictype(b), hdlfimath);
    b2  = fi(0, numerictype(b), hdlfimath);
    b3  = fi(0, numerictype(b), hdlfimath);
    b4  = fi(0, numerictype(b), hdlfimath);
    b5  = fi(0, numerictype(b), hdlfimath);
    b6  = fi(0, numerictype(b), hdlfimath);
    b7  = fi(0, numerictype(b), hdlfimath);
    b8  = fi(0, numerictype(b), hdlfimath);
    b9  = fi(0, numerictype(b), hdlfimath);
```

```
b10 = fi(0, numerictype(b), hdlfimath);
b11 = fi(0, numerictype(b), hdlfimath);
b12 = fi(0, numerictype(b), hdlfimath);
b13 = fi(0, numerictype(b), hdlfimath);
b14 = fi(0, numerictype(b), hdlfimath);
b15 = fi(0, numerictype(b), hdlfimath);
b16 = fi(0, numerictype(b), hdlfimath);
% %
beta1  = fi(0, numerictype(theta), hdlfimath);
beta2  = fi(0, numerictype(theta), hdlfimath);
beta3  = fi(0, numerictype(theta), hdlfimath);
beta4  = fi(0, numerictype(theta), hdlfimath);
beta5  = fi(0, numerictype(theta), hdlfimath);
beta6  = fi(0, numerictype(theta), hdlfimath);
beta7  = fi(0, numerictype(theta), hdlfimath);
beta8  = fi(0, numerictype(theta), hdlfimath);
beta9  = fi(0, numerictype(theta), hdlfimath);
beta10 = fi(0, numerictype(theta), hdlfimath);
beta11 = fi(0, numerictype(theta), hdlfimath);
beta12 = fi(0, numerictype(theta), hdlfimath);
beta13 = fi(0, numerictype(theta), hdlfimath);
beta14 = fi(0, numerictype(theta), hdlfimath);
beta15 = fi(0, numerictype(theta), hdlfimath);
% %
a1n  = fi(0, numerictype(a), hdlfimath);
a2n  = fi(0, numerictype(a), hdlfimath);
a3n  = fi(0, numerictype(a), hdlfimath);
a4n  = fi(0, numerictype(a), hdlfimath);
a5n  = fi(0, numerictype(a), hdlfimath);
a6n  = fi(0, numerictype(a), hdlfimath);
a7n  = fi(0, numerictype(a), hdlfimath);
a8n  = fi(0, numerictype(a), hdlfimath);
a9n  = fi(0, numerictype(a), hdlfimath);
a10n = fi(0, numerictype(a), hdlfimath);
a11n = fi(0, numerictype(a), hdlfimath);
a12n = fi(0, numerictype(a), hdlfimath);
a13n = fi(0, numerictype(a), hdlfimath);
a14n = fi(0, numerictype(a), hdlfimath);
a15n = fi(0, numerictype(a), hdlfimath);
a16n = fi(0, numerictype(a), hdlfimath);
% %
b1n  = fi(0, numerictype(b), hdlfimath);
b2n  = fi(0, numerictype(b), hdlfimath);
b3n  = fi(0, numerictype(b), hdlfimath);
```

```
b4n  = fi(0, numerictype(b), hdlfimath);
b5n  = fi(0, numerictype(b), hdlfimath);
b6n  = fi(0, numerictype(b), hdlfimath);
b7n  = fi(0, numerictype(b), hdlfimath);
b8n  = fi(0, numerictype(b), hdlfimath);
b9n  = fi(0, numerictype(b), hdlfimath);
b10n = fi(0, numerictype(b), hdlfimath);
b11n = fi(0, numerictype(b), hdlfimath);
b12n = fi(0, numerictype(b), hdlfimath);
b13n = fi(0, numerictype(b), hdlfimath);
b14n = fi(0, numerictype(b), hdlfimath);
b15n = fi(0, numerictype(b), hdlfimath);
b16n = fi(0, numerictype(b), hdlfimath);
% %
beta1n  = fi(0, numerictype(theta), hdlfimath);
beta2n  = fi(0, numerictype(theta), hdlfimath);
beta3n  = fi(0, numerictype(theta), hdlfimath);
beta4n  = fi(0, numerictype(theta), hdlfimath);
beta5n  = fi(0, numerictype(theta), hdlfimath);
beta6n  = fi(0, numerictype(theta), hdlfimath);
beta7n  = fi(0, numerictype(theta), hdlfimath);
beta8n  = fi(0, numerictype(theta), hdlfimath);
beta9n  = fi(0, numerictype(theta), hdlfimath);
beta10n = fi(0, numerictype(theta), hdlfimath);
beta11n = fi(0, numerictype(theta), hdlfimath);
beta12n = fi(0, numerictype(theta), hdlfimath);
beta13n = fi(0, numerictype(theta), hdlfimath);
beta14n = fi(0, numerictype(theta), hdlfimath);
beta15n = fi(0, numerictype(theta), hdlfimath);
% %
i1n  = ufi(0,1,0);
i2n  = ufi(0,1,0);
i3n  = ufi(0,1,0);
i4n  = ufi(0,1,0);
i5n  = ufi(0,1,0);
i6n  = ufi(0,1,0);
i7n  = ufi(0,1,0);
i8n  = ufi(0,1,0);
i9n  = ufi(0,1,0);
i10n = ufi(0,1,0);
i11n = ufi(0,1,0);
i12n = ufi(0,1,0);
i13n = ufi(0,1,0);
i14n = ufi(0,1,0);
```

```
    i15n = ufi(0,1,0);
    i16n = ufi(0,1,0);
    % %
    i1  = ufi(0,1,0);
    i2  = ufi(0,1,0);
    i3  = ufi(0,1,0);
    i4  = ufi(0,1,0);
    i5  = ufi(0,1,0);
    i6  = ufi(0,1,0);
    i7  = ufi(0,1,0);
    i8  = ufi(0,1,0);
    i9  = ufi(0,1,0);
    i10 = ufi(0,1,0);
    i11 = ufi(0,1,0);
    i12 = ufi(0,1,0);
    i13 = ufi(0,1,0);
    i14 = ufi(0,1,0);
    i15 = ufi(0,1,0);
    i16 = ufi(0,1,0);
end
% %

%Definition of Constant Values

p   =  fi(pi,numerictype(theta), hdlfimath);
p2  =  fi(pi/2,numerictype(theta), hdlfimath);    %1.570796
p2m =  fi(-pi/2,numerictype(theta), hdlfimath);
theta0 = fi(theta,numerictype(theta), hdlfimath);


%Determining section of the theta
%---------                              # First pipeline stage

  if(theta0 <= fi(p2m,numerictype(theta), hdlfimath))
     beta1n = fi(theta0 + p,numerictype(theta), hdlfimath);
         i1n    = ufi(0,1,0);
elseif(theta0 >= fi(p2,numerictype(theta), hdlfimath))
     beta1n = fi(theta0 - p,numerictype(theta), hdlfimath);
         i1n    = ufi(0,1,0);
else
     beta1n = theta0;
     i1n    = ufi(1,1,0);

end
```

```
      a1n = fi(a, numerictype(a), hdlfimath);
      b1n = fi(b, numerictype(b), hdlfimath);


%---------                                  # Second pipeline stage

 if(beta1 <fi(0,numerictype(theta), hdlfimath))
         a2n    = fi(a1 + b1, numerictype(a), hdlfimath);
         b2n    = fi(b1 - a1, numerictype(b), hdlfimath);
         beta2n = fi(beta1 + fi(0.78539816339745,numerictype(theta),...
                  hdlfimath),numerictype(theta), hdlfimath);
 else
         a2n    = fi(a1 - b1, numerictype(a), hdlfimath);
         b2n    = fi(b1 + a1, numerictype(b), hdlfimath);
         beta2n = fi(beta1 - fi(0.78539816339745,numerictype(theta),...
                  hdlfimath),numerictype(theta), hdlfimath);
 end


%---------                                  # Third pipeline stage

 if(beta2<fi(0,numerictype(theta), hdlfimath))
         a3n    = fi(a2 +  bitsra(b2,1), numerictype(b), hdlfimath);
         b3n    = fi(b2 - bitsra(a2,1), numerictype(a), hdlfimath);
         beta3n = fi(beta2 + fi(0.46364760900081,numerictype(theta),...
                  hdlfimath),numerictype(theta), hdlfimath);
 else
         a3n    = fi(a2 - bitsra(b2,1), numerictype(a), hdlfimath);
         b3n    = fi(b2 + bitsra(a2,1), numerictype(b), hdlfimath);
         beta3n = fi( beta2 - fi(0.46364760900081,numerictype(theta),...
                  hdlfimath),numerictype(theta), hdlfimath);
 end


%---------                                  # Fourth pipeline stage

 if(beta3<fi(0,numerictype(theta), hdlfimath))
         a4n    = fi(a3 + bitsra(b3,2), numerictype(a), hdlfimath);
         b4n    = fi(b3 - bitsra(a3,2), numerictype(b), hdlfimath);
         beta4n = fi(beta3 + fi(0.24497866312686,numerictype(theta),...
                  hdlfimath),numerictype(theta), hdlfimath);
 else
```

```
        a4n    = fi(a3 - bitsra(b3,2), numerictype(a), hdlfimath);
        b4n    = fi(b3 + bitsra(a3,2), numerictype(b), hdlfimath);
        beta4n = fi( beta3 - fi(0.24497866312686,numerictype(theta),...
                 hdlfimath),numerictype(theta), hdlfimath);
 end


%---------                                  # Fifth pipeline stage

 if(beta4<fi(0,numerictype(theta), hdlfimath))
        a5n    = fi(a4 + bitsra(b4,3), numerictype(a), hdlfimath);
        b5n    = fi(b4 - bitsra(a4,3), numerictype(b), hdlfimath);
        beta5n = fi( beta4 + fi(0.12435499454676,numerictype(theta),...
                 hdlfimath),numerictype(theta), hdlfimath);
 else
        a5n    = fi(a4 - bitsra(b4,3), numerictype(a), hdlfimath);
        b5n    = fi(b4 + bitsra(a4,3), numerictype(b), hdlfimath);
        beta5n = fi(beta4 - fi(0.12435499454676,numerictype(theta),...
                 hdlfimath),numerictype(theta), hdlfimath);
 end


%---------                                  # Sixth pipeline stage

 if(beta5<fi(0,numerictype(theta), hdlfimath))
        a6n    = fi(a5 + bitsra(b5,4), numerictype(a), hdlfimath);
        b6n    = fi(b5 - bitsra(a5,4), numerictype(b), hdlfimath);
        beta6n = fi( beta5 + fi(0.06241880999596,numerictype(theta),...
                 hdlfimath),numerictype(theta), hdlfimath);
 else
        a6n    = fi(a5 - bitsra(b5,4), numerictype(a), hdlfimath);
        b6n    = fi( b5 + bitsra(a5,4), numerictype(b), hdlfimath);
        beta6n = fi(beta5 - fi(0.06241880999596,numerictype(theta),...
                 hdlfimath),numerictype(theta), hdlfimath);
 end


%---------                                  # Seventh pipeline stage

 if(beta6<fi(0,numerictype(theta), hdlfimath))
        a7n    = fi(a6 + bitsra(b6,5), numerictype(a), hdlfimath);
        b7n    = fi(b6 - bitsra(a6,5), numerictype(b), hdlfimath);
        beta7n = fi( beta6 + fi(0.03123983343027,numerictype(theta),...
                 hdlfimath),numerictype(theta), hdlfimath);
```

```
 else
         a7n    = fi(a6 - bitsra(b6,5), numerictype(a), hdlfimath);
         b7n    = fi(b6 + bitsra(a6,5), numerictype(b), hdlfimath);
         beta7n = fi(beta6 - fi(0.03123983343027,numerictype(theta),...
                  hdlfimath),numerictype(theta), hdlfimath);
 end


%---------                                  # Eighth pipeline stage

 if(beta7<fi(0,numerictype(theta), hdlfimath))
         a8n    = fi(a7 + bitsra(b7,6), numerictype(a), hdlfimath);
         b8n    = fi(b7 - bitsra(a7,6), numerictype(b), hdlfimath);
         beta8n = fi(beta7 + fi(0.01562372862048,numerictype(theta),...
                  hdlfimath),numerictype(theta), hdlfimath);
 else
         a8n    = fi(a7 - bitsra(b7,6), numerictype(a), hdlfimath);
         b8n    = fi(b7 + bitsra(a7,6), numerictype(b), hdlfimath);
         beta8n = fi(beta7 - fi(0.01562372862048,numerictype(theta),...
                  hdlfimath),numerictype(theta), hdlfimath);
 end


%---------                                  # Ninth pipeline stage

 if(beta8<fi(0,numerictype(theta), hdlfimath))
         a9n    = fi(a8 + bitsra(b8,7), numerictype(a), hdlfimath);
         b9n    = fi(b8 - bitsra(a8,7), numerictype(b), hdlfimath);
         beta9n = fi( beta8 + fi( 0.00781234106010,numerictype(theta),...
                  hdlfimath),numerictype(theta), hdlfimath);
 else
         a9n    = fi(a8 - bitsra(b8,7), numerictype(a), hdlfimath);
         b9n    = fi( b8 + bitsra(a8,7), numerictype(b), hdlfimath);
         beta9n = fi(beta8 - fi( 0.00781234106010,numerictype(theta),...
                  hdlfimath),numerictype(theta), hdlfimath);
 end


%---------                                  # Tenth pipeline stage

 if(beta9<fi(0,numerictype(theta), hdlfimath))
         a10n    = fi(a9 + bitsra(b9,8), numerictype(a), hdlfimath);
         b10n    = fi(b9 - bitsra(a9,8), numerictype(b), hdlfimath);
         beta10n = fi(beta9 + fi(0.00390623013197,numerictype(theta),...
```

```
                    hdlfimath),numerictype(theta), hdlfimath);
 else
         a10n    = fi(a9 - bitsra(b9,8), numerictype(a), hdlfimath);
         b10n    = fi(b9 + bitsra(a9,8), numerictype(b), hdlfimath);
         beta10n = fi( beta9 - fi(0.00390623013197,numerictype(theta),...
                   hdlfimath),numerictype(theta), hdlfimath);
 end


%---------                                  # Eleventh pipeline stage

 if(beta10<fi(0,numerictype(theta), hdlfimath))
         a11n    = fi(a10 + bitsra(b10,9), numerictype(a), hdlfimath);
         b11n    = fi(b10 - bitsra(a10,9), numerictype(b), hdlfimath);
         beta11n = fi(beta10 + fi(0.00195312251648,numerictype(theta),...
                   hdlfimath),numerictype(theta), hdlfimath);
 else
         a11n    = fi(a10 - bitsra(b10,9), numerictype(a), hdlfimath);
         b11n    = fi(b10 + bitsra(a10,9), numerictype(b), hdlfimath);
         beta11n = fi(beta10 - fi(0.00195312251648,numerictype(theta),...
                   hdlfimath),numerictype(theta), hdlfimath);
 end


%---------                                  # Twelfth pipeline stage

 if(beta11<fi(0,numerictype(theta), hdlfimath))
         a12n    = fi( a11 + bitsra(b11,10), numerictype(a), hdlfimath);
         b12n    = fi(b11 - bitsra(a11,10), numerictype(b), hdlfimath);
         beta12n = fi( beta11 + fi(0.00097656218956,numerictype(theta),...
                   hdlfimath),numerictype(theta), hdlfimath);
 else
         a12n    = fi(a11 - bitsra(b11,10), numerictype(a), hdlfimath);
         b12n    = fi( b11 + bitsra(a11,10), numerictype(b), hdlfimath);
         beta12n = fi(beta11 - fi(0.00097656218956,numerictype(theta),...
                   hdlfimath),numerictype(theta), hdlfimath);
 end


%---------                                  # Thirteenth pipeline stage

 if(beta12<fi(0,numerictype(theta), hdlfimath))
         a13n    = fi( a12 + bitsra(b12,11), numerictype(a), hdlfimath);
         b13n    = fi(b12 - bitsra(a12,11), numerictype(b), hdlfimath);
```

```
         beta13n = fi( beta12 + fi(0.00048828121119 ,numerictype(theta),...
                   hdlfimath),numerictype(theta), hdlfimath);
 else
         a13n    = fi(a12 - bitsra(b12,11), numerictype(a), hdlfimath);
         b13n    = fi( b12 + bitsra(a12,11), numerictype(b), hdlfimath);
         beta13n = fi(beta12 - fi(0.00048828121119 ,numerictype(theta),...
                  hdlfimath),numerictype(theta), hdlfimath);
 end


%---------                                  # Fourteenth pipeline stage

 if(beta13<fi(0,numerictype(theta), hdlfimath))
        a14n    = fi( a13 + bitsra(b13,12), numerictype(a), hdlfimath);
        b14n    = fi(b13 - bitsra(a13,12), numerictype(b), hdlfimath);
        beta14n = fi( beta13 + fi( 0.00024414062015,numerictype(theta),...
                  hdlfimath),numerictype(theta), hdlfimath);
 else
        a14n    = fi(a13 - bitsra(b13,12), numerictype(a), hdlfimath);
        b14n    = fi( b13 + bitsra(a13,12), numerictype(b), hdlfimath);
        beta14n = fi(beta11 - fi( 0.00024414062015,numerictype(theta),...
                  hdlfimath),numerictype(theta), hdlfimath);
 end


%---------                                  # Fifteenth pipeline stage

 if(beta14<fi(0,numerictype(theta), hdlfimath))
        a15n    = fi( a14 + bitsra(b14,13), numerictype(a), hdlfimath);
        b15n    = fi(b14 - bitsra(a14,13), numerictype(b), hdlfimath);
        beta15n = fi( beta14 + fi(0.00012207031189,numerictype(theta),...
                  hdlfimath),numerictype(theta), hdlfimath);
 else
        a15n    = fi(a14 - bitsra(b14,13), numerictype(a), hdlfimath);
        b15n    = fi( b14 + bitsra(a14,13), numerictype(b), hdlfimath);
        beta15n = fi(beta14 - fi(0.00012207031189,numerictype(theta),...
                  hdlfimath),numerictype(theta), hdlfimath);
 end

%---------                                  # Sixteenth pipeline stage

 a16n = fi(a15 * sfi( 0.60725293538591,18,17), numerictype(a), hdlfimath);
 b16n = fi(b15 * sfi( 0.60725293538591,18,17), numerictype(b), hdlfimath);
```

```
%Output Update of Registers
i2n  = i1;
i3n  = i2;
i4n  = i3;
i5n  = i4;
i6n  = i5;
i7n  = i6;
i8n  = i7;
i9n  = i8;
i10n = i9;
i11n = i10;
i12n = i11;
i13n = i12;
i14n = i13;
i15n = i14;
i16n = i15;
% %
a1  = a1n;
a2  = a2n;
a3  = a3n;
a4  = a4n;
a5  = a5n;
a6  = a6n;
a7  = a7n;
a8  = a8n;
a9  = a9n;
a10 = a10n;
a11 = a11n;
a12 = a12n;
a13 = a13n;
a14 = a14n;
a15 = a15n;
a16 = a16n;
% %
b1 = b1n;
b2 = b2n;
b3 = b3n;
b4 = b4n;
b5 = b5n;
b6 = b6n;
b7 = b7n;
b8 = b8n;
b9 = b9n;
b10 = b10n;
```

```
 b11 = b11n;
 b12 = b12n;
 b13 = b13n;
 b14 = b14n;
 b15 = b15n;
 b16 = b16n;
 % %
 beta1 = beta1n;
 beta2 = beta2n;
 beta3 = beta3n;
 beta4 = beta4n;
 beta5 = beta5n;
 beta6 = beta6n;
 beta7 = beta7n;
 beta8 = beta8n;
 beta9 = beta9n;
 beta10 = beta10n;
 beta11 = beta11n;
 beta12 = beta12n;
 beta13 = beta13n;
 beta14 = beta14n;
 beta15 = beta15n;
 % %
 i1  = i1n;
 i2  = i2n;
 i3  = i3n;
 i4  = i4n;
 i5  = i5n;
 i6  = i6n;
 i7  = i7n;
 i8  = i8n;
 i9  = i9n;
 i10 = i10n;
 i11 = i11n;
 i12 = i12n;
 i13 = i13n;
 i14 = i14n;
 i15 = i15n;
 i16 = i16n;
 % %
%Generating output

 if(i16 == ufi(0,1,0))
    d = -a16;
```

```
    q = -b16;
else
    d = a16;
    q = b16;
end
```

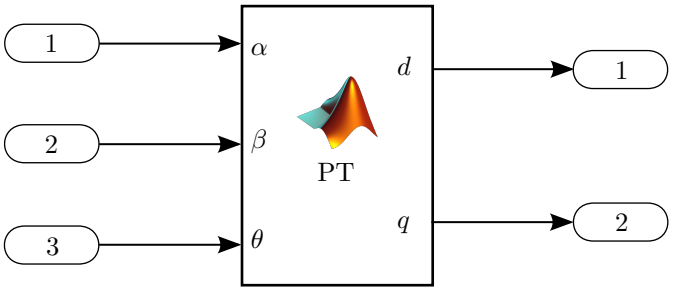

FIGURE A.1: Block diagram of Park transformation

Appendix B

Analog-Digital Converter

B.1 Analog Digital Converter

```
--- Author: Saeid Saeidi
--- EAL-TUM
---
---
--- ONLY ONE CHANNEL IS AQUAERED PER EOC PERIOD
--- The output data type is sfi(16,10)

LIBRARY ieee;
LIBRARY IEEE;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_signed.all;

entity ADC_8330_0CH is
port(CLK           : in std_logic;
     RST           : in std_logic;
     EOC           : in std_logic;
     SDO           : in std_logic;
     CONFG         : in std_logic_vector(15 DOWNTO 0);

     CONVST_N      : out std_logic;
     CS_N          : out std_logic;
     SCLK          : out std_logic;
     SDI           : out std_logic;
     DATA_A        : out std_logic_vector(15 DOWNTO 0) --sfi(16,10)
        );
```

```
end entity ADC_8330_0CH;

architecture DATA_GEN of ADC_8330_0CH is

SIGNAL DATA_ADS : STD_LOGIC_VECTOR(16 DOWNTO 0);
SIGNAL DATA_OUT : STD_LOGIC_VECTOR(16 DOWNTO 0);
SIGNAL DATA_ADS_OUT : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL DATA_ADS_32_OUT : STD_LOGIC_VECTOR(31 DOWNTO 0);
SIGNAL DATA_ADS_16_OUT : STD_LOGIC_VECTOR(15 DOWNTO 0);
SIGNAL RD_WR   : STD_LOGIC := '0';
SIGNAL COUNTER : std_logic_vector(8 DOWNTO 0) := "000000000";
SIGNAL CONFIG_REG  : std_logic_vector(15 DOWNTO 0);
SIGNAL CONFIG_COUNTER  : std_logic_vector(15 DOWNTO 0) := CONV_STD_LOGIC_VECTO
SIGNAL CYCLE_ST   : STD_LOGIC_VECTOR(1 DOWNTO 0) := "00";

begin

 ----****************************************

COUNT : process (CLK)

VARIABLE COUNTER_N : STD_LOGIC_VECTOR(8 DOWNTO 0);
VARIABLE CS_OUT : STD_LOGIC;

begin

if (CLK'event and CLK='1') then

if(EOC = '1' and CYCLE_ST = "00") then
CYCLE_ST <= "01";
CONFIG_COUNTER <= CONFIG_COUNTER + 1;
end if;

        if(CYCLE_ST = "01") then
                if(COUNTER < CONV_STD_LOGIC_VECTOR(20,9)) then
                        COUNTER_N := COUNTER + 1;
                else
                        CYCLE_ST <= "00";
                        COUNTER_N := CONV_STD_LOGIC_VECTOR(0,9);

                end if;

     end if;
```

```
-----------------------------------
        if(COUNTER_N = CONV_STD_LOGIC_VECTOR(0,9)) then
                CS_OUT := '1';
        elsif(COUNTER_N < CONV_STD_LOGIC_VECTOR(20,9)) then
             CS_OUT := '0';
        else
        CS_OUT := '1';
        end if;

-----------------------------------

CASE COUNTER_N IS

WHEN  CONV_STD_LOGIC_VECTOR(1,9) =>
SDI <= CONFIG_REG(15);

WHEN  CONV_STD_LOGIC_VECTOR(2,9) =>
SDI <= CONFIG_REG(14);

WHEN  CONV_STD_LOGIC_VECTOR(3,9) =>
SDI <= CONFIG_REG(13);

WHEN  CONV_STD_LOGIC_VECTOR(4,9) =>
SDI <= CONFIG_REG(12);

WHEN  CONV_STD_LOGIC_VECTOR(5,9) =>
SDI <= CONFIG_REG(11);

WHEN  CONV_STD_LOGIC_VECTOR(6,9) =>
SDI <= CONFIG_REG(10);

WHEN  CONV_STD_LOGIC_VECTOR(7,9) =>
SDI <= CONFIG_REG(9);

WHEN  CONV_STD_LOGIC_VECTOR(8,9) =>
SDI <= CONFIG_REG(8);

WHEN  CONV_STD_LOGIC_VECTOR(9,9) =>
SDI <= CONFIG_REG(7);

WHEN  CONV_STD_LOGIC_VECTOR(10,9) =>
SDI <= CONFIG_REG(6);

WHEN  CONV_STD_LOGIC_VECTOR(11,9) =>
```

```
SDI <= CONFIG_REG(5);

WHEN  CONV_STD_LOGIC_VECTOR(12,9) =>
SDI <= CONFIG_REG(4);

WHEN  CONV_STD_LOGIC_VECTOR(13,9) =>
SDI <= CONFIG_REG(3);

WHEN  CONV_STD_LOGIC_VECTOR(14,9) =>
SDI <= CONFIG_REG(2);

WHEN  CONV_STD_LOGIC_VECTOR(15,9) =>
SDI <= CONFIG_REG(1);

WHEN  CONV_STD_LOGIC_VECTOR(16,9) =>
SDI <= CONFIG_REG(0);

WHEN OTHERS =>
SDI <= '0';

END CASE;

COUNTER <= COUNTER_N;

-------------------------------
CASE COUNTER is

WHEN  CONV_STD_LOGIC_VECTOR(1,9) =>
DATA_ADS(15) <= SDO;

WHEN  CONV_STD_LOGIC_VECTOR(2,9) =>
DATA_ADS(14) <= SDO;

WHEN  CONV_STD_LOGIC_VECTOR(3,9) =>
DATA_ADS(13) <= SDO;

WHEN  CONV_STD_LOGIC_VECTOR(4,9) =>
DATA_ADS(12) <= SDO;

WHEN  CONV_STD_LOGIC_VECTOR(5,9) =>
DATA_ADS(11) <= SDO;

WHEN  CONV_STD_LOGIC_VECTOR(6,9) =>
DATA_ADS(10) <= SDO;
```

```
WHEN  CONV_STD_LOGIC_VECTOR(7,9) =>
DATA_ADS(9) <= SDO;

WHEN  CONV_STD_LOGIC_VECTOR(8,9) =>
DATA_ADS(8) <= SDO;

WHEN  CONV_STD_LOGIC_VECTOR(9,9) =>
DATA_ADS(7) <= SDO;

WHEN  CONV_STD_LOGIC_VECTOR(10,9) =>
DATA_ADS(6) <= SDO;

WHEN  CONV_STD_LOGIC_VECTOR(11,9) =>
DATA_ADS(5) <= SDO;

WHEN  CONV_STD_LOGIC_VECTOR(12,9) =>
DATA_ADS(4) <= SDO;

WHEN  CONV_STD_LOGIC_VECTOR(13,9) =>
DATA_ADS(3) <= SDO;

WHEN  CONV_STD_LOGIC_VECTOR(14,9) =>
DATA_ADS(2) <= SDO;

WHEN  CONV_STD_LOGIC_VECTOR(15,9) =>
DATA_ADS(1) <= SDO;

WHEN  CONV_STD_LOGIC_VECTOR(16,9) =>
DATA_ADS(0) <= SDO;

WHEN  CONV_STD_LOGIC_VECTOR(17,9) =>
DATA_ADS(16) <= SDO;

WHEN OTHERS =>

DATA_OUT <= DATA_ADS;

END CASE;
end if;
-------------------------------
SCLK <= CLK and (not CS_OUT) ;
CS_N <= CS_OUT;
end process;
```

```
----*****************************************
----*****************************************


config : process(CLK, RST) is

begin


if(RST = '1' or CONFIG_COUNTER = CONV_STD_LOGIC_VECTOR(0,16)) then
CONFIG_REG <= CONFG;
RD_WR <= '1';
elsif(CONFIG_COUNTER = CONV_STD_LOGIC_VECTOR(1,16)) then

CONFIG_REG <= "0000000000000000";
RD_WR <= '1';
else
CONFIG_REG <= "1101000000000000";
RD_WR <= '0';
end if;

end process;


----*****************************************
----*****************************************
----*****************************************

CH_DATA : process (CLK) is

variable TO_BITV : STD_LOGIC_VECTOR(31 DOWNTO 0);
variable MULT_DATA : STD_LOGIC_VECTOR(31 DOWNTO 0);

begin

if(CLK'EVENT and CLK = '1') THEN
      if(RD_WR = '0' )then
DATA_ADS_OUT <= DATA_OUT(15 DOWNTO 0) - CONV_STD_LOGIC_VECTOR(32768,16);
MULT_DATA := (DATA_ADS_OUT *CONV_STD_LOGIC_VECTOR(13107,16));
TO_BITV := (TO_BITVECTOR(MULT_DATA) sra 13);
DATA_ADS_32_OUT <= TO_STDLOGICVECTOR(TO_BITV);
```

```
        DATA_ADS_16_OUT <= DATA_ADS_32_OUT(15 DOWNTO 0);
        end if;


                        DATA_A <= DATA_ADS_16_OUT;


end if;

end process;

CONVST_N  <= '1';

end DATA_GEN;
```

Appendix C

FPGA Implementation of PI Controller

Fixed point model of the parameterizable PI controller is illustrated on figure C.1. The red lines show signal paths of the proportional part, which represents the critical computational delay.

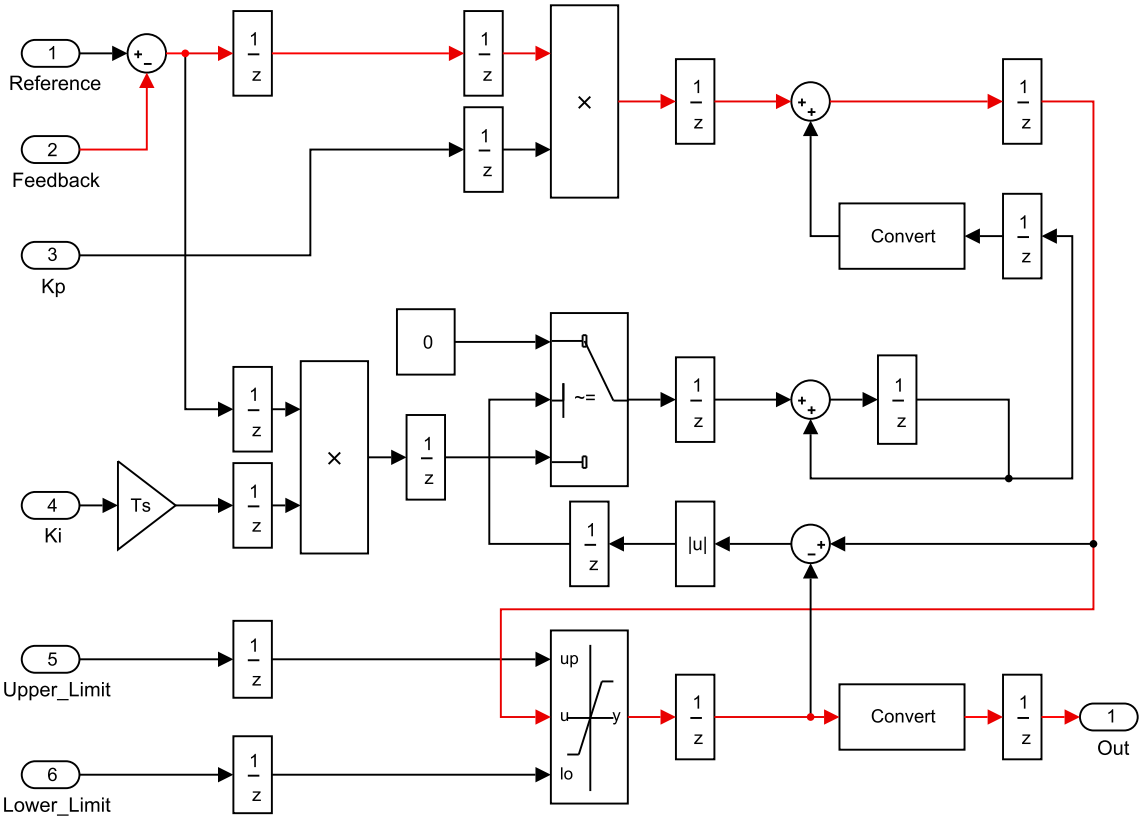

FIGURE C.1: Simulink model for PI controller

Appendix D

Flux Observer of Induction Motor

Figure D.1 shows Simulink model for the rotor and stator flux estimation and torque calculation. This model is optimized for VHDL code generation by the *HDL coder* tool. Clock frequency of this model is set to 10 MHz which is relatively low. It allows longer signal paths without necessity of so many pipeline registers.

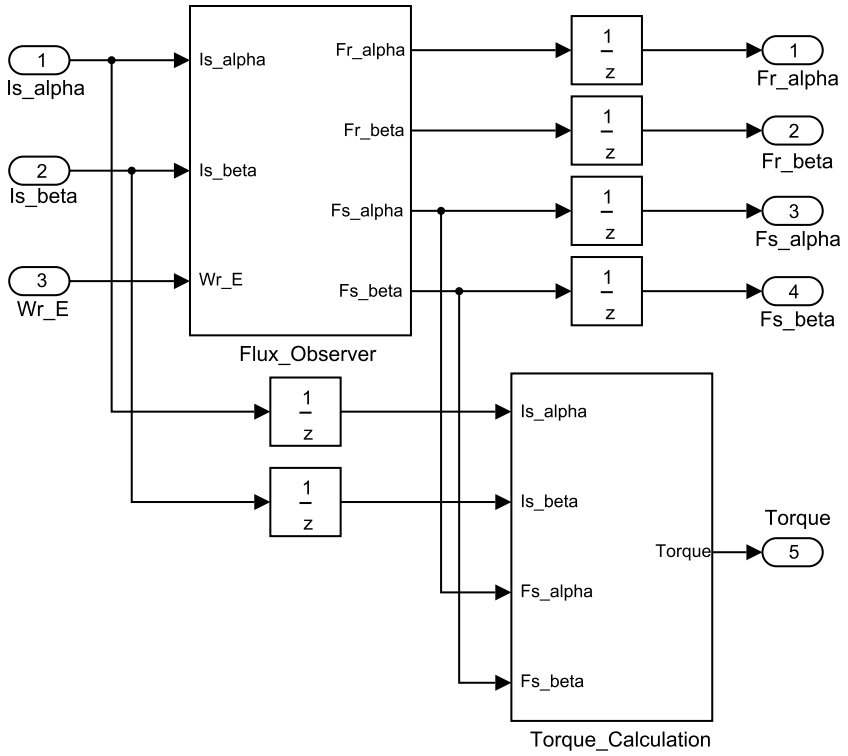

FIGURE D.1: Simulink model for flux estimation and torque calculation

Structure of the flux observer is shown on figure D.2. Two first order lag blocks are used to realize behavior of the rotor model.

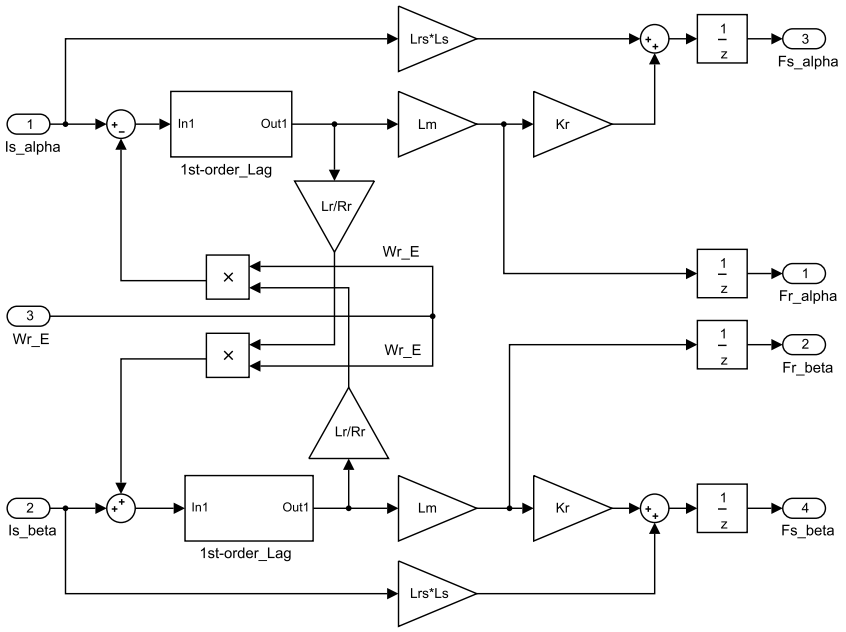

FIGURE D.2: Simulink model of the flux observer

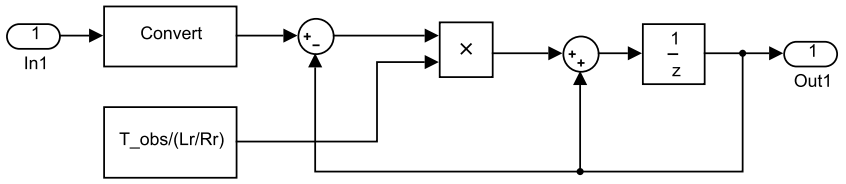

FIGURE D.3: First order lag system

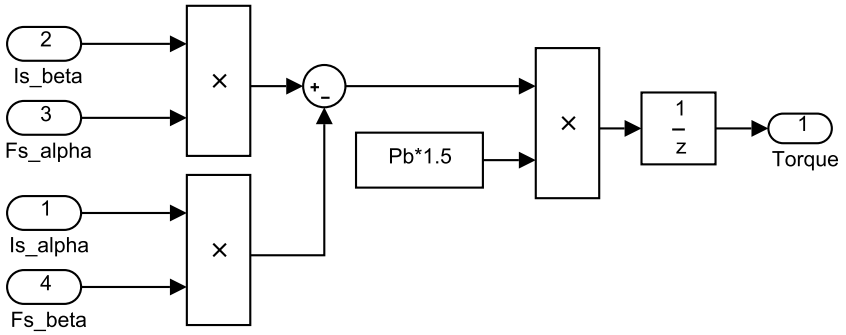

FIGURE D.4: Simulink model for torque calculation

Appendix E

Simulink Model of CS-NMPC for Induction Motor

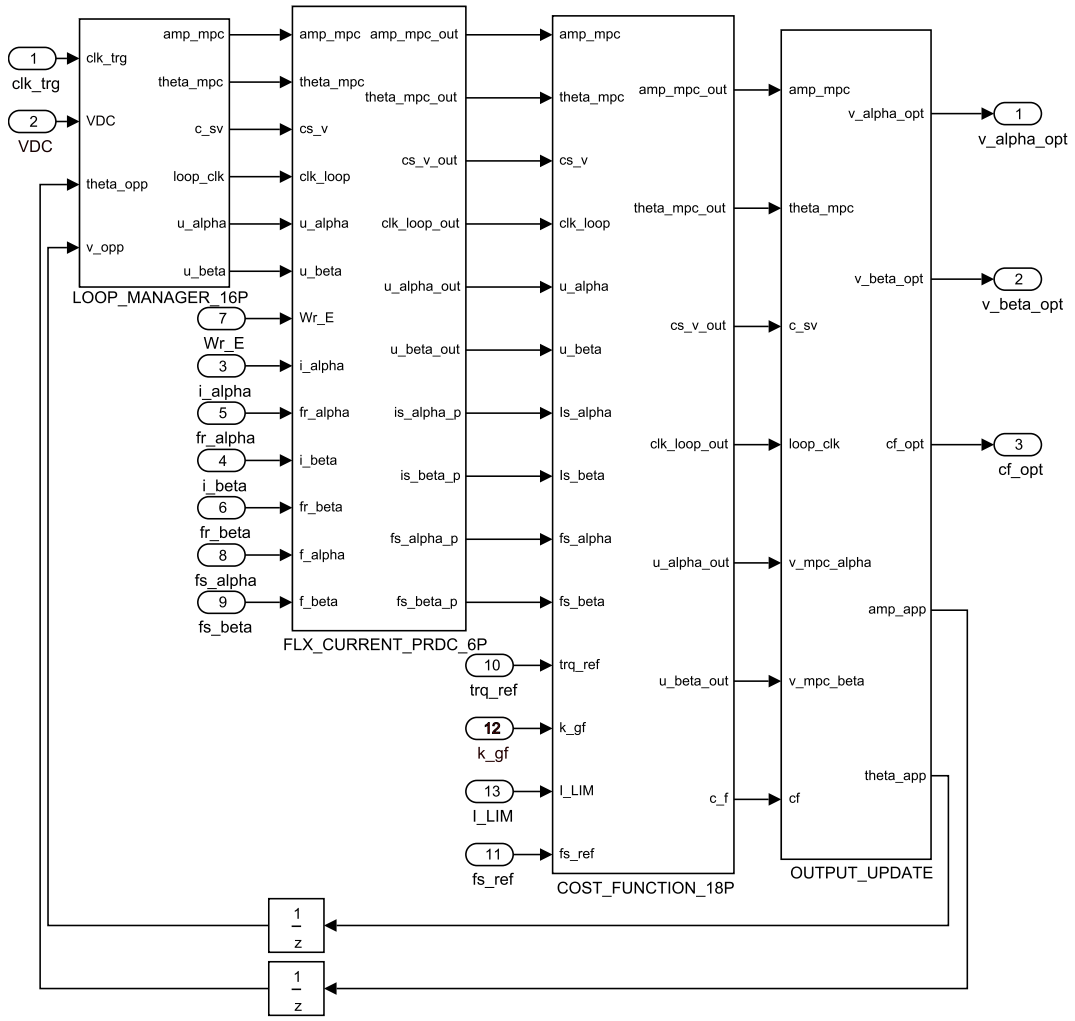

FIGURE E.1: A screenshot of the Simulink model of CS-NMPC for the induction motor

E.1 Loop Manager

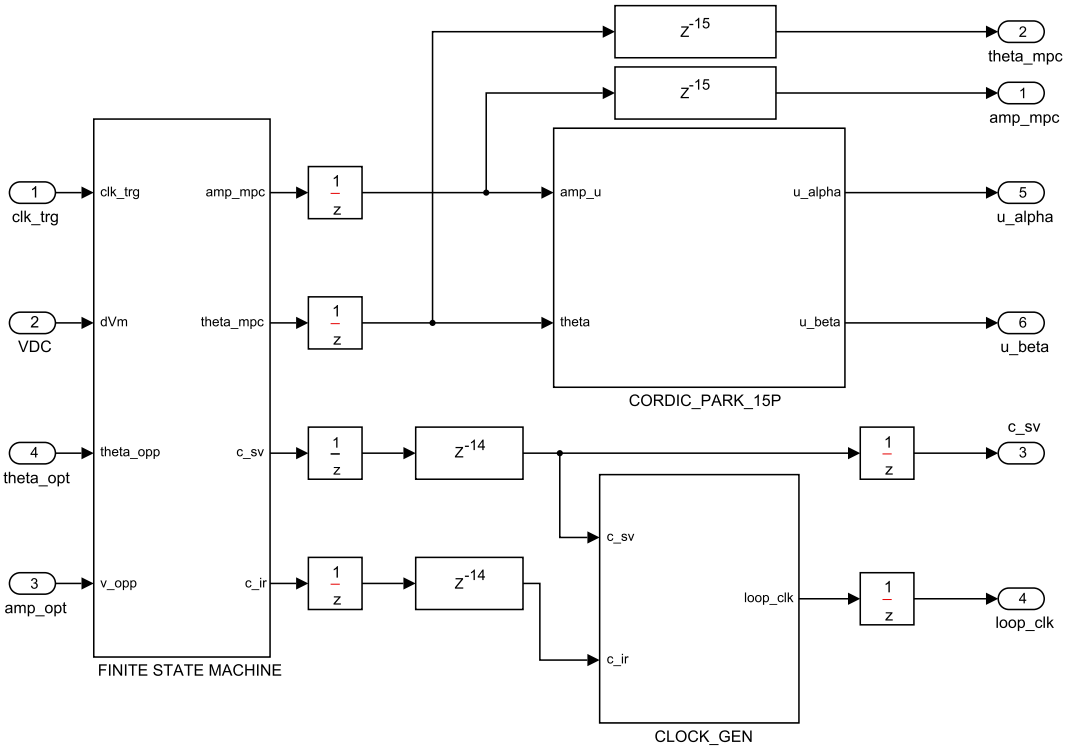

FIGURE E.2: Simulink model of the Loop Manager

E.2 Current and Flux Prediction

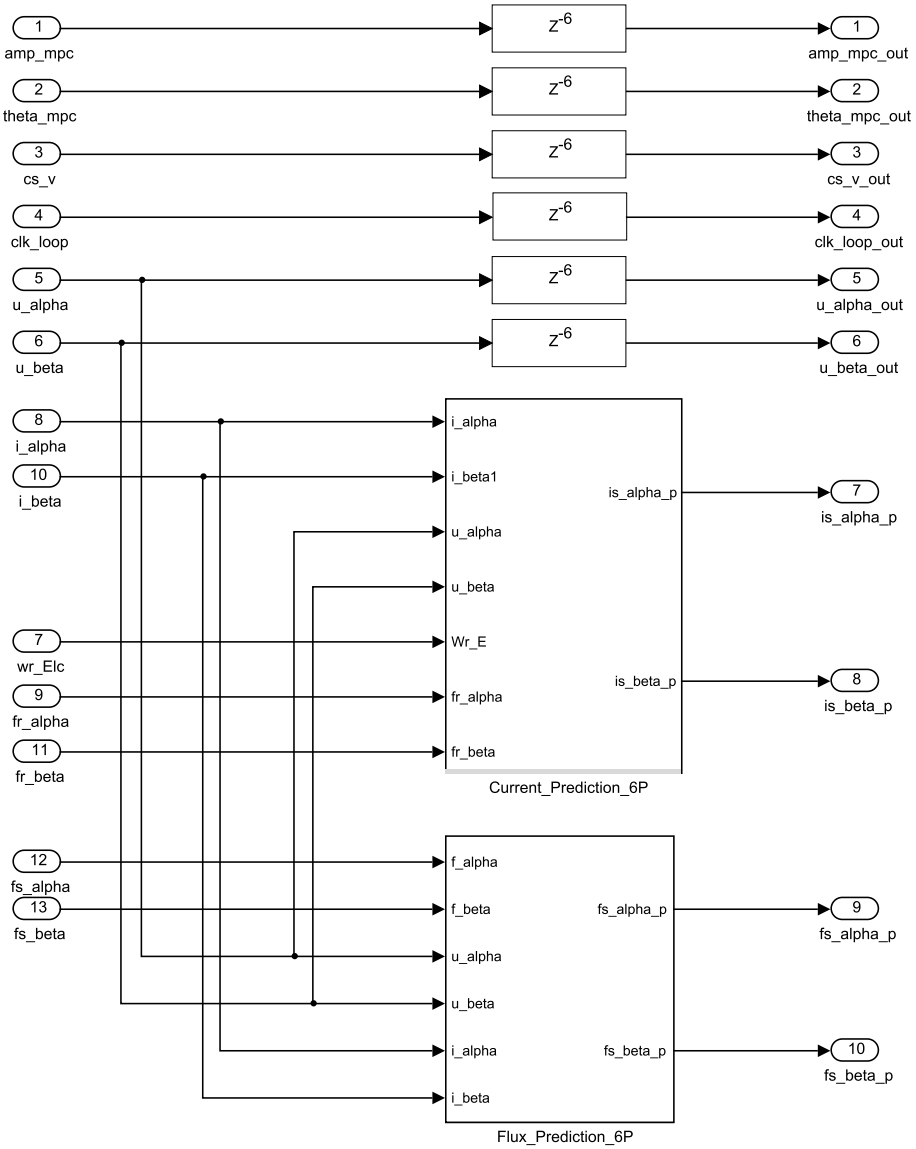

FIGURE E.3: A screenshot of the Simulink model for the flux and current prediction

E.2.1 Flux Prediction Model

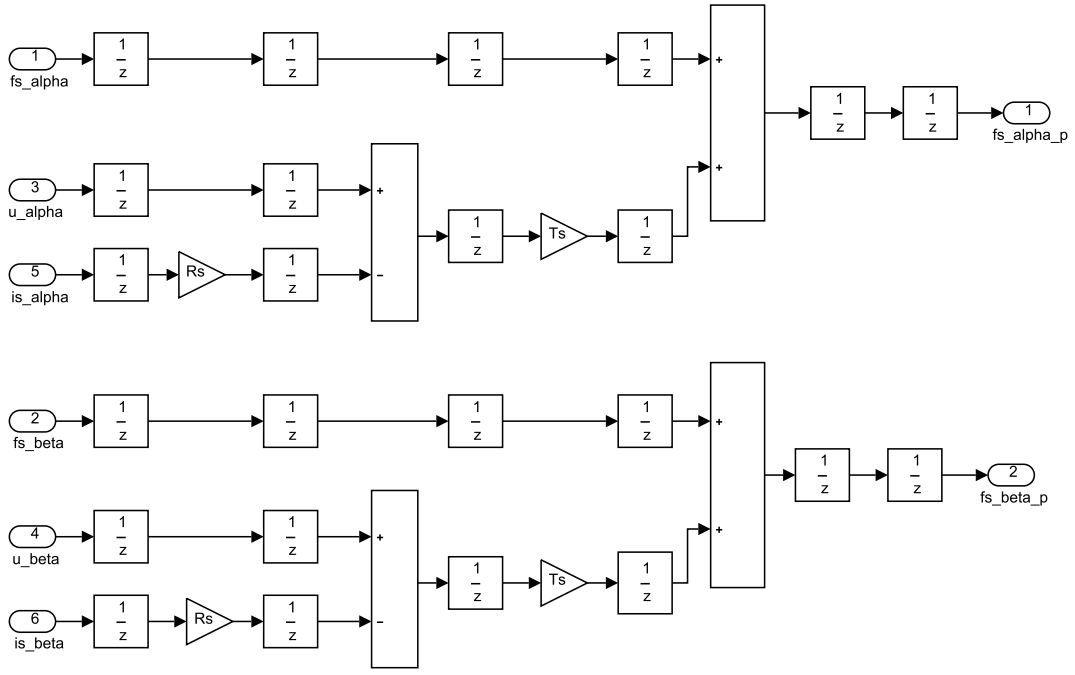

FIGURE E.4: A screenshot of the Simulink model for the flux prediction

E.3 Cost Function Model

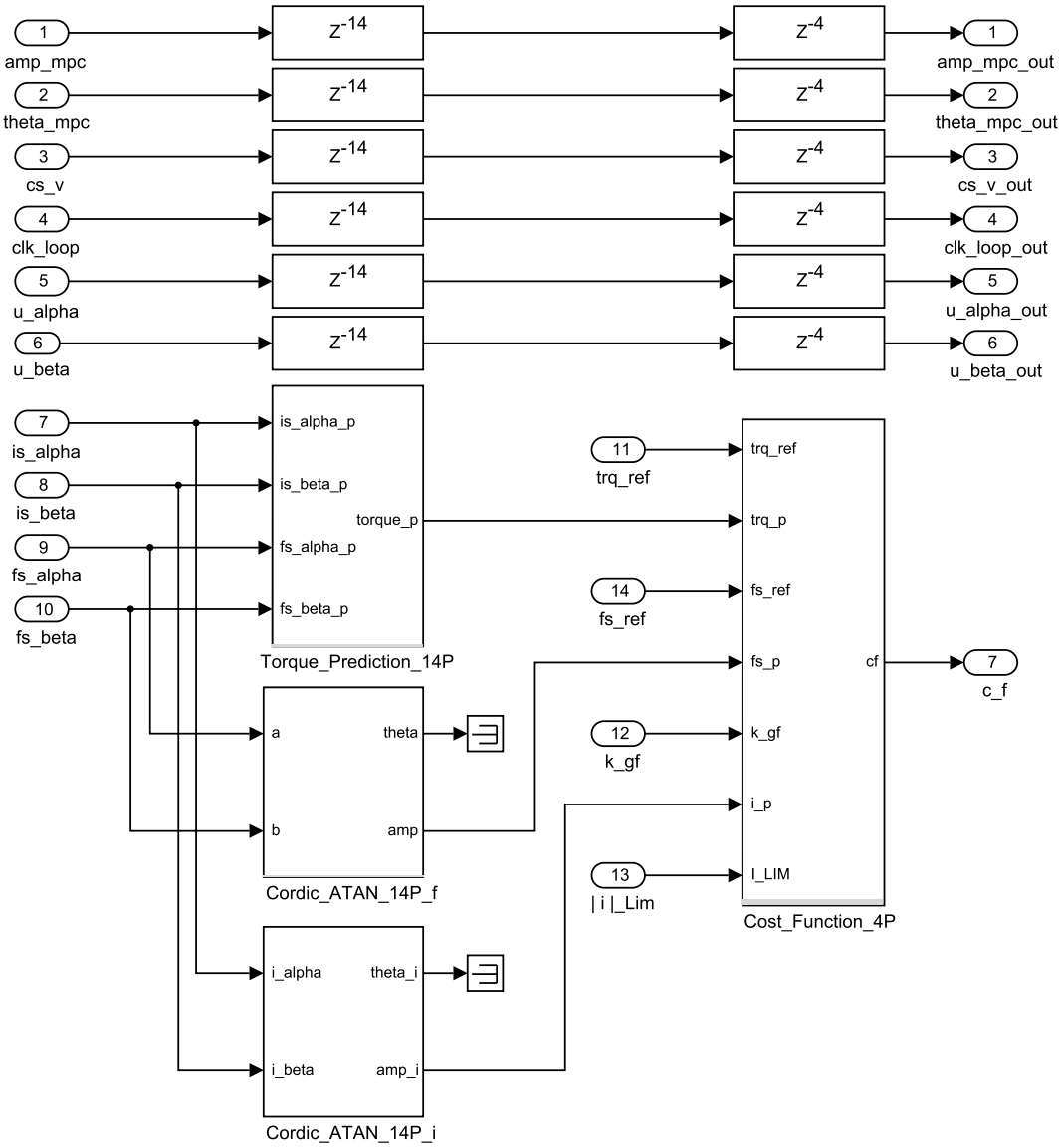

FIGURE E.5: The Simulink model for the cost function calculation

Appendix F

Parameters of the Electrical Motors

F.1 Permanent Magnet Synchronous Motor

Rated power P_n	2.7 (kW)
Rated torque T_n	8.5 (Nm)
Rated voltage V_n	400 (V)
Rated current I_n	5 (A)
Pole pairs p	3
Rated speed ω_n	3000 (rpm)
Stator resistance R_s	1.3 (ohm)
Stator inductance L_s	9 (mH)
PM Flux ψ_m	0.41 (Wb)

Table F.1: Parameter of the PMSM

F.2 Induction Motor

Symbol		Value
P_n	Rated power	2,2 (kW)
V_n	Rated voltage	230 (V)
I_n	Rated current	7,98 (A)
F_n	Rated frequency	50 (Hz)
p	Pole pairs	1
T_n	Rated torque	7,4 (Nm)
ω_n	Rated speed	2840 (rpm)
R_s	Stator resistance	2,7389 (ohm)
R_r	Rotor resistance	2,1385 (ohm)
L_s	Stator self inductance	328 (mH)
L_r	Rotor self inductance	328 (mH)
L_m	Mutual inductance	317 (mH)
Θ	Moment of inertia	0.007 kgm^2

TABLE F.2: Parameters of the induction motor

Bibliography

[1] J. Holtz. The representation of AC machine dynamics by complex signal flow graphs. *IEEE Trans. Industrial Electronics*, 42(3):263 – 271, June 1995.

[2] *DE2-115 User Manual*. Terasic, 2012.

[3] *IPM L1/S1-Series Application Note*. Mitsubishi, September 2008.

[4] W.C. Duesterhoeft, Max W. Schulz, and Edith Clarke. Determination of instantaneous currents and voltages by means of alpha, beta, and zero components. *American Institute of Electrical Engineers*, 70:1248 – 1255, July 1951.

[5] R.H. Park. Two-reaction theory of synchronous machines generalized method of analysis - part I. *American Institute of Electrical Engineers*, 48:716 – 727, 1929.

[6] José Rodríguez, Ralph M. Kennel, José R. Espinoza, Mauricio Trincado, César A. Silva, and Christian A. Rojas. High-performance control strategies for electrical drives: An experimental assessment. *IEEE Trans. Industrial. Electronics*, 59(2): 812 – 820, February 2012.

[7] J. Holtz and Qi Xin. Optimal control of medium-voltage drives — an overview. *IEEE Trans. Industrial. Electronics*, 60(12):5472 – 5481, December 2013.

[8] Robert D. Lorenz and Sheng-Ming Yang. Efficiency-optimized flux trajectories for closed-cycle operation of field-orientation induction machine drives. *IEEE Trans. Industrial. Applications*, 28(3):574 – 580, June 1992.

[9] Sikyung Kim, Prasad N. Enjeti, Paul Packebush, and Ira J. Pitel. A new approach to improve power factor and reduce harmonics in a three-phase diode rectifier type utility interface. *IEEE Trans. Industrial Applications*, 30(6):1557 – 1564, November/December 1994.

[10] R.D. Lorenz and Donald B Lawson. A simplified approach to continuous, online tuning of field oriented induction machine drives. In *Conference Record of the IEEE Industry Applications Society Annual Meeting*, volume 1, pages 444 – 449, 1988.

[11] Robert D. Lorenz and Donald B. Lawson. Flux and torque decoupling control for field-weakened operation of field-oriented induction machines. *IEEE Trans. Industrial Applications*, 26(2):290 – 295, March/April 1990.

[12] Isao Takahashi and Toshihiko Noguchi. A new quick-response and high-efficiency control strategy of an induction motor. *IEEE Trans. Industrial. Applications*, IA-22(5):820–827, Sebtember/October 1986.

[13] Dierk Schröder. *Elektrische Antriebe Regelung von Antriebssystemen.* Springer, Springer-Verlag Berlin Heidelberg, 2009.

[14] Timothy M. Rowan and Russel J. Kerkman. A new synchronous current regulator and an analysis of current-regulated PWM inverters. *IEEE Trans. Industrial Applications*, IA-22(4):678 – 690, July/August 1986.

[15] R.D. Lorenz and Donald B. Lawson. Performance of feedforward current regulators for field-oriented induction machine controllers. *IEEE Trans. Industrial Applications*, IA-23(4):597 – 602, July/August 1987.

[16] Julio C. Moreira, Kam Tim Hung, Thomas A. Lipo, and Robert D. Lorenz. A simple and robust adaptive controller for detuning correction in field-oriented induction machines. *IEEE Trans. Industrial Applications*, 28(6):1359 – 1392, November/December 1992.

[17] J. Holtz. Pulsewidth modulation for electronic power conversion. *Proceedings IEEE*, 80(8), August 1994.

[18] Giuseppe S. Buja and Marian P. Kazmierkowski. Direct torque control of PWM inverter-fed AC motors — a survey. *IEEE Trans. Industrial Electronics*, 51(4):744 – 757, August 2004.

[19] Jose Rodriguez, Jorge Pontt, Cesar Silva, Samir Kouro, and Heman Miranda. A novel direct torque control scheme for induction machines with space vector modulation. In *35th Annul IEEE Power Electronics Specialists Conference*, volume 2, pages 1392 – 1397, Achen, 2004.

[20] J. Monteiro, J. Fernando Silva, S. F. Pinto, and J. Palma. Matrix converter-based unified power-flow controllers: Advanced direct power control method. *IEEE Trans. Power Delivery*, 26(1):420 – 430, January 2011.

[21] Romeo Ortega, Nikita Barabanov, and Gerardo Escobar Valderrama. Direct torque control of induction motors: Stability analysis and performance improvement. *IEEE Trans. Automatic Control*, 46(8):1209 – 1222, August 2001.

[22] Thomas Burtscher and Tobias Geyer. Deadlock avoidance in model predictive direct torque control. *IEEE Trans. Industrial Applications*, 49(5):2126 – 2135, September/October 2013.

[23] Yongchang Zhang, Jianguo Zhu, Zhengming Zhao, Wei Xu, and David G. Dorrell. An improved direct torque control for three-level inverter-fed induction motor sensorless drive. *IEEE Trans. Power Electronics*, 27(3):1502 – 1512, March 2012.

[24] J. Holtz. The induction motor — a dynamic system. In *20th International Conference on Industrial Electronics, Control and Instrumentation,IECON 94*, volume 1, pages 1 – 6, 1994.

[25] J. Holtz. Sensorless control of induction motor drives. *Proceedings of the IEEE*, 36:1359 – 1394, 2002.

[26] J. Holtz. Sensorless position control of induction motors — an emerging technology. *IEEE Trans. Industrial Electronics*, 8(6):840 – 851, December 1998.

[27] Andrew Smith1, Shady Gadoue1, Matthew Armstrong, and John Finch. Improved method for the scalar control of induction motor drives. *IET Electric Power Applications*, 7:487–498, April 2013.

[28] Philip T. Krein, Francisco Disilvestro, Ioannis Kanellakopoulos, and Jonathan Locker. Comparative analysis of scalar and vector control methods for induction motors. In *24th Annual IEEE Power Electronics Specialists Conference*, pages 1139 – 1145. IEEE Conference Publications, 1993.

[29] Iordanis Kioskeridis and Nikos Margaris. Loss minimization in scalar-controlled induction motor drives with search controllers. *IEEE Trans. Power Electronics*, 11(2):213 – 220, March 1996.

[30] M. Cacciato, A. Consoli, G. Scarcella, G. Scelba, and A. Testa. Efficiency optimization techniques via constant optimal slip control of induction motor drives. In *International Symposium on Power Electronics, Electrical Drives, Automation and Motion*, pages 33 – 38. IEEE Conference Publications, 2006.

[31] Robert D. Lorenz and donald B. Lawson. Performance of feedforward current regulators for field-oriented induction machine controllers. *IEEE Trans. Industrial Applications*, IA-23(4):597 – 602, July/August 1987.

[32] Gabriel Rupprecht, Leonhard Werner, and Craig J. Nordby. Field-oriented control of a standard AC motor using microprocessors. *IEEE Trans. Industrial Applications*, IA-46(2):186 – 192, March/April 1980.

[33] W. Leonhard. Field-orientation for controlling AC machines — principle and application. In *Third International Conference on Power Electronics and Variable-Speed Drives*, pages 277 – 282, 1988.

[34] Ichiro Miki, Osamu Nakao, and Sakae Nishiyama. A new simplified current control method for field-oriented induction motor drives. *IEEE Trans. Industrial Applications*, 27(6):1081 – 1086, November/December 1991.

[35] Fernando Briz del Blanco, Michael W. Degner, and Robert D. Lorenz. Dynamic analysis of current regulators for AC motors using complex vectors. *IEEE Trans. Industrial Applications*, 35(6):1424 – 1432, November/December 1999.

[36] R.D. Lorenz. The emerging role of dead-beat, direct torque and flux control in the future of induction machine drives. In *11th International Conference on Optimization of Electrical and Electronic Equipment, OPTIM*, volume 1, pages XIX – XXVII, May 2008.

[37] Joachim Holtz and Bernd Beyer. Fast current trajectory tracking control based on synchronous optimal pulsewidth modulation. *IEEE Trans. Industrial Applications*, 31(5):1110 – 1120, September/October 1995.

[38] C. Bastin, F. Jaritchi, and I.M.Y. Mareelst. Dead-beat control of recursive nonlinear systems. In *Proceedings of the 32nd IEEE Conference on Decision and Control*, volume 4, pages 2965 – 2971, 1993.

[39] Barbara H. Kenny and Robert D. Lorenz. Stator- and rotor-flux-based deadbeat direct torque control of induction machines. *IEEE Trans. Industrial Applications*, 39(4):1093 – 1101, July/August 2003.

[40] Takao Kawabata, Takeshi Miyashita, and Yushin Yamamoto. Dead beat control of three phase PWM inverter. *IEEE Trans. Power Electronics*, 5(1):21 – 28, January 1990.

[41] A. Linder and R. Kennel. Model predictive control for electrical drives. In *IEEE 36th Power Electronics Specialists Conference*, pages 1793 – 1799, Recife, June 2005.

[42] Patrick L. Jansen and Robert D. Lorenz. A physically insightful approach to the design and accuracy assessment of flux observers for field oriented induction machine drives. *IEEE Trans. Industrial Applications*, 30(1):101 – 110, January/February 1994.

[43] Joachim Holtz. Sensorless control of induction machines — with or without signal injection. *IEEE Trans. Industrial Electronics*, 53(1), February 2006.

[44] Hidehiko Sugimoto and Shinzo Tamai. Secondary resistance identification of an induction motor applied model reference adaptive system and its characteristics. *IEEE Trans. Industrial Applications*, IA-23(2):296–303, September/October 1987.

[45] Colin Schauder. Adaptive speed identification for vector control of induction motors without rotational transducers. *IEEE Trans. Industrial Applications*, 28(5): 1054–1061, September/October 2006.

[46] Mihai Comanescu and Longya Xu. Sliding-mode MRAS speed estimators for sensorless vector control of induction machine. *IEEE Trans. Industrial Electronics*, 53(1):146–153, February 2006.

[47] T Du and M A Brdys. Shaft speed, load torque and rotor flux estimation of induction motor drive using an extended luenberger observer. In *Sixth International Conference on Electrical Machines and Drives*, pages 179 – 184, 1993.

[48] Markus A. Vogelsberger, Stefan Grubic, Thomas G. Habetler, and Thomas M. Wolbank. Using PWM-induced transient excitation and advanced signal processing for zero-speed sensorless control of AC machines. *IEEE Trans. Industrial Electronics*, 57(1):365–374, January 2010.

[49] Reiko Raute, Cedric Caruana, Cyril Spiteri Staines, Joseph Cilia, Mark Sumner, and Greg M. Asher. Sensorless control of induction machines at low and zero speed by using PWM harmonics for rotor-bar slotting detection. *IEEE Trans. Industrial Applications*, 46(5):1989 – 1998, January 2010.

[50] Oscar Cabral Ferreira and Ralph Kennel. Encoderless control of industrial servo drives. In *12th International of Power Electronics and Motion Control Conference, EPE-PEMC*, pages 1962 – 1967. IEEE Conference Publications, 2006.

[51] R. D. Lorenz. Self-sensing as an integration focus for motor drives and power devices. In *Proceeding of International Conference on Electrical Machines and Systems*, pages 386 – 391, Seoul, Korea, October 2007. IEEE Conference Publications.

[52] Fernando Briz, Michael W. Degner, Alberto Diez, and Robert D. Lorenz. Measuring, modeling, and decoupling of saturation-induced saliencies in carrier-signal injection-based sensorless AC drives. *IEEE Trans. Industrial Applications*, 37(5): 1356–1364, September/October 2001.

[53] *TMS320C28x CPU and Instruction Set Reference Guide.* Texas Instrument, August 2009.

[54] Thanh Tran, G. Frantz, and Cheng Peng. System-on-chip choices. In *IEEE International SOC Conference*, pages 259 – 260, 2003.

[55] T.S. Rajesh Kumar, R. Govindarajan, and C. P. Ravi Kumar. Optimal code and data layout in embedded systems. In *16th International Conference on VLSI Design*, pages 573 –578, 2003.

[56] Kiyoo Itoh. Embedded memories: Progress and a look into the future. *Design and Test of Computers*, 28(1):10 – 13, 2011.

[57] R. Duma, P. Dobra, M. Abrudean, and M. Dobra. Rapid prototyping of control systems using embedded target for TI C2000 DSP. In *Mediterranean Conference on Control and Automation*, pages 1–5, July 2007.

[58] Concettina Buccella, Carlo Cecati, and Hamed Latafat. Digital control of power converters — a survey. *IEEE Trans. Industrial Informatics*, 8(3):437 – 446, August 2012.

[59] *TI SYS/BIOS v6.35 Real-time Operating System User's Guide*. Texas Instrument, June 2013.

[60] Nenad Cetic, Miroslav Popovic, Miodrag Djukic, and Momcilo Krunic. A run-time library for parallel processing on a multi-core DSP. In *3rd Eastern European Regional Conference on the Engineering of Computer Based Systems*, pages 41 – 47, 2013.

[61] W. Miller. Real world applications for field programmable gate array devices — an overview. In *Proceeding of WESCON*, pages 259 – 260, 94.

[62] Jonathan Rose, Abbas El Gamal, and Alberto Sangiovanni-Vincentelli. Architecture of field-programmable gate arrays. *Proceedings of the IEEE*, pages 1013 – 1029, 1993.

[63] Stephen Trimberger. A reprogrammable gate array and applications. *Proceedings of the IEEE*, pages 1030 – 1041, 1990.

[64] Umer Farooq, Zied Marrakchi, and Habib Mehrez. *Tree-based Heterogeneous FPGA Architectures*. Springer, 2012.

[65] S. Dai and E. Bozorgzadeh. CAD tool for FPGAs with embedded hard cores for design space exploration of future architectures. In *14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 329 – 330, 2006.

[66] *Cyclone IV Device Handbook*. Altera, December 2013.

[67] Ian Kuon, Russell Tessier, and Jonathan Rose. FPGA architecture: Survey and challenges. *Foundations and Trends in Electronic Design Automation*, 2(2):135–253, 2008.

[68] Moe Shahdad, Roger Lipsett, Erich Marschner, Kellye Sheehan, and Howard Cohen. VHSIC hardware description language. *Computer*, 18:94 – 103, 1985.

[69] A. Dewey. The VHSIC hardware description language (VHDL) program. In *21st Conference on Design Automation*, pages 556 – 557, 1984.

[70] Gary Gannot and Michiel Ligthart. Verilog HDL based FPGA design. In *International Verilog HDL Conference*, pages 86 – 92, March 1994.

[71] Victor Berman. Standard verilog-VHDL interoperability. In *International Verilog HDL Conference*, pages 2 – 9, March 1994.

[72] Tejas M. Bhatt and Dennis McCain. Matlab as a development environment for FPGA design. In *Proceedings of 42nd Design Automation Conference*, pages 607 – 610, March 2005.

[73] *HDL Coder: Getting Started Guide*. MathWorks, March 2012.

[74] Eric Monmasson and Marcian N. Cirstea. FPGA design methodology for industrial control systems — a review. *IEEE Trans. Industrial Elecronics*, 54(4):1824 – 1842, August 2007.

[75] Shahram Karimi, Philippe Poure, and Shahrokh Saadate. An HIL-based reconfigurable platform for design, implementation, and verification of electrical system digital controllers. *IEEE Trans. Industrial Elecronics*, 57(4):1226 – 1236, April 2010.

[76] Mahmoud Matar and Reza Iravani. The reconfigurable-hardware real-time and faster-than-real-time simulator for the analysis of electromagnetic transients in power systems. *IEEE Trans. Power Delivery*, 28(2):619 – 627, April 2013.

[77] Óscar Lucía, Isidro Urriza, Luis. A. Barragán, Denis Navarro, Óscar Jiménez, and José M. Burdío. Real-time FPGA-based hardware-in-the-loop simulation test bench applied to multiple-output power converters. *IEEE Trans. Industrial Applications*, 47(2):619 – 627, March/April 2011.

[78] S. Trimberger. A reprogrammable gate array and applications. *Proceedings of the IEEE*, 72(12):1030 – 1041, July 1993.

[79] Hau-Jean Hsu Ying-Yu Tzou. FPGA realization of space-vector PWM control IC for three-phase PWM inverters. *IEEE Trans. Power Electronics*, (6):953 – 963, November .

[80] *Embedded Peripherals IP User Guide.* Altera, June 2011.

[81] *Quartus II Handbook Version 13.1.* Altera, November 2013.

[82] *DSP Builder Handbook, Volume 1: Introduction to DSP Builder.* Altera, November 2013.

[83] Eric Monmasson, Lahoucine Idkhajine, Marcian N. Cirstea, Imene Bahri, Alin Tisan, and Mohamed Wissem Naouar. FPGAs in industrial control applications. *IEEE Trans. Industrial Informatics*, 7(2):224 – 243, May 2011.

[84] Mohamed-Wissem Naouar, Eric Monmasson, Ahmad Ammar Naassani, Ilhem Slama-Belkhodja, and Nicolas Patin. FPGA-based current controllers for AC machine drives — a review. *IEEE Trans. Industrial Electronics*, 54(4):1907 – 1925, August 2007.

[85] P. Patel and M. Moallem. Reconfigurable system for real-time embedded control applications. *IET Control Theory and Applications*, 4(11):12, September 2010.

[86] Patrick Schaumont. A senior-level course in hardware–software codesign. *IEEE Trans. Education*, 51(3):306 – 311, August 2008.

[87] Welson Sun, Michael J. Wirthlin, and Stephen Neuendorffer. FPGA pipeline synthesis design exploration using module selection and resource sharing. *IET Control Theory and Applications*, 26(2):12, February 2007.

[88] Bruno dos Santos, Rui Esteves Araujo, Diogo Varajao, and Claudio Pinto. Rapid prototyping framework for real-time control of power electronic converters using Simulink. In *39th Annual Conference of the IEEE Industrial Electronics Society*, pages 2303 – 2308, 2013.

[89] Robert Grepl. Real-time control prototyping in MATLAB/Simulink: review of tools for research and education in mechatronics. In *Proceedings of the 2011 IEEE Intemational Conference on Mechatronics*, pages 881 – 886, 2011.

[90] Yam P. Siwakoti and Graham E. Town. Design of FPGA-controlled power electronics and drives using MATLAB Simulink. In *5th IEEE Annual International Energy Conversion Congress and Exhibition*, pages 571 – 577, Melbourne - Australia, June 2013.

[91] Syed Manzoor Qasim, Shuja Ahmad Abbasi, and Bandar Almashary. A review of FPGA-based design methodology and optimization techniques for efficient hardware realization of computation intensive algorithms. In *Multimedia, Signal Processing and Communication Technologies, IMPACT 09. International*, pages 313 – 316, Aligarh, 2009.

[92] E. Monmasson, L. Idkhajine, I. Bahri, M-W-Naouar, and L. Charaabi. Design methodology and FPGA-based controllers for power electronics and drive applications. In *IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pages 2328 – 2338, 2010.

[93] I. Bahri, E. Monmasson, F. Verdiers, and M-A.Ben Khelifa. Design and validation methodology of FPGA-based motor drive for high-temperature environment. In *Electrical Systems for Aircraft, Railway and Ship Propulsion*, pages 1 – 6, 2010.

[94] Gustavo G. Parma and Venkata Dinavahi. Real-time digital hardware simulation of power electronics and drives. *IEEE Trans. Power Delivery*, 22(2):1235 – 1246, April 2007.

[95] Barry Fagin and Cyril Renard. Field programmable gate arrays and floating point arithmetic. *IEEE Trans. VLSI Systems*, 2(3):365 – 367, September 1994.

[96] Chun Hok Ho, ChiWai Yu, Philip Leong, Wayne Luk, and Steven J. E. Wilton. Floating-point FPGA: Architecture and modeling. *IEEE Trans. Very Larg Scale Integration Systems*, 17(12):1709 – 1718, December 2009.

[97] ChiWai Yu, Alastair M. Smith, Wayne Luk, Philip H. W. Leong, and Steven J. E. Wilton. Optimizing floating point units in hybrid FPGAs. *IEEE Trans. VLSI Systems*, 20(7):1295 – 1303, July 2012.

[98] *7 Series DSP48E1 Slice User's Guide.* Xilinx, August 2013.

[99] P. Karlstroem, A. Ehliar, and D. Liu. High-performance, low-latency fieldprogrammable gate array-based floating-point adder and multiplier units in a virtex 4. *IET Computers and Digital Techniques*, 2(4):305 – 313, January 2008.

[100] Z. Jovanovic and V. Milutinovic. FPGA accelerator for floating-point matrix multiplication. *IET Computers and Digital Techniques*, 6(4):249 – 256, May 2012.

[101] *Fixed-Point Designer User's Guide.* Mathworks, November 2014.

[102] R. Kennel, T. Boller, and J. Holtz. Replacement of electrical (load) drives by a hardware-in-the-loop system. *Proceedings of the IEEE*, pages 17 – 25, July 2011.

[103] Wai Chung Lee and David Drury. Development of a hardware-in-the-loop simulation system for testing cell balancing circuits. *IEEE Trans. Power Electronics*, 28 (12):5949 – 5959, December 2013.

[104] Michael Steure, Chris S. Edrington, Michael Sloderbeck, Wei Ren, and James Langston. A megawatt-scale power hardware-in-the-loop simulation setup for motor drives. *IEEE Trans. Industrial Electronics*, 57(4):1254 – 1260, April 2010.

[105] Bin Lu, Xin Wu, Hernan Figueroa, and Antonello Monti. A low-cost real-time hardware-in-the-loop testing approach of power electronics controls. *IEEE Trans. Industrial Electronics*, 54(2):919 – 931, April 2007.

[106] *Low-Power, 16-BIT, 1-MHz, Single/Dual Unipolar Input, Analog-to-Digital Coverters with Serial Interface.* Texas Instruments, December 2006.

[107] P. Landsmann, J. Jung, M. Kramkowski, P. Stolze, D. Paulus, and R. Kennel. Lowering injection amplitude in sensorless control by means of current oversampling. In *IEEE Symposium on Sensorless Control for Electrical Drives (SLED)*, pages 1 – 6, 2012.

[108] Ralph Kennel. Encoders for simultaneous sensing of position and speed in electrical drives with digital control. *IEEE Trans. Industrial Applications*, 43(6):1572 – 1577, November/December 2007.

[109] *3.3-V RS-485 Transceivers.* Texas Instruments, February 2002.

[110] *Hall Effect Current Sensor.* Chen Yang Technologies GmbH, September 2012.

[111] *Precision Lowes-Cost Isolation Amplifier.* Texas Instruments, September 1997.

[112] Carlos E. Garcia, David M. Prett, and Manfred Morari. Model predictive control: Theory and practice - a survey. *Automatica*, 25(3):14, February 1989.

[113] Arne Linder, Rahul Kanchan, Ralph Kennel, and Peter Stolze. *Model Based Predictive Control of Electric Drives.* Cuvillier Verlag Goettingen, Nonnenstieg 8, 37075, 2010.

[114] Alberto Bemporad and Manfred Morari. Robust model predictive control: A survey. *Lecture Notes in Control and Information Sciences*, 245:207 – 226, 1999.

[115] Manfred Morari and Jay H. Lee. Model predictive control: past, present and future. *Elsevier*, (5):667 —- 682, May 1999.

[116] Simone Loureiro, Oliveira Kothare, and Manfred Morari. Contractive model predictive control for constrained nonlinear systems. *IEEE Trans. Industrial Electronic*, 45(6):19, June 2000.

[117] J. Holtz and S. Stadtfeld. A predictive controller for stator current vector of AC-machines fed from a switched voltage source. In *International Semiconductor Power Converter Confrence IPEC*, volume 2, pages 1665 – 1675, Tokio, 1983.

[118] J. Holtz and U. Schwellenberg. A new fast response current control scheme for line controlled converters. In *Power Converter Confrence*, pages 175 – 183, Orlando, 1982.

[119] R. Kennel. *Prädiktives Führungsverfahren für Stromrichter*. PhD thesis, Kaiserslautern University, 1984.

[120] R. Kennel and A. Linder. Predictive control of inverter supplied electrical drives. In *IEEE 31st Annual Power Electronics Specialists Conference*, pages 761 – 766, 2000.

[121] R. Kennel, A. Linder, and M. Linke. Generalized predictive control (GPC) — ready for use in drive applications? In *IEEE 32nd Annual Power Electronics Specialists Conference*, volume 4, pages 839 – 1844, Vancouver, BC, 2001.

[122] Al-Sheakh Ameen N., Galal B.S., Kennel R.M., and Kanchan R.S. The explicit solution of model-based predictive control by considering the nonlinearities in drive applications. In *15th International Power Electronics and Motion Control Conference (EPE/PEMC)*, Navi Sad, September 2012.

[123] Georgios Papafotiou, Jonas Kley, Kostas G. Papadopoulos, Patrick Bohren, and Manfred Morari. Model predictive direct torque control — part II: Implementation and experimental evaluation. *IEEE Trans. Industrial Electronics*, 56(6):12, June 2009.

[124] R. Kennel and D. Schroeder. Predictive control strategy for converters. In *Proc. of the third IFAC Symposium*, volume 2, pages 415 – 422, Lausanne, 1983.

[125] Jose Rodriguez, Jorge Pontt, César A. Silva, Pablo Correa, Pablo Lezana, Patricio Cortés, and Ulrich Ammann. Predictive current control of a voltage source inverter. *IEEE Trans. Industrial Electronics*, 54(1), February 2007.

[126] Tobias Geyer, Georgios Papafotiou, and Manfred Morari. Model predictive direct torque control — part I: Concept, algorithm, and analysis. *IEEE Trans. Industrial Electronics*, 56(6):12, June 2009.

[127] James Scoltock, Tobias Geyer, and Udaya K. Madawala. A comparison of model predictive control schemes for MV induction motor drives. *IEEE Trans. Industrial Informatics*, 9(2):11, May 2013.

[128] P. Stolze, P. Landsmann, R. Kennel, and T. Mouton. Finite-set model predictive control of a flying capacitor converter with heuristic voltage vector preselection. In *8th International Conference on Power Electronics ECCE Asia*, volume 2, pages 210 – 217, The Shilla Jeju, Korea, May 2011.

[129] Jose Rodriguez, Marian P. Kazmierkowski, José R. Espinoza, Pericle Zanchetta, Haitham Abu-Rub, Héctor A. Young, and Christian A. Rojas. State of the art

of finite control set model predictive control in power electronics. *IEEE Trans. Industrial Informatics*, 9(2):14, May 2013.

[130] Samir Kouro, Patricio Cortés, René Vargas, Ulrich Ammann, and José Rodríguez. Model predictive control — a simple and powerful method to control power converters. *IEEE Trans. Industrial Electronics*, 56(6):1826 – 1838, June 2009.

[131] S. Alireza Davari, Davood Arab Khaburi, and Ralph Kennel. An improved FCS–MPC algorithm for an induction motor with an imposed optimized weighting factor. *IEEE Trans. Power Electronics*, 27(3):12, March 2012.

[132] Stephan P. Bradley, Arnoldo C. Hax, and Thomas L. Magnanti. *Applied Mathematical Programming*. Addison-Wesley, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, 1977.

[133] Esteban J. Fuentes, Cesar A. Silva, and Juan I. Yuz. Predictive speed control of a two-mass system driven by a permanent magnet synchronous motor. *IEEE Trans. Industrial Electronics*, 59(7), July 2012.

[134] Ashwin M. Khambadkone and Joachim Holtz. Compensated synchronous PI current controller in overmodulation range and six-step operation of space-vector-modulation-based vector-controlled drives. *IEEE Trans. Industrial Electronics*, 49 (3):574 – 580, June 2002.

[135] Vadim I. Utkin. Variable structure systems with sliding modes. *IEEE Trans. Automatic Control*, 22(1):212 – 222, February 1977.

[136] Vadim I. Utkin. Sliding mode control design principles and applications to electric drives. *IEEE Trans. Industrial Electronics*, 40(1):23 – 36, February 1993.

[137] Hashimoto H., Ishikawa Y., Harashima F., Rudev A., and Utkin V.I. Application of sliding mode control using reduced order model in induction motor. In *Power Electronics Specialists Conference*, volume 2, pages 259 – 264. IEEE Conference Publications, 1992.

[138] Hermann Haken. *Synergetics: An Introduction*. Springer, Berlin, 1983.

[139] Hermann P. J. Haken. Synergetics. *IEEE Circuits and Devices Magazine*, 4:3 – 7, November 1988.

[140] Hermann Haken. *Principles of Brain Functioning. A Synergetic Approach to Brain Activity, Behavior, and Cognition*. Springer, 1996.

[141] Michael Schanz and Axel Pelster. Synergetic system analysis for the delay-induced hopf bifurcation in the wright equation. *SIAM J. Applied Dynamical Systems*, 2 (3):277–296, 2003.

[142] Anatoly A. Kolesnikov. *Synergetic Control Theory.* Moscow-Taganrog, 1994.

[143] Anatoly A. Kolesnikov. *Modern Applied Control Theory: Synergetic approach in control theory.* TSURE press, Moscow-Taganrog, 2000.

[144] Anatoly A. Kolesnikov. *Synergetic Control of Complex Systems: Theory of systematic synthesis.* Editorial URACC, Moscow, 2005.

[145] Enrico Santi, Antonello Monti, Donghong Li, Karthik Proddutur, and Roger A. Dougal. Synergetic control for DC–DC boost converter: Implementation options. *IEEE Trans. Industrial Applications*, 39(6):1803 – 1813, November/December 2003.

[146] Zhenhua Jiang and Roger A. Dougal. Synergetic control of power converters for pulse current charging of advanced batteries from a fuel cell power source. *IEEE Trans. Power Electronics*, 19(4):1140 – 1150, July 2004.

[147] Chuanjiang Li, Jing Huang, Guangfu Ma, and Zhongzhao Zhang. Spacecraft attitude tracking based on nonlinear synergetic optimal control. In *2010 3rd International Symposium on Systems and Control in Aeronautics and Astronautics (ISSCAA)*, pages 7 – 12. IEEE Conference Publications, 2010.

[148] I. Kondratiev and R. Dougal. Current distribution control design for paralleled DC-DC converters using synergetic control theory. In *Power Electronics Specialists Conference*, pages 851 – 857. IEEE Conference Publications, 2007.

[149] Alessandro Astolfi and Romeo Ortega. Immersion and invariance: A new tool for stabilization and adaptive control of nonlinear systems. *IEEE Trans. Automatic Control*, 48(4), April 2003.

[150] Nusawardhana, S. H. Żak, and M. W. A. Crossley. Nonlinear synergetic optimal controllers. *Guidance, Control and Dynamics*, 30(4), August 2007.

[151] P. Mutschler. A new speed-control method for induction motors. In *PCIM*, pages 131 – 136, Nuremberg, May 1998.

[152] P. Mutschler. Digital implementation of predictive direct control algorithms for induction motors. In *Industry Applications Conference*, volume 1, pages 444 – 451, May 1998.

[153] Matthias Preindl and Silverio Bolognani. Model predictive direct torque control with finite control set for PMSM drive systems, part 1: Maximum torque per ampere operation. *IEEE Trans. Industrial Informatics*, 9(4):1912 – 1921, November 2013.