NTNU – Trondheim
Norwegian University of
Science and Technology

Predictive Control applied to Power Electronics Conversion Systems

From Simulation to Implementation

Katrine Skjelanger Kvinge

Master of Energy and Environmental Engineering
Submission date: February 2015
Supervisor: Marta Molinas, ELKRAFT
Co-supervisor: Kjell Ljøkelsøy, SINTEF

Norwegian University of Science and Technology
Department of Electric Power Engineering

Problem description

A model to simulate predictive control applied to a matrix converter has been developed in the specialization project. This model is going to be used as a base to develop a model predictive control for a two level inverter that will eventually be tested in a prototype.

In the specialization project the algorithm for the predictive control was executed on a real-time simulator OP5600 HILBOX produced by OPAL-RT Technologies. This method resulted in overruns when running the simulation in real-time for a sampling period of 10 us. As a high sampling frequency is crucial to ensure that the control method delivers a good reference tracking, the algorithm will be executed on a FPGA. Xilinx Design Tools software, combined with RT-Lab, will be used to adapt the model to be running on the FPGA. When the model is working, the control method will be tested in the lab.

Start date: September 8th 2014

Supervisor: Marta Molinas

Preface

First, I would like to thank my supervisor Professor Marta Molinas for providing me an interesting project. I would also like to thank my co-supervisor Kjell Ljøkelsøy for advice and assistance during the project work. PhD candidate Nathalie Holtsmark deserves my gratitude for assisting me and providing valuable advice and support throughout the project work. Next, I would like to thank Thomas S. Haugan for sharing experience from his project with a similar experimental setup and RT-Lab. Finally, I want to express my appreciation to Jesper Bjerre and Siri Hårklau, who assisted me with proofreading and layout for my report.

Katrine Skjelanger Kvinge

Trondheim, Norway

February 20th 2015

Abstract

Model predictive control has been a topic of research for approximately three decades, and is considered one of the most important advances in process control. In this project, a model to simulate predictive control applied to a two level inverter has been developed. The first step was to develop a Simulink model for testing of predictive current control. Sampling periods of 25 µs and 1 µs for the predictive control algorithm was tested, to investigate the effect the frequency of predictions has on the performance. The conclusion was that a higher sampling frequency results in a significantly better performance.

For experimental purposes, the predictive control algorithm will be executed on a FPGA. Xilinx System Generator (XSG) was used to adapt this part of the Simulink model to consist of blocks from the XSG, and the predictive control algorithm was altered to be suitable for execution on a FPGA. Switching frequency reduction was added to the predictive control. Simulation with only current control resulted in a switching frequency of 43.8 kHz, which is not feasible for use in a physical circuit. The weight assigned to the current reference tracking versus the weight of the switching frequency reduction, can be specified by the use of a weighing factor, A. Simulations with different values of A were performed, and this showed that the switching frequency can easily be reduced by increasing A, but the cost is a less accurate current reference tracking.

The control method was tested in an experimental setup with a 20 kW IGBT inverter and a resistive inductive load. For this purpose the model was implemented in the RT-Lab environment, and custom RT-XSG communication blocks were inserted in the model to handle data transferring between the CPU, FPGA and inverter. The testing of the predictive control model on the experimental setup was not successful. There was no response from the system, and no switching occurred. It is believed that this is a result of an error in the data transfer from the CPU to the FPGA model. Troubleshooting to locate the error will be included in further work.

Samandrag

Modell prediktiv kontroll har vore eit tema for forsking i nærmare tre tiår, og er ansett som eit av dei viktigaste framstega innan prosesstyring. Ein modell for simulering av prediktiv kontroll av ein vekselrettar er utforma i dette prosjektet. Første steg var å utvikla ein Simulink modell for testing av prediktiv straumregulering. Sampling periodar på 25 µs og 1 µs for den prediktive kontroll algoritmen vart testa, for å undersøka effekten frekvensen på prediksjonane har på ytinga til kontroll metoden. Konklusjonen vart at ein høgare sampling frekvens resulterar i ei betydeleg betre yting.

For eksperimentelle formål vert den prediktive kontroll algoritmen utført på ein FPGA. Xilinx System Generator (XSG) vart brukt til å tilpassa denne delen av Simulink modellen til å bestå av XSG-blokker, og den prediktive kontroll algoritmen vart modifisert for å kunne utførast på ein FPGA. Reduksjon av svitsjefrekvens vart inkludert i den prediktive kontrollen. Simuleringar med berre straumregulering resulterte i ein svitsjefrekvens på 43.8 kHz, noko som ikkje er anvendbart på ein fysisk krets. Vekta tillagt straumregulerings kriteriet samanlikna med vektlegginga av svitsjefrekvens reduksjon, kan spesifiserast ved bruk av ein vektleggingsfaktor, A. Simuleringar med ulike verdiar for A vart utført, og dette viste at svitsjefrekvensen enkelt kan reduserast ved å auka verdien av A, men prisen for dette er at referanseføLginga på straumen vert mindre presis.

Kontrollmetoden vart testa i eit eksperimentelt oppsett med ein 20 kW IGBT vekselrettar og ei resistiv induktiv last. For dette formålet vart modellen implementert i RT-LAB, og spesielle RT-XSG kommunikasjonsblokker vart innført i modellen for å handtera dataoverføringa mellom CPU, FPGA og vekselrettaren. Forsøka med den prediktive kontrollen i det eksperimentelle oppsettet var ikkje vellukka. Det var ingen respons frå systemet, og ingen svitsjing vart utført. Ein teori er at dette er eit resultat av ein feil i kommunikasjonen mellom CPU og FPGA modellen. Feilsøking for å lokalisera feilen må inkluderast i vidare arbeid.

Contents

List of Figures

List of Tables

1 Introduction

The control of power converters is a significant topic for research. Model predictive control (MPC) have been developed for approximately three decades, and is considered one of the most important advances in process control [4]. The control method requires a high amount of computations, and this has previously been a challenge holding the development back. Today this is no longer a problem, as there are control platforms that have a lot of computational power, and this makes this type of control very interesting. Predictive control has several advantages that makes it very suitable for the control of power converters: Concepts are intuitive and easy to understand, it can be applied to a variety of systems, constraints and nonlinearities can be easily included, multivariable cases can be considered and the resulting controller is easy to implement.

The MPC have been used in practical applications, which has been very successful [2]. Some of the main practical applications where MPC has been employed recently are distributed generation systems, active filtering and power conditioning, drives, non-conventional renewable energy and uninterruptable power supplies [4].

Three-phase current-controlled voltage source inverters (VSIs), have been considered as the most popular structure for supplying three-phase loads [4]. The fact that the two-level voltage source inverter is one of the most popular converter topologies in industry, combined with the fact that it has a generic structure that can be easily extended to other converter topologies, makes it very suitable for studying the basic principles of the model predictive control.

In this work, current control will be applied to a two-level VSI. However, if the only criterion for the predictive control is current reference tracking, and the algorithm is running with a high sampling frequency, it is expected that the switching frequency of the inverter will be very high. The switching frequency might be too high for a physical circuit to follow. Another

issue is the switching losses caused by a high switching frequency, as efficiency is an important factor in energy conversion. One of the advantages with the MPC is the simplicity of adding several control objectives. The switching losses will be added to the evaluation criterions of the control method, in order to limit the switching losses in the inverter.

In chapter 3 the predictive control is introduced, and a mathematical model of the power circuit containing the inverter, load and predictive control algorithm is presented. Chapter 4 presents a Simulink model derived from the mathematical model, and the results from the simulation in Simulink. The software RT-Lab and Xilinx System Generator, which will be used to adapt the model for use in an experimental setup, are presented in chapter 5. The first step in preparing the predictive control algorithm for execution on an FPGA, is to exchange the Simulink blocks with Xilinx blocks, and this new model is presented in chapter 6. Results from simulation with this Xilinx model, combined with simulated hardware, are also presented in this chapter. Finally the model is adapted to RT-Lab, and tested on an experimental setup. This process is described in chapter 7. Chapter 8 contains the conclusion and further work.

2 The inverter

2.1 Power circuit of the inverter

The power circuit of the inverter is shown in Figure 2-1. The electrical power is supplied by a DC-source, with a voltage V_{dc}, and is transformed to AC by controlling the current flow through the switches S1-S6.

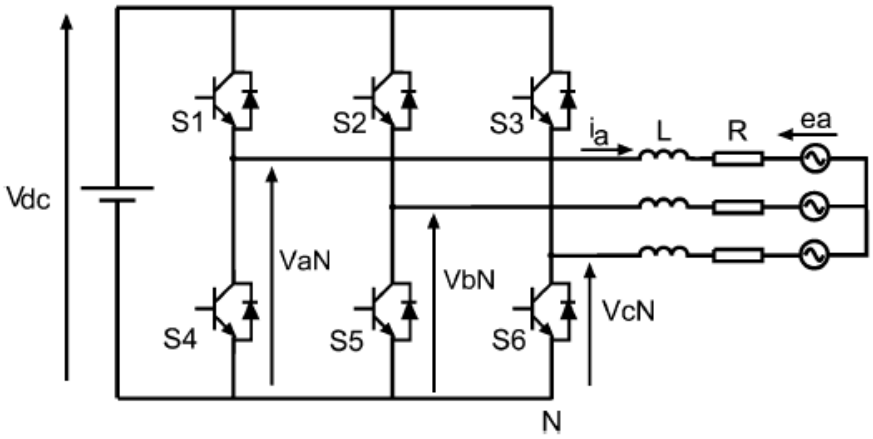

Figure *2-1: Voltage source inverter power circuit [3]*.

The inverter consists of three legs, where each leg is coupled to one of the three phases in the load. In order to avoid short circuiting the DC-source, the two switches of each leg operates in complementary mode. The switching states can then be represented by the three switching signals defined in equation (1) – (3).

$$S_a = \begin{cases} 1, & S_1 \; is \; conducting \; and \; S_4 \; is \; blocking \\ 0, & S_1 \; is \; blocking \; and \; S_4 \; is \; conducting \end{cases} \quad (1)$$

$$S_b = \begin{cases} 1, & S_2 \; is \; conducting \; and \; S_5 \; is \; blocking \\ 0, & S_2 \; is \; blocking \; and \; S_5 \; is \; conducting \end{cases} \quad (2)$$

$$S_c = \begin{cases} 1, & S_3 \; is \; conducting \; and \; S_6 \; is \; blocking \\ 0, & S_3 \; is \; blocking \; and \; S_6 \; is \; conducting \end{cases} \quad (3)$$

This can be expressed in vectorial form as

$$\boldsymbol{S} = \frac{2}{3}(S_a + \boldsymbol{a}S_b + \boldsymbol{a}^2 S_c) \quad (4)$$

where

$$\boldsymbol{a} = e^{j2\pi/3} = -\frac{1}{2} + j\frac{\sqrt{3}}{2} \tag{5}$$

The unity voltage vector **a** represents the phase displacement of 120° between the phases. Given the DC source voltage, V_{dc}, the output voltages can be found from the switching states.

$$v_{aN} = S_a V_{dc} \tag{6}$$

$$v_{bN} = S_b V_{dc} \tag{7}$$

$$v_{cN} = S_c V_{dc} \tag{8}$$

This is the phase-to-neutral voltages, where N is the neutral point, as indicated in figure 1. Different switching states will create different configurations of the three-phase load. The output load voltage vector created by the inverter can be described as

$$\mathbf{v} = \frac{2}{3}(v_{aN} + \boldsymbol{a} v_{bN} + \boldsymbol{a}^2 v_{cN}). \tag{9}$$

As the switches in each leg works in complementary mode, considering all the possible combinations of switching states results in eight valid combinations. The voltage vectors generated by these states are presented in Table 2-1

Table 2-1: Switching states and voltage vectors

Voltage vector **V**	Sa	Sb	Sc	Value
$\mathbf{V}_0$	0	0	0	0
$\mathbf{V}_1$	1	0	0	$\frac{2}{3} V_{dc}$
$\mathbf{V}_2$	1	1	0	$\frac{1}{3} V_{dc} + \frac{\sqrt{3}}{3} V_{dc}$
$\mathbf{V}_3$	0	1	0	$-\frac{1}{3} V_{dc} + \frac{\sqrt{3}}{3} V_{dc}$
$\mathbf{V}_4$	0	1	1	$-\frac{2}{3} V_{dc}$
$\mathbf{V}_5$	0	0	1	$-\frac{1}{3} V_{dc} - \frac{\sqrt{3}}{3} V_{dc}$
$\mathbf{V}_6$	1	0	1	$\frac{1}{3} V_{dc} - \frac{\sqrt{3}}{3} V_{dc}$
$\mathbf{V}_7$	1	1	1	0

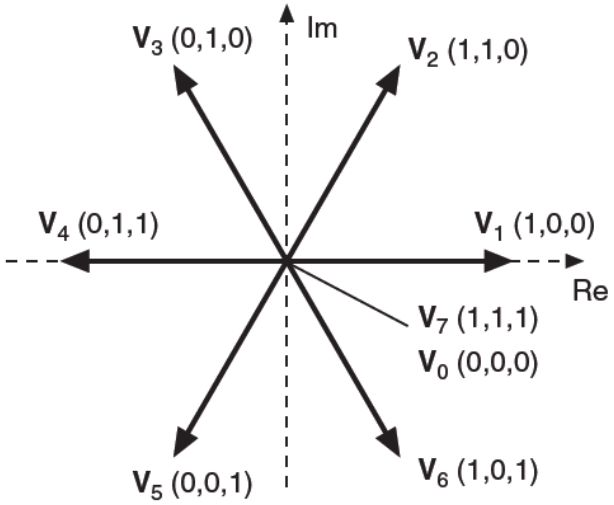

Figure 2-2: Voltage vectors generated by the inverter [1].

The switching states resulting in each voltage vector are indicated in the figure. It can be observed that the vectors $\mathbf{V}_0$ and $\mathbf{V}_7$ results in the same voltage vector. This gives a finite set of seven different voltage vectors. It should be noted that a more accurate model of the converter could be used for higher switching frequencies. Subjects of interest could be modeling dead time, diode forward voltage drop and IGBT saturation voltage. However, in order to keep the model simple and focus on the control strategy, such topics have not been included in this work.

2.2 Switching losses

Power dissipation in semiconductor power devices is fairly generic in nature, which means the same basic factors governing power dissipation apply to all devices in the same manner [5]. Figure *2-3* shows the generic-switch switching characteristics, where the current that is flowing through the switch also flows through some series inductance. This case is a very common occurrence in many power electronic applications. Part A in the figure shows the switch waveforms, with upper graph showing the switch control signal. A linearized version of the current flowing through the switch, i_T, and the voltage over the switch, v_T, is presented in the second part of the figure. It takes some time for both the current and voltage to change value during switching, and this is called the rise and fall time. When the switch turns on, there is a short delay time, t_d, before the rise time of the current, t_{ri}. After the current flows entirely through the switch, the voltage can fall to a low on-state value, V_{on}, which takes a

given time, t_{fv}. Together these time intervals make up the time it takes for the switch to turn on. The same behavior can be found when turning the switch off, though the rise and fall time for voltage and current are different in this case, as illustrated in Figure *2-3*. The turn-on crossover interval is defined as:

$$t_{c(on)} = t_{ri} + t_{fv} \tag{10}$$

and the turn-off interval as

$$t_{c(off)} = t_{rv} + t_{fi}. \tag{11}$$

At the turn-on and turn-off intervals, both a high current and voltage is present, and this results in losses in the switch. Part B of the figure illustrates the instantaneous power loss that occurs in the switch during the commutation. The losses can then be estimated by the following equations.

$$W_{c(on)} = \frac{1}{2} V_d \, I_o t_{c(on)} \tag{12}$$

$$W_{c(off)} = \frac{1}{2} V_d \, I_o t_{c(off)} \tag{13}$$

This shows that the switching power loss in a semiconductor switch varies linearly with the switching frequency and the switching times.

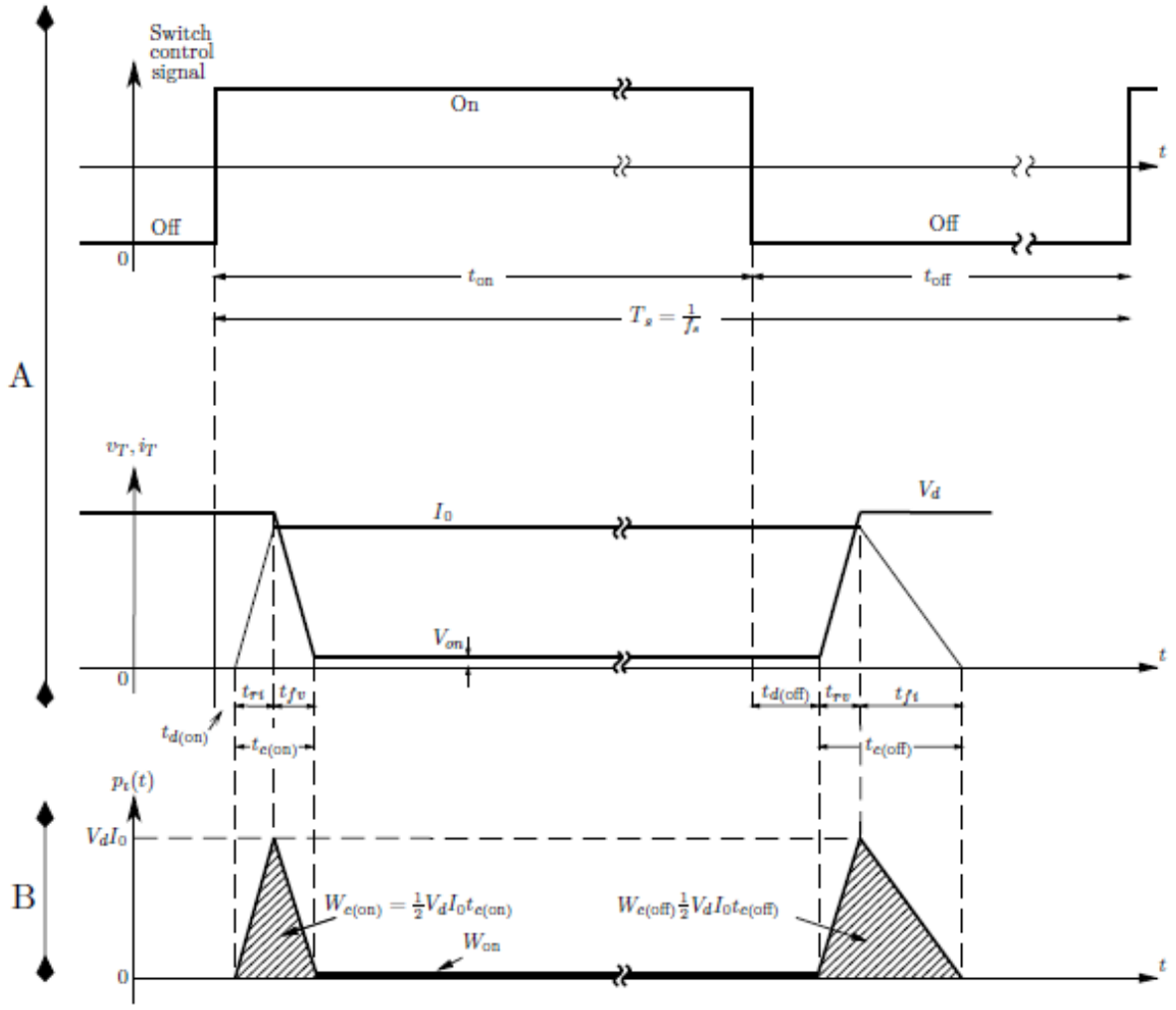

Figure 2-3: Generic-switch switching characteristics (linearized): (A) switch waveforms, (B) instantaneous switch power loss.

However, it should be noted that this is a theoretical approximation, and in reality the losses will deviate from the linear approximation. Figure 2-4 presents a graph from a datasheet for an IGBT module with resembling characteristics as the one that will be used in the experimental setup. The losses vary with the current flowing through the transistor, and the additional energy loss due to the reverse-recovery current flowing through the diode in parallel with the transistor, E_{rr}, is also included.

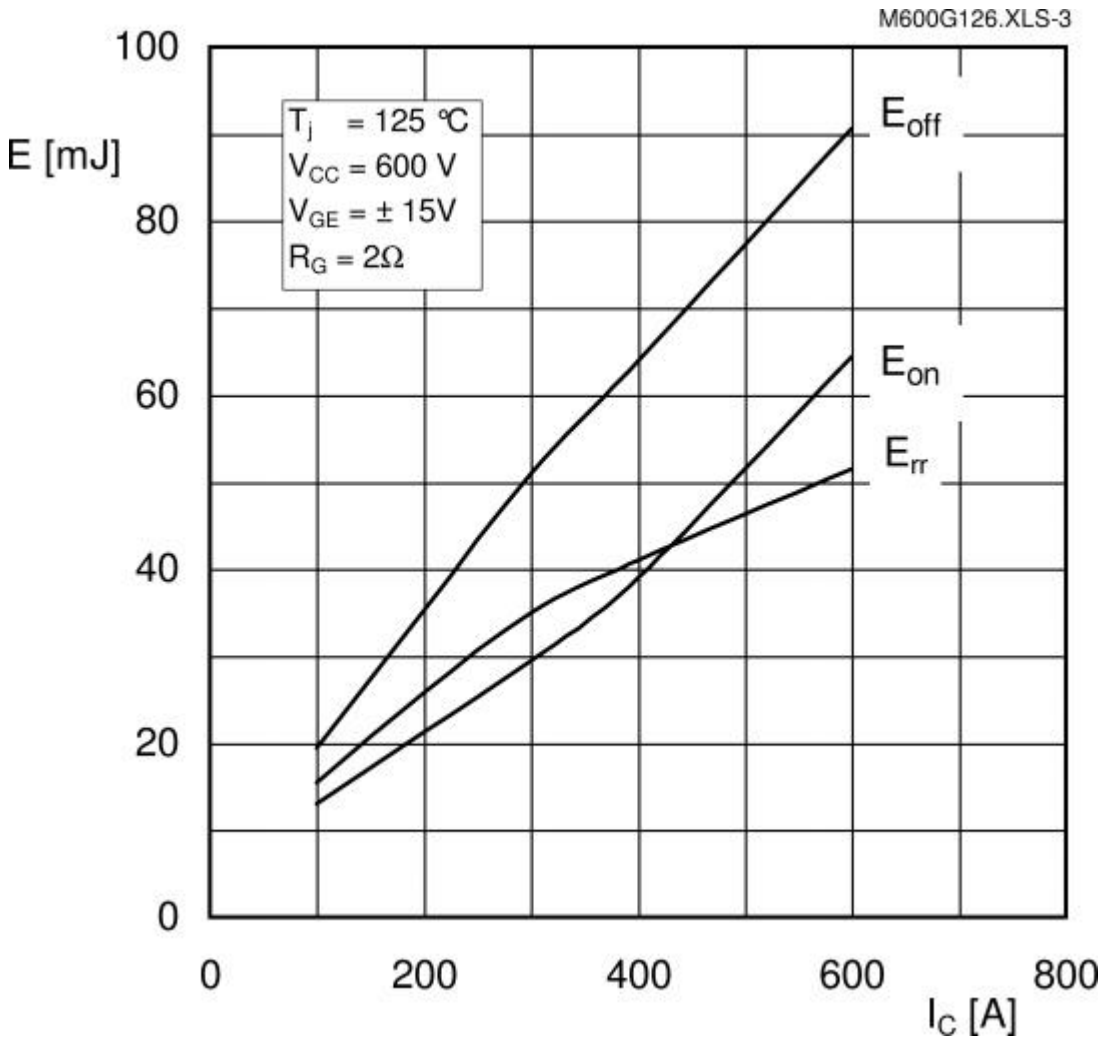

Figure 2-4: Typ. Turn-on/-off energy = f(Ic).

3 Predictive control

3.1 Introduction to predictive control [1]

The main characteristics of predictive control are the use of a model to predict the future behavior of the system. An optimization criterion is defined, and the optimal actuation is then decided based on this criterion and the predictions made. Predictive control covers a wide class of controllers, and a classification for different predictive control methods proposed in [2] is presented Figure 3-1.

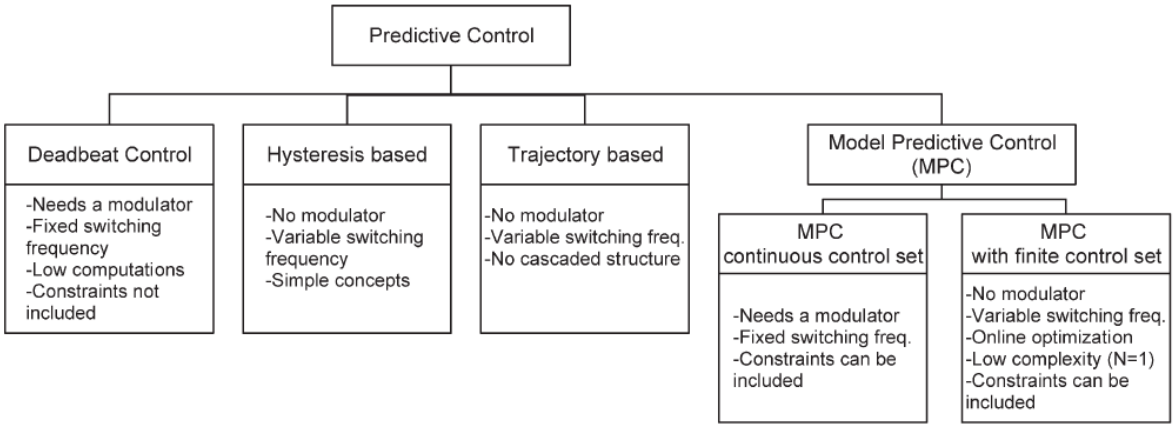

Figure 3-1: Classification of predictive control methods [2].

The different types of predictive control have different optimization criterions. In hysteresis based control, the criterion is to keep the controlled variable within the boundary of the hysteresis area, while in deadbeat control it is to find the actuation that gives zero error in the next sampling instant. For trajectory-based control the variables follows a predefined trajectory. For model predictive control the optimization criterion is a cost function that should be minimized, which is the most flexible [1].

This report will focus on the model predictive control (MPC) with a finite control set and finite prediction horizon. The finite control set makes it easy to implement, needs no modulator and therefore operates with a variable switching frequency. The name of the

[1] Section 3.1 is a modified extract from the report written on the specialization project «Real-time Simulation of Predictive Control with a Matrix Converter» leading up to this Master thesis.

method is based on the finite number of switching states and following actuations, and includes the discrete nature of the power converters. The MPC have been extremely successful in practical applications, and has had a great influence on research in the recent decades [2]. Figure 3-3 shows the general scheme for MPC, where $\mathbf{x}^*(k)$ is the reference value and **S** is the switching function.

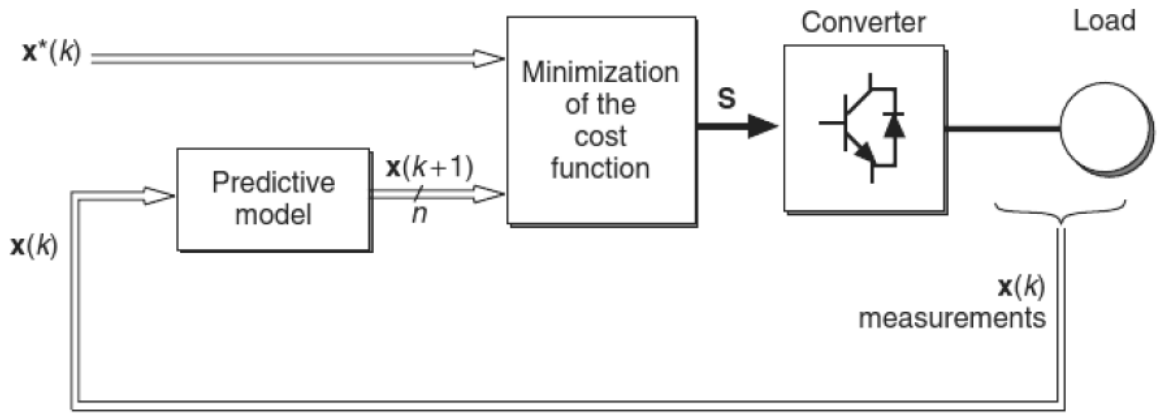

Figure 3-3:General MPC scheme for power converters [1].

Based on the measured value **x**(k), a prediction of the next value **x**(k+1) is made. The time from k to (k+1) is the prediction horizon. This value is then compared to the reference value, and by minimizing the cost function the optimal switching state, **S**, is found. This is repeated for every new sampling instant. Figure 3-4 illustrates this basic working principle. The future values are predicted until a given horizon in time (k + N), using the system model and the measured values at time k. The optimal switching state is found, and this is applied until the next sampling instant (k+1) where the calculations are performed again.

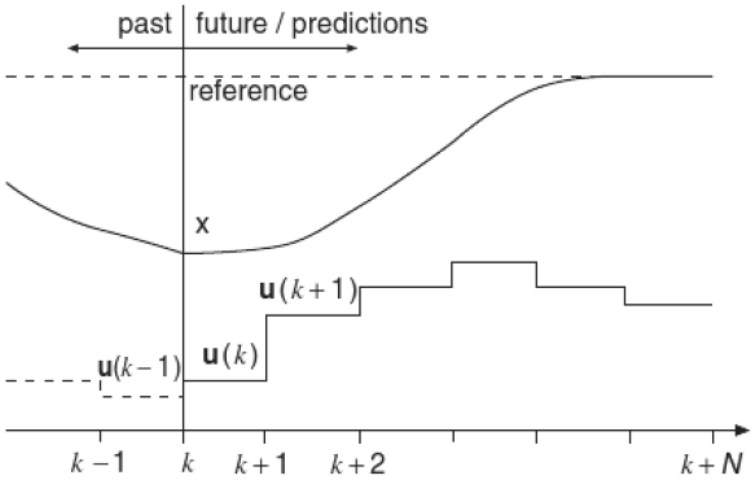

Figure 3-4: Working principle of model predictive control [1].

3.2 Model of the system

A model of the system is needed to calculate the predicted values of the variables. The system that will be modelled corresponds to the power circuit of the inverter presented in Figure 2-1. This section will present a mathematical model used for predictive current control including switching frequency reduction.

3.2.1 Model of the inverter

The working principles of the inverter were introduced in Chapter 2. Equation (9) presented the definition of the output load voltage vector. This can also be expressed as

$$\begin{bmatrix} v_{aN}(t) \\ v_{bN}(t) \\ v_{cN}(t) \end{bmatrix} = V_{dc} \cdot \begin{bmatrix} S_a(t) \\ S_b(t) \\ S_c(t) \end{bmatrix} \tag{14}$$

where the switching states are as defined in equations (1) - (3).

3.2.2 Load model

To predict the load current, a model of the load is established. The resistive-inductive load can be described by the following equation

$$L\frac{d\boldsymbol{i_o}(t)}{dt} = \boldsymbol{v_o}(t) - R\boldsymbol{i_o}(t) - \boldsymbol{e}(t) \tag{15}$$

where L and R are the inductance and resistance of the load and **e** is the electromotive force, emf. With a sampling period equal to T_s, the following approximation can be made.

$$\frac{d\boldsymbol{i_o}}{dt} \approx \frac{\boldsymbol{i_o}(k+1) - \boldsymbol{i_o}(k)}{T_s} \tag{16}$$

Substituting (15) into (16) gives the equation needed to calculate the predicted value of the load current at time (k + 1).

$$\boldsymbol{i_o}(k+1) = \left(1 - \frac{R\boldsymbol{T_s}}{L}\right)\boldsymbol{i_o}(\boldsymbol{k}) + \frac{T_s}{L}\left(\boldsymbol{v_o}(k) - \hat{\mathrm{e}}(k)\right) \tag{17}$$

The corresponding load voltage vector is calculated for every valid switching state, using (14). Extrapolation from present and past values can be used to estimate the present value of the emf, **e**(t). For sufficiently small sampling time, the approximation e(k-1) ≈ e(k) is used, and no extrapolation is needed.

3.2.3 The cost function

The cost function will represent the desired behavior of the system. One approach is to make a cost function that when minimized gives the optimal switching state. Several objectives can be implemented in the cost function. In this case the objectives are to control the output current and keep it close to the reference value, as well as confining the switching frequency in order to limit the switching losses.

3.2.3.1 Current control

The idea is to assign cost to the cost function, g, whenever a predicted value deviates from the reference value. The current control can then be described by

$$g_{current} = \left|i^*_{o\alpha}\,(k+1) - \, i^p_{o\alpha}\,(k+1)\right| + \left|i^*_{o\beta}\,(k+1) - \, i^p_{o\beta}\,(k+1)\right| \qquad (18)$$

where the subscript * indicates the reference values and p the predicted values.

3.2.3.2 Reduction of switching cost

The second objective is to control the switching frequency in order to limit the switching losses. A simplified model for estimating the switching losses, based on a practical approximation, was presented in [6]. After analyzing experimental data and studying previous work regarding models to estimate switching losses [7], [8], [9], [10], [11], the conclusion was that a simple model based on the switched current and voltage involved in the commutation, can be used to approximate the switching losses. By assuming that the losses vary linearly with the voltage and current at the switching instant, and considering a proportional constant, the loss for one commutation can be considered as

$$E_{loss} \propto \Delta i_c * \Delta v_{ce} \tag{19}$$

where i_c is the collector current and v_{ce} is the collector-emitter voltage. If Δi_c represents the current change for the commutation, and Δv_{ce} is the change in voltage over the transistor, then the loss can be estimated as

$$E_{loss} = \int_{t_c} p(t)dt == \int_{t_c} i_c(t)v_{ce}(t)dt = \frac{t_c}{6} * \Delta i_c(t) * \Delta v_{ce}(t) \tag{20}$$

where p(t) is the electric power, and t_c represents the turn-on or turn-off interval described in equations (10) and (11). This is a very simplified calculation, which does not take into account all the aspects of a switching process, and neither considers the losses due to soft commutation. However, it is a valid approximation that will be very useful for the purpose in this application, which is to reduce the switching frequency based on a cost function that represents that the losses will rise with increasing frequency.

To penalize the power consumption, an extra term is added to the cost function (18), as described in equation (21).

$$g_{sw} = A * \sum_{i=1}^{3} \Delta i_c^{(i)} \Delta v_{ce}^{(i)} + e_0 \tag{21}$$

i represents the three phases, and as the two switches in each phase leg of the inverter works complementary, the change in switching state of the leg will be represented as one term. Here v_{ce} correspond to the voltage from the DC-source, V_{dc}, and i_c will be equal to the phase current for the leg i. e_0 represents the loss that will occur when the current has a low value or is close to zero. Figure *2-4* presented a graph showing the relation between the collector current and switching loss in an IGBT with similar characteristics as the one used in the experimental setup. The value of e_0 should be scaled according to this relation between current and energy loss. In this work, the term e_0 has been included mainly to illustrate the fact that a loss can still occur even when the current is measured to zero, and that it is not cost-free to switch at a low

current. For this reason, a simple approximation of the value for e_0 has been done, and the priority have been to select a value in a representable range, not on selecting an exact value.

The loss approximation found by the current and voltage change should be multiplied by a proportional constant. However, it is not necessary to add an extra constant, as this will be compensated for by using a weighing factor, A.

3.2.3.3 The weighing factor

The two control objectives are combined by simply adding the two cost functions, resulting in one cost function for the system.

$$g = g_{current} + g_{sw} \tag{22}$$

In order to be able to control the significance of the different control objectives, weighing factors are introduced. The weighing factor A is inserted in the equation to control the relevance of the switching loss reduction control.

$$g = \left|i^*_{o\alpha}\,(k+1) - i^p_{o\alpha}\,(k+1)\right| + \left|i^*_{o\beta}\,(k+1) - i^p_{o\beta}\,(k+1)\right| + A * \sum_{i=1}^{3} \Delta i_c^{(i)}\,\Delta v_{ce}^{(i)} + e_0 \tag{23}$$

Selection of the value of weighing factor A is still an open topic for research, and at the present state of the art, these values are determined empirically. The reason why deciding the weighing factors is challenging is the changing value of the reference. There are different optimal values for the cost function with different values of the parameters controlled [12]. Work has been done to develop an approach based on an empirical procedure to obtain suitable weighing factors. The method suggested when a cost function includes a primary control objective and secondary terms, is to set A = 0 as a starting point, then test increments of A until the desired behavior is achieved [13].

3.2.4 Performance variables

The proposed control method has two objectives. Current reference tracking and switching frequency reduction. In order to evaluate the performance of the control method, some evaluation criterions needs to be established. For that purpose, two performance variables are defined. The average switching frequency will be defined as

$$f_s = \frac{f_{sa} + f_{sb} + f_{sc}}{3} \tag{24}$$

where f_{sa}, f_{sb} and f_{sc} are the switching frequency of the three phases. It should be noted that since the two switches in in each converter leg works in complementary mode, for every switching there will be two switches changing state, and two commutations causing power loss.

To evaluate the current reference tracking, the mean reference tracking error will be defined by the following equation.

$$\bar{e} = \frac{1}{m}\sum_{k=0}^{m} \left(\left| i_\alpha^* - i_\alpha^p \right| + \left| i_\beta^* - i_\beta^p \right| \right) \tag{25}$$

4 Simulink

4.1 Simulink model

Simulink was used for simulation of the power circuit for the inverter presented in Figure 2-1 together with the predictive control algorithm, with the aim to later adapt it in the OPAL-RT environment. For simplicity, the same model as presented in [1] Appendix A was used, although some changes were made. The Simulink model is derived from the theory and mathematical model presented in chapter 3. However, only the current control was tested in the Simulink model. An extra criterion in the cost function, concerning the switching frequency, was added to the algorithm when the model was adapted to the RT-Lab/Xilinx environment, with experimental implementation in mind. The following chapter will explain the model and changes made from the example in [1].

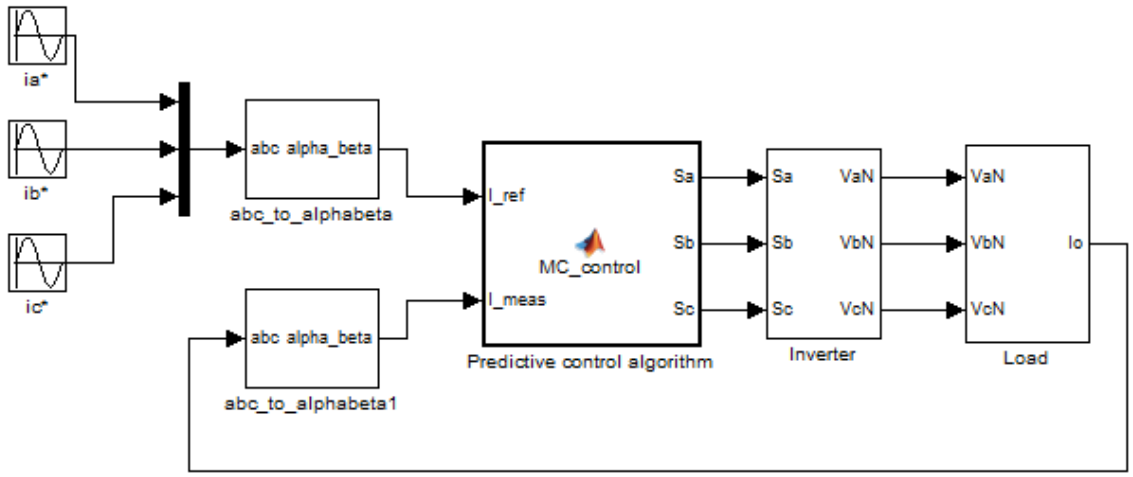

Figure 4-1: System model in Simulink.

Matlab 2011b (32-bit) was used, due to compatibility with RT-LAB. Figure 4-1 shows the system model. The system consists of a Matlab Function block containing the predictive control algorithm, an inverter, a resistive-inductive load, reference currents and two blocks for coordinate transformation of the currents. A constant representing the DC-source feeding the inverter is placed inside the inverter block. In addition to this model, a parameter-file was made to have a good overview of the different parameters, and to make them easily accessible.

4.1.1 Load

The model for the resistive-inductive load is shown in Figure 4-2. As presented in Figure 1, the load is star-connected. Kirchhoff's current law can then be used to describe the load phase voltages as

$$v_{an} = v_{aN} - v_{nN} \quad (26)$$

$$v_{bn} = v_{bN} - v_{nN} \quad (27)$$

$$v_{cn} = v_{cN} - v_{nN} \quad (28)$$

The common-mode voltage can then be found by adding the phase voltages.

$$v_{aN} + v_{bN} + v_{cN} = L\frac{d}{dt}(i_a + i_b + i_c) + R(i_a + i_b + i_c) + (e_a + e_b + e_c) + 3v_{nN} \quad (29)$$

For a star-connected load, $i_a + i_b + i_c = 0$. Assuming that the back-emf is balanced $e_a + e_b + e_c$ will also be zero. This gives a simpler expression for the common-mode voltage.

$$v_{nN} = \frac{1}{3}(v_{aN} + v_{bN} + v_{cN}) \quad (30)$$

This can easily be implemented in the Simulink model. The back-emf is represented with Sine Wave blocks and the dynamics of the load by use of linear continuous-time transfer functions. The transfer function is found by taking the basic equation for a resistive-inductive circuit, here presented for phase a

$$v_{an} = L\frac{di_a}{dt} + Ri_a + e_a \quad (31)$$

and taking the Laplace transform of this.

$$\frac{I_a}{V_{an} - E_a} = \frac{1}{L_s - R} \quad (32)$$

Upper case letters represent the Laplace transforms of the respective lower case variables in the time domain.

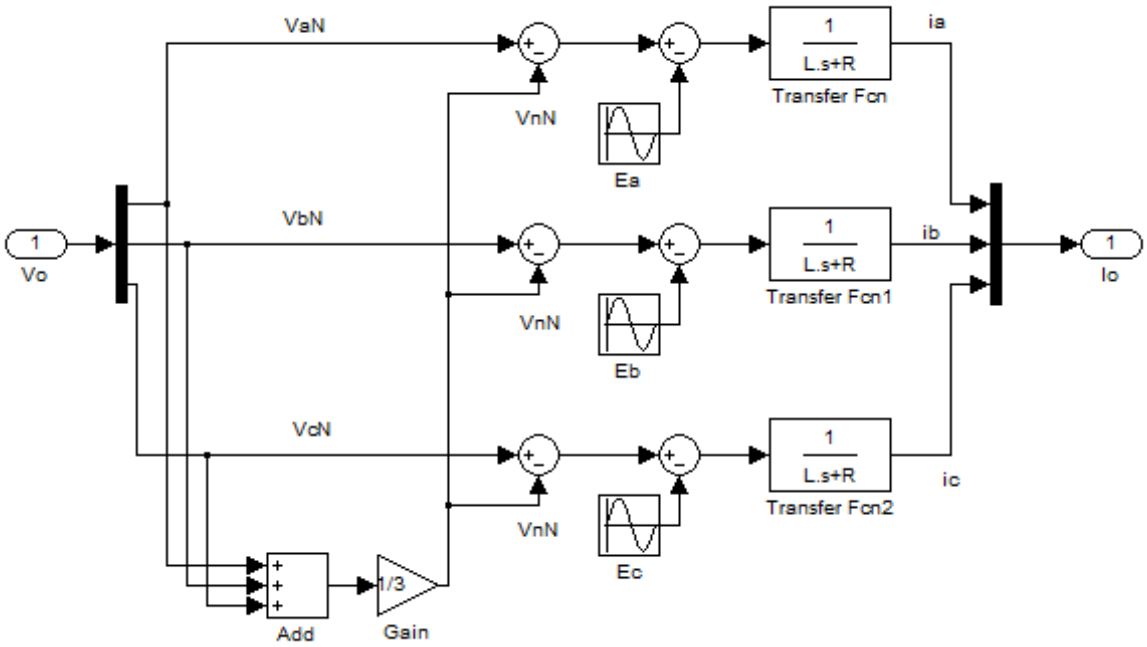

Figure 4-2: Load model.

4.1.2 Inverter

The inverter is shown in Figure 4-3 and takes the optimal switching signals as inputs. A constant is used to represent the DC-source voltage. For most practical cases, the DC link voltage will not be constant, and needs to be controlled. The Vdc block in this model would then be replaced with a model describing the variable DC-voltage. By multiplying the gating signals with the DC link voltage, the output voltage of each inverter leg with respect to the negative busbar, N, is calculated.

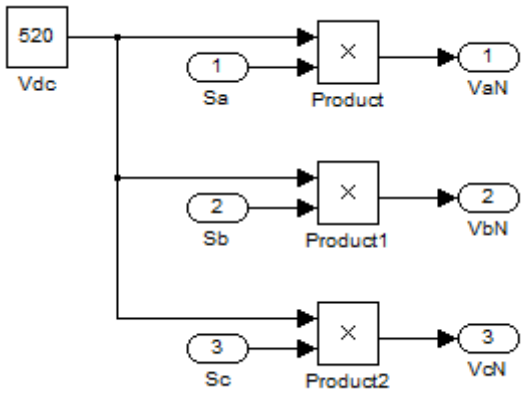

Figure 4-3: Inverter model.

4.1.3 Coordinate transformation

The reference current and the measured load currents are given in abc coordinates representing the three phases. By transforming the currents to αβ coordinates, the number of predictions required in the predictive control algorithm can be reduced. Equation (33) and (34) shows the relation between the two coordinate systems.

$$i_\alpha = \frac{2}{3}\left(i_a - \frac{1}{2}\,i_b - \frac{1}{2}\,i_c\right) \tag{33}$$

$$i_\beta = \frac{2}{3}\left(\frac{\sqrt{3}}{2}\,i_b - \frac{\sqrt{3}}{2}\,i_c\right) \tag{34}$$

Figure 4-4 shows the practical implementation of these equations in Simulink.

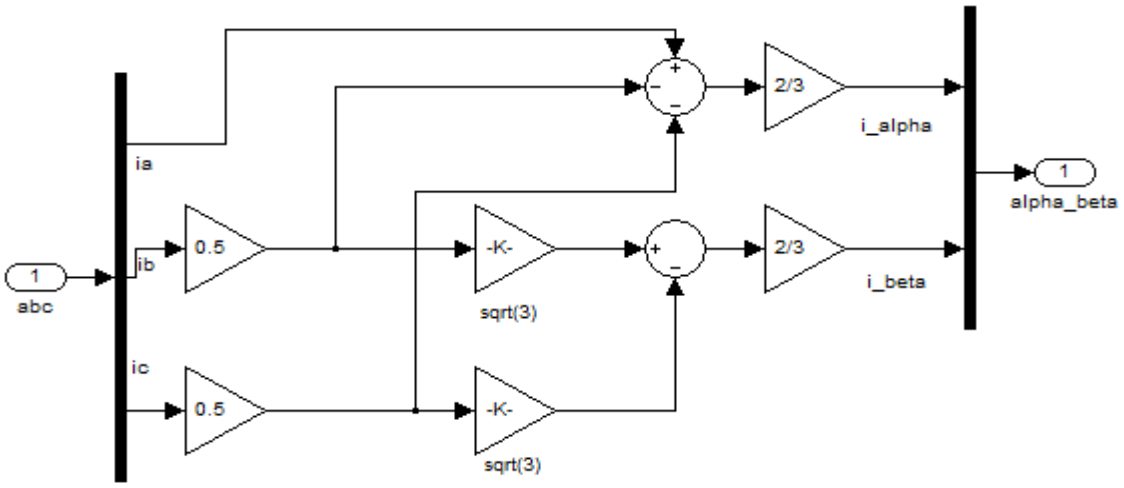

Figure 4-4: abc to alpha-beta coordinate transformation.

4.1.4 Parameter-file

The necessary parameters for the simulation are given in the file parameters.m, which can be found in appendix 1. *Table 4-1* shows the parameter names and values that are defined in the m-file. Simulations were made with T_s for the algorithm set to both 1µs and 25µs, to examine the effect of the sampling frequency on the performance of the control method. The sampling of the Matlab function block needs to be set under Subsystem Parameters.

Table 4-1: Parameters set in the parameters.m file

Parameter	Parameter name	Value
DC-link voltage	V	520 V
Load resistance	R	10 Ω
Load inductance	L	10 mH
Load back-emf amplitude	e	100 V
Load back-emf frequency	f_e	50 Hz
Reference current amplitude	i_ref	10 A
Reference current frequency	w_ref	50 Hz
Sampling time predictive control algorithm	Ts	Tested for 1 µs and 25 µs

In addition to setting the parameters presented in *Table 4-1*, the different voltage vectors are calculated. In the last line of the m-file all the valid switching states for the converter are defined.

Simulink requires some simulation configuration parameters to be set, in addition to the parameters set by running the file parameters.m. *Table 4-2* contains the parameters selected. The rest of the simulation parameters are set by default.

Table 4-2: Simulation configuration parameters

Parameter	Value
Start time	0 s
Stop time	0,3 s
Solver type	Fixed step
Solver	Ode 5 (Dormand-Prince)
Fixed-step size	1e-6 s

4.1.5 Predictive control algorithm

The predictive control algorithm is implemented using a Matlab Function block. This is an edited version of the Embedded Matlab Function block which was used in the example in [1]. This block has been changed and renamed with different versions of Matlab. It still has the same basic functions, but one of the differences is the use of global variables. In [1] the variables needed for the algorithm were set as global variables, and could be changed from the parameters.m file. The Matlab Function block for Matlab 2011b needs an allocated memory to store these global variables, and this can be done by using Data Store Memory blocks. This block defines and initializes a named shared data store, which can be used to read and write from. In this case it is only used for memory allocation, and setting the initial value. The initial value has to be set for every block, and the values are as given in the parameters file. All the Data Store Memory blocks are placed at the top level and then linked to the Matlab Function block. The linking is done within the Matlab Function block by declaring the parameters as global, and adding it in the Ports and Data Manager.

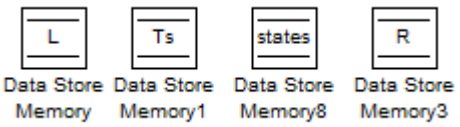

Figure 4-5: Data Store Memory blocks

The Matlab Function block uses the voltage vector that was calculated in the parameters-file based on the DC-link voltage. Using a Data Store Memory block to make the voltage vector

accessible to the algorithm in the Matlab Function block presented a new problem, as the Data Store Memory block does not accept complex numbers. To solve this problem the voltage vector was initialized inside the Matlab Function block, with the values calculated by running the parameter file. Another solution could be to separate the complex numbers into their real and imaginary parts. A new vector can then be made, containing eight elements for the real values and eight for the imaginary values. Some changes would be needed in the Mcode to apply the new vector. This solution would result in an easier way to adjust the DC value, and might be a better option if different DC values will be tested.

Code 1, presented in the next page, shows the code for the predictive control algorithm. The inputs are the reference current ant the measured load current . At line 19 and 22 the current reference and measurements are read into the variables ik_ref and ik. The prefix k indicates the sampling instant k. At line 11 x_old and i_old are initialized as persistent variables. If the variable does not exist the first time you issue the persistent statement, it will be initialized as an empty matrix. Persistent variables are similar to global variables, with the exception that they are local to the function. This results in the value being retained between function calls, as Matlab creates permanent storage for the variable.

In the for-loop covering line 33 to 45, the cost function for each of the 8 valid switching states is calculated, and the optimal switching state is decided. Prediction of the next value for the load current is carried out at line 37, according to equation (17). The cost function is then estimated at line 39, corresponding to equation (18). The switching combination with the lowest cost is selected and set as the output from the function.

One of the changes from the code in [1] is the addition of the initial value of x_opt, set in line 31. The original lack of an initial value caused problems when the requirement for the if-loop, g < gopt, were never met. In this case, x_opt would never be assigned a value, and the optimal switching state could not be found. Another change made in the code from the example code, is line 7 and 8, where the voltage vector is initialized. This change is due to the problems with Data store memory and complex numbers, as previously described.

```
function [g_opt, Sa, Sb, Sc] = MC_control(I_ref,I_meas)

%Variables defined in the parameters file
Global Ts R L states

%Initialize voltage vector
v = (1.0e+002)* [0 3.4667 (1.7333 + 3.0022i) (-1.7333 + 3.0022i)
-3.4667
(-1.7333 - 3.0022i) (1.7333 - 3.0022i) 0];

% Optimum vector and measured current at instant k-1
persistent x_old i_old

% Initialize values
if isempty(x_old), x_old = 1; end
if isempty(i_old), i_old = 0 + 1j*0; end
g_opt = 1e10;

% Read current reference inputs at sampling instant k
ik_ref = I_ref(1) + 1j*I_ref(2);

% Read current measurements at sampling instant k
ik = I_meas(1)+ 1j*I_meas(2);

% Back-emf estimate
e = v(x_old) - L/Ts*ik - (R - L/Ts)*i_old;

% Store the measured current for the next iteration
i_old = ik;

%Initialize x_opt
 x_opt = 0;%Denne linja er lagt til koden. x_opt vart berre
initialisert dersom kravet i if-løkka blei oppfylt.

for i = 1:8
    % i-th voltage vector for current prediction
    v_o1 = v(i);
    % Current prediction at instant k+1
    ik1 = (1 - R*Ts/L)*ik + Ts/L*(v_o1 - e);
    % Cost function
    g = abs(real(ik_ref - ik1)) + abs(imag(ik_ref - ik1));
    % Selection of the optimal value
    if (g<g_opt)
        g_opt = g;
        x_opt = i;
    end
end
% Store the present value of x_opt
x_old = x_opt;

% Output switching states
Sa = states(x_opt, 1);
Sb = states(x_opt, 2);
Sc = states(x_opt, 3);
```

Code 1: Predictive control algorithm

4.2 Simulation results from Simulink

4.2.1 Simulation with algorithm time step 25µs

Figure 4-6 to *Figure 4-8* presents some results from the simulation performed in Simulink where the sampling period for the whole system was set to 1µs, and the Mcode block containing the predictive control algorithm had a sampling of 25µs.

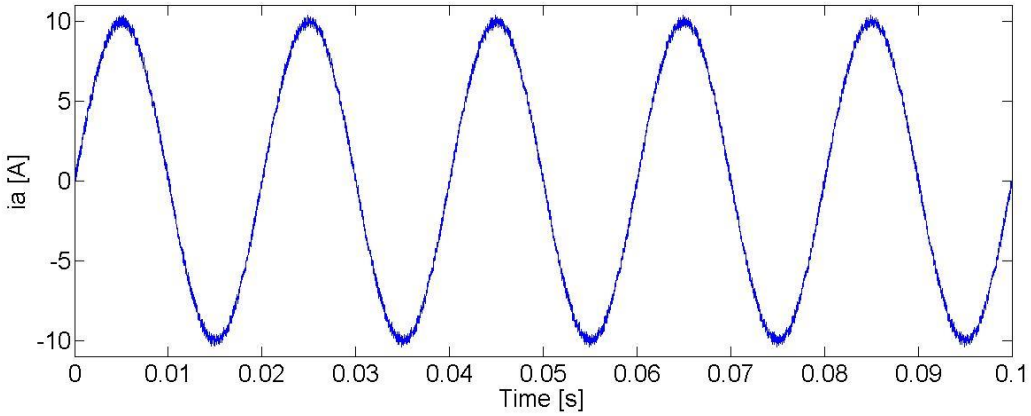

Figure 4-6: Output current, ia.

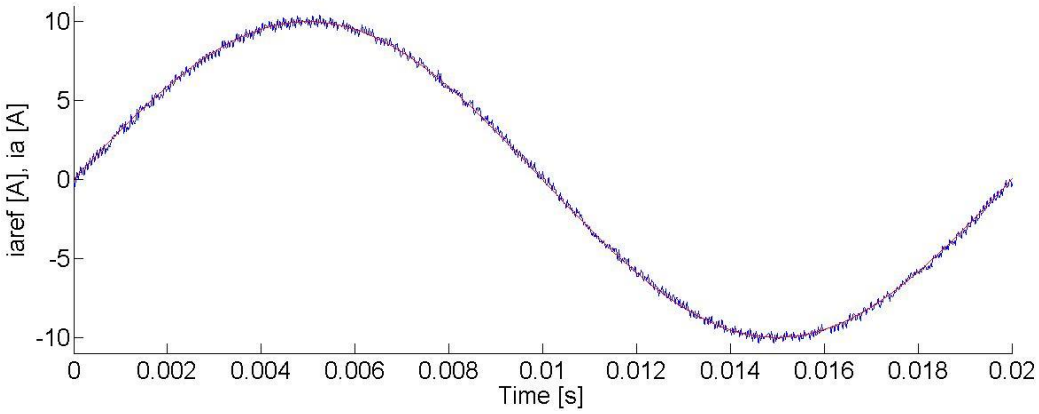

Figure 4-7: Output current ia (blue) and reference current (red).

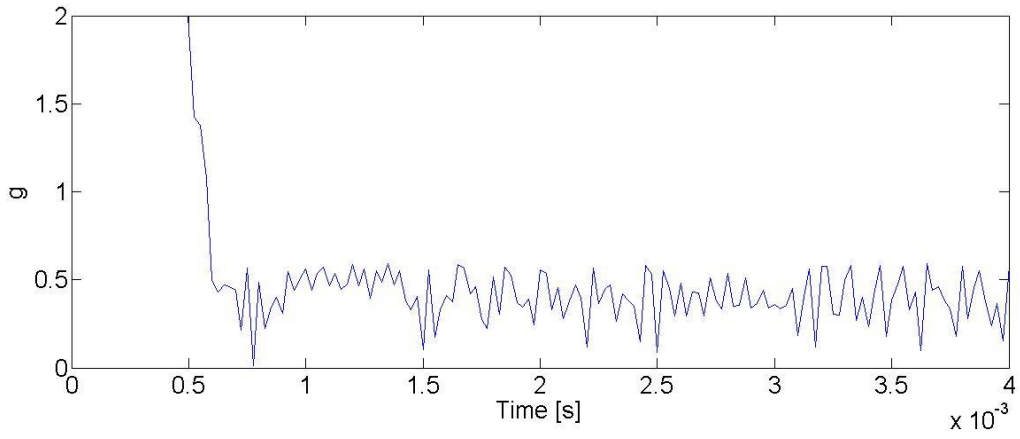

Figure 4-8: Cost function, g.

Figure 4-6 shows the output current delivered to the load, which has a sinusoidal shape with some ripple. As can be observed from Figure 4-7, a good reference tracking is accomplished. The cost function is presented in Figure 4-8. This shows the deviation between the output current and the reference in more detail. As expected, the system needs some time to stabilize, but after approximately 0.6 ms the value of g will be centered around 0.4. Appendix B contains a figure of g for a longer simulation period, to illustrate that this value stays stable.

4.2.2 Simulation with algorithm time step 1 µs

To study the effect the sampling frequency has on the performance of the controller, a simulation was executed with the sampling time for the Mcode block containing the predictive control algorithm increased to 1 µs. The results from this simulation is presented in Figure 4-9 to *Figure 4-11*.

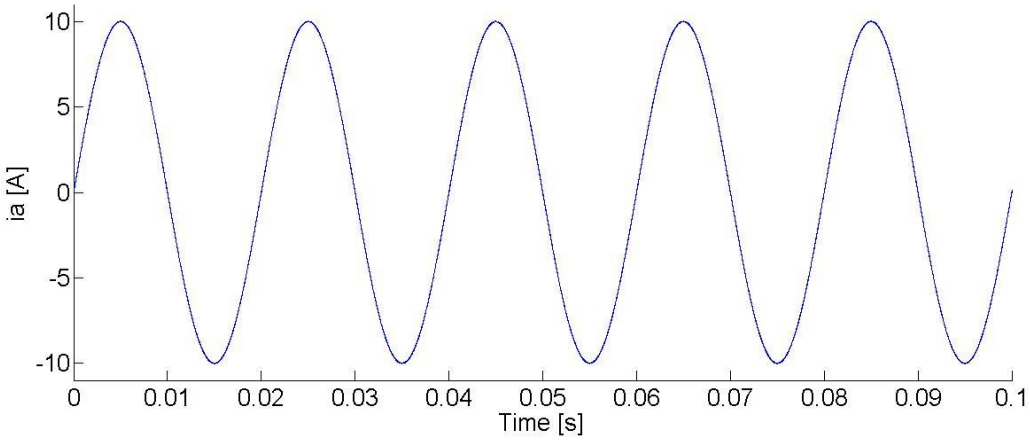

Figure 4-9: Output current.

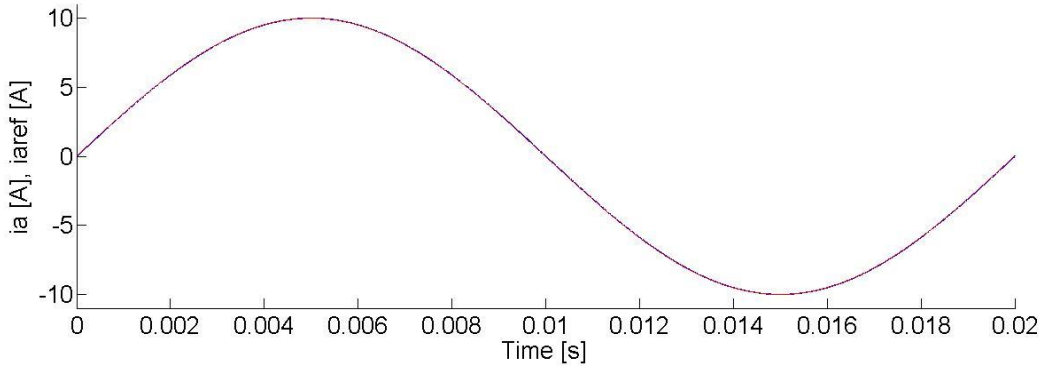

Figure 4-10 : Output current (blue) and reference current (red).

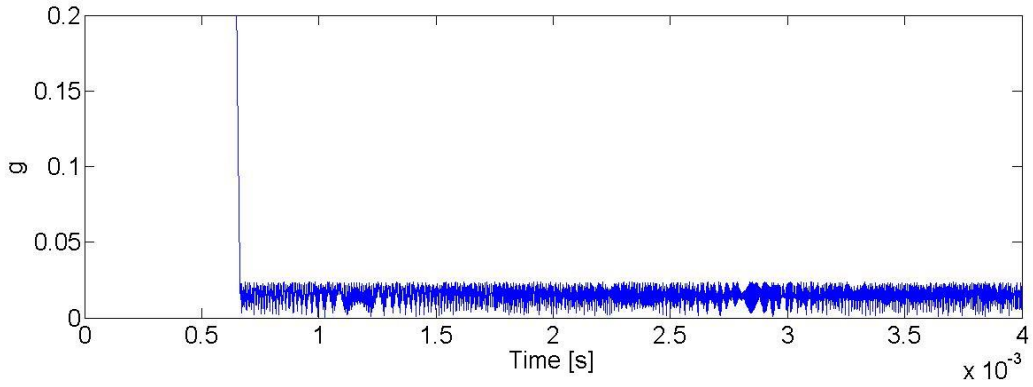

Figure 4-11: Cost function, g

Figure 4-9 shows that the ripple in the output current is reduced. The reference tracking is improved significantly, as can be observed from figure 4-10. An average value at approximately 0.015 for the cost function can be found from the plot in figure 4-11. This corresponds to the output currents deviation from the reference, which clearly demonstrates a very good performance of the control method. These results are as expected, as the calculations performed in the predictive control algorithm will work with an increased precision as the sampling frequency is raised.

This suffices as motivation for running the algorithm on an FPGA, which has a very high computing power, as the calculation of the switching states can be performed at a very high speed. However, in a physical circuit there are some factors that should be taken into consideration before deciding on a switching frequency. Increased frequency gives a better reference tracking, but will also result in higher switching losses. The next chapter will explain how the predictive control method tested in Simulink, will be adapted for the use in an experimental setup. For this purpose, switching frequency and losses will be of high importance, and an extra control objective concerning the switching frequency reduction will be included in the predictive control algorithm.

5 RT-Lab and Xilinx System Generator

5.1 Introduction to RT-Lab and XSG

The results from the simulation with Simulink show a good performance, and indicate that the predictive control can be considered a promising control method for the inverter. However, in order to draw this conclusion the method should be verified experimentally, and this will need a control system that can operate with a short calculation time to be able to follow the physical system in real-time. To solve this challenge, the software RT-Lab is used, combined with the Xilinx System Generator, with the objective to adapt the control method to be used in an experimental setup. RT-Lab is a distributed real-time platform that makes it possible to conduct real-time simulation of Simulink models with hardware in the loop. The Xilinx System Generator (RT-XSG) is a Simulink toolbox that enables engineers to generate custom, application specific models that can be implemented onto a field-programmable gate array (FPGA).

RT-Lab version 10.5.7.344 was used to build the model. Configuration of the RT-Lab software is performed on a Windows XP/Vista/7 or Red Hat linux computer called the command station. The command station is the computer that serves as the user interface. When the model is adapted to use in RT-Lab, the model is automatically coded in C, and sent to the target nodes for execution. In this case, the target nodes are parallel cores located in a real-time simulator OP5600 HILBOX produced by OPAL-RT Technologies. The simulator has six cores available, and it runs on the real-time operating system QNX 6.5. One core is reserved for the operating system. The physical connection to the inverter and FPGA will be added by introducing custom blocks to the model, which handles the interface for I/O devices.

As the predictive control algorithm requires a very short calculation time, this will be executed on the FPGA. An FPGA is a general-purpose integrated circuit that is "programmed" by the user using flash memory. This makes it possible to easily reprogram the FPGA. The FPGA is programmed by downloading a configuration file called a bitstream into static on-chip random-access memory. Using the Xilinx System Generator, a high-level

Simulink model can easily be translated to a low-level executable FPGA program. The FPGA configuration is generally described with a Hardware Description Language (HDL), and the Xilinx System Generator will compile this code, in the same way that a RTW is used to compile the C code for a Simulink model. This is a very useful tool for users who are familiar with high-level programming and Matlab/Simulink, but not with the low-level programming required to control a FPGA. However, one should have a certain knowledge about the working principle and limitations of the FPGA in order to make a program that will be functional.

An FPGA can be described as a two-dimensional array of configurable resources that can implement a wide range of arithmetic and logical operations. A great number of semiconductor devices are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects. There are some contstraints that needs to be taken into consideration when designing a FPGA program. The FPGA does not have an infinite number of resources, and the capacity depends on the FPGA used. This means that some optimization of the logic might be necessary, in order to save some resources. Another factor that needs to be taken into consideration is the timing restrictions of the FPGA. The FPGA works with synchronous logic where a clock signal synchronizes the different part of the model. This clock frequency depends on the FPGA, and will set some constraints on the time available for logic operations. Functions that can be used in Matlab will also need to be adapted to be applicable for execution on a FPGA.

5.2 Mandatory blocks for the RT-XSG model

To use RT-XSG with I/O-blocks, there are four blocks that are mandatory in the design. That is the System Generator block, the Opal-RT FPGA Synthesis Manager block, the Version block and the Hardware configuration block.

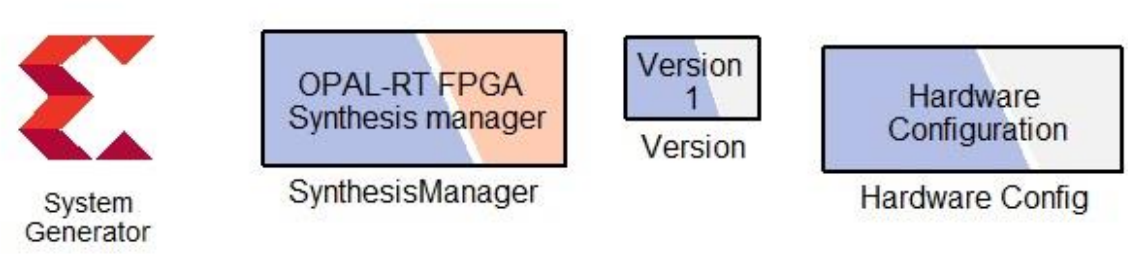

Figure 5-1: Mandatory RT-XSG blocks.

The System Generator block is mandatory for all Xilinx Simulink designs. In this block, several parameters can be set, including the type of FPGA chip that will be programmed, the FPGA language that should be used as well as the clock frequency of the design. The System Generator token is a member of the Xilinx Blocksets Basic Elements and Tools libraries. *Table 5-1* presents the parameters set for this block.

Table 5-1: Parameters for the System Generator block.

Compilation	HDL Netlist
Part	Kintex7 xc7k325t-3fbg676
Synthesis tool	XST
Hardware description language	VHDL
FPGA clock period (ns)	10
Simulink system period (sec)	1e-8

HDL Netlist is the compilation type most commonly used. In this mode, the results of the generation is a collection of HDL and EDIF files, and a few auxiliary files that simplify downstream processing. Part describes the physical FPGA type that will be used. Synthesis tool is the tool used to synthesize the design. VHDL is selected as the hardware description language. The FPGA clock period is given by the FPGA used, and is set to 10 ns. The Simulink system period needs to be the gcd, greatest common divisor, for any sampling period set to any block in the model. In this model it was set to 1e-8 sec.

The Synthesis Manager block allows to generate a programming file for the FPGA, and it is from this block that the generation process is launched. Clock frequency is set to 100 MHz, which results in a sample period of 10 ns for the FPGA. The FPGA development board used is the Xilinx ML605 PCIe-XSG development (Virtex 6 XC6VLX249T device). To generate the programming file, the "Enable partition" box should be checked, before clicking the "Generate programming file" button starts the process.

Version is the block where the name of the bitstream file that will be generated is set. When a new bitstream is generated, the name of the version should be changed in order to avoid problems with the files. This can easily be done by giving each version a number, and increasing this every time a new file is generated.

The Hardware Configuration block describes the hardware configuration of the target computer I/O module. Configuration and location of eah module should be specified in this block. A table presenting the configuration for the setup used in this work can be found in appendix C.

6 Simulation with XSG & simulated hardware

6.1 From Simulink to Xilinx

The Simulink model needs to be adapted so that it can be executed on a FPGA. That means that all Simulink blocks must be replaced with blocks from the Xilinx blockset. For experimental purposes the model of the load and the inverter will be replaced by hardware. However, in order to test the new model, the Simulink model of the load and inverter was used in the first stage of the model adaption, in order to provide calculations for the measured load current. All the other calculations should be executed on the FPGA, and is therefore rebuilt with Xilinx blocks.

6.2 Data formatting with Xilinx blocks

6.2.1 Gateway In/Out

The Xilinx Simulink blockset works with its own signal format, which is a fixed-point format. Standard Simulink signals are formatted as doubles, and so the signals needs to be converted before they can be applied to the Xilinx blocks. Xilinx provides blocks that can handle this conversion. The Gateway In block is used to change a signal from the Simulink double format, to the Xilinx fixed-point format. A Gateway Out block is used to convert the signal from fixed-point to double again. That means that every part of the model that will be executed on the FPGA should be between the Gateway In and Gateway Out blocks. In the settings for the Gateway In block, the format of the fixed-point number can be selected. By setting the format to fixed-point, the arithmetic type can be set to either Signed (Fix) or Unsigned (UFix). If the signal can be both negative and positive, the Signed format should be selected. For a signed signal, the most significant bit (MSB) will be used to indicate the sign of the signal. The number of bits, as well as the placement of the binary point is also selected in the settings. When deciding on the format there is two things that should be taken into consideration; the magnitude and the precision of the signal. To ensure that the full magnitude of the signal can be represented, there must be enough bits dedicated before the binary point. The precision that can be used for the signal is decided by the number of bits after the binary

point. For this simulation, the format is set to Fix_16_10, which means that the number is signed, contains 16 bits and has at binary point before the 10^{th} least significant bit.

6.2.2 Reinterpret

The Reinterpret block reinterprets a word from its actual format to another format specified by the user. By selecting force arithmetic type, the type can be changed to either Signed, Unsigned or floating-point. The location of the binary point can be changed, but the total number of bits stay the same.

6.2.3 Slice

If it is desirable to change the number of bits in a signal, the slice block can be used. This block extract specific bits or a range of bits from a signal. The number of bits to extract can be selected, as well as the offset from the least significant bit or most significant bit.

6.2.4 Assert

The Assert block is used to assert a rate and/or a type of signal. It can be useful in situations where designer intervention is required to resolve rates and/or types of a signal. The type can be set to either signed or unsigned, and the precision can be specified. Asserting the sample rate can also be done, either explicitly or from an input port. If the rates or type at the blocks input is not the same as the one specified, an error message will occur.

6.2.5 Convert

The Convert block converts a signal from its actual format to another format that can be specified by the user. Data types that can be selected for the output is Boolean, Fixed-point or Floating-point. If the fixed-point format is selected, the number of bits and binary point can be specified by the user. The signal can also be specified as Signed or Unsigned.

6.3 The component_values block

As previously mentioned, the model of the load and inverter will be replaced with hardware in the experimental setup. For this purpose, the model will later be adapted into the RT-Lab environment. In RT-Lab there is a console that, when the system is running, will be able to communicate with the rest of the model. This means that measured values can be monitored here, and variables can also be changed during simulation. To be able to test the effect of changing the values of V_{dc}, A and T_s during simulation, these variables will be placed in the console. In the Simulink/XSG model, the console is represented by a block named Component_values. Figure 6-1 shows the block with its outputs. Inside the system there is simply four Constant blocks, where the value can be adjusted, and the format of the signal specified.

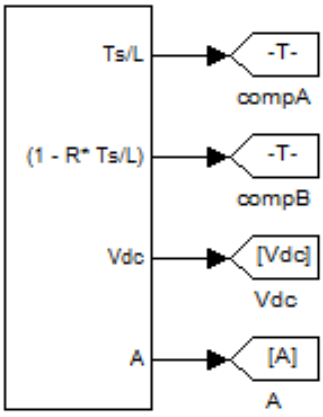

Figure 6-1: Component_values block.

6.4 Coordinate transformation

Both the reference and measured current should be transformed from abc- to αβ-coordinates before being applied to the predictive control function. The coordinate transformation subsystem is very similar to the one in the Simulink model. All Simulink blocks have been replaced with Xilinx blocks, as can be seen in Figure 6-2. The precision of the output from the blocks can be user specified, but for this model the precision was set to full, which lets the blocks use as many bits as needed to reach full precision. The latency on each gain block at the outputs, was set to one. As the input signals will be merged in the AddSub blocks, the delay on each input was adjusted to ensure that the signals for an AddSub block will reach the

input at the same time. The upper Gain block on the iref_b input has a delay of one, while the other input Gain blocks and the Delay block has the delay set to two.

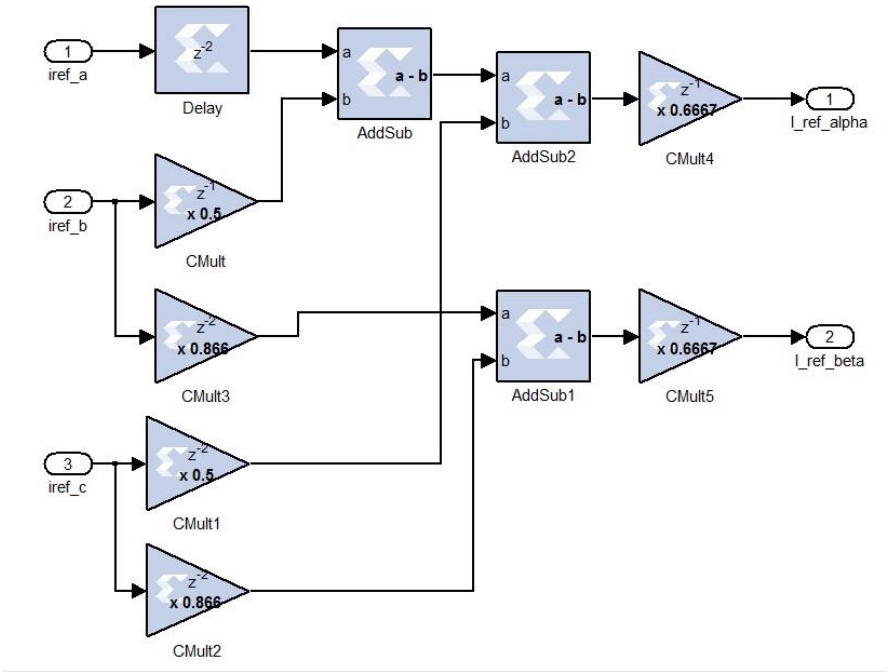

Figure 6-2: abc- to αβ-coordinates.

6.5 The Mcode block and predictive control algorithm

As previously mentioned, the aim is to execute the calculation of the predictive control algorithm on the FPGA. In order to do this, the algorithm needs to be adapted to a model consisting of Xilinx blocks. A Xilinx Mcode block is used for this purpose. The Mcode block executes a user-supplied Matlab function, and the link to the m.-file should be set as a parameter in the block. When performing a simulation with Simulink, the function is executed, and the outputs from the function is provide for the block. When hardware is generated, the code is translated in a straightforward way to give the same behaviour in Hardware Description language (HDL).

Only a limited subset of the Matlab language that is useful for implementing arithmetic functions, finite state machines and control logic is supported by the Mcode block. This causes that the algorithm used in Simulink will have to be adapted to be compatible as well.

All variables in the block should be of the Xilinx fixed-point type. Appendix D contains the m-function used to describe the predictive control algorithm. This m.-file should be kept in the same Matlab path as the block calling it.

The function takes several inputs. Measured currents both in αβ and abc coordinates, as well as the reference currents in αβ coordinates. The back-emf of the inductive load, is calculated in a separate block, divided into real and imaginary parts, and then set as inputs to the Mcode block. Variables controlling the DC-source, weighing factor A, Ts and component values for the load are fed to the function. The last inputs are the values of the voltage vector. Output variables used to evaluate the performance is sent from the output of the block, to the console. The optimal switching state, x, is also given as an output.

The first part of the code is mainly dedicated to initializing and defining different variables used in the predictions. As previously mentioned, the Xilinx block operates with fixed-point numbers. The precision for every variable needs to be defined, as well selecting between the Fix and Ufix format, depending on if the variable needs to be signed. Line 8 gives an example of this, by defining a variable, prec, that sets the precision for a selected number to 30 bits with a binary point equal to 20. The format xlSigned is selected, which corresponds to a fixed-point signed number. The Mcode block also support data of the type unsigned fixed-point, xlUnsigned and Boolean, xlBoolean.

Persistent variables works the same way in the Xilinx block, as previously explained for the Simulink model. The results in the value is being retained between function calls, as Matlab creates permanent storage for the variable. This can be used to create registers in the Mcode block. All the variables named with an ending _reg are set up as registers. To create a register the variable first needs to be initialized as persistent. This should be followed by reading from the variable memory, before finally writing a new value to the variable memory. The variables defined as registers in this code are the reference and measured current, initialized at line 17 to 25, as well as the voltage vector and back-emf, initialized at line 81 to 86. All the register values are used in calculations in the for-loop, before they are updated at the end of the code, line 278 to 303.

The majority of the calculations are placed in the for-loop stretching from line 127 to 181. An extract from the control algorithm containing the these lines is presented in Code 2. This is where the cost function for all the possible switching states are calculated. All calculations are divided into a real and an imaginary part. In line 142 to 145 the prediction for the real part of the current at instant (k+1) is calculated, as described in equation (17). Only one mathematical operation is performed at the time, as the Mcode block can not handle several operations in the same line. This is the motivation for establishing the variables n, o and p, as the values are temporarily stored before the next operation is performed. The same procedure is followed to calculate the imaginary part of the current.

g1 and g2 are the terms in the cost function representing the real and imaginary part error in current reference following. As Matlab functions such as abs() are not supported by the Mcode block, the same function is obtained by using if statements, as can be seen in line 153 to 164. The last term in the cost function, g3, represents the cost caused by switching losses, and is calculated in line 153 to 164. As the variables S_a, S_b and S_c is declared as persistent, the current state for the switches on each leg will be stored in these variables. An if statement is used to check if the switching state for the current iteration of the for loop, found from the states vector, deviates from the current state stored in S_x. If the two values are not equal, cost is added to g3. This is performed for all three inverter legs.

Code 2: For loop containing cost function calculation

```
127  %For-loop estimating the cost function for all possible
switching
128 %states
129 for i = 0:2:14
130
131    if i > 0
132    a = a + 1;
133    b = b + 3;
134    end
135
136    %i-th voltage vector for current prediction
137    v_o1_re_reg(a) = v(i);
138    v_o1_im_reg(a) = v(i+1);
139
140   %Current prediction at instant k+1, real part
```

```
141     %Calculating ik1_re = (1 - R*Ts/L)*ik_re + Ts/L*(v_o1_re -
e_re);
142     n(a) = (m * ik_re_reg);
143     o(a) = v_o1_re_reg(a) - e_re_reg;
144     p(a) = l * o_(a);
145     ik1_re(a) = n(a) + p(a);
146
147     %Current prediction at instant k+1, imaginary part
148     q(a) = (m * ik_im_reg);
149     r(a) = v_o1_im_reg(a) - e_im_reg;
150     s(a) = l * r(a);
151     ik1_im(a) = q(a) + s(a);
152
153     %Cost function
154     if ik_ref_re_reg >= ik1_re(a)
155         g1(a) = ik_ref_re_reg - ik1_re(a);
156     else
157         g1(a) = ik1_re(a) - ik_ref_re_reg;
158     end
159
160     if ik_ref_im_reg >= ik1_im(a)
161         g2(a) = ik_ref_im_reg - ik1_im(a);
162     else
163         g2(a) = ik1_im(a) - ik_ref_im_reg;
164     end
165
166     %Calculate cost switching losses
167     if Sa ~= states(b)
168         g3(a) = g3(a) + tap_a + tap_init;
169     end
170
171     if Sb ~= states(b+1)
172         g3(a) = g3(a) + tap_b + tap_init;
173     end
174
175     if Sc ~= states(b+2)
176         g3(a) = g3(a) + tap_c + tap_init;
177     end
178
179     g(a) = g1(a) + g2(a) + A* g3(a);
180
181 end
```

When the cost functions for every possible switching state is calculated, the next step is to select the optimal switching state. The section reaching from line 183 to 254 covers this, and can be found in Code 3. Initially the cost function, g_opt is set to a very high value. For every possible switching state the value of g_opt is compared to the previous value, starting with the initial value, which results in the cost function with the lowest cost being set as g_opt. The variables, S_a, S_b and S_c are set, and are used for estimating the switching cost. e_cur is

updated to contain the deviation between the reference and measured current for the given switching state.

Code 3: Extraction showing the selection of optimal values

```
183 %Selection of the optimal value
184     if (g(0)<g_opt)
185        g_opt = g(0);
186        x_opt = 0;
187        Sa = 0;
188        Sb = 0;
189        Sc = 0;
190        e_cur = g1(0) + g2(0);
191     end
```

As can be observed from the code, S_a, S_b and S_c are given constant values. For the first attempts at this section of this code, the values given to S_a, S_b and S_c were called from the states vector. An example of such a call could be:

$$Sa = states(index);$$

This resulted in problems when compiling to binary. The following error message occurred when generating the bitstream file.

Summary of Errors:

Error 0001: caught standard exception

Block: Unspecified

Details:

standard exception: XNetlistEngine:

An exception was raised:

com.xilinx.sysgen.netlist.NetlistException: The following error

was encountered while running xtclsh.

Figure 6-3: Extract from the generated error log

This problem was solved by the use of constants to set the values of the vector elements.

There are some rules that needs to be followed when using vectors in a Mcode block. The order different vector operations can be performed in is not random. All query methods of a

vector must be invoked before any update method is invocation during any simulation cycle. If this is done differently, it will result in an error message during model compilation. A method of a vector that queries a state variable is called a query method. It has a return value. Example of query methods are: v(idx), v.front, v.back, v.full, v.empty, v.length, v.maxlen. A method of a vector that changes a state variable is called an update method. An update method does not return any value. Some examples of update methods are: v(idx) = val, v.push_front(val),v.pop_front,v.push_back(val),v.pop_back, and v.push_front_pop_back(val).

As described in chapter 3.2.4, some variables are needed to evaluate the performance of the system. In line 256 to 267, the old switching states are compared to the updated ones. If there is a change in state and a switching occurs, the variable count_x is set to 1. The number of commutations can then be counted outside the the Mcode block, as the count_x variables are set as outputs. This provides the information needed to find the mean switching frequency.

The variable m is simply used to count the number of samples during the simulation time. Finding the mean current is a process of several steps. The error for the optimal switching state estimated for the current sample, is stored in the variable e_cur. Variable e_sum is defined as persistent, and can store the sum of errors for all predictions performed during the simulation time. This value is set as an output, with the aim to easily get an overview and read the value of the error during the simulation. Setting the variable e-sum directly as an output resulted in an error, due to the variable being defined as persistent. To avoid this problem a new variable, e_out was established, with the purpose to transfer the value of e_sum to the output of the Mcode block. The sum of the errors can then be divided by the number of samples to calculate the mean error for the current reference tracking.

6.6 Inputs and outputs to the Mcode

The input and output ports of the Mcode block was described in the previous chapter. This chapter will present the blocks used to adjust the timing of the signals before being fed to the Mcode, as well as the blocks on the outputs for the count_a, count_b and count_c signals. Outputs e_out and m_out are sent through a Gateway Out block to a Scope for easy access to the values. The handling of the x_opt output signal will be presented in the next chapter.

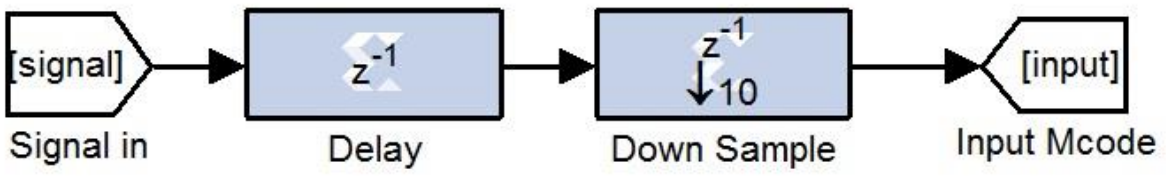

Figure 6-4: Timing blocks before Mcode input ports.

Figure 6-4 shows the two blocks placed before every input port at the Mcode block. Every input port has a delay, but in order to adjust the different signals to each others, the delay is set to different values for the input signals. The reference and measured currents spend approximately the same time before arriving to the Mcode block. One of the input values to the Mcode is the back-emf, and this is calculated in a separate subsystem placed outside the Mcode containing the algorithm. A latency of four is introduced due to the time it takes for the signal to pass through this block, and this latyency is added to the signal on every Mcode input port on all the other variables except e_re and e_im. A delay block is used for this purpose, as illustrated in *Figure 6-4*. The value indicating the optimal switching state, x_opt, is sent through a feedback loop from the output on the Mcode, and back as an input to be used in the next prediction. A Register block on the output port of the Mcode gives x_opt the initial value. For this reason, the value of the delay on all input port is increased with one.

Between every delay block and Mcode input port, there is a Down Sample block, as illustrated in *Figure 6-4*. This is a block that explicitly changes the rate of a signal by a fixed multiple that can be user specified. In this model the rate is down sampled with a factor of ten.

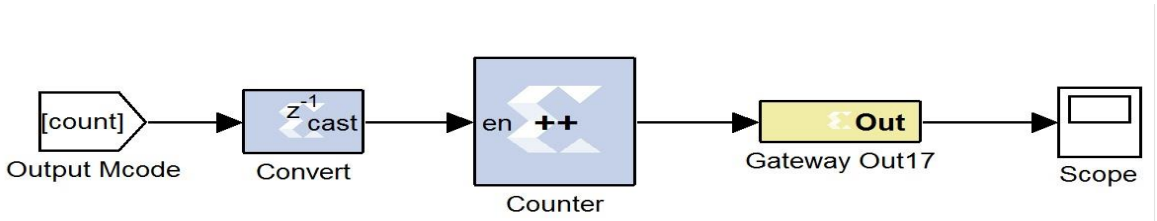

Figure 6-5: Switcing counter output.

At the output of the Mcode block, there is an Up Sample block, which increase the rate with a factor of ten, to adjust the sampling to the original value for the rest of the model. The motivation for this down sampling is to provide the Mcode block with more time to execute the calculations in the function. Models that will be executed on a FPGA have to meet certain

timing requirements, and as the execution time for the calculations for the predictive control exceeded the time limits, down sampling was used as a simple solution to solve this problem. Chapter 6.10 will explain the timing requirements and the problems faced for this algorithm in more detail.

Figure 6-5 shows an output of the Mcode block which is used for the variables count_a, count_b and count_c. The output signal from the Mcode is either 0 or 1, and indicates if there has been a change in the switching state for the given phase. A convert block is used to change the signal from fixed-point to boolean, as this is the required data format on the input of the Counter block. The Counter block is used to count how many time a switching will occur during the simulation time. By checking 'Provide enable port' for the block, the enable input port can be used to let the input signal decide when the block should count. Every time the count signal from the Mcode takes the value 1, the value on the output port of the counter increases. The user should specify the initial value, step size and number of bits available for the output, in the blocks parameters box. Here the initial value is set to 0, and the step size is 1. This signal will then be transferred through a Gateway Out block to a scope, where the switching course can be studied.

6.7 Back-emf calculation

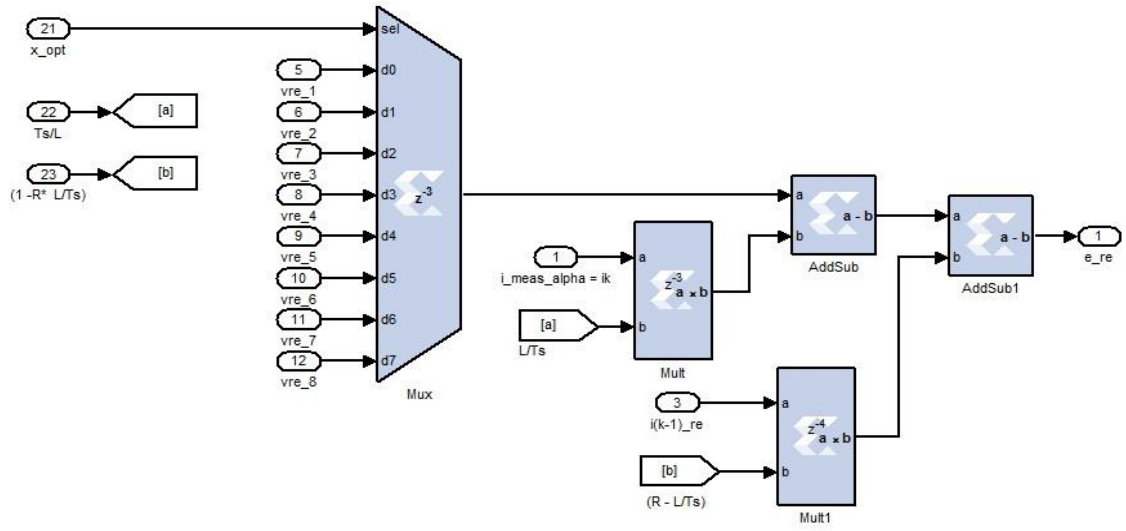

Figure 6-6: Back-emf real part calculation.

The calculations for the back-emf of the load, used in the predictive control algorithm, is placed in a separate block. Appendix E presents an overview of the contents of this block. Figure 6-6 shows the diagram for the real-part calculations of the emf. The calculations for the imaginary part is implemented the same way, with the difference being the voltage vectors at the input of the Mux. Inputs include the component values from the component_block, voltage values, the switch state signal, x_opt, as well as the currents i(k) and i(k+1).

Equation (17) giving the predicted load current, can be rearranged to give an expression for the back-emf.

$$\hat{e}(k) = \boldsymbol{v_o}(k) - \frac{L}{T_s}\ \boldsymbol{i_o}(k+1) - \left(R - \frac{L}{T_s}\right) \boldsymbol{i_o}(k) \qquad (35)$$

Considering the assumption e(k-1) ≈ e(k), the currents in the equation can be shifted one sampling back in time. io(k+1) will then correspond to the measured current at sampling k, and io(k) will correspond to the measured current at sampling (k-1). This means that the input port marked i_meas_alpha in Figure 6-6 will take the α-component of the measured current as an input. The signal feeding the input port marked i(k-1) in the figure, is the same measured current, but a Register block is used to delay it with one sampling time step before it is passed to the input port.

6.8 The states_selection block

The Mcode block gives the optimal switching state as an output, with the name x_opt. Figure 6-7 shows the model from the point where the x_opt signal has passed the Up Sample block on the Mcode output, and until it reaches the Simulink model for the inverter. A slice block is used to change the x_opt signal to a Ufix_3_0 format, which is required from the input on the states_selction block. The switching signal emerges the states_selection block as Fix_16_2 signals. In order to be able to use the communication blocks that handle the digital output communication with the FPGA when hardware is included, the signal should be of a UFix_1_0 format. A Reinterpret block is used to change the signal to unsigned, before a Slice

block extracts the one relevant bit. The signal can then be passed on to the Simulink inverter model, through a Gateway Out block.

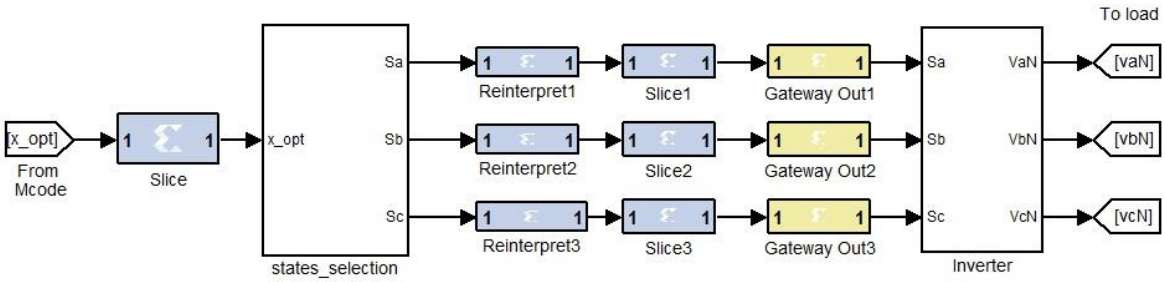

Figure 6-7: The states_selection block feeding the inverter.

Figure 6-8 presents an extract from the contents of the states_selection block. A diagram of the whole subsystem can be found in Appendix F. The figure contains the part of the subsystem generating the switching signal for phase a. Depending on x_opt, the right input of the Mux is selected and passed on to the output.

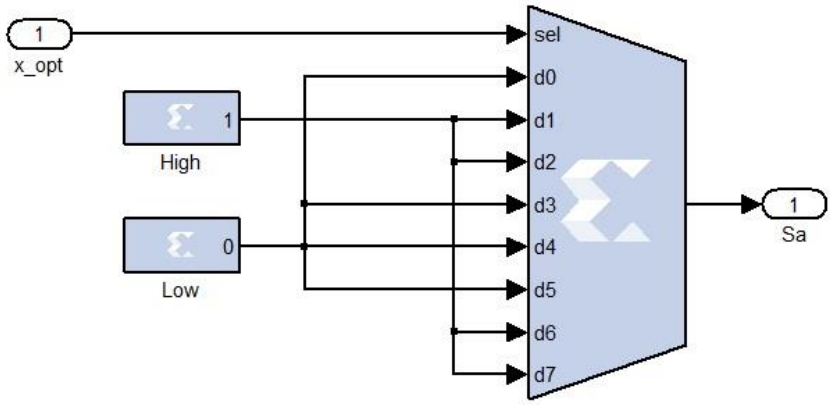

Figure 6-8: Switching signal generation for phase a.

6.9 Voltage vector generation

The Mcode-block in the Xilinx model takes the voltage values, where every voltage vector is divided into separate signals for the real and imaginary part, as inputs. By giving these values as inputs to the function, it is ensured that every part of the script in the function has access to the values. Testing reviled that if the values are set in the function itself, calling the values inside the for-loop resulted in problems. These problems appeared when the bitstream-file was generated. By looking at the VHDL-code, it was observed that when the function was calling for the voltage value, no initialization of the variable was found. This problem was solved by giving the variables as inputs to the Mcode block.

Merging all the voltage values into a vector before sending them to the output would reduce the number of output ports, and make the model more surveyable. However, the Mcode block did not accept an array as an input. The error message "An internal error occurred in the Xilinx Blockset Library" appeared when trying to run the model with an array input, hence the voltage values was set as separate outputs.

In chapter 2, Table 2-1, the eight valid switching states as well as the corresponding generated output voltage vectors for the inverter was presented. Due to the fact that the Mcode block does not handle complex numbers, the voltage vectors are divided into two variables, one for the real part, and one for the imaginary part. *Table 6-1* presents the new values and numbering for the voltage vectors, where a is the variable used in the for-loop in the Mcode to indicate the eight predictions made.

Table 6-1: Voltage values and numbering

a	**Sa**	**Sb**	**Sc**	**v_re**	**v_im**	**v_re value**	**v_im value**
0	0	0	0	Vo	V1	0	0
1	1	0	0	V2	V3	$\frac{2}{3} V_{dc}$	0
2	1	1	0	V4	V5	$\frac{1}{3} V_{dc}$	$\frac{\sqrt{3}}{3} V_{dc}$
3	0	1	0	V6	V7	$-\frac{1}{3} V_{dc}$	$\frac{\sqrt{3}}{3} V_{dc}$
4	0	1	1	V8	V9	$-\frac{2}{3} V_{dc}$	0
5	0	0	1	V10	V11	$-\frac{1}{3} V_{dc}$	$-\frac{\sqrt{3}}{3} V_{dc}$
6	1	0	1	V12	V13	$\frac{1}{3} V_{dc}$	$-\frac{\sqrt{3}}{3} V_{dc}$
7	1	1	1	V14	V15	0	0

Figure 6-9 shows how this is implemented in the model. A subsystem is created where every possible output voltage vector is calculated given the value of V_{dc}. V_{dc} is given as a constant, but in the case of a variable V_{dc}, this block should be replaced with an input connected to the rectifier system output.

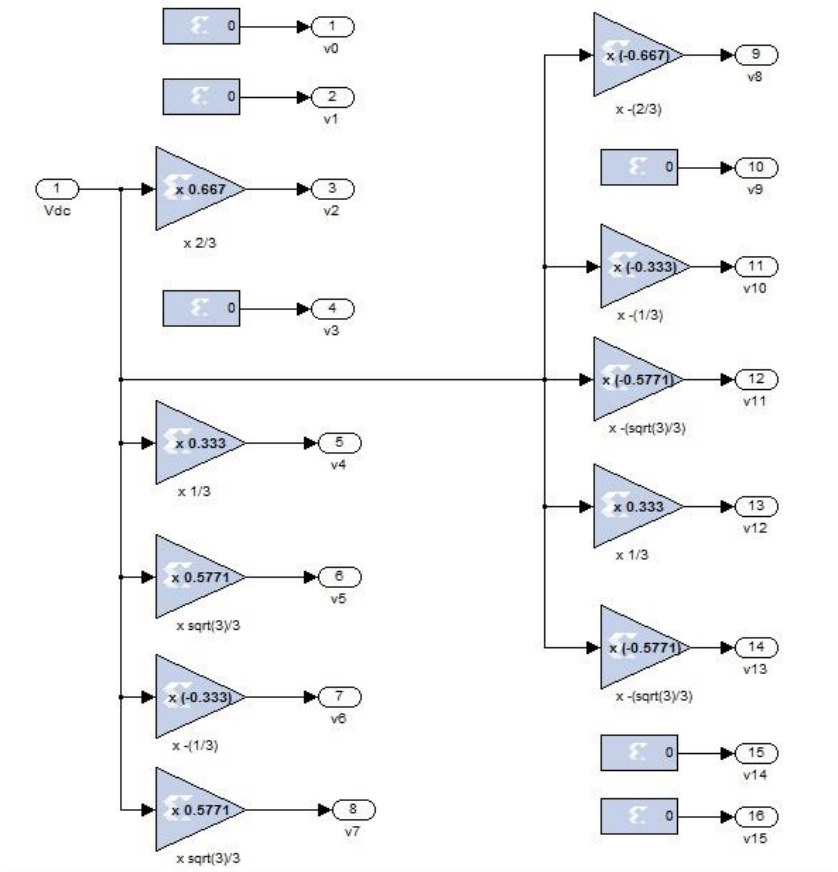

Figure 6-9: The voltage vector calculations.

6.10 Timing constraints

A problem that occurred while developing the model was the timing error. This error message reviles itself when the bitstream file is generated. Timing errors is a very common challenge faced when working with digital circuits. This model works with synchronous logic, which means that the logic is encapsulated between registers. The input signals on all registers needs to have enough time to reach a stable value before the next rising edge of the system clock. In the Xilinx model all memory elements, e.g. registers, shift registers and delays should be seen as registers. All blocks or calculations in between is defined as logic with no memory, and introduces a logic delay. This delay is due to the time it takes for the signal to perform the logical operations, and is called propagation delay. Another name used for this delay is data path delay, and it is the sum of the logic and routing delay.

In addition to the time it takes for the signal to pass all logical operations, there is some additional delay caused by the system clock. Clock path skew is caused when the clock signal

will follow different paths between the registers, and therefore can arrive to the next register at different times. It is common to have a dedicated path for distributing the clock on the FPGA, which will minimize this skew. Another delay could be caused by jitter, and will introduce a clock uncertainty due to the clock signals slightly differing. If the total delay exceeds the system clock period, a timing error will occur, and the FPGA will not be able to execute the model.

When a timing error is discovered by the system generator during compilation, an error log is presented. This error log can then be opened in the program Timing Analyzer. Timing Analyzer is a graphical user interface tool that perform static timing analysis of the FPGA design. The Timing Analyzer will be used after a model is implemented going through the faces of mapping, placing and routing the logic to adapt it to the specified FPGA board. By opening the error log in Timing Analyzer you will get an overview of the time consumption of different paths in the design. The log will specify the different delay times, as well as the source and destination of the path. An extract from an error log generated while compiling the Xilinx model can be found in Appendix G. This is meant as an example to illustrate the information found from the Timing Analyzer.

One way to handle the problem with timing errors is to insert register or delays in the data paths that has too much delay. Register and Delay blocks was placed in the Xilinx model to shorten the path, and this solved some of the timing errors. However, the algorithm in the Mcode did not meet the requirements. Registers can also be established in the code by using persistent variables, as explained in chapter 6.5. Looking in the error log in Appendix G, it can be observed that e_re_reg is established as a register, so this method seems to work. However, introducing the registers presented in the Mcode block, did not reduce the delay of the critical path sufficiently.

Analysis were made where the variables n, o, p, q, r and s were defined as registers, following the same procedure as for the emf, reference and measured currents. This did not have a noticeable effect on the measured delay. The cause for this could be that the variables never is defined as registers in the code, due to an user fault in the code structure, or it could be a

result of the variables being defined inside the Mcode block and not defined as inputs. However, the error log shows the number of logical levels in the critical path, and some small changes was observed when testing with different combinations of the previously mentioned variables set as registers. This indicates that it could be a possible solution to the problem, if the structure of the algorithm is studied in more detail. In this work, it was decided to use another solution. By using Down and Up Sample blocks, the whole Mcode block is downsampled by a factor of ten. This gives more time for the calculations in the code, and solves the problem with the algorithm not meeting the timing constraints. Arguments for choosing this solution was that the algorithm will still be running with a satisfyingly high sampling, and it saved time to proceed to work with other parts of the model.

6.11 Simulation results with XSG & simulated hardware

6.11.1 Model parameters

The simulation were executed with a time step of 1 µs for the Mcode block. *Table 6-2* presents the values set for the model. The parameters.m-file should be run to provide the load model with values.

Table 6-2:Model parameters.

Parameter	Parameter name	Value
DC-link voltage	V	520 V
Load resistance	R	10 Ω
Load inductance	L	10 mH
Load back-emf amplitude	e	100 V
Load back-emf frequency	f_e	50 Hz
Reference current amplitude	i_ref	10 A
Reference current frequency	w_ref	50 Hz
Sampling time step model	T_{sys}	10^{-8} s
Sampling time step Mcode block	T_{mcode}	10^{-6} s (Down sampled 100)
Variable Ts applied to the predictive control algorithm	T_s	10^{-6} s

6.11.2 Simulation with no switching frequency control

The first test was performed with the weighing factor, A, set to 0. This means that no consideration will be taken to the switching losses in the selection of the optimal switching state. Figure 6-10 to Figure 6-12 presents the results for the output current and the cost function.

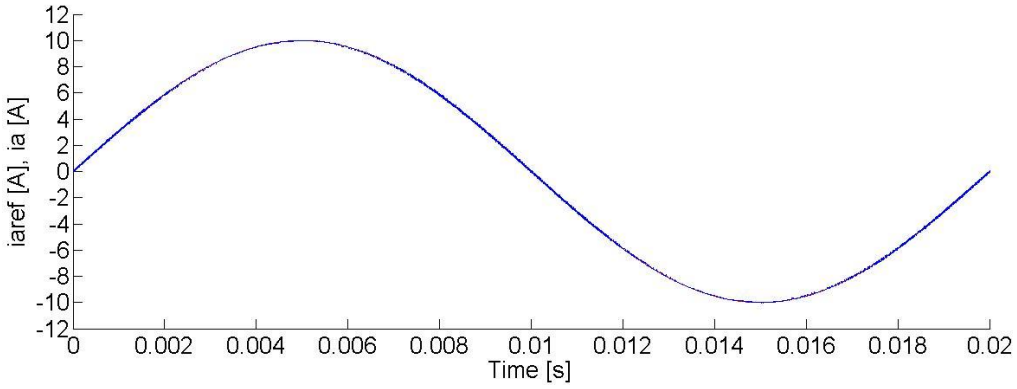

Figure 6-10 : Output current (blue) and reference current (red), phase a.

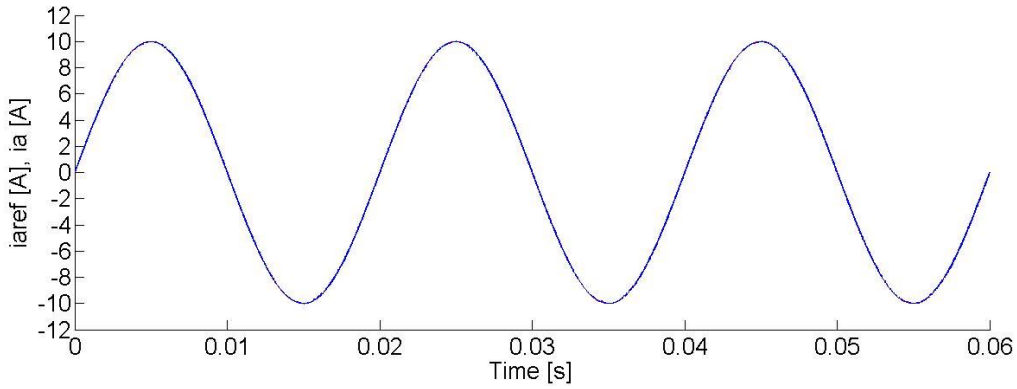

Figure 6-11: Output current (blue) and reference current (red), phase a.

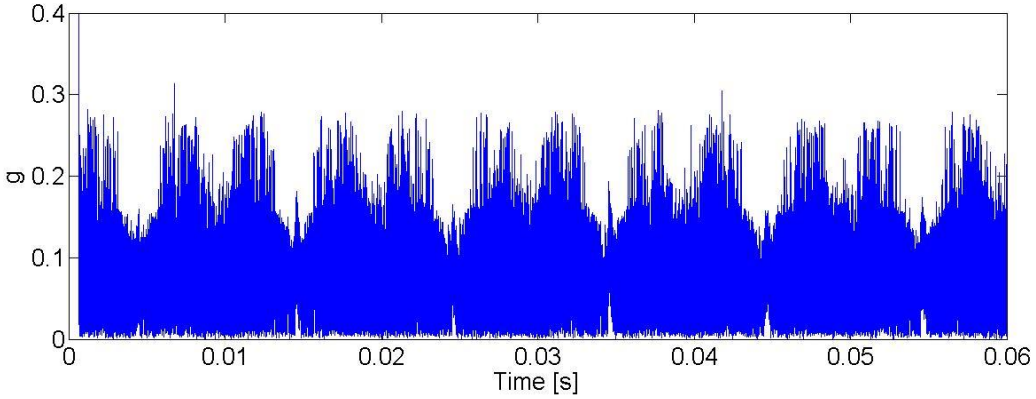

Figure 6-12: The cost function, g.

From the figures, it is observed that a very accurate reference tracking is accomplished. By comparing the cost function to the results from the current control simulation with the Simulink model, it is observed that the cost function will have a higher value. The cost function from the Simulink model, given the same sampling frequency, had a peak value of approximately 0.025. Studying Figure 6-12, the peak value is found to be around 0.3. The variations in the value of the function is also more significant in the Xilinx model. Both models works with the same parameters and sampling frequency, performing only currentcontrol of the inverter.

A reason for the deprecated performance in the Xilinx model, compared to the model in Simulink, could be that some delay is introduced due to the working principles of the FPGA. The blocks are synchronized by the clock signal, and the predictive control algorithm implemented in the Mcode block, will be translated into VHDL language, which will decide how to execute the function on a physical board. This might cause a small displacement of the signals, which can be observed when studying the cost function in detail.

Another reason for the difference between the Simulink and Xilinx model, is that the Xilinx blocks work with data of fixed point format. This restricts the precision of the data, and might affect the performance to some extent. The precision used for different data needs to be assessed, as a higher number of bits and better precision comes at the cost of more data to process, which again will require a longer calculation time.

Regardless of the deprecated performance compared to the Simulink model, the reference tracking of the Xilinx model is still very good. The performance variables for average switching frequency, f_s, and the mean reference current tracking error, $\bar{e}$, were defined in equation (24) and (25) in chapter 3.2.4. The value for $\bar{e}$ was measured to 0.1280, which indicates a small deviation between the reference and output current. However, the average switching frequency has a value of 43.8 kHz. This is not a feasible option for control of a physical circuit, as the switching time will be too short, and the switching losses will be high.

6.11.3 Simulation including switching frequency control

By simulating a system with no switching frequency control, it became clear that this needs to be included in the control system to be able to use the predictive control method on a physical circuit. *Table 6-3* presents some of the results from simulation with different values of the weighing factor, A.

Table 6-3: Simulation results for different weighing factors.

A	t [s]	fs [kHz]	$\bar{e}$
0	0.06	43.8	0.1280
0.01	0.06	25.6	0.6800
0.02	0.06	18.1	1.0775
0.03	0.06	13.2	1.4313
0.04	0.06	10.8	1.7475
0.08	0.06	6.4	2.9160

Outputs from the model includes the counter signals for each phase, where the count signal registers the change in switching state of the given inverter leg. This number should be divided by two in order to find the switching frequency, as one switching period will consist of two changes; switching on and off. The average switching frequency for phase x can then be calculated from the following equation.

$$f_{sx} = \frac{m}{2t} \tag{36}$$

Here m is the number of samples during the simulation, and t is the simulation time. By calculating the average switching frequency for each phase, and using the definition given in equation (24), the average switching frequency for the whole inverter can be found. A detailed overview of the output signals and phase switching frequency can be found in appendix H.

The simulation time, t, is 0.06 s, which correspond to three periods for the 50 Hz current. It should be taken into consideration that the simulation time will affect the results for the evaluation parameters, as the control method takes a certain time to reach a stable value for the cost function. This means that the value for the current error, which is used to find $\bar{e}$, will increase rapidly at the beginning of the simulation, and the first period of the current will deviate from the following performance to some extent. This means that by using a longer simulation time the mean reference tracking error would decrease, as would the switching frequency. As the exact value of the switching losses and the deviation from the reference will not be given too much weight in this evaluation, it was decided that a simulation time of

0.06 s was sufficient for demonstrating the influence of the adjustment of the weighing factor. What is interesting here, is the correlation between the reduction in switching frequency and the performance of the current control.

The inverter used in the experimental setup is design for a switching frequency in the range between 0 and 25 kHz. Current capacity will also depend on the switching frequency used. A higher switching frequency results in lower current capacity. This means that a weighing factor needs to be selected that keeps the switching frequency in the proper range, depending on the load current that will be expected. The value of $\bar{e}$ gives an impression of the performance of the current control, but to better illustrate the results from the control system, some plots of the current and cost function for two different values of A are presented in this chapter. Plots for simulations with the residual weighing factors from *Table 6-3* can be found in appendix I. The first plots that is present is from simulation with the weighing factor set to 0.03.

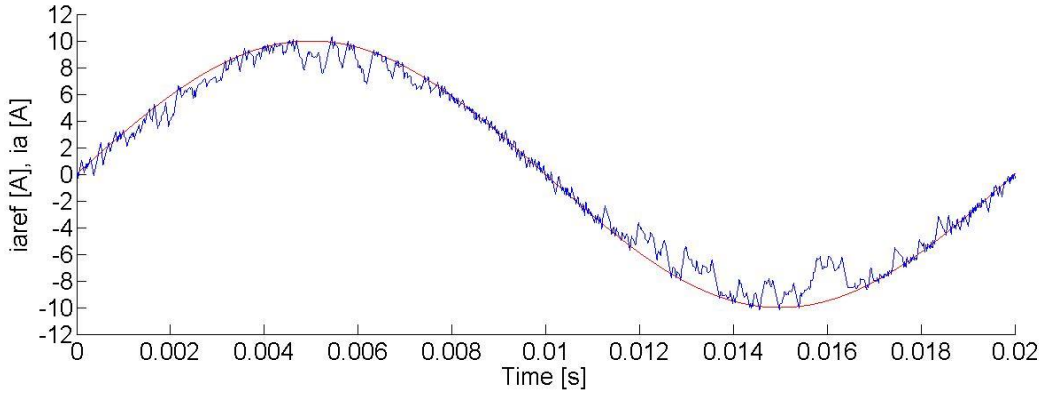

Figure 6-13: A = 0.03. Output current (blue) and reference current (red), phase a.

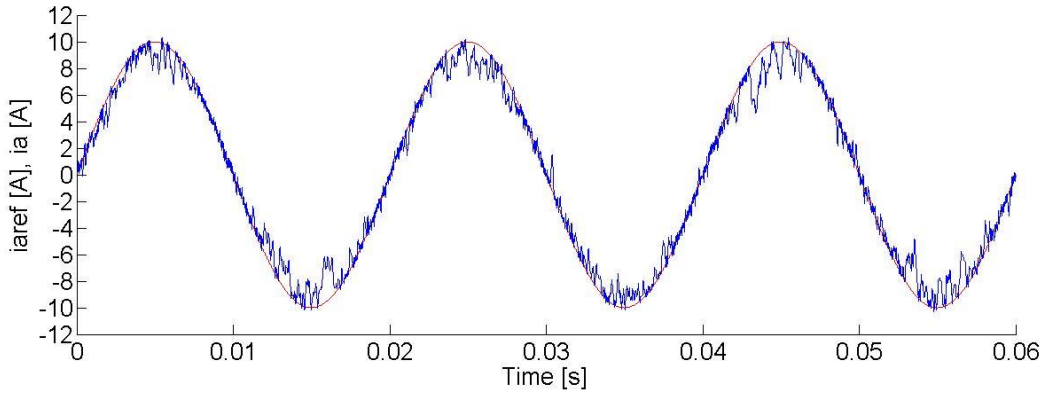

Figure 6-14:A = 0.03. Output current (blue) and reference current (red), phase a.

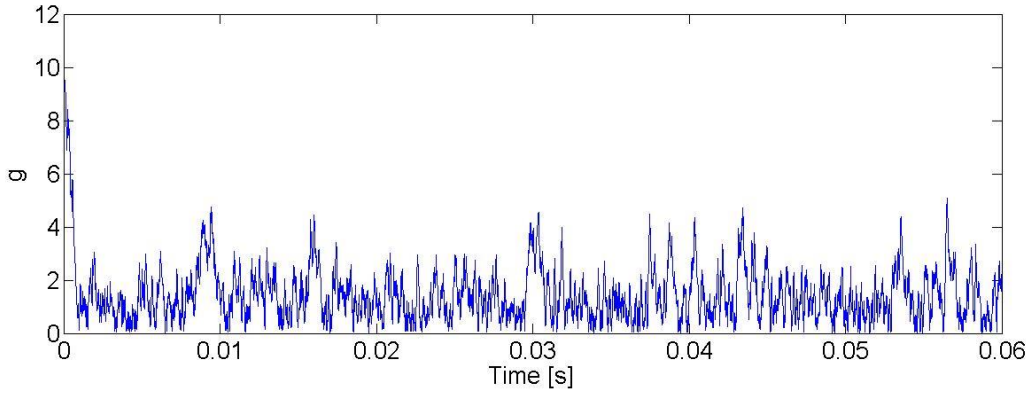

Figure 6-15: The cost function, g for A = 0.03.

It can be observed from Figure 6-13 and Figure 6-14 that the reference tracking is still good, but there is a significant reduction in the precision compare to the results with no switching frequency control. This is as expected, as the reduced switching losses comes at the cost of a reduced accuracy in the current control. Studying Figure 6-15, the cost function values is increased, which corresponds well to the increased value of $\bar{e}$. As can be observed from the figure, the average value of g will be higher than $\bar{e}$, which has a value of 1.4313 in this simulation. The difference will be equal to the term in the cost function representing the switching losses. Next, plots from the simulation with A = 0.08 will be presented, to study the effect of increasing the priority of the switching frequency reduction in the control algorithm.

Figure 6-16 and Figure 6-17 clearly shows that the current control is significantly affected by the increased weighing factor. The plot of the cost function in Figure 6-18, has taken on a much higher average value, compare to the case with A = 0.03, and the variations is much higher. The value of $\bar{e}$ has increased from 1.4313 to 2.9160, which is a 49.1 % rise of the value. However, the reduction in the switching frequency is also consequential. A switching frequency of 6.4 kHz corresponds to a 48.5 % reduction from the simulation with A = 0.03. It is clear that by including the switching frequency reduction in the predictive control algorithm, the designer can easily control the prioritizing of the current control versus the switching frequency reduction.

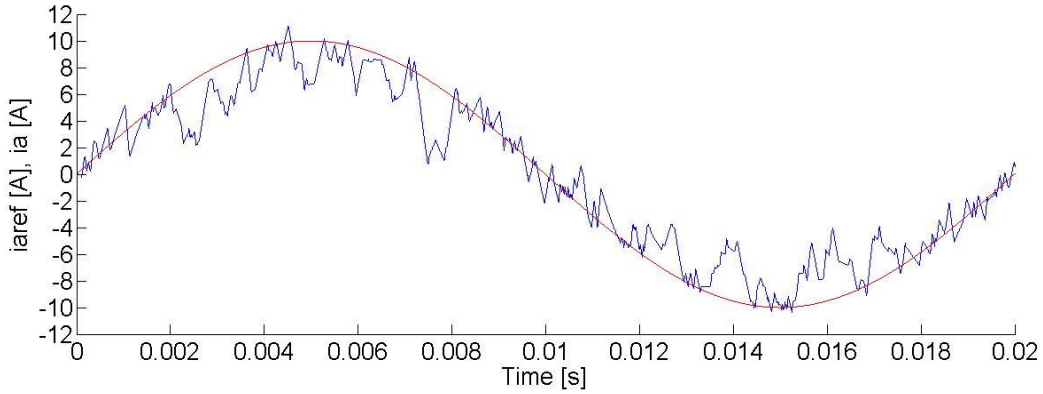

Figure 6-16: A = 0.08. Output current (blue) and reference current (red), phase a.

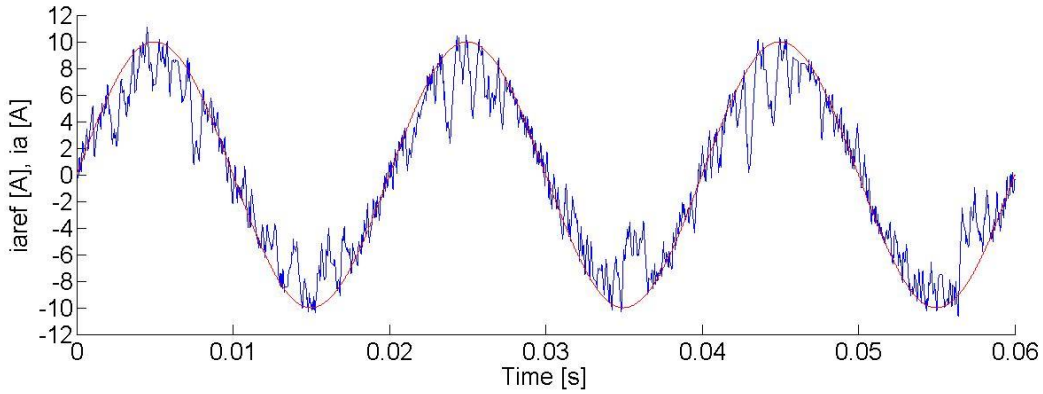

Figure 6-17: A = 0.08. Output current (blue) and reference current (red), phase a.

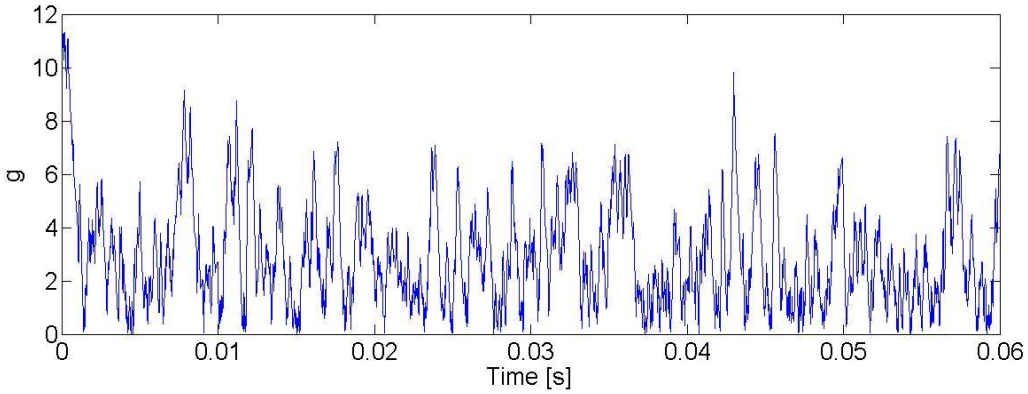

Figure 6-18: The cost function, g for A = 0.08.

In chapter 7, the adaption of the model to the RT-Lab environment, which is suitable for working with hardware connected, will be presented. In the RT-Lab model, the Component_values block will be replaced with a model that allows all the parameters set in this block to be adjusted while control system is running. This gives the possibility to adjust the weighing factor, and study the effect for many values in a short time. When testing the predictive control in an experimental setup, the designer will need to observe the performance of the system, and adjust the weighing factor to a value that gives both a satisfying current reference tracking and switching frequency. It is also expected that the performance of the predictive control will deviate from the results in these simulations, as delay will be introduced to the system.

7 RT-Lab and experimental implementation

7.1 Adapting the model to RT-Lab

In the previous chapter a model was developed using Xilinx blocks, which makes it possible to generate the necessary files for running the model on an inverter. Simulink models for the load and inverter were used to find the output current, and a block named Component_values was used as a user interface to adjust key variables. In order to replace the simulated load and inverter with hardware, the model needs to be adapted to contain some means of communication with the hardware. The physical connection to the inverter and FPGA will be added by introducing custom blocks to the model, which handles the interface for I/O devices. In the previous model, the Component_values block was used to specify variables. This should be adapted to a system that can communicate with the FPGA while the simulation is running, in order to easily adjust variables. It is also desirable to have access to the measurements of output current and the performance variables, in order to study the behaviour of the system. The software RT-Lab, which was introduced in chapter 5 makes this setup possible.

When adapting a Simulink model to a RT-Lab model, there are two main tasks that must be carried out. Dividing the model into subsystems and inserting OpComm communication blocks. When the model is divided into subsystems, there are some procedures that need to be followed. A console must be made, and this subsystem's name must start with the prefix SC_. The console works as the user interface for a simulation. It is used to display the acquired signals, and gives the possibility to set control signals during the simulation. The console can be operated from the command station. This subsystem will replace the Component_values block, as well as the Scopes used for monitoring the signals in the Simulink/Xilinx model.

Every model also needs a subsystem called the master, with a name starting with the prefix SM_. The master is responsible for the overall synchronization of the network, and there can be only one master per model. In addition to the master, several slave subsystems can be added to the model to distribute the calculations over several cores in the simulator. In this

model most calculations is performed on the FPGA, and only one core is used in the simulator. The main task of the master subsystem in the RT-Lab model, will be the communication between the hardware and FPGA, to the control station and console.

7.2 The console

The first block that should be included in a RT-Lab model is he OpComm block. No signal can enter a subsystem without going through an OpComm block first. The block serves a number of purposes:

- Provides RT-Lab with information about the size and type of the data received.
- Defines acquisition groups and parameters.
- The block waits until all of its inputs are updated before updating its outputs.
- Specifies the sample time for the subsystem where it is placed.
- Specifies the communication sample time of a calculation subsystem with another calculation subsystem.

It should be noted that the communication between the console and the target nodes is not in real-time. If several subsystems were to be executed on the simulator, each subsystem would need one OpComm block for the real-time communication between the cores, and one OpComm block for the communication with the console.

The OpComm block in the console for this model takes the three measured phase currents, the counter signal for phase a, b, and c, the cost function, g, the evaluation parameter, e_cur and the sample counter, m, as inputs. These input signals are passed through the OpComm block, and sent to Scopes where the values can be monitored and saved to the workspace for plotting. Outputs from the console includes the three phase reference currents, the DC-source voltage, V_{dc}, the component values given as (Ts/L) and R, as well as the weighing factor, A. While the control method is running these variables can be adjusted. This will be very time saving compared to running simulations in Simulink, as many values and combinations can be tested for key variables in real time, and the effect can be studied directly. An overview of the blocks used for sending and receiving signals in the console can be found in appendix J.

In addition to the variables used in the control method, two more signals should be generated from the console. The inverter needs an enable signal, and the I/O interface card between the simulator/FPGA and the inverter needs a watchdog signal. Generating these two signals in the console provides an easy way to control the inverter operation for the user. Figure 7-1 presents the system that creates the two signals.

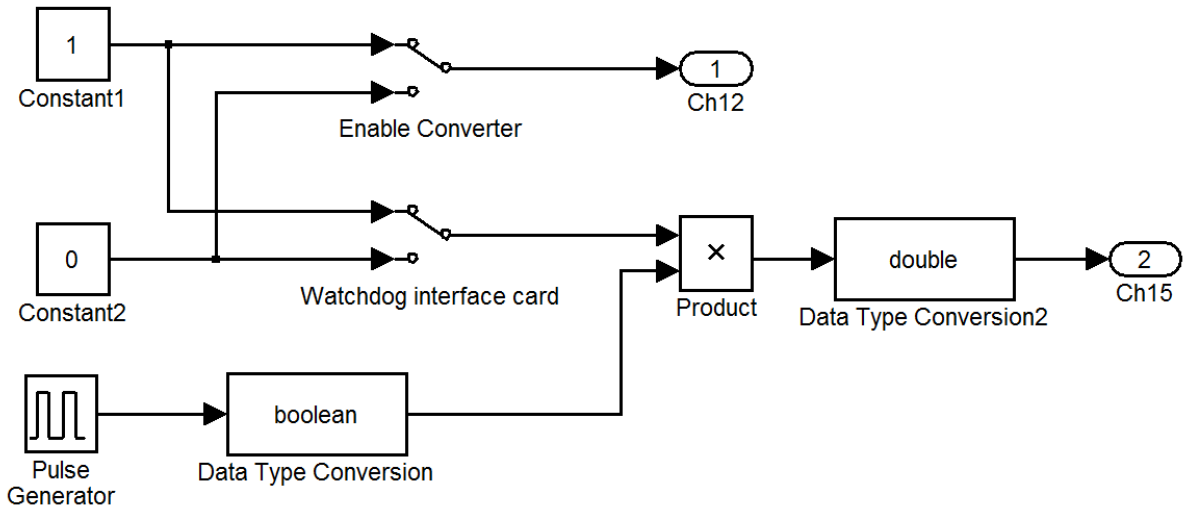

Figure 7-1: Generation of enable and watchdog signal.

7.3 The Master

Communication between the FPGA and the console is done through the master subsystem in RT-Lab. Appendix K presents figures of the contents of this model. Globally, RT-Lab and RT-XSG are using DataOut Recv, DataIn Send, DataIn and DataOut blocks to allow communication between the CPU and the FPGA model. The first block that is needed is the OpCtrlML605EX1, presented in Figure 7-2. This block controls the programming of one ML605 card, its initialization and the selection of the hardware synchronization mode of the card. It also enables binding of Send and Receive and I/O blocks to that specific card. In the parameters for this block, the bitstream filename that will be applied to the FPGA needs to be specified. This is the .bin-file that was generated with the Xilinx Synthesis Manager block. The file needs to be put in the same directory as the RT-Lab model of the console.

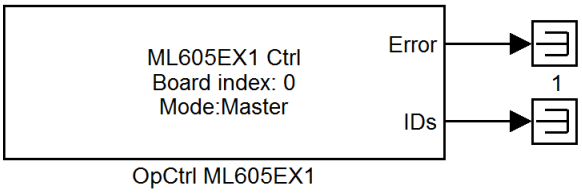

Figure 7-2: The OpCtrlML605EX1 block.

Send and receive blocks placed in the master, is of the type “ML605EX1 DataIn Send” and “ML606EX1 DataOut Recv”. Input values from the console passes through an OpComm block, before it is sent to a DataIn Send block. The DataIn Send block communicates with a DataIn block in the FPGA model. Both doubles and uint32 data format is accepted as input, and can be selected in the parameter settings for the block. All the input signals has the double format initially, but only the enable and watchdog signal are sent to the DataIn Send blocks in this format. The rest of the signals are converted to the uint32 format by using a Rescale Double to Integer block. In this block the total number of bits, binary point and type of numerical format can be specified. This is done in order to get control over the signal, in order to transfer it back to the right value after it is transferred to the DataIn block in the FPGA model. The DataIn block performs data conversion from uint32 to the System Generator UFix33_0 data format. From this format the desired data needs to be extracted from the 32 least significant bits, binary point needs to be defined if necessary, and the sign of the signal needs to be defined.

A simple calculation is added in the master subsystem, to calculate the output signal (1-R*Ts/L). The implementation of the calculations can be seen in Figure 7-3 where the input signals are received from the OpComm block, and the output signal is sent through a Rescale block before the DataIn Send block.

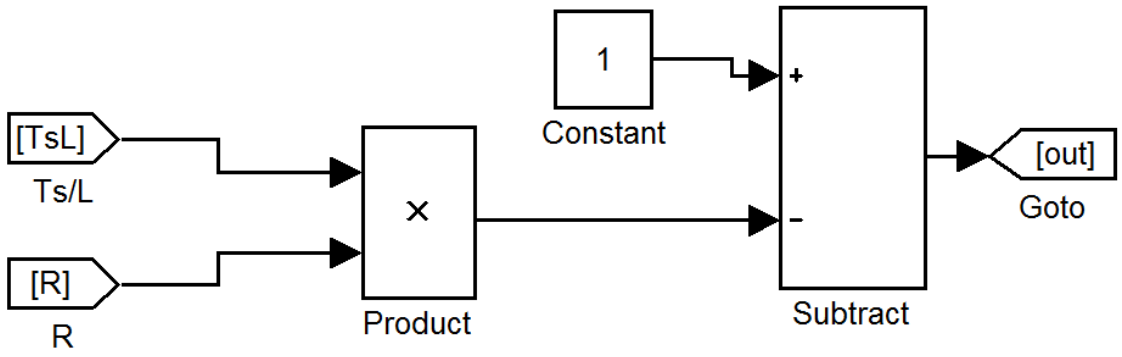

Figure 7-3: Calculations in the master subsystem.

The measured current, counter signals for the switches, the cost function and the error evaluation parameters are set as inputs to the master subsystem, in order to transfer these signals to the console for monitoring and saving. This is done by the use of DataOut Recv blocks. The DataOut Recv blocks gets their data input from the DataOut block placed in the FPGA model. Output data from the block can be selected to be either double or uint32. Figure *7-4* shows an example of one of the DataOut Recv blocks in the model, and the data handling before it is sent to the console. For an overview of all the DataOut Recv blocks in the model, see appendix K.

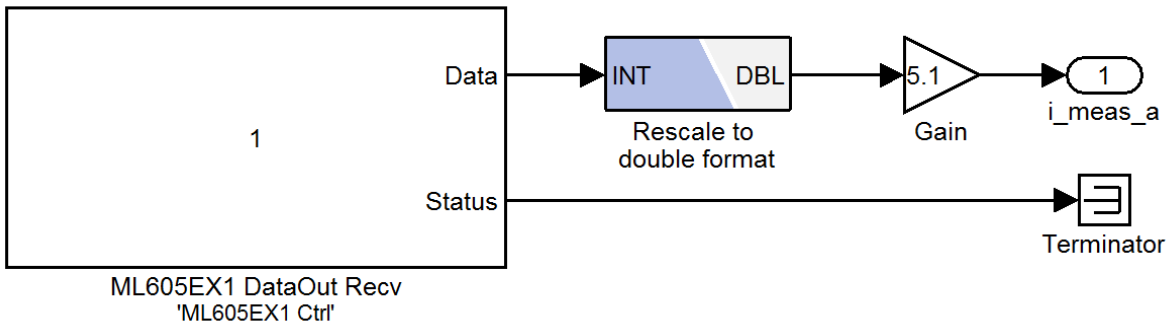

Figure 7-4 : DataOut Recv block with rescaling.

The handling of the data after it is collected by the DataOut Recv block, depends on the format of the signal at the input of the DataOut block sending the values. Figure *7-4* presented an example, in this case the measured current of phase a. Here the signal originally

has the format Fix_16_10 before being sent to the DataOut block, and therefor this bitcombination is used in the parameters for the Rescale block, converting it back to the right range. The current measurements are done with LEM sensors, and for this reason needs to be scaled to achieve the exact value. Using a test circuit to find the right scaling factor for the equipment used in the experimental setup, as will be explained in chapter 7.7, the gain factor is set to 5.1.

7.4 Communication blocks in the FPGA model

7.4.1 The DataIn block

The FPGA model contains two blocks for communication with the console and two blocks for communication with the inverter through I/O ports for analog in and digital out signals. The DataOut block send data to the DataOut Recv block in the master subsystem, before they are transferred to the console. Inputs to the DataOut block consists of the three measured phase currents, count signals for the three phases, the cost function value, g, the evaluation parameter e_cur and the variable m, which gives the number of samples for the Mcode block. An overview of all the input signals and the data formatting performed before the input port to the DataIn block can be found in appendix L. Figure 7-5 presents a DataIn block with only the three measured phase currents as inputs.

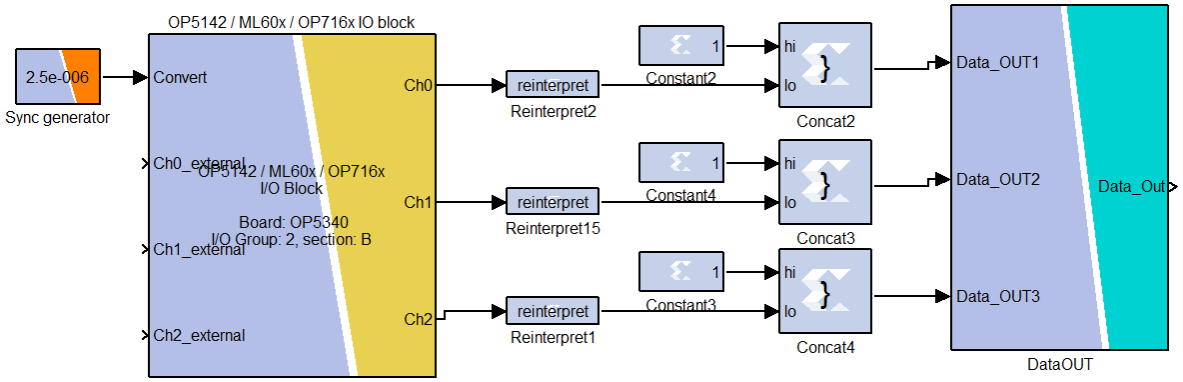

Figure 7-5: Analog Input and DataIn blocks.

The input ports of the DataOut ports, marked Data_OUT, is in the UFix_33_0 format. One bit is needed to set the most significant bit, indicating that the data is ready to be transferred, and can be seen as a write signal to the buffer. The buffer is emptied and transferred to the CPU model at the beginning of each calculation step. A Constant block is used to set the first bit to 1, and the Concat block merges this bit with the rest of the signal sent to the input. The binary point needs to be removed from every signal before the DataOut block. For the measured currents, the data arrives from the Analog Input block in the format Fix_16_10. A Reinterpret block is used to change the arithmetic type to Unsigned and set the binary point to position 0. The data is then sent to the input in the format UFix_16_0.

The rest of the input signals goes through a similar data reformatting in order to ensure the right format at the input ports. All three count signals pass through an Assert block that specifies that the output signal should be of the type UFix_16_0. g and e_cur pass through a Slice block, where 16 bits are extracted. The offset of the slicing needs to be adjusted so that ten bits after the binary point and six bits before the binary point are extracted. This gives the format UFix_16_0 at the outputs on the Slice blocks. The samples signal, m, pass through an Assert block, and is sent to the input port with the type UFix_32_0. The DataOUT port of the DataOut block is used for offline simulation only, and does not need to be connected.

7.4.2 The analog input block

In addition to the DataIn block, Figure *7-5* also presented the Analog Input block used to receive the measured currents from the interface card between the FPGA and the inverter. This is a Xilinx block of the type OP5142/ML605/OP716x I/O. This block gives access to all I/O modules controlled by the OP5142 board, ML605 development board and the OP716x board. The Hardware Synchronization block, introduced in chapter 5.2 determines all the interface modules available , given the requested signal type and direction. The signal type can be set to analog or digital, and the direction is either input or output. Input ports marked with Chx_External, can be used for offline simulation. It should be noted that the blocks input and output ports depend on the interface board selected.

Parameters set in the block properties includes the type of I/O, which in this case is set to Analog Input. The hardware interface selection is specific for the interface board, and is set to "Slot #2, section B". Three channels are selected.

On the input marked convert, a Synch Generator block is connected. This block generates a synchronization pulse train with the specified period. The width of the pulse equals the FPGA board clock period, and gives an output signal of the type unsigned integer with the value 1. A period of 2.5 µs is selected. The analog outputs deliver signals received from the analog-to-digital conversion module. Data at the output ports are in the format Fix_16_10, which provides a dynamic range of [-16, 15.9995] and a resolution of 0.0005V.

7.4.3 The digital output block

The block used for the digital output communication is of the same type as the one used for analog input, but for this purpose with the I/O type selection set to Digital Output. Hardware interface selection is set to "Front Panel Slot #4, Section B". This block handles the communication between the FPGA and the interface board that delivers the switching signals to the inverter. Input data to this block must be of the type UFix_1_0. For the switching signals arriving form the States_selection block in the model, this is achieved by first passing the data through a Reinterpret block to change the format to unsigned. A Slice block is then used to extract the one relevant bit. *Table 7-1* gives an overview of which input port channel the different signals should be connected to. This was decided through running a test model in the lab. The rest of the input ports are not in use, and are given a constant input of 0.

Table 7-1: Input ports of the Digital Out block

Channel	Signal
0	S_a
1	S_b
2	S_c
3	S_a inverted
4	S_b inverted

5	S_c inverted
12	Enable
15	Watchdog

7.4.4 The DataIn block

The DataIn block receives data input from the DataIn Send block in the CPU model. This block performs data conversion from the type uint32 to the System Generator format UFix_33_0. The data needs to be reformatted to the desired format for use in the FPGA model. Chapter 7.3 described how the data format was specified before being sent to the DataIn Send block.

With the exception of the enable and watchdog signal, every input signal passes through the same blocks for data reformatting. The specific number of bits and binary point position depends on what was specified in the CPU model. Figure 7-6 shows an example of how the reference current for phase a is implemented.

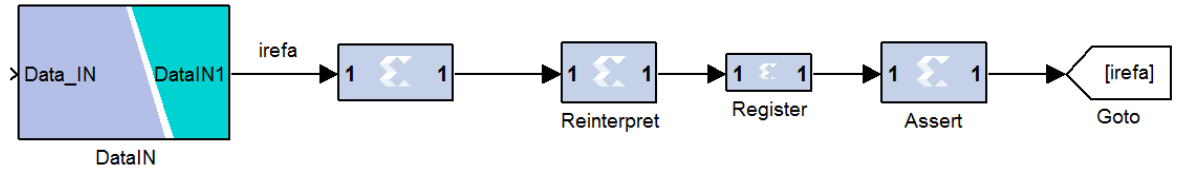

Figure 7-6: DataIn block and output signal .

The output signal from the DataIn block is in the UFix_33_0 format. A Slice block is used to extract the sixteen relevante bits, and delivers a UFix_16_0 signal. By using a Reinterpret block, the number of fractional bits is set to ten, and the format is changed to signed. This results in a Fix_16_10 signal. The Register block is applied to every input for the model, to provide a delay. Sample rate is set by the Assert block, to make sure that this is specified before the signal is sent to the Mcode block. The same procedure is implemented for the other input signals, and a figure giving an overview of this is presented in appendix L.

7.5 Experimental setup

The aim of the project was to test the predictive control in an experimental setup. As previously mentioned, a real-time simulator of the type OP5600 HILBOX produced by OPAL-RT Technologies, combined with a FPGA development board of the type Xilinx ML605 PCIe-XSG development (Virtex 6 XC6VLX249T device), is used for the control system. A 20 kW IGBT inverter will be the control objective. The load is a three phase symmetrical, star connected, resistive inductive load, as described in Figure 2-1. For the tests performed, three resistors of 10 Ω in combination with inductances of 1.4 mH were used. An adjustable DC-source delivered the power. Between the digital output from the FPGA and the inverter, a test card was inserted. This test card has light emitting diodes mounted on it, which will indicate when the enable and gating signals for the switches is high. There is also a diode dedicated to indication of the status received from the inverter. If this signal is high, the inverter is in operating mode. If an error occurs, the status signal will be low, and a display on the inverter will give an error message.

Before the predictive control was tested on the circuit, some test models were made with the aim to establish working communication links between the equipment. Chapter 7.6 will present a model used for testing the digital output from the FPGA. The analog input was tested with the model presented in chapter 7.7. This model was also used to adjust the gain on the input signal, to be in accordance with the transformation ratio of the LEM sensors and wiring used for output current measurements.

7.6 Digital output test

The digital output block used in the FPGA model has 32 possible output channels to select from when the output signals are routed. A simple model was made, where the console contained a constant for every output channel to every output. This way the output to every channel for the digital output block could be changed while the same program was loaded onto the FPGA. The values were sent to the master subsystem, running on the simulator, where the communication with the FPGA is established. Figure *7-7* is used to illustrate the structure of the model in the master subsystem. In this figure, only one of the channels are included, but the same procedure is used for all of the 32 channels. The input signal is passed

through an OpComm block, before it is sent to the FPGA model through a DataIn Send block. Linking the CPU model to the FPGA model is done with the use of a OpCtrlML605EX1 block. The DataOut Recv block receive data from the FPGA model containing the signals sent to the digital outputs. These blocks are not necessary for the model to work, but were included to test that the link between the FPGA model and console were working properly.

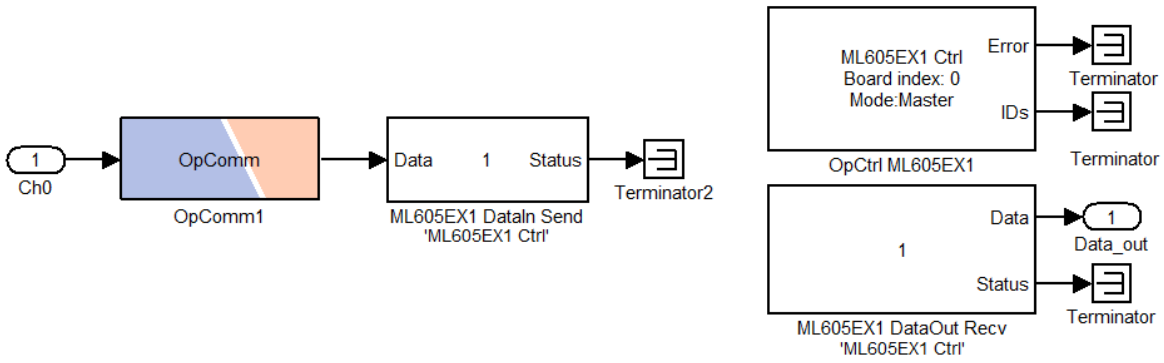

Figure 7-7: Communication blocks in the CPU model.

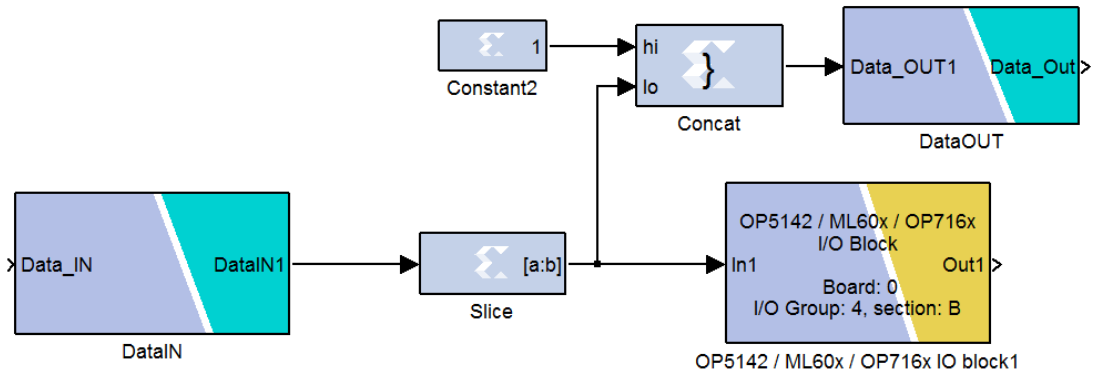

Figure 7-8 : FPGA model for one output channel.

Figure *7-8* presents an illustration of how the FPGA model would look for one output channel. The same procedure was followed in the test, but with the number of output channels increased to 32. Data is received from the CPU model through the DataIn block. A Slice block is used to extract the one relevant bit, and to adjust the data format to UFix_1_0, which

is required for the input of the digital output block. After concatenation with a constant, the signal is sent back to the console via the master, through a DataOut block. In the console this signal is passed through an OpComm block, before being displayed on a Scope. The digital output block sends the signal from the FPGA to the inverter, via the test card.

A simple method for finding the right channels to connect the gating and enabling signals, is to set one output channel at the time high in the console, while monitoring the test card to see when an indication light turn on. *Table 7-2* presents the results for this test. The output current can then be studied to ensure that the IGBTs are actually receiving the signal and switching. A model to measure the output current is presented in the next chapter.

Table 7-2: Output channels for the digital output block.

Signal	Channel
S_{a+}	0
S_{a-}	1
S_{b+}	2
S_{b-}	3
S_{c+}	4
S_{c-}	5
Enable	12
Watchdog	15

7.7 Analog input test and adjustment

The analog input block receives the measured output load current of the inverter. The wiring from the inverter to the load, passes through a LEM current transducer, which is used for electronic measurement of currents. This sensor measures the current with a high accuracy, but the output value is scaled, depending on the conversion ratio for the LEM sensor, and also on the wiring. The cable from the inverter to the load was arranged to make three loops

through the sensors measuring area, in order to increase the value of the measured signal. When the current is measured, the data are sent to the analog input block at the FPGA.

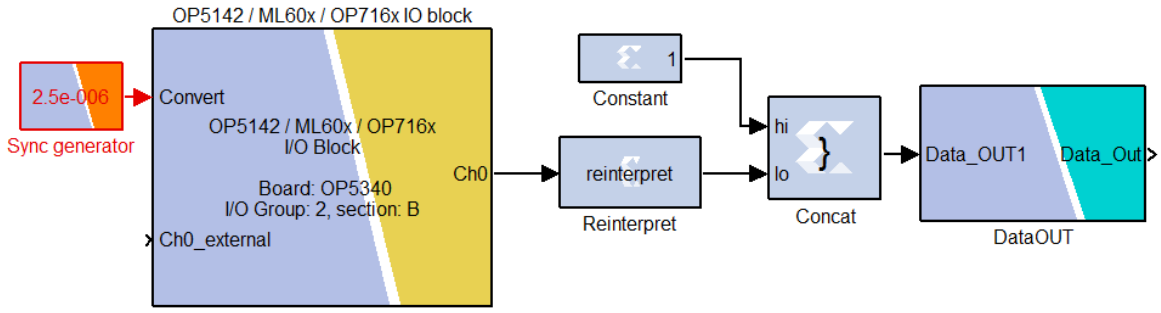

Figure 7-9 : Analog input and DataOut in the FPGA model.

Figure *7-9* presents the transmission of the signal for phase a, from the analog input block in the FPGA model, to the CPU model. The Reinterpret block changes the data format to Ufix_16_0. Digital output in the FPGA model was explained in the previous chapter, and the FPGA model also contains a DataIn and a digital output block, where the data is reinterpreted to the UFix_1_0 format before being sent to the inverter.

The model for the master system in the CPU model works the same way as explained in the previous chapter, with the three measured phase currents as inputs, and the switching signals, enable and watchdog as outputs. In chapter 7.2, the generation of the enable and watchdog signal was demonstrated, and the same system is applied in the console for this model. Figure *7-10* shows the other blocks included in the console, presented for one phase.

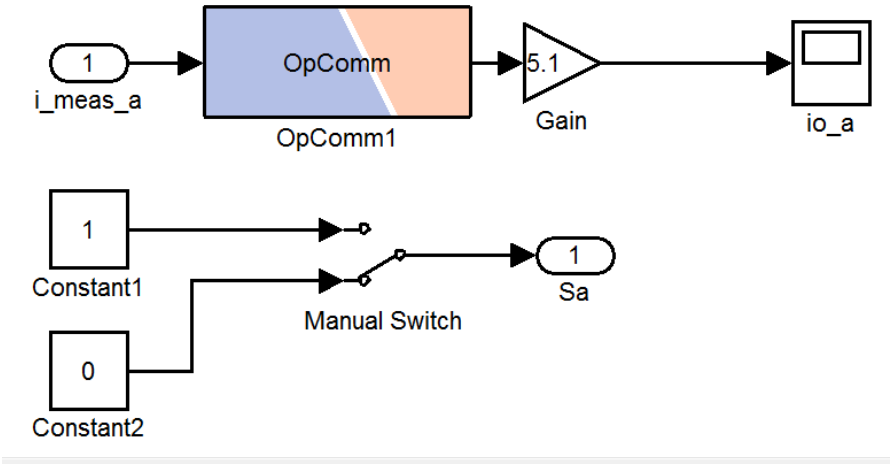

Figure 7-10: Console for one phase.

The manual switch controls the switching signal to one of the phases, in this case phase a. By setting the switch for phase a high, and the switches for phase b and c low, a positive direct current will flow from phase leg a in the inverter, to the load. The value of the current can be found from the display on the variable DC-source. For accuracy, the current was also measured by using a current clamp combined with an oscilloscope. When the real value of what the output current in the physical circuit is known, the gain block placed after the OpComm block in the console can be adjusted. When the value of the real current, and the current displayed in the Scope in the console show the same value, the right scaling factor is found. For this setup, a factor of 5.1 gave the right scaling. This scaling factor should also be added in the FPGA model after the analog input block, to attain the right current value for the predictive control algorithm.

A current clamp combined with an oscilloscope was used to measure the load current. The simulator also provides the possibility to connect an oscilloscope to the analog inputs, which was used for observing the measured load currents from the LEM sensors. As previously mentioned, the communication between the master subsystem running on the simulator, and the console does not happen in real-time. This becomes very clear when the measured currents are observed by the use of Scopes in the console. A great amount of data is lost, and the scopes will start to interpolate to fill in for the data loss. This problem can be appeased by adjusting some settings in the Scopes used, as well as the Probe Control Panel that can be opened from RT-Lab.

For the Scopes used in the model, the decimation factor can be adjusted. The Probe Control Panel can be launch using the **Tools > Probe Control** menu or using the toolbar in RT-Lab.

It enables you to specify the acquisition parameters for a given acquisition group. The acquisition group is defined in the OpComm blocks used in the model. For the model used in this project, consisting of a console and only one subsystem, the master, there will only be one acquisition group. Some of the acquisition and transmission parameters that can be adjusted are:

- Decimation factor
- Number of samples per signal
- Max number of samples per signal
- Duration

When the model is running, these parameters needs to be tuned to give a combination that will give the best possible data display in the console.

The lack of real-time monitoring is a clear disadvantage of using the console for data collection. This means that if the performance of the control method will be studied in detail, the use of other measuring equipment, such as an oscilloscope, is necessary to get reliable results. However, the monitoring in the console can be useful to get a quick overview of what is happening in the system.

7.8 Testing the predictive control algorithm

After the digital output and analog input communication was tested as described in the previous chapters, the RT-Lab model containing the predictive control algorithm was applied to the system. This did not give the desired results. The programming of the FPGA worked with no error messages, and the model can be executed from RT-Lab. However, there seems to be no response from the circuit. Studying the output variables and the test card, it is clear that no switching occurs. In order to try to locate the error, a selection of signals were sent to the monitor to study the behaviour in different parts of the system. The following signals were sent to the console:

- The counter signals, count_a, count_b, count_c
- Cost function, g
- Sum error in current control, e_cur
- Number of samples, m
- Alpha reference current, iref_alpha
- Alpha measured current, imeas_alpha

- Phase a measured current, imeas_a
- Back-emf real part, e_re
- Voltage value input to the Mcode, v2
- Component values at the input of the Mcode, (Ts/L)

None of these signals gave any feedback to the console, and all values were constantly measured to 0.

Signals were monitored from different locations in the model, and both signals that depends on the results from the predictive control algorithm as well as signals that should be unaffected by the Mcode block were sent to the console. This indicates that the error might lie within the communication between the CPU part of the model and the FPGA. One possibility is that the error occurs in the transmission of the data from the DataOut block in the FPGA model to the DataOut Recv block in the master. The data is both reformatted before the transition and after, and a design fault from the user might be present in the system.

Another critical part of the model is the communication between the DataIn Send block in the master, and the DataIn block in the FPGA model. It is interesting that the refrence currents that were sent to the console gave a constant value of zero. If this is the cause of the error, then this will affect the whole model, and might cause every other variable to be zero, as no currents will flow, resulting in no switching. A comment to this is that the reference current signals that were tested, were extracted from the model after the data formatting through several blocks editing type and rate of the signal, including Slice, Reinterpret and Assert blocks. The signal has also been changed from abc- to $\propto\beta$-coordinates. The next logical test would be to test the signal at the output of the DataIn block, to see if there is any data received in this block. If so, there is a problem with the reinterpreting of the signal before it is used in the model. One more factor that should be taken into consideration, is that no scaling is implemented for the measured current in the FPGA model. A gain of 5.1 should be introduced at the output from the analog in block, and no gain block should be applied in the console.

Unfortunately, when the work reached this point, the deadline for the master thesis was approaching, and there was not enough time to make the tests required to find the error source. Troubleshooting in the model will be included in further work.

8 Conclusion and further work

8.1 Conclusion

Predictive current control was tested in a Simulink model with a sampling period of 1 µs. Different sampling periods for the predictive control algorithm were tested to study the effect the frequency of the prediction has on the performance of the control method. Simulations with a sampling period of 25 µs for the Matlab Function block, containing the predictive control algorithm, resulted in good reference tracking of the current, with some ripple in the output current. The cost function stabilized around a value of approximately 0.4, which confirms a low reference tracking error. Simulations were performed with a sample period for the predictive control algorithm adjusted to 1 µs. This improved the current reference tracking significantly, and a mean value of approximately 0.015 for the cost function was achieved. From these results, it is clear that the sampling frequency of the predictive control algorithm will affect the performance of the control. This contributed to the motivation for executing the control algorithm on a FPGA, which can perform a great amount of calculations at a very high speed.

The predictive control method tested in Simulink, was adapted into the RT-Lab/XSG environment, with the aim to implement the predictive control in an experimental setup. For a physical circuit, switching frequency and losses will be of high importance, and an extra control objective concerning the switching frequency reduction was included in the predictive control algorithm. Some performance variables, used to calculate the average switching frequency and mean current error from the cost function, were introduced in the model. The first simulations with XSG were performed with no switching frequency reduction, and resulted in a very good reference tracking with the value of the cost function centering around approximately 0.1. Compared to the results in Simulink, the XSG current control presented a deprecated performance. This might be due to the working principles of the FPGA, which introduces some delay in the model. Another reason might be that XSG works with the fixed point data format, and this puts some limitation on the precision of the data, which might propagate through the model. The average switching frequency was measured to 43.8 kHz, and this is not feasible for use with a physical circuit.

Switching frequency cost was included in the cost function, by adjusting the weighing factor, A. A decides the weight put on the switching frequency compared to the current reference tracking. Several simulations with an increasing value of A, showed that the switching frequency easily could be reduced, though at the cost of decreased precision in the current reference tracking. These results demonstrates the benefit of adding several control objectives to the control system of the inverter, and emphasize the advantage of how easily this can be done for the predictive control method. By implementing the control method in an experimental setup, the weighing factor can be adjusted during execution, and a value should be found that provides both a satisfactory switching frequency and current control. It should be taken into consideration that despite the good results achieved with simulation, it is expected that the performance of the predictive control in an experimental setup will deviate from these results, as delay is introduced in the system.

The model was adapted to the RT-Lab environment, and I/O blocks were introduced in the model for communication with external hardware. This model was tested in an experimental setup with a 20 kW IGBT inverter and a star connected resistive inductive load. Tests were performed to establish working communication links between the console in RT-Lab and the digital output and analog input communication from the model to the inverter. Switching signals were controlled successfully from the console, and a DC-current test was performed to adjust the scaling of the analog input current from the LEM current sensors. However, when the model containing the predictive control algorithm was tested, there was no response from the circuit, and no switching occurred. Troubleshooting in the model indicated that an error is introduced in the data transmission between the CPU model and the FPGA model. Unfortunately, the problem could not be solved within the given time limits, and this will have to be included in further work for the model.

8.2 Further work

Further work will include troubleshooting in the model, to find the error that causes the model to fail. The next logical test would be to monitor the signal at the output of the DataIn block, to see if there is any data received in this block. If so, there is a problem with the reinterpreting of the signal before it is used in the model, and the formatting needs to be revaluated. A gain of 5.1 at the output from the analog in block, should also be implemented in the FPGA model, due to the conversion ratio in the LEM sensors measuring the output current for the inverter.

When the model is working, the next step would be to introduce some form of delay compensation to compensate for the time required for execution in an experimental setup. The method presented in [14] is a relevant option, and might contribute to a better performance especially during transients. Another factor that should be taken into consideration is that a prediction horizon length of one was used in this work. Previous work have indicated that a longer prediction horizon can contribute to a better steady-state performance [4]. The influence of the prediction horizon on the performance would be an interesting topic of investigation, and should be included in the predictive control algorithm for further studies.

Adjustment of the weighing factor to find an acceptable combination of reference tracking performance and switching frequency should be done for the modified model. The focus of this work has been to illustrate the correlation between the current control and the switching frequency, and study how the reference tracking performance will be affected by a reduced switching frequency. For a given system, more weight could be put on investigating variables such as energy efficiency, THD and voltage spectra in the output voltage, to get a better evaluation of the performance of the predictive control method for the inverter.

9 References

1. Rodriguez, J.C., P. , *Predictive control of power converters and electrical drives*. 2 ed. 2012, United kingdom: John Wiley & sons.
2. Cortes, P., Kazmierkowski, M. P., Kennel, R. M., Quevedo, D. E. & Rodriguez, J. , *Predictive Control in Power Electronics and Drives.* IEEE Transactions on Industrial Electronics, 2008. **55**(12): p. 4312-4324.
3. Rodriguez, J., Pontt, J., Silva, C.A., Correa, P., Lezana, P., Cortes, P. & Ammann, U., *Predictive Current Control of a Voltage Source Inverter.* Industrial Electronics, IEEE Transactions on, 2007. **54**(1): p. 495 - 503.
4. J. Rodriguez, M.P.K., J. R. Espinoza, P. Zanchetta, H. Abu-Rub, H. A. Young, & C. A. Rojas, *State of the Art of Finite Control Set Model Predictive Control in Power Electronics.* IEEE Transactions on Industrial Informatics, 2013. **9**(2): p. 1003 - 1016.
5. Mohan, N., Undeland. T. M. & Robbins, W. P., *Power Electronics. Converters, Applications and Design.* Third ed. 2003: John Wiley & Sons, Inc.
6. Vargas, R., Ammann, U., Rodriguez, J., & Ponnt, J., *Reduction of Switching Losses and Increase in Efficiency of Power Converters using Predictive Control*, in *Power Electronics Specialists Conference*. 2008, IEEE: Rhodes. p. 1062 - 1068.
7. Rajapakse, A.D., Gole, A.M. & Wilson, P.L. , *Electromagnetic Transients Simulation Models for Accurate Representation of Switching Losses and Thermal Performance in Power Electronic Systems.* IEEE Transactions on Power Delivery, 2005. **20**(1): p. 319 - 327.
8. Al-Naseem, O., Erickson, R. W. & Carlin, P., *Prediction of Switching Loss Variations by Averaged Switch Modeling*, in *Applied Power Electronics Conference and Exposition. APEC 2000.* 2000, IEEE: New Orleans, LA. p. 242 - 248.
9. Bernet, S., Ponnaluri, S. & Teichmann, R. , *Design and Loss Comparison of Matrix Converters and Voltage-Source Converters for Modern AC Drives.* IEEE Transactions on Industrial Electronics, 2002. **49**(2): p. 304 - 314.
10. Helle, L., Larsen, K., Jorgensen, A. H., Munk-Nilsen, S. & Blaabjerg, F. , *Evaluation of Modulation Schemes for Three-Phase to Three-Phase Matrix Converters.* IEEE Transactions on Industrial Electronics, 2004. **51**(1): p. 158 - 171.
11. Apap, M., Clare, J. C., Wheeler. P. & Bradley, K. J. , *Analysis and Comparison of AC-AC Matrix Converter Control Strategies*, in *IEEE PE Society Annual Meeting, PESC 2003*. 2003, IEEE: Acapulco, Mexico. p. 1287 - 1292.
12. Rivera, M., Wilson, A., Rojas, C. A., Rodriguez, J., Espinoza, J. R., Wheeler, P. W. & Empringham, L. , *A Comparative Assessment of Model Predictive Current Control and Space Vector Modulation in a Direct Matrix Converter.* IEEE Transactions on Industrial Electronics, 2013. **60**(2): p. 578-588.
13. Cortes, P., Kouro, S., La Rocca, B., Vargas, R., Rodriguez, J., Leon, J. I., … Franquelo, L.G. , *Guidelines for Weighting Factors Design in Model Predictive Control of Power Converters and Drives*, in *IEEE International Conference on Industrial Technology, ICIT 2009*. 2009, IEEE: Gippsland, VIC. p. 1-7.
14. Cortez, P., Rodriguez, J., Silva, C. & Flores, A., *Delay Compensation in Model Predictive Current Control of a Three-Phase Inverter*, in *IEEE Transactions on Industrial Electronics*. 2011, IEEE. p. 1323 - 1325.

Appendix A Parameters file

Code A.1

```
%Variables required in the control algorithm
global Ts R L v states

%Sampling time for the predictive control algorithm [s]
Ts = 1e-6;

% Load parameters
R = 10;      %Resistance [ohm]
L = 10e-3;       %Inductance [H]
e = 100;         %Back-emf amplitude [V]
f_e = 50 * (2*pi) ;      %Back-emf frequency [rad/s]
Vdc = 520;       %DC-link voltage [V]

% %Current reference
i_ref = 10;
w_ref = 2*pi*50;

%Voltage vectors
v0 = 0;
v1 = 2/3 *Vdc;
v2 = (1/3) * Vdc + 1j*(sqrt(3)/3)* Vdc;
v3 = -(1/3) * Vdc + 1j*(sqrt(3)/3) * Vdc;
v4 = -(2/3) * Vdc;
v5 = -(1/3) * Vdc - 1j*(sqrt(3)/3) *Vdc;
v6 = (1/3) * Vdc - 1j*(sqrt(3)/3) *Vdc;
v7 = 0;
v = [v0 v1 v2 v3 v4 v5 v6 v7];

%Switching states
states =[0 0 0; 1 0 0; 1 1 0; 0 1 0; 0 1 1; 0 0 1; 1 0 1; 1 1 1];
```

Appendix B Cost function plot

Results from simulation in Simulink with a sampling period of 25 µs for the predictive control algorithm. The value for g stabilizes after a short time.

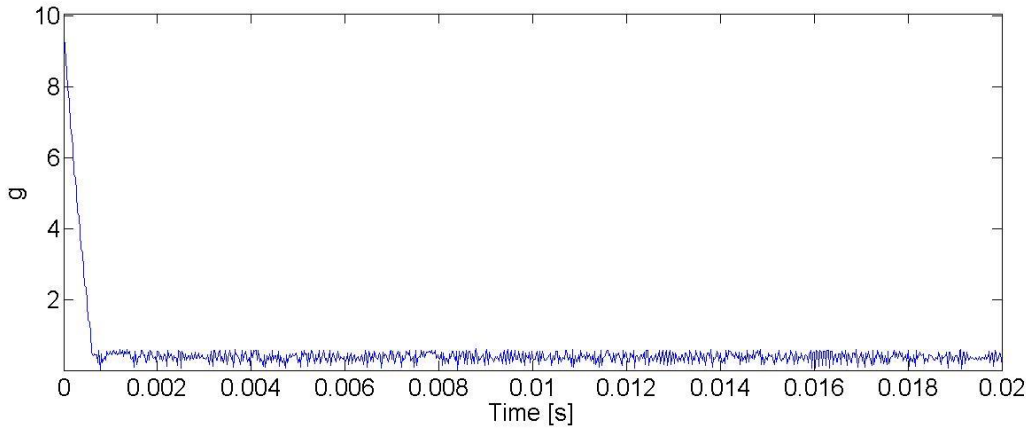

Figure B-1: Cost function, g.

Appendix C Hardware Configuration block

Settings for the Hardware Configuration block.

Table C-1: Settings Hardware Configuration block.

Active control card	Xilinx ML605 Development Platform
Chassis form factor	FlatIO
Carrier type	Opal-RT OP56008-Slot Flat I/O Carrier
Section 1A I/O module	Opal-RT OP5330-3 SCMB, D/A 14 Ch@15ma Digital to Analog Module
Section 1B I/O module	Opal-RT OP5340 SCMB, 16 ch, 16 bit, 1us, A/D, +5 V to +- 100V
Section 2A I/O module	<empty>
Section 2B I/O module	OPAL-RT OP5340 SCMB, 16 ch, 16 bit, 1us, A/D, +5 V to +- 100V
Section 3A I/O module	<unavailable>
Section 3B I/O module	<unavailable>
Section 4A I/O module	Opal_RT OP5353 Opto-Isolated Digital Mezzanine – 32 Din
Section 4B I/O module	Opal_RT OP5354 Opto-Isolated Digital Mezzanine – 32 Dout

Appendix D Predictive control algorithm

The predictive control algorithm implemented in the Xilinx Mcode block.

Code D.1

```
function [x, count_a, count_b, count_c, g_opt_output, e_out,
m_out]  = system_test_with_emf_compblock_and_switch( I_ref_alpha,
I_ref_beta,I_meas_alpha,I_meas_beta, ioa, iob, ioc, e_re, e_im,
v0, v1, 4  v2, v3, v4, v5, v6, v7, v8, v9, v10, v11, v12, v13,
v14, v15, compA, compB, Vdc, A)

%Initialize voltage vector
prec = {xlSigned, 30, 20};
persistent v, v = xl_state(zeros(1,16), prec);

%Initial values
g_opt = xfix({xlUnsigned, 60, 30}, 1e10);
x_opt = xfix({xlUnsigned, 3, 0}, 0);
a = 0;
b = 0;

%Initialize reference current variables
persistent ik_ref_re_reg, ik_ref_re_reg  = xl_state(0,
{xlSigned, 16, 10});
persistent ik_ref_im_reg, ik_ref_im_reg  = xl_state(0,
{xlSigned, 16, 10});

% Initialize measured current variables
persistent ik_re_reg, ik_re_reg  = xl_state(0, {xlSigned,16,10});
persistent ik_im_reg, ik_im_reg  = xl_state(0, {xlSigned,16,10});

%Establish switching states variables and vector
persistent Sa, Sa  = xl_state(0, {xlUnsigned, 1, 0});
persistent Sb, Sb  = xl_state(0, {xlUnsigned, 1, 0});
persistent Sc, Sc  = xl_state(0, {xlUnsigned, 1, 0});

Sa_old = Sa;
Sb_old = Sb;
Sc_old = Sc;

persistent states, states=xl_state(zeros(1,24),{xlUnsigned,1,0});

states(0) = 0;
states(1) = 0;
states(2) = 0;

states(3) = 1;
states(4) = 0;
states(5) = 0;

states(6) = 1;
```

```
47 states(7) = 1;
48 states(8) = 0;
49
50 states(9) = 0;
51 states(10) = 1;
52 states(11) = 0;
53
54 states(12) = 0;
55 states(13) = 1;
56 states(14) = 1;
57
58 states(15) = 0;
59 states(16) = 0;
60 states(17) = 1;
61
62 states(18) = 1;
63 states(19) = 0;
64 states(20) = 1;
65
66 states(21) = 1;
67 states(22) = 1;
68 states(23) = 1;
69
70 %Read values from console
71 l = xfix({xlSigned, 22, 20}, compA);
72 m = xfix({xlSigned, 16, 10}, compB);
73
74 %Initialize variables as arrays
75 persistent n, n = xl_state(zeros(1,8), prec);
76 persistent o, o = xl_state(zeros(1,8), prec);
77 persistent p, p = xl_state(zeros(1,8), prec);
78 persistent q, q = xl_state(zeros(1,8), prec);
79 persistent r, r = xl_state(zeros(1,8), prec);
80 persistent s, s = xl_state(zeros(1,8), prec);
81
82 persistent v_o1_re_reg, v_o1_re_reg = xl_state(zeros(1,8), prec);
83 persistent v_o1_im_reg, v_o1_im_reg = xl_state(zeros(1,8), prec);
84
85 persistent e_re_reg, e_re_reg = xl_state(0, prec);
86 persistent e_im_reg, e_im_reg = xl_state(0, prec);
87
88 persistent ik1_re, ik1_re = xl_state(zeros(1,8), prec);
89 persistent ik1_im, ik1_im = xl_state(zeros(1,8), prec);
90
91 persistent g, g = xl_state(zeros(1,8), {xlUnsigned, 20, 16});
92 persistent g1, g1 = xl_state(zeros(1,8), {xlUnsigned, 20, 16});
93 persistent g2, g2 = xl_state(zeros(1,8), {xlUnsigned, 20, 16});
94 persistent g3, g3 = xl_state(zeros(1,8), {xlUnsigned, 20, 16});
95
96 %Calculate switching losses for sampling instant k
97  if ioa >= 0
98        tap_a = (ioa * Vdc);
99  else
100       tap_a = (-ioa * Vdc);
101 end
102
```

```
103 if iob >= 0
104        tap_b = (iob * Vdc);
105 else
106        tap_b = (-iob * Vdc);
107 end
108
109 if ioc >= 0
110        tap_c = (ioc * Vdc);
111 else
112        tap_c = (-ioc * Vdc);
113 end
114
115 %Switching loss, independent of the current
116 tap_init = xfix({xlUnsigned, 6, 5}, 0.3);
117
118 %Initialize values for evaluation parameters
119 count_a = xfix({xlUnsigned, 1, 0}, 0);
120 count_b = xfix({xlUnsigned, 1, 0}, 0);
121 Count_c = xfix({xlUnsigned, 1, 0}, 0);
122
123 persistent e_sum, e_sum  = xl_state(0, {xlUnsigned, 25, 10});
124 e_out = xfix({xlUnsigned, 25, 10}, 0);
125 e_cur = xfix({xlUnsigned, 16, 10}, 0);
126
127 %For-loop estimating the cost function for all possible
128 %switching states
129 for i = 0:2:14
130
131    if i > 0
132    a = a + 1;
133    b = b + 3;
134    end
135
136    %i-th voltage vector for current prediction
137    v_o1_re_reg(a) = v(i);
138    v_o1_im_reg(a) = v(i+1);
139
140  %Current prediction at instant k+1, real part
141  %Calculating ik1_re = (1 - R*Ts/L)*ik_re + Ts/L*(v_o1_re-e_re);
142    n(a) = (m * ik_re_reg);
143    o(a) = v_o1_re_reg(a) - e_re_reg;
144    p(a) = l * o(a);
145    ik1_re(a) = n(a) + p(a);
146
147    %Current prediction at instant k+1, imaginary part
148    q(a) = (m * ik_im_reg);
149    r(a) = v_o1_im_reg(a) - e_im_reg;
150    s(a) = l * r(a);
151    ik1_im(a) = q(a) + s(a);
152
153    %Cost function
154    if ik_ref_re_reg >= ik1_re(a)
155        g1(a) = ik_ref_re_reg - ik1_re(a);
156    else
157        g1(a) = ik1_re(a) - ik_ref_re_reg;
158    end
```

```
159
160     if ik_ref_im_reg >= ik1_im(a)
161         g2(a) = ik_ref_im_reg - ik1_im(a);
162     else
163         g2(a) = ik1_im(a) - ik_ref_im_reg;
164     end
165
166     %Calculate cost switching losses
167     if Sa ~= states(b)
168         g3(a) = g3(a) + tap_a + tap_init;
169     end
170
171     if Sb ~= states(b+1)
172         g3(a) = g3(a) + tap_b + tap_init;
173     end
174
175     if Sc ~= states(b+2)
176         g3(a) = g3(a) + tap_c + tap_init;
177     end
178
179       g(a) = g1(a) + g2(a) + A* g3(a);
180
181 end
182
183 %Selection of the optimal value
184      if (g(0)<g_opt)
185         g_opt = g(0);
186         x_opt = 0;
187         Sa = 0;
188         Sb = 0;
189         Sc = 0;
190         e_cur = g1(0) + g2(0);
191      end
192
193      if (g(1)<g_opt)
194         g_opt = g(1);
195         x_opt = 1;
196         Sa = 1;
197         Sb = 0;
198         Sc = 0;
199         e_cur = g1(1) + g2(1);
200      end
201
202      if (g(2)<g_opt)
203         g_opt = g(2);
204         x_opt = 2;
205         Sa = 1;
206         Sb = 1;
207         Sc = 0;
208         e_cur = g1(2) + g2(2);
209      end
210
211      if (g(3)<g_opt)
212         g_opt = g(3);
213         x_opt = 3;
214         Sa = 0;
```

```
215         Sb = 1;
216         Sc = 0;
217         e_cur = g1(3) + g2(3);
218      end
219
220      if (g(4)<g_opt)
221         g_opt = g(4);
222         x_opt = 4;
223         Sa = 0;
224         Sb = 1;
225         Sc = 1;
226         e_cur = g1(4) + g2(4);
227      end
228
229      if (g(5)<g_opt)
230         g_opt = g(5);
231         x_opt = 5;
232         Sa = 0;
233         Sb = 0;
234         Sc = 1;
235         e_cur = g1(5) + g2(5);
236      end
237
238      if (g(6)<g_opt)
239         g_opt = g(6);
240         x_opt = 6;
241         Sa = 1;
242         Sb = 0;
243       Sc = 1;
244         e_cur = g1(6) + g2(6);
245      end
246
247      if (g(7)<g_opt)
248         g_opt = g(7);
249         x_opt = 7;
250         Sa = 1;
251         Sb = 1;
252         Sc = 1;
253         e_cur = g1(7) + g2(7);
254      end
255
256  %Register alteration in switching states
257       if Sa ~= Sa_old
258        count_a = 1;
259       end
260
261       if Sb ~= Sb_old
262        count_b = 1;
263       end
264
265       if Sc ~= Sc_old
266        count_c = 1;
267       end
268
269  %variables for calculating the mean current-cost function value
270  persistent m, m  = xl_state(0, {xlUnsigned, 16, 0});
```

```
271 m_out = xfix({xlUnsigned, 16, 0}, 0);
272 m = m + 1;
273 m_out = m;
274 
275 e_sum = e_sum + e_cur;
276 e_out = e_sum;
277 
278 %Update registers
279 ik_re_reg = I_meas_alpha;
280 ik_im_reg = I_meas_beta;
281 
282 ik_ref_re_reg = I_ref_alpha;
283 ik_ref_im_reg = I_ref_beta;
284 
285 v(0) = v0;
286 v(1) = v1;
287 v(2) = v2;
288 v(3) = v3;
289 v(4) = v4;
290 v(5) = v5;
291 v(6) = v6;
292 v(7) = v7;
293 v(8) = v8;
294 v(9) = v9;
295 v(10) = v10;
296 v(11) = v11;
297 v(12) = v12;
298 v(13) = v13;
299 v(14) = v14;
300 v(15) = v15;
301 
302 e_re_reg = e_re;
303 e_im_reg = e_im;
304 
305 end
```

Appendix E Back-emf calculation

The back-emf calculation model with Xilinx blocks.

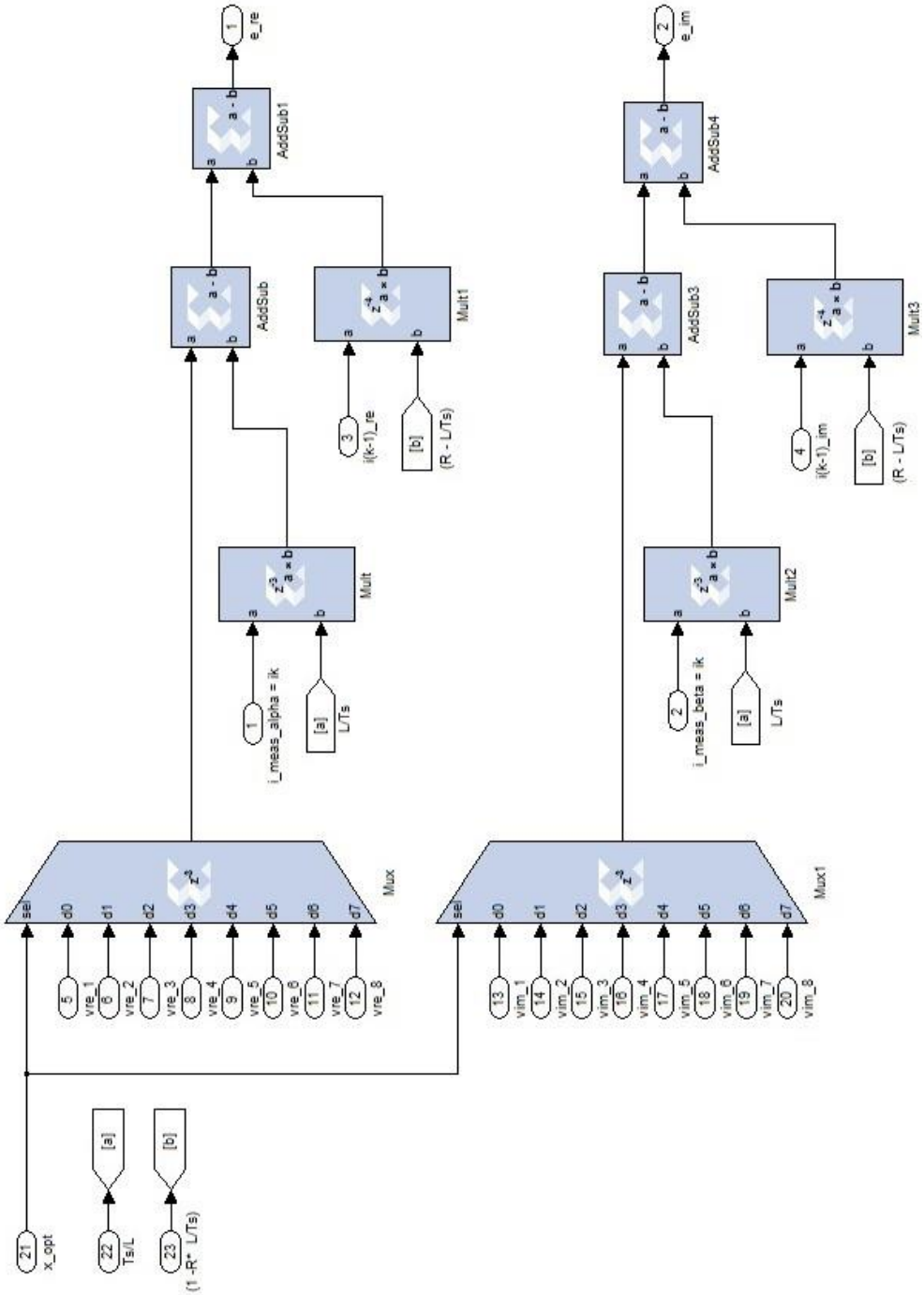

Figure E-1: Back-emf calculation.

Appendix F States_selection block

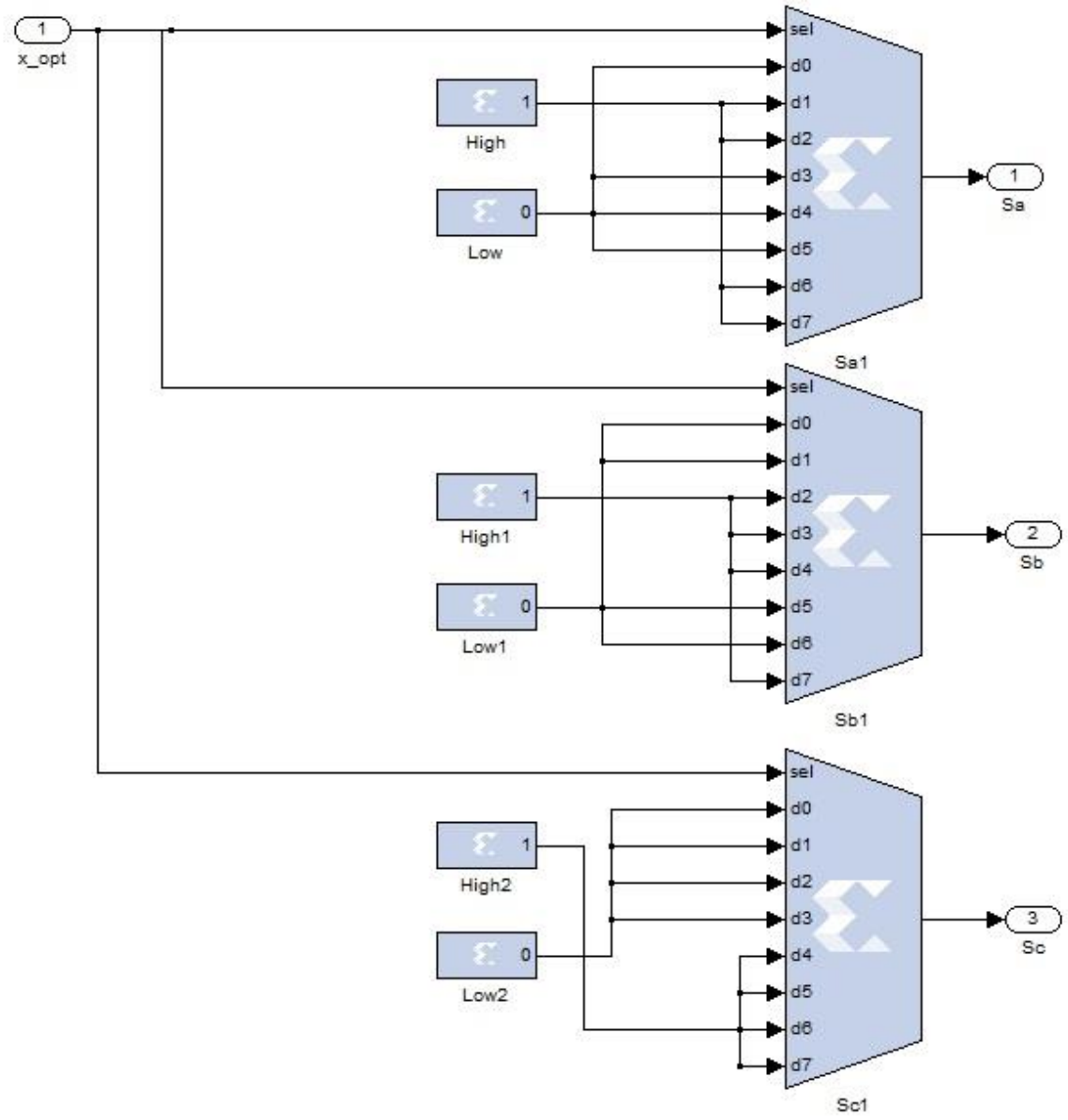

Figure F-1: Contents of the states_selection block in the XSG model.

Appendix G Timing error

Table G-1 Shows an extract from the error log opened in Timing Analyzer. The path exceeds the timing requirement of 10 ns with 28.333 ns. It can also be observed that the clock path skew and clock uncertainty is negligible compared to the data path delay, so this is not the cause for the problem. The source of the critical path is identified as the register containing the variable e_re_reg, and the destination is the block Register4, which is located at the output of the Mcode block.

Table G-1: Extract from the error log opened in Timing Analyzer.

```
Slack (setup path): -28.333ns (requirement - (data path - clock path
skew + uncertainty))
   Source:
ml605_pcie_xsg_core_u0/fpga_model_u0/fpga_model_x0/mcode/e_re_reg_50
_22_3 (FF)
   Destination:
ml605_pcie_xsg_core_u0/fpga_model_u0/fpga_model_x0/register4/synth_r
eg_inst/latency_gt_0.fd_array[1].reg_comp/fd_prim_array[0].bit_is_0.
fdre_comp (FF)
   Requirement:          10.000ns
   Data Path Delay:       38.180ns (Levels of Logic = 67)(Component
delays alone exceeds constraint)
   Clock Path Skew:      -0.089ns (0.960 - 1.049)
   Source Clock:         user_clk_c rising at 0.000ns
   Destination Clock:    user_clk_c rising at 10.000ns
   Clock Uncertainty:    0.064ns

   Clock Uncertainty:           0.064ns  ((TSJ^2 + DJ^2)^1/2) / 2 +
PE
     Total System Jitter (TSJ):  0.070ns
     Discrete Jitter (DJ):       0.105ns
     Phase Error (PE):           0.000ns
```

Appendix H Simulation results XSG model

Simulation results from the Xilinx model.

Table H-1 Count signals and average switching frequency.

A	t [s]	Count_a	Count_b	Count_c	fsa[kHz]	fsb [kHz]	fsc [kHz]	fs [kHz]
0	0.06	4399	5714	5662	36.7	47.6	47.2	43.8
0.01	0.06	3140	3075	3015	26.2	25.6	25.1	25.6
0.02	0.06	2160	2164	2181	18.0	18.0	18.2	18.1
0.03	0.06	1589	1598	1583	13.2	13.3	13.2	13.2
0.04	0.06	1293	1296	1295	10.8	10.8	10.8	10.8
0.08	0.06	777	740	764	6.5	6.2	6.4	6.4

Table H-2: Mean refrence tracking error.

A	t [s]	e_out	m_out	$\bar{e}$
0	0.06	7677	6*10^4	0.1280
0.01	0.06	4.08*10^4	6*10^4	0.6800
0.02	0.06	6.465*10^4	6*10^4	1.0775
0.03	0.06	8.588*10^4	6*10^4	1.4313
0.04	0.06	1.0485*10^5	6*10^4	1.7475
0.08	0.06	1.7496*10^5	6*10^4	2.9160

Appendix I Simulation results XSG model:

Plots of current and cost function

I.1 A= 0

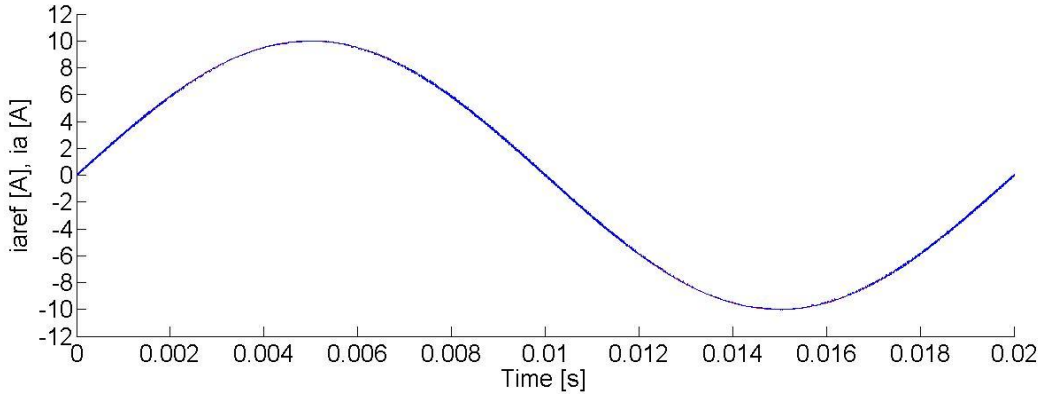

Figure I-1: A = 0.. Output current (blue) and reference current (red), phase a.

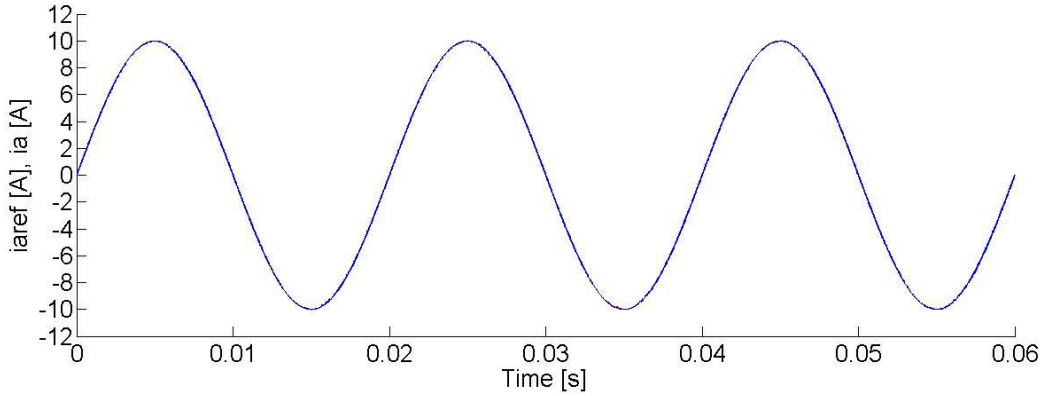

Figure I-2: A = 0. Output current (blue) and reference current (red), phase a.

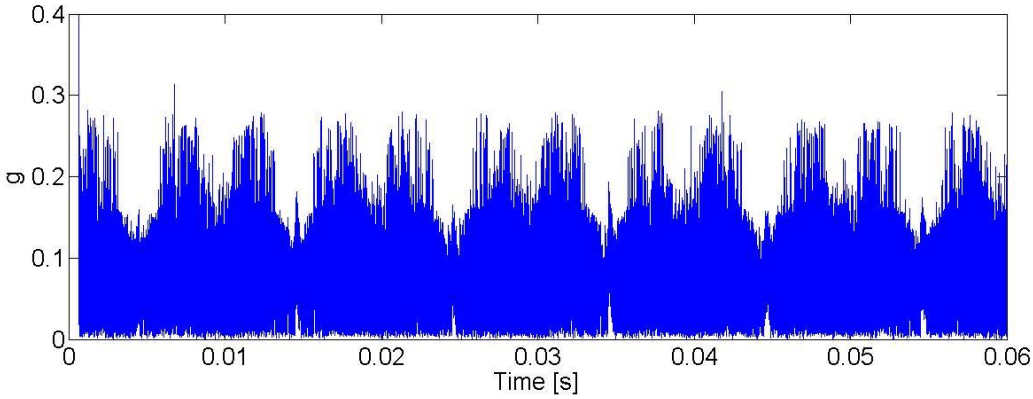

Figure I-3: The cost function, g for A = 0.

I.2 A= 0.01

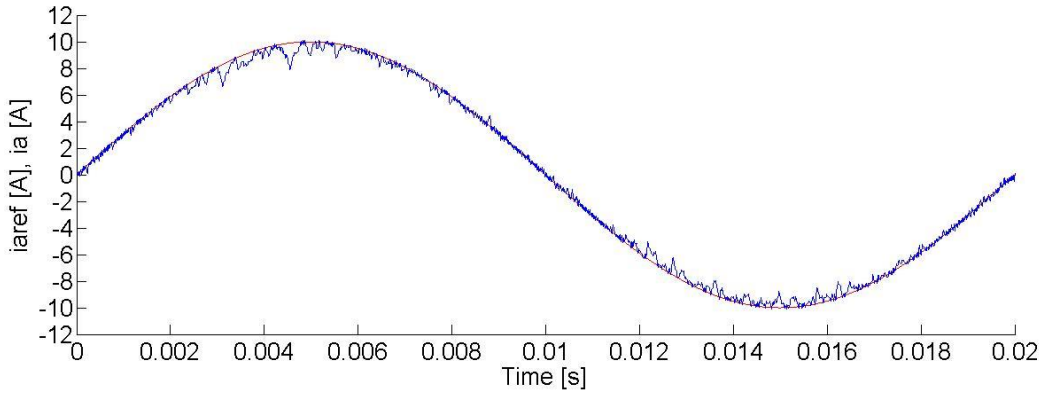

Figure I-4: A = 0.01. Output current (blue) and reference current (red), phase a

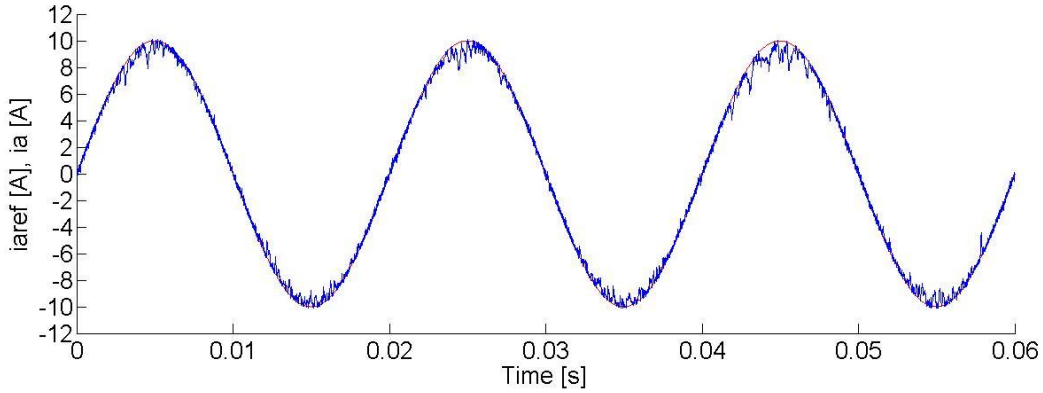

Figure I-5: A = 0.01. Output current (blue) and reference current (red), phase a

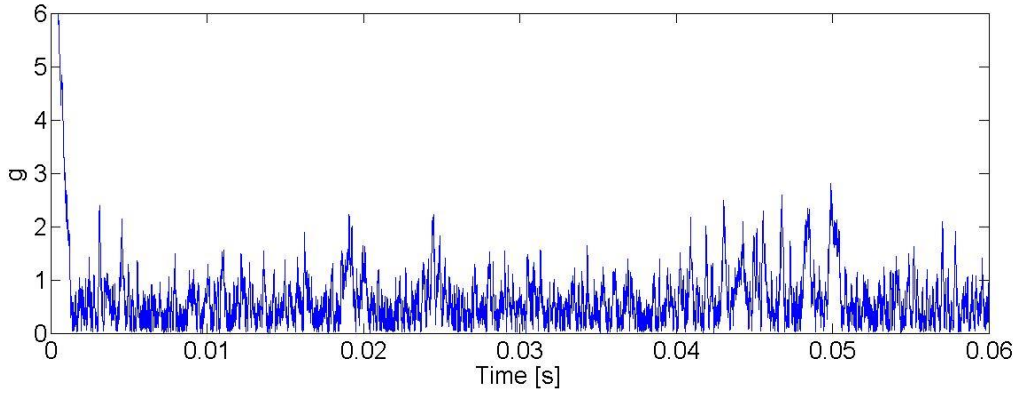

Figure I-6: The cost function, g for A = 0.01.

I.3 A= 0.02

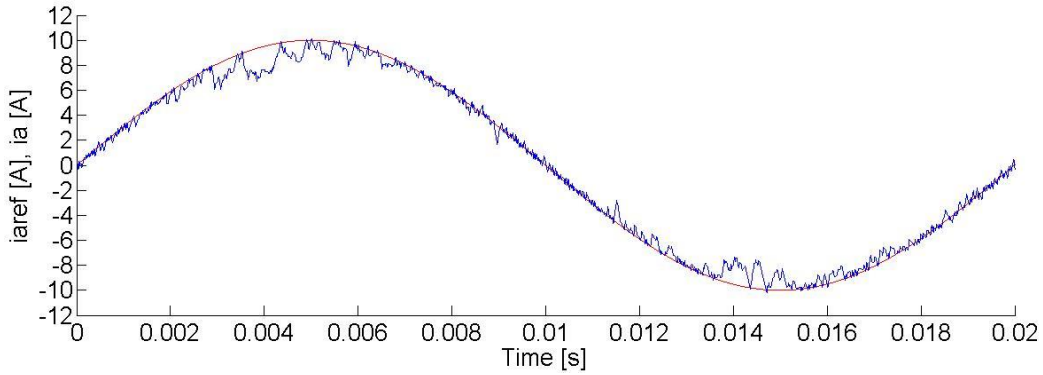

Figure I-7: A = 0.02. Output current (blue) and reference current (red), phase a

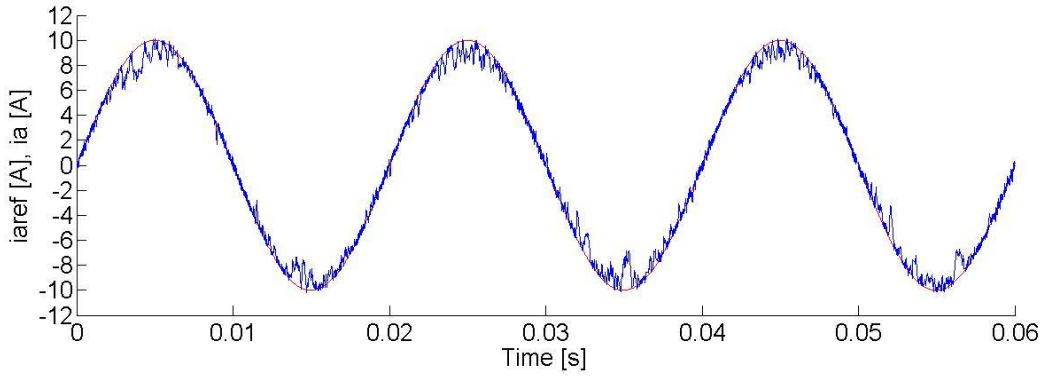

Figure I-8: A = 0.02. Output current (blue) and reference current (red), phase a

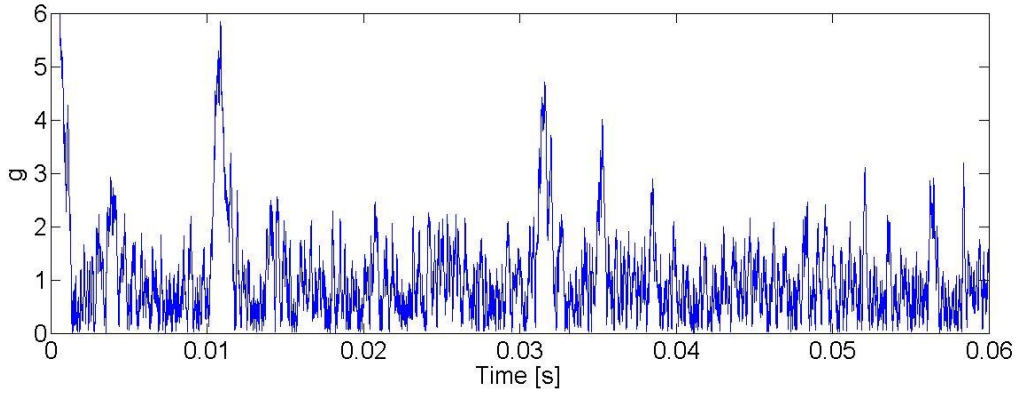

Figure I-9: The cost function, g for A = 0.02.

I.4 A= 0.04

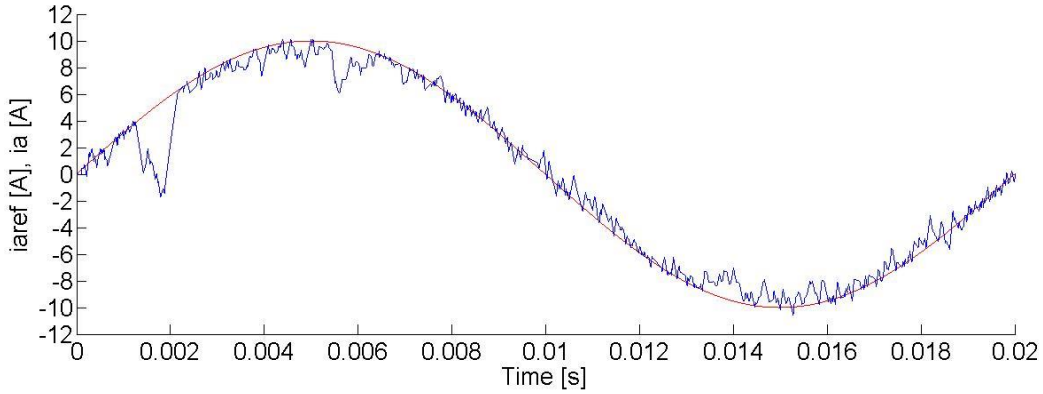

Figure I-10: A = 0.04. Output current (blue) and reference current (red), phase a

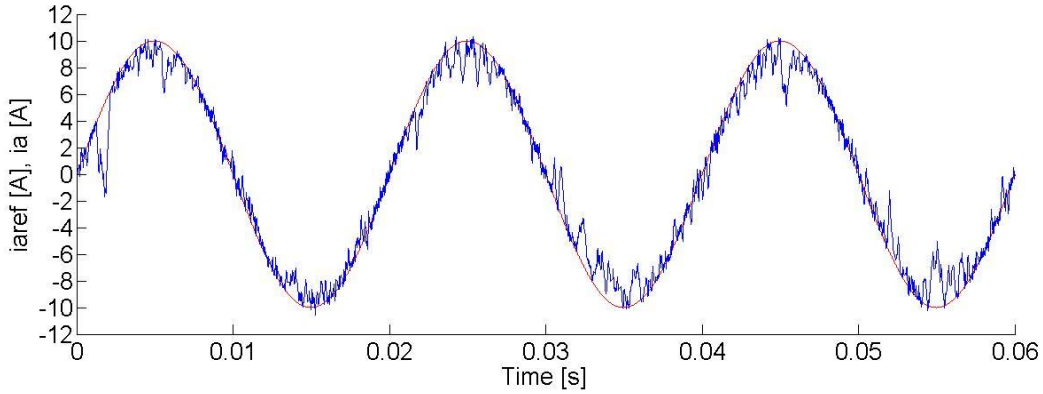

Figure I-11: A = 0.04. Output current (blue) and reference current (red), phase a

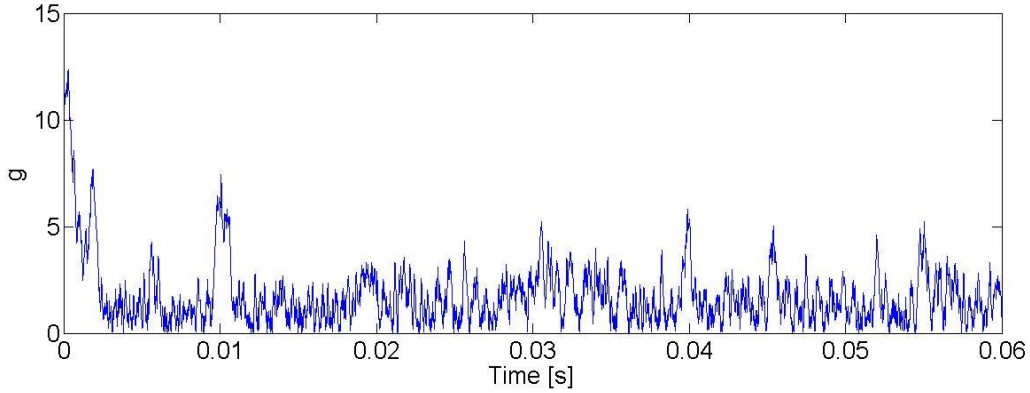

Figure I-12: The cost function, g for A = 0.04.

Appendix J Console in RT-Lab

Blocks for sending and receiving signals in the console.

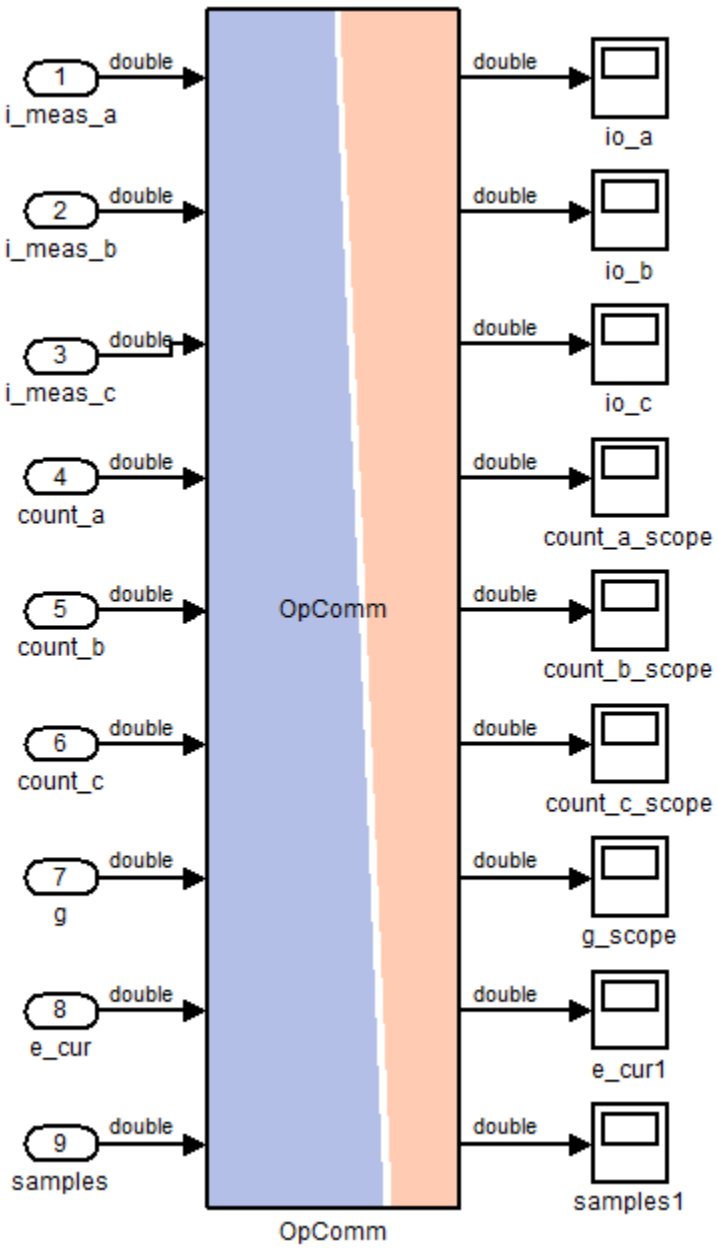

Figure J-1: OpComm block and input signals in the console.

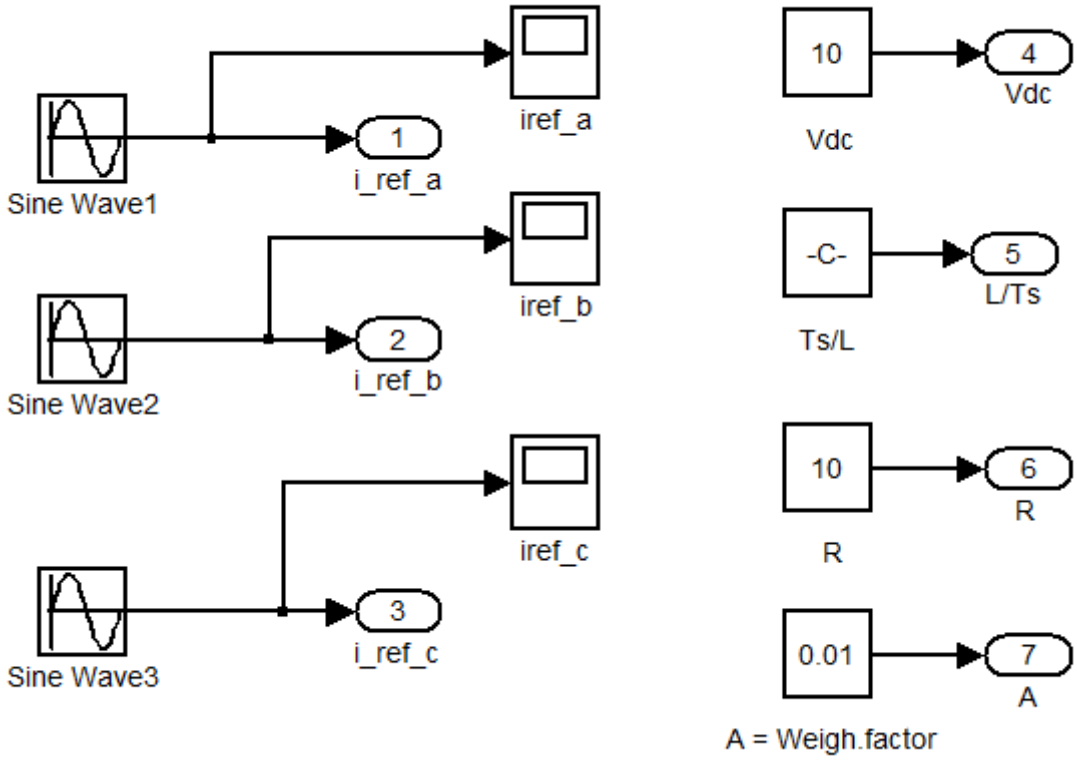

Figure J-2: Outputs from the console.

Appendix K The master in RT-Lab

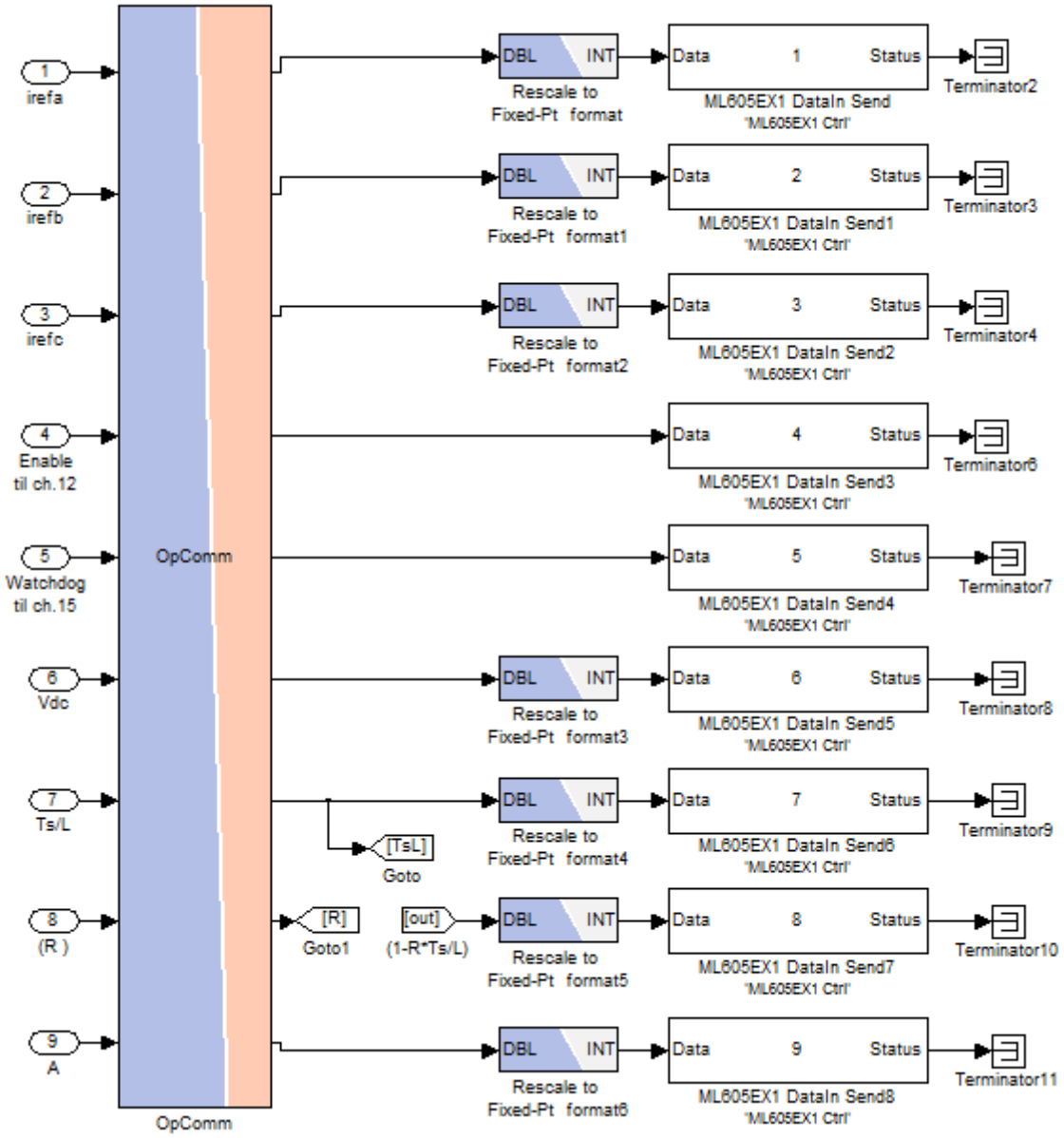

Figure K-1: Input from the console and output to the FPGA.

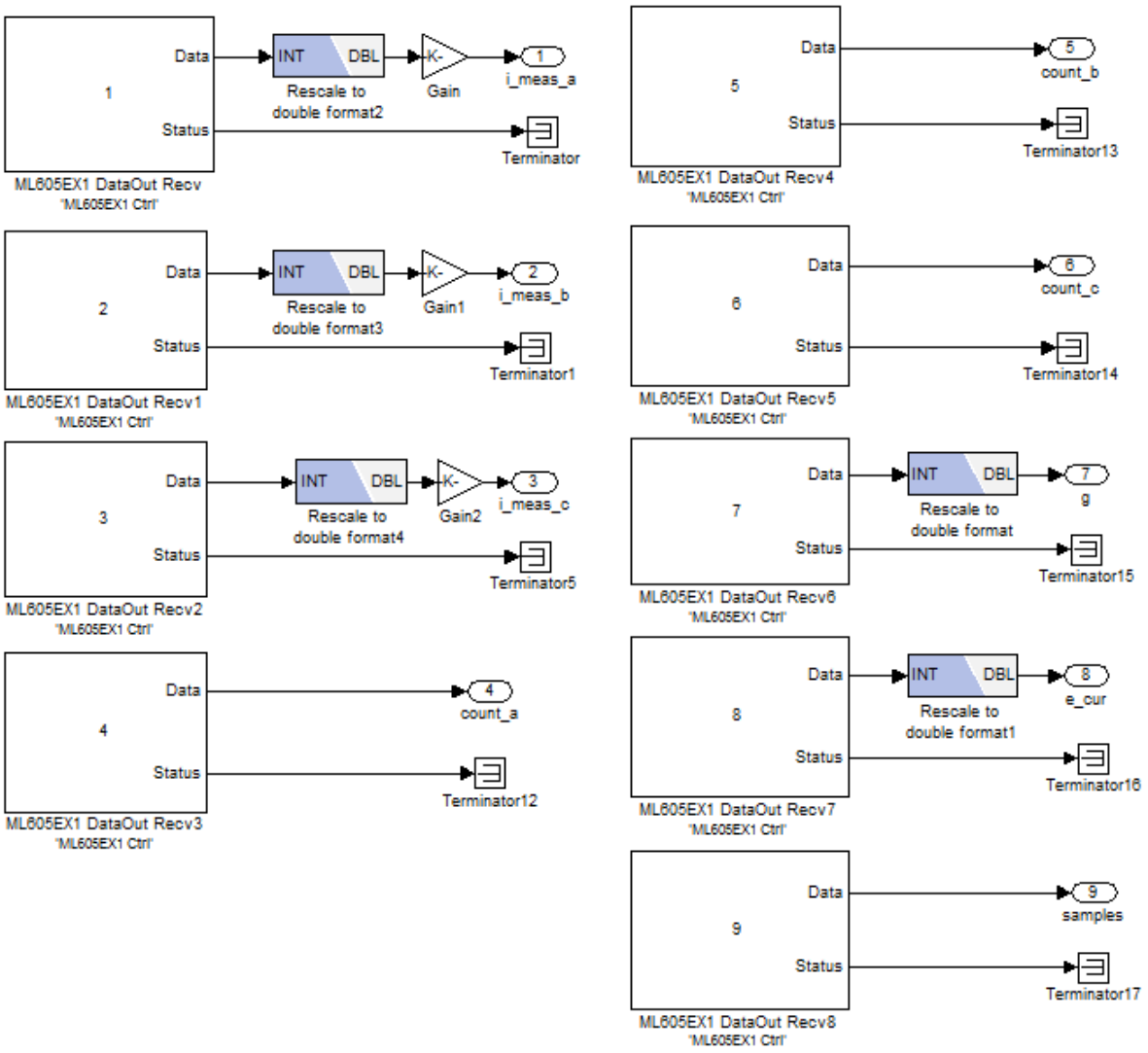

Figure K-2: DataOut Recv blocks.

Appendix L Communication blocks in the FPGA model

The DataOut and DataIn blocks with all input signals from the FPGA model.

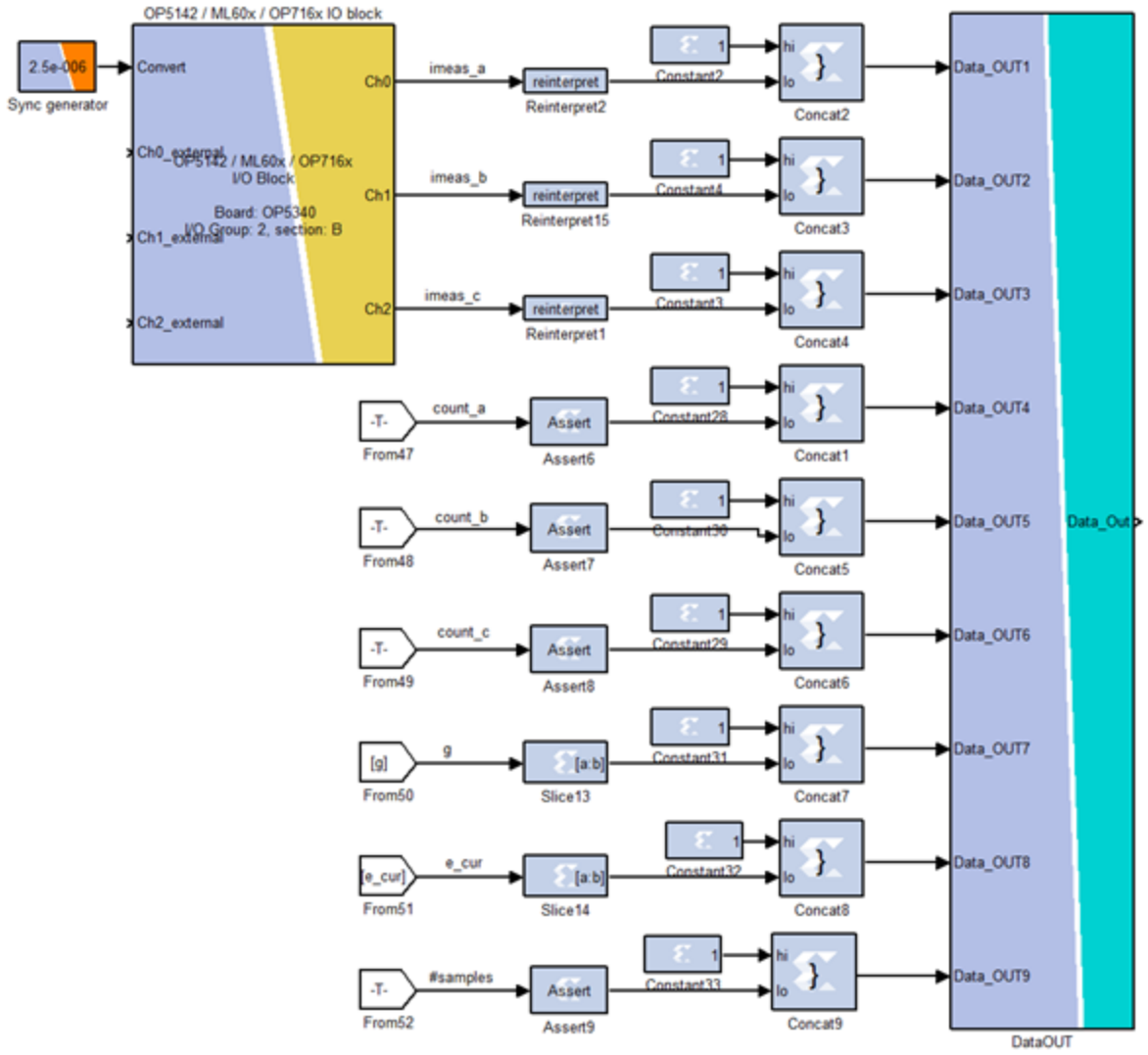

Figure L-1 The DataOut block with inputs.

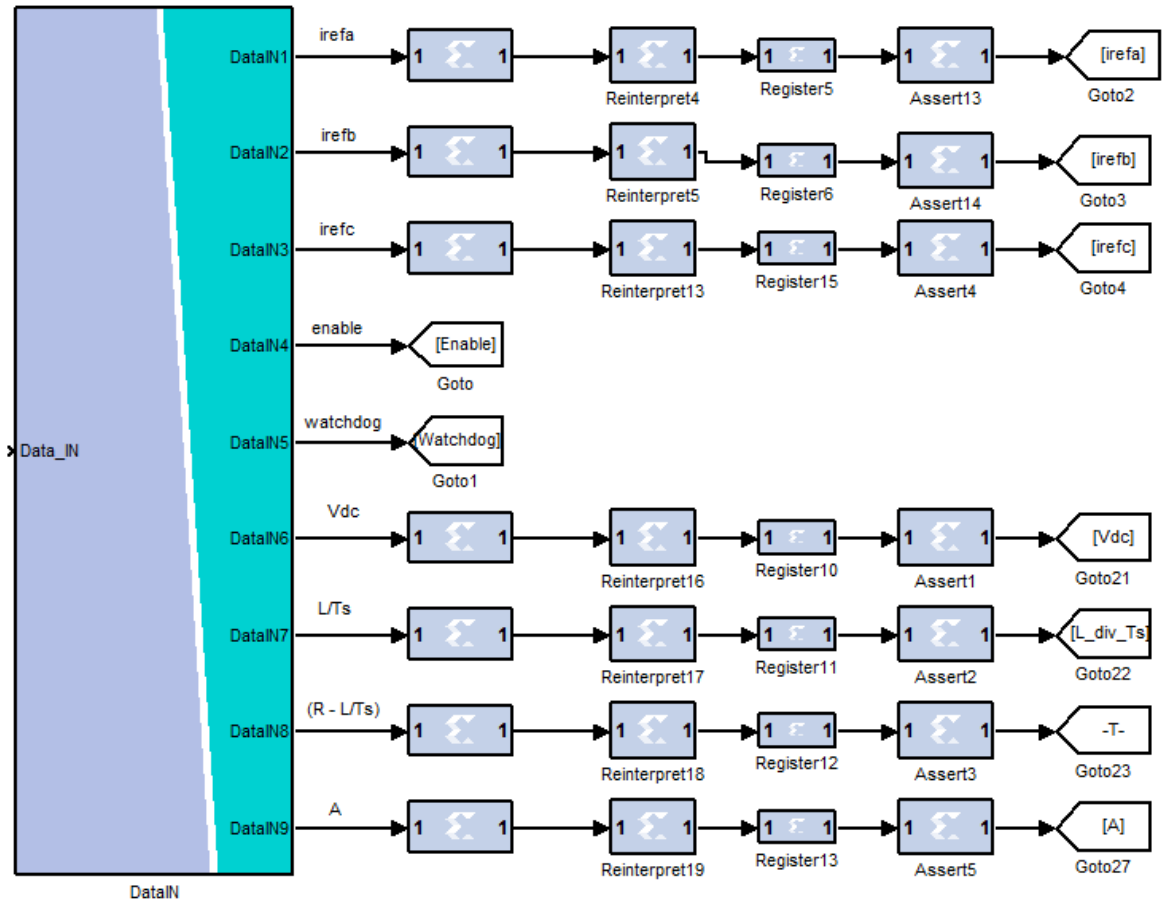

Figure L-2 The DataIn block with outputs.