Budapest University of Technology and Economics

Gergő Zoltán Sipos

Real-time, Hardware-In-The-Loop (HIL) based modelling of an electric drive system built on an asynchronous machine

CONSULTANT

Balázs Farkas

BUDAPEST, 2018

Table of Contents

Abstract

In my dissertation I implement a Hardware-In-The-Loop system, which is able to model the behaviour of an electric drive. The drive model consists of a 3-phase rectifier, a DC link, a three-phase two-level voltage inverter and an asynchronous machine. The model works functionally and is also able to model such secondary functions as semiconductor losses, transistor dead times and saturation of the asynchronous machine.

I build my model using Matlab Simulink toolbox elements in such a way, that the Verilog code - generated from the model - can run on the Xilinx Zynq-7000 platform. I need to generate a bitstream, using Matlab and Xilinx Vivado, and the model needs to be suitable for a fast file generation for the FPGA. This is essential, because it speeds up the development significantly.

I create a Matlab application to control this FPGA based, real-time drive simulator, on which I also carry out measurements.

At the end of my dissertation I present the capabilities of my system and what it can be used for during the development of a drive control system.

Kivonat

A dolgozatomban egy Hardware-in-the-Loop rendszert valósítok meg, amely képes valós időben modellezni egy hálózatról működő, 3 fázisú egyenirányítóból, egyenáramú oldalból, 3 fázisú, kétszintű feszültség inverterből és aszinkron gépből álló hajtás működését. A modell nem csak a funkcionális működés emulációjára képes, hanem olyan másodrendű hatásokat is figyelembe vesz, mint a félvezetők veszteségei, kapcsolási holtidők, illetve az aszinkron gép telítése.

A modellt a Matlab Simulink toolbox elemeiből építem fel, úgy, hogy az ebből generált Verilog kód a Xilinx Zynq-7000 platformon futtatható legyen. A hatékony fejlesztés érdekében a modellel szemben további elvárás, hogy a Matlab és Xilinx Vivado fejlesztő eszközökre épülő toolchain segítségével könnyen lehessen a FPGA-n futtatható bitfájlt generálni. Ez azért fontos, mert ez nagyban felgyorsítja a fejlesztés közi iterációkat.

Az így létrejövő FPGA alapú, valós idejű hajtásszimulátor vezérléséhez létrehozok egy Matlab alapú applikációt és a létrejött szimulátoron méréseket végzek.

Mindemellett, dolgozatom végén bemutatom, hogyan használható az általam fejlesztett eszköz egy hajtásrendszer vezérlőjének fejlesztése során.

1 Introduction

The main goal of my Hardware-in-The-Loop system is to make the development of an electric drive control unit easier. First, let's take a look at how the electric drive system works.

The drive system that consist of the grid, a three-phase rectifier, a DC link, a three-phase two-level inverter model and an asynchronous machine.

The grid provides the three phase voltages, 230 Volt RMS each. The three-phase rectifier converts this three-phase voltage into ~540 Volt DC voltage. The DC Link contains a capacitor and a reactor, which reduce the voltage and current ripple, which results in a more stable DC supply. The voltage inverter, controlled by the drive control unit, converts the DC voltage into a controlled AC voltage for the asynchronous motor.

The control unit provides the control signals for the gate driver unit and regulates certain motor parameters. The gate driver unit gives voltage to the gates of the transistors in the inverter. This determines the output voltage of the inverter. Measured parameters, which make the regulation possible, are fed back to the control unit.

If you want to test the control system's algorithm, you can use non-real-time tests on a PC platform, but if you want to test the control unit itself, you either need a testbench with an inverter and a motor, or a HIL system. The benefit of a HIL system is that it is much cheaper, and easier to test the control unit, and you can also monitor motor parameters, which are difficult to measure.

It is very useful and important to be able to easily and cheaply test the drive control unit. Therefore, it is used widely in the industry for advanced development.

2 The fundamentals of real-time modelling

2.1 Mathematical tools

2.1.1 Simulation time

The FPGA clock cycles are discrete steps, so the HDL coder only supports discrete time codes. Therefore, the Matlab Simulink model is in discrete time as well.

2.1.2 Number representation

There are two options for number representation, these are floating-point and fixed-point data types.

Floating-point data type numbers consist of three parts, a signum bit, exponent bits and fraction bits. The fraction bits represent the number itself, and the exponent bits determine where the binary point is. Floating-point representation can represent numbers in an enormous scale.

Fixed-point numbers, on the other hand, have a signum bit and an integer part, with a pre-defined binary point. If the range of the representable number is known precisely, fixed-point representation is more accurate than floating-points.

But if we don’t know the exact range, then the following problems may occur:

- Its range is much lower than using floating-points. If the representable number is bigger than the limit, the fixed-point number overflows.
- If the representable number is small compared to the maximum representable value, it will contain a lot of unnecessary zeros, and it will be inaccurate.

I use only fixed-point data types in the model, because it is much faster for FPGA calculations, and it is pretty easy to know the ranges of the variables in the model.

2.1.3 Discrete numerical approximation methods

For approximating differential equations solutions, we can use any of the Runge-Kutta methods. The first-order Runge-Kutta method is the Euler method. The explicit Euler method looks like the following:

$$y[n+1] = y[n] + T_s \cdot f(y[n], t[n])$$

Where *y* stands for the variable, t stands for time and T_s stands for the fundamental step size.

While the implicit Euler-method:

$$y[n+1] = y[n] + T_s \cdot f(y[n+1], t[n+1])$$

These are one-step solutions, which are really simple and can be calculated very quickly.

There are more complicated Runge-Kutta methods, like the fourth-order and the second-order Runge-Kutta. For example, the second-order explicit Runge-Kutta method is a two-step method, which calculates as follows:

$$k_1 = f(y[n], t[n]), k_2 = f(y[n-1] + 0.5 \cdot Ts \cdot k_1, t[n-1] + 0.5 \cdot Ts)$$

$$y[n+1] = y[n] + Ts \cdot \ k_2$$

The error of the Runge-Kutta methods are determined by the fundamental step size. The global error of each method is proportional to the step size raised to the power of the Runge-Kutta's order.

Therefore, these higher-order Runge-Kutta methods are much more precise, than the Euler method, but they are more complicated, and are much harder to compute.

With using higher-order Runge-Kutta methods, the simulation can be more precise. However, with the Euler-method, the easier computations make me able to reduce the fundamental step size. The step size can be reduced so low, that it will not be comparable to the system time constants, so higher-order Runge-Kutta methods will not be necessary. Reducing the step size is very good for the other calculations in the model as well.

For example, here is the differential equation of the inductivity, which I want to implement in my model in discrete time:

$$U_L(t) = L \cdot \frac{di_L(t)}{dt}$$

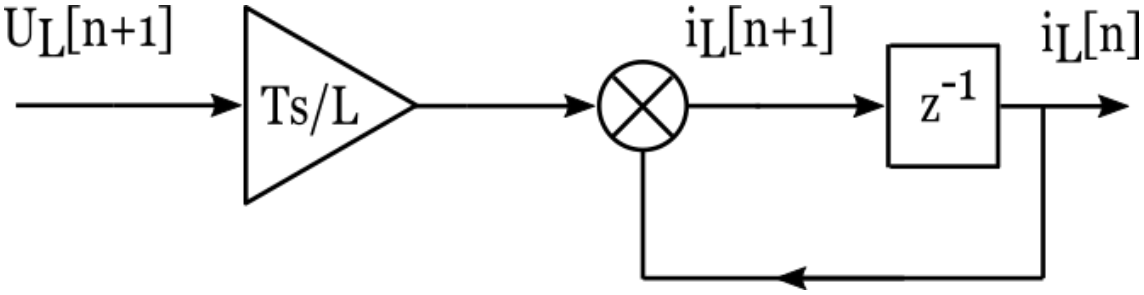
Examining a step size interval:

$$\int_0^{T_s} di_L(t) = \int_0^{T_s} \frac{U_L(t)}{L} dt$$

Converting the equation using the backward Euler method:

$$i_L[n+1] = i_L[n] + \frac{U_L[n+1]}{L} T_s$$

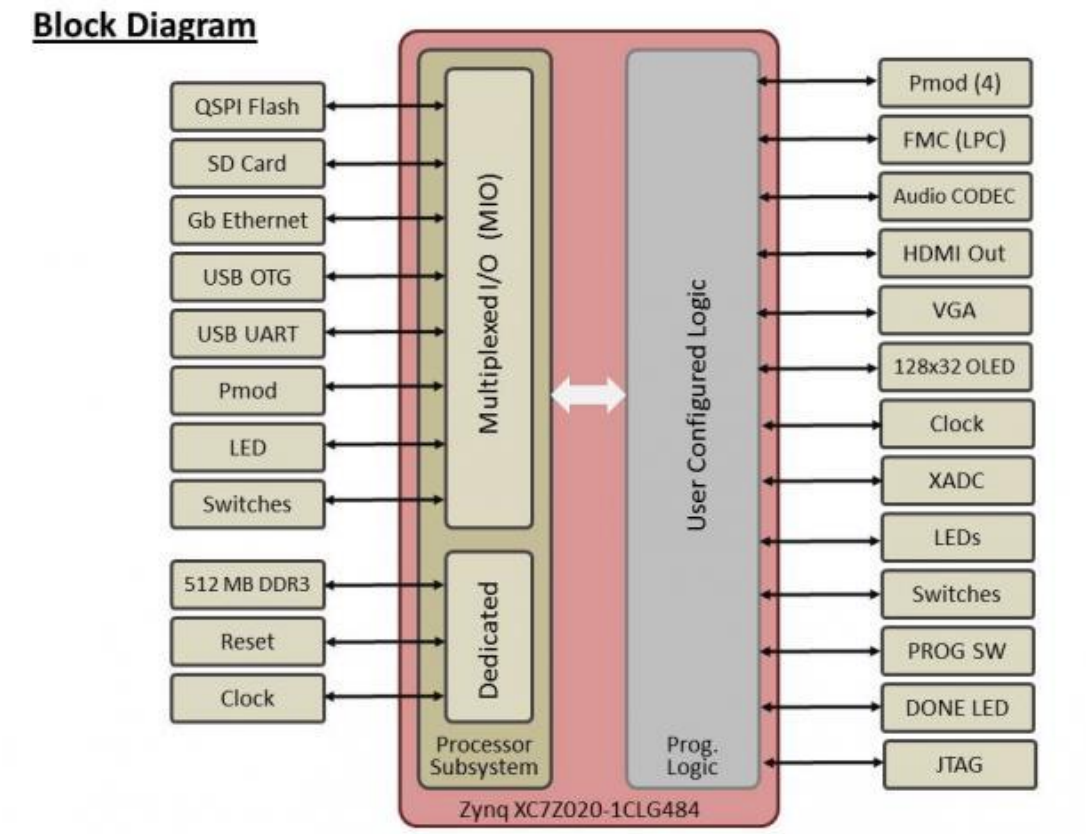
I use the implicit (forward) Euler method, because I want to determine the current of the inductivity for the same clock cycle, that is in the input voltage, not the previous one. The signal flow:



1. Figure: Signal flow of the inductivity Euler

2.2 Technical environment

For the implementation, I use the Zedboard Rev D. The Zedboard is a development kit containing a high-performance FPGA Artix and an ARM A9 processor with an SD card. It also has plenty of interfaces, like micro-USBs, ethernet, Pmods and switches. The block diagram of the Zedboard:



2. Figure: Zynq 7000 Block diagram [4]

The ARM processor functions as an application processor, it is responsible for signal processing and communication, while the FPGA is running the model. They have on-chip connection, therefore they are called system on chip (SoC).

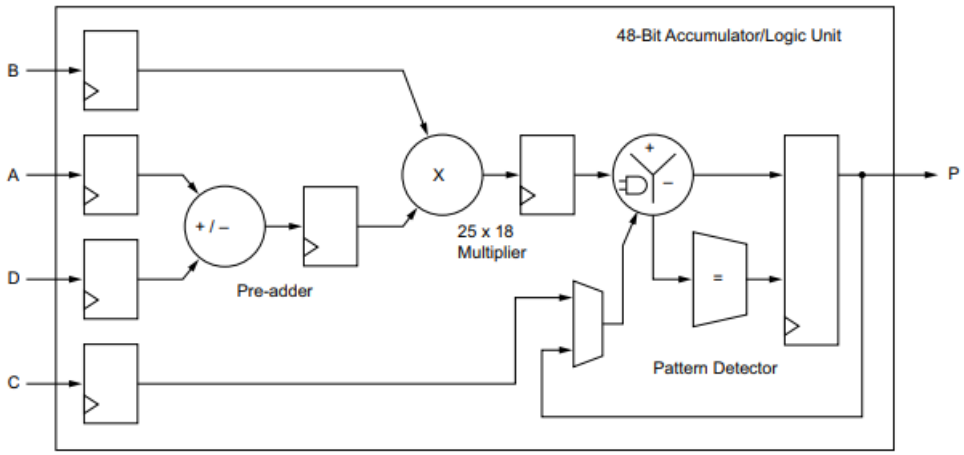
The SoC also has a 512 MB DDR RAM, which can be reached from the FPGA and accepts data with FPGA clock rate. It is a huge data container that improves traceability.

The FPGA stands for Field-Programmable Gate Array. It is an integrated circuit designed to be configurable. The advantage of the FPGA compared to a processor is that the FPGA can operate in parallel and process multiple signal flows at the same time. While processors have faster clock, they follow instructions and can process only one signal at a time.

The basic components of the FPGA are flip-flops (registers) and lookup-tables (LUTs). Flip-flops are responsible to store data between operations in the FPGA. They require a clock signal.

Lookup-tables are basic blocks in the FPGA, which can implement any logic functions of the input variable, like a truth table.

Besides these blocks, the FPGA contains digital signal processing (DSP48) blocks, which are made for fast arithmetic calculations (adding, multiplying).



3. Figure: The block diagram of the DSP 48 in the FPGA

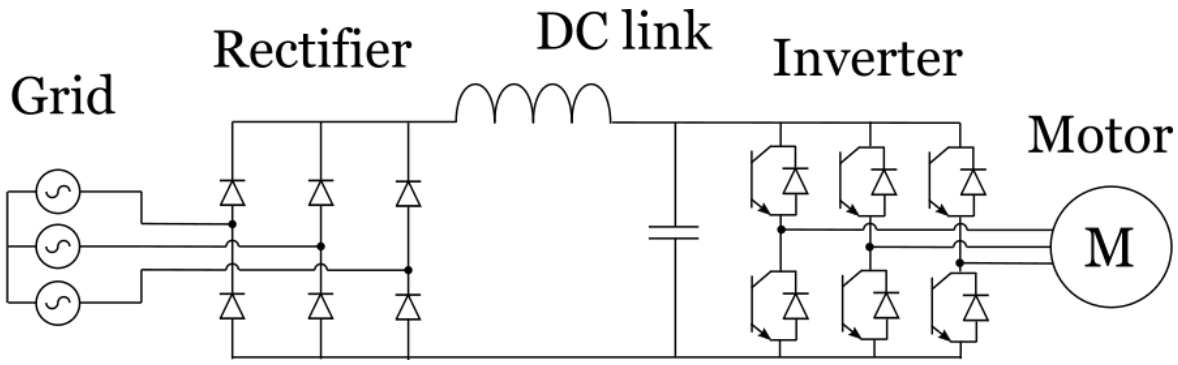
The DSP blocks have 18x25 bit multipliers, which I will take into consideration when choosing the fixed-point data types. The FPGA can implement multiplications with wider bits, but it costs more resources, more DSP48 blocks.

The Zedboard's FPGA has a clock range of 5-500 MHz. The implemented design can have multiple clock signals, and it is really important to choose the proper ones to operate with. A faster clock enables more accurate results and make the output easier to filter by a low pass filter. On the other hand, if the clock is too fast, the FPGA will not have enough time for its calculations and will not be able to meet its timing constraints. This will lead to the "worst negative slack" of the design to be negative. The optimal design has as fast of a clock as possible, with the worst negative slack staying positive.

An FPGA can add or subtract pretty fast compared to multiplication, which takes more time. Division is much slower than multiplication, so I want to avoid that if possible.

3 Creation of the system elements real-time models

The model is a three-phase converter model with a grid and an asynchronous machine:

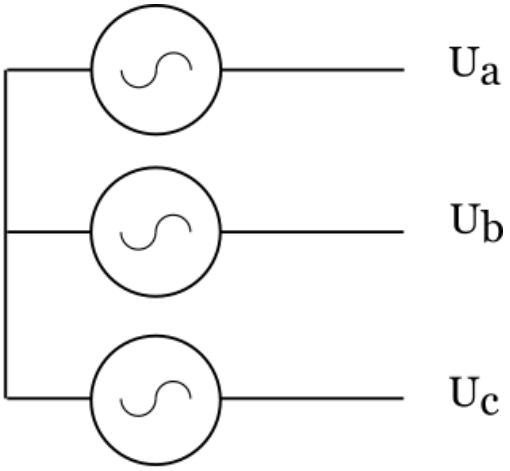


4. Figure: The model

I will interpret the models one by one, starting from the grid to the motor outputs.

3.1 Grid model

The Grid model has three phase voltage generators and there are no impedances like grid inductances. Each phase has a 50 Hz frequency, 230 V effective voltage. The three phases are symmetrical by default, but the model supports asymmetric grid voltages. The model calculates the grid phase voltages (U_a, U_b, U_c) as its output. The schematic circuit diagram of the grid:

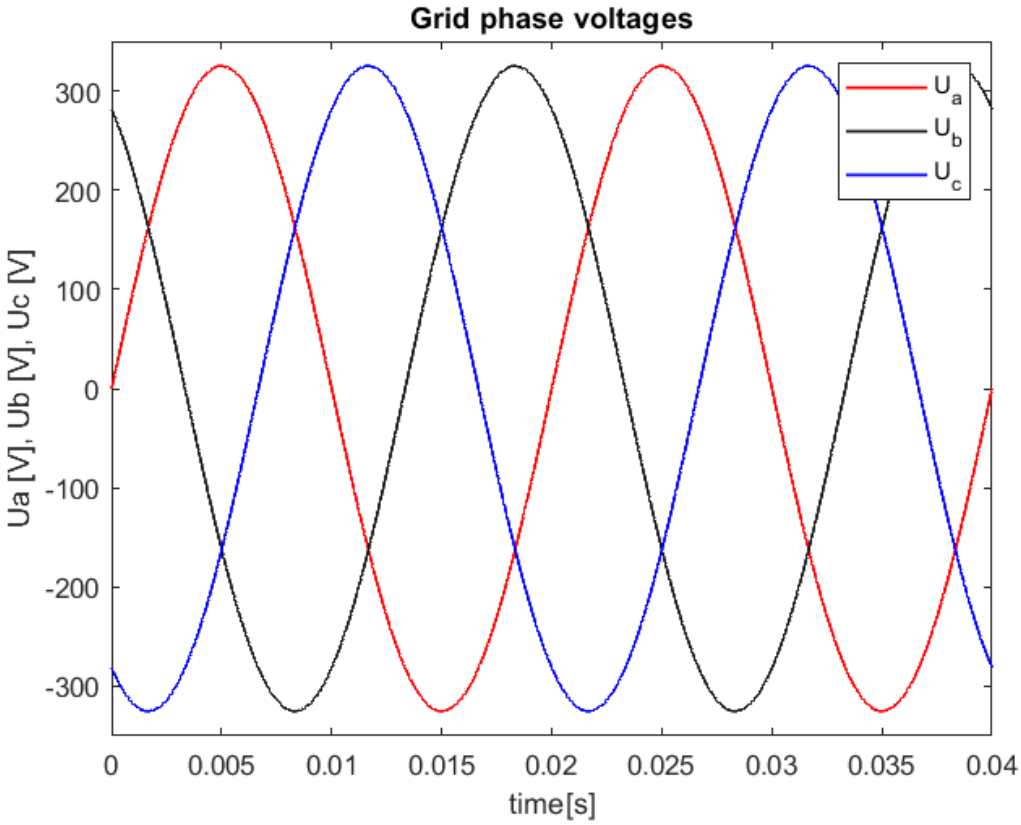


5. Figure: Schematic of the grid

The model has three sine wave generators, with 50 Hz frequency and 325 amplitude, because I want to model a 230 V effective phase. The U_a phase doesn't have an offset, the U_b phase has a 120° offset and the U_c phase has a 240° offset. These will add up to a symmetrical three-phase system. The model only includes fundamental frequency voltages, can be improved to model harmonics later on.

The sine wave generators are from Matlab Simulink DSP toolbox and compute their output from a lookup-table.

The output time function of the grid, with the phase voltages, plotted via Matlab:

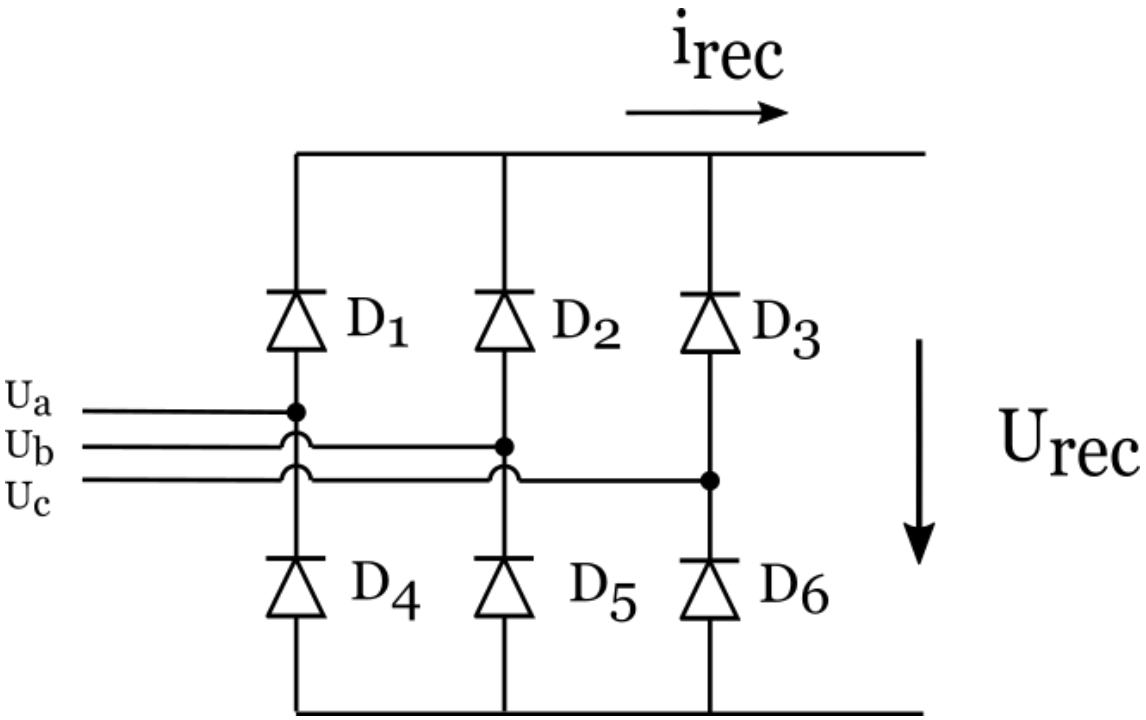


6. Figure: Grid phase voltages

The picture shows that the model produces what I expected, symmetrical three phases, each 50 Hz and 325 V peak.

3.2 Rectifier

The rectifier model contains 6 diodes, 2 for each phase. Its input is the 3 phase voltages (U_a, U_b, U_c) from the grid model and the rectified current (i_{rec}) from the DC link. Its output is the rectified voltage (U_{rec}). The schematic circuit diagram of the rectifier:



7. Figure: Schematic of the rectifier

I use logic operations to compute the output voltage. The basic idea is that on the high side, the phase with the highest potential has the conducting diode. On the low side, the phase with the lowest potential has the conducting diode.

First, the model checks, if the U_a phase voltage is higher than the U_b and U_c voltages. If it is, the model passes it into the positive potential of the rectifier, U_{rec}+. If not, the model checks whether U_b or U_c is higher and passes it to the U_{rec}+ signal.

The low side of the bridge computes in a completely analogic way. The model checks if the U_a phase voltage is lower than the U_b and U_c voltages. If it is, the model passes it into the negative potential of the rectifier, U_{rec}-. If not, the model checks whether U_b or U_c is lower and passes it to the U_{rec}- signal.

Then, the model computes the original rectified voltage ($U_{rec,org}$), with the following equation. This does not contain the diode voltage drops.

$$U_{rec,org} = U_{rec+} - U_{rec-}$$

As shown in a table:

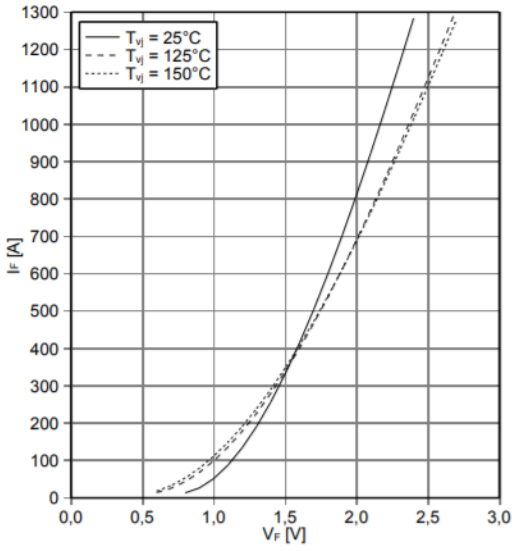
Phase voltages	**U_{rec}+**	**U_{rec}-**	**$U_{rec,org}$**
$U_a > U_b > U_c$	U_a	U_c	$U_a - U_c$
$U_a > U_c > U_b$	U_a	U_b	$U_a - U_b$
$U_b > U_a > U_c$	U_b	U_c	$U_b - U_c$
$U_b > U_c > U_a$	U_b	U_a	$U_b - U_a$
$U_c > U_a > U_b$	U_c	U_b	$U_c - U_b$
$U_c > U_b > U_a$	U_c	U_a	$U_c - U_a$

The average voltage $U_{rec,avg}$ of the rectifier can be calculated from the phase peak voltages (U_{peak}, assumes symmetrical phase voltages), with the following equation:

$$U_{rec,avg} = \frac{3 \cdot \sqrt{3} \cdot U_{peak}}{\pi}$$

With my model having 325 V peak voltages, the result of the equation is around 537.5 V

The diode drops (U_{diode}) are determined by the rectified current. The model contains a lookup table, which assigns the voltage drop to the rectified current. I modelled the forward characteristics of a power diode:



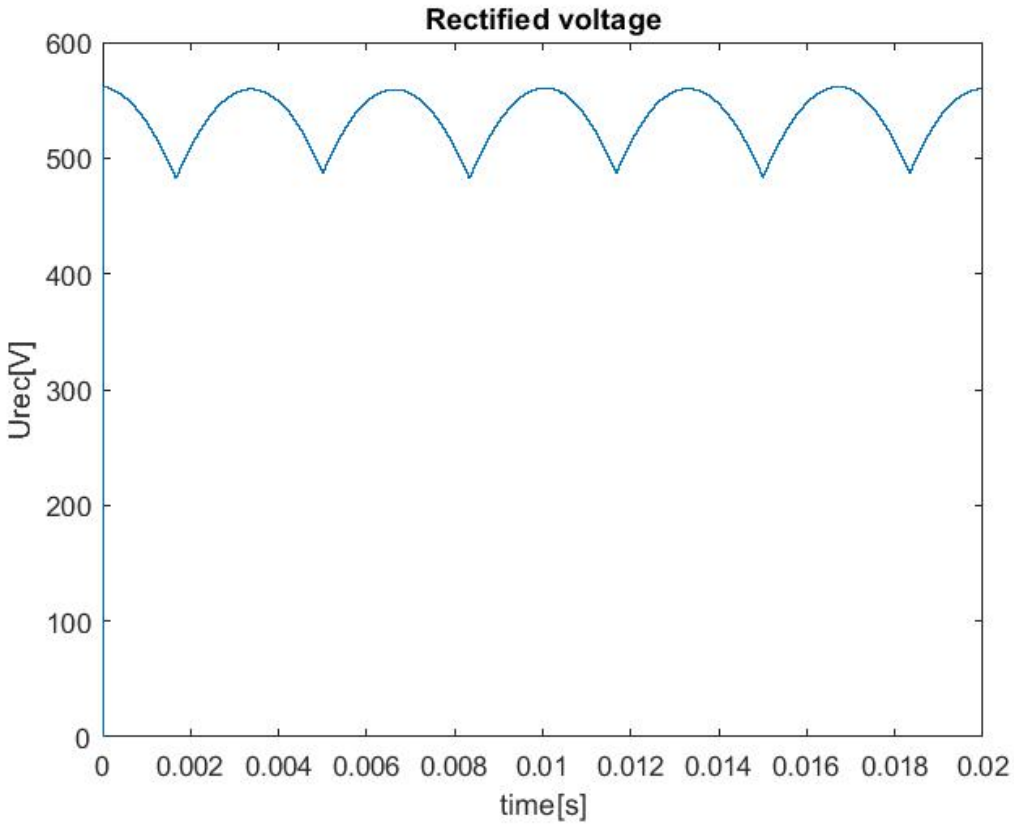
8. Figure: Forward diode characteristics [2]

The 1-dimensional lookup table models the 25°C line. HDL Coder doesn't support interpolation (it would be too much computing anyway), so I filled up a 64-element lookup-table, so it is accurate without the extrapolation.

The diode drops at a certain moment are the same on the high and the low side, because the diodes are usually same (they are modelled as being equivalent), and the same current flows through them. Therefore, the calculated rectified voltage (U_{rec}):

$$U_{rec} = U_{rec,org} - 2 \cdot U_{diode}$$

The output time function of the rectifier, with the rectified voltage, plotted via Matlab:

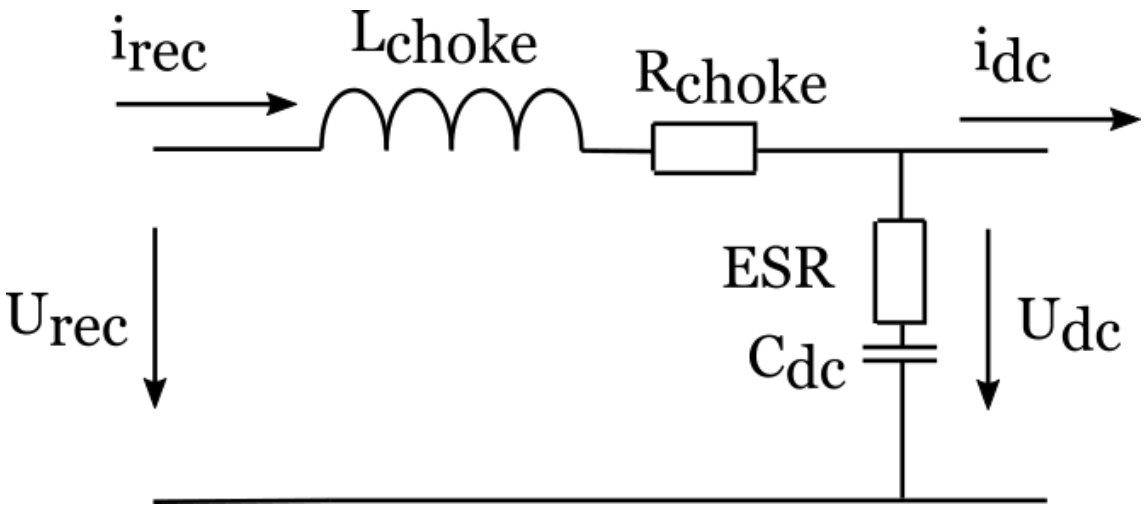


9. Figure: Rectified voltage

The expected average voltage of ideal rectifier would be 537.5 V. The rectified voltage on the picture is a tiny bit lower, which is expected due to the diode drops.

3.3 DC link

The DC link model contains a choke inductivity and a DC capacitor. The DC link's input is the rectified voltage (U_{rec}) from the rectifier and the DC current (i_{dc}) from the inverter model. Its outputs are the DC voltage (U_{dc}) and the rectified current (i_{rec}). The schematic circuit diagram of the DC link:



10. Figure: Schematic of the DC link

The inductivity and capacitor models use the formerly presented Euler method. The choke's input is its voltage (U_{choke}), its output is its current (i_{choke}). The inductivity Euler:

$$i_L[n+1] = i_L[n] + \frac{U_L[n+1]}{L} T_s$$

The choke inductivity model has a resistor model connected in series, modelling the choke resistance. The simple resistor model calculates its voltage from its current via Ohm's law:

$$U_R[n] = R \cdot i_R[n]$$

The choke inductivity and resistance are connected in series, so their current will be the same and their voltage will add up:

$$i_R[n] = i_L[n] = i_{choke}[n]$$

$$U_L[n] = U_{choke}[n] - U_R[n]$$

The DC capacitor's input is its current (i_{DCcap}), its output is its voltage (U_{DCcap}). The capacitor Euler:

$$U_C[n+1] = U_C[n] + \frac{i_C[n+1]}{C} T_s$$

The capacitor also has a resistor model in series with it, modelling the equivalent series resistance (ESR) of the capacitor, calculating the same way as the choke resistance.

The capacitor and its ESR are also in series, so their current will be the same and their voltage will add up.

$$i_{ESR}[n] = i_C[n] = i_R[n] = i_{DCcap}[n]$$

$$U_{DCcap}[n] = U_C[n] + U_{ESR}[n]$$

Based on the schematic circuit diagram of the DC link, we can calculate the following equations:

$$U_{choke}[n] = U_{rec}[n] - U_{DC}[n]$$

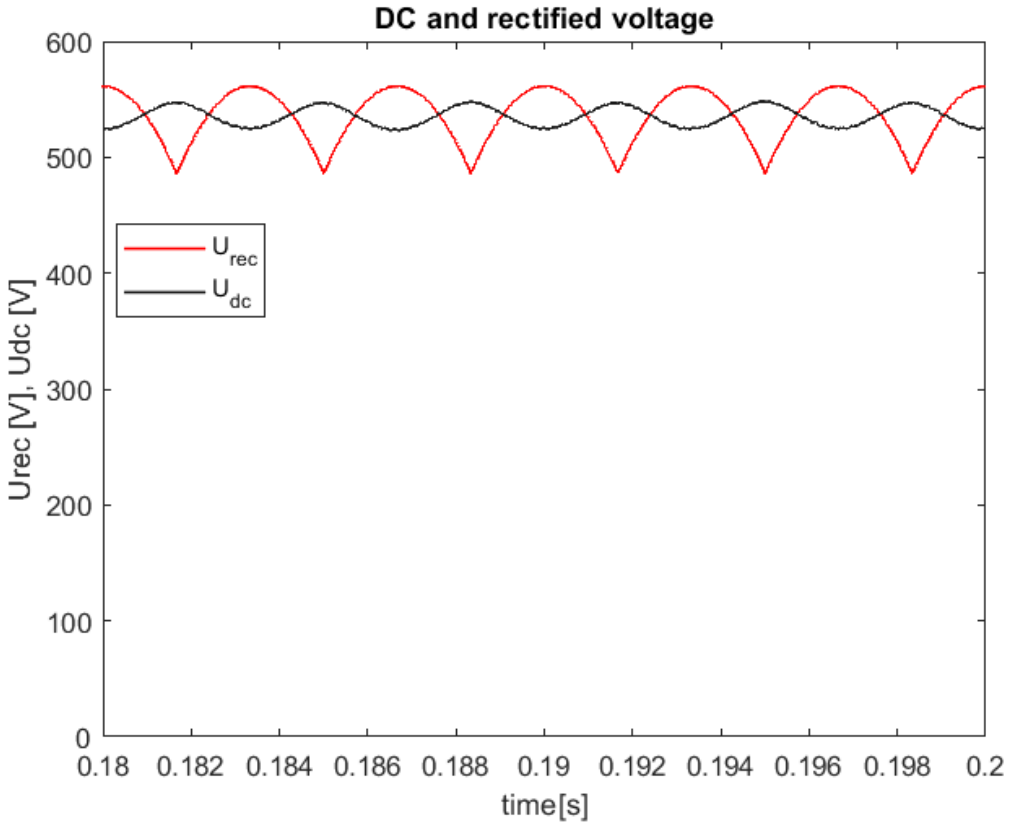
$$i_{DCcap}[n] = i_{rec}[n] - i_{DC}[n]$$

$$i_{rec}[n] = i_{choke}[n]$$

$$U_{DC}[n] = U_{DCcap}[n]$$

So, we have the inputs of the choke and the DC capacitor, and the outputs of the DC link.

The DC link's purpose is to reduce the voltage and the current ripple. The DC voltage (blue) compared to the rectified voltage (red), plotted via Matlab:

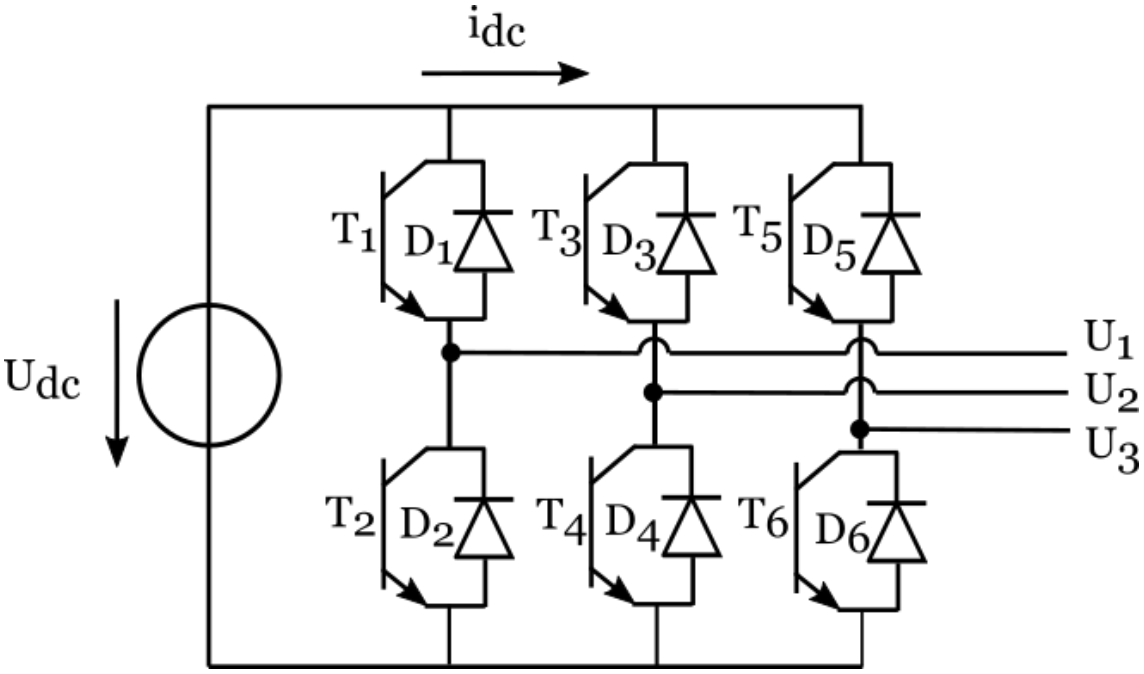


11. Figure: DC voltage and Rectified voltage

As we can see in the plot, the DC voltage ripple is lower than the rectified voltage ripple, as the effect of the DC capacitor.

3.4 Inverter

The inverter contains three bridge branches, each of them having a 2 transistor-diode pairs. Its input is the DC voltage (U_{dc}) from the DC link, the transistor control signals from the control system, (T_1, T_2, T_3, T_4, T_5, T_6) and the motor phase currents (i_1, i_2, i_3). Its outputs are the three inverted phase voltages (U_1, U_2, U_3) for the motor, the DC current (i_{dc}), and the error signal. The schematic circuit diagram of the inverter:



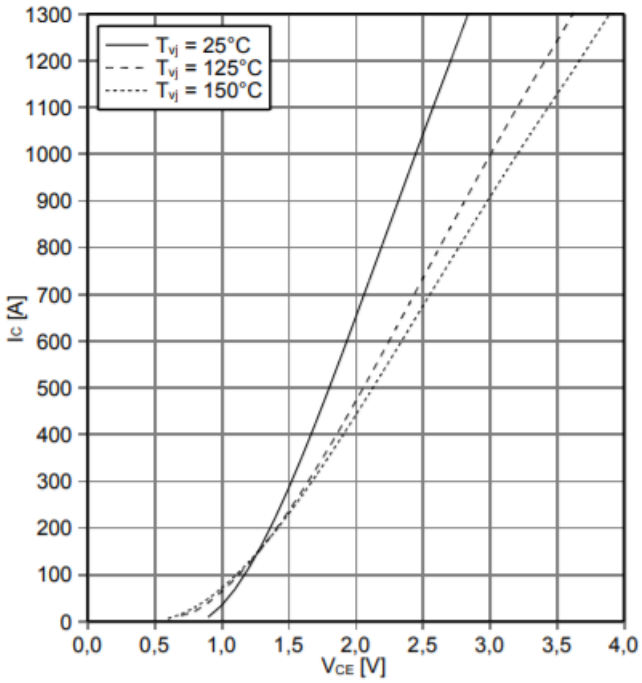
12. Figure: Schematic of the inverter

The inverter model separates into the three bridge branches. Each branch determines the phase voltage (U_{ph}) and the error signal via the high and low transistor signals (T_{high}, T_{low}) and the phase current (i_{ph}).

T_{high}	T_{low}	i_{ph}	Error	U_{ph}
1	0	+	0	$U_{dc} - U_{tr,high}$
1/0	0	-	0	$U_{dc} + U_{diode,high}$
0	1	-	0	$0 + U_{tr,low}$
0	0/1	+	0	$0 - U_{diode,low}$
1	1	+/-	1	short circuit

Once the T_{high} and the T_{low} are true at the same time, the simulation throws an error signal, and the other results will be irrelevant. The error signal is fed into an SR flip-flop, so the error signals remains as long as we don't reset the SR.

The model computes the semiconductor drops, U_{tr} and U_{diode} via lookup-tables. The diode uses the same lookup as the rectifier does, but it can be set to a different one if we know what can of diode we want to model. The transistor lookup models the output characteristics of an IGBT:



13. Figure: The output characteristic of the IGBT [2]

The model also takes transistor deadtimes into consideration. The deadtime block's inputs are the control signals, its outputs are the T signals to the bridge branch blocks. It starts a counter when the control signal changes from zero to one and this change will only appear on its output when the counter has counted up to a certain value, determined by the turn-on delay and rise time of the transistor [2]. The same happens in case of a turn off, a counter starts, and the output only turns off when the counter says so.

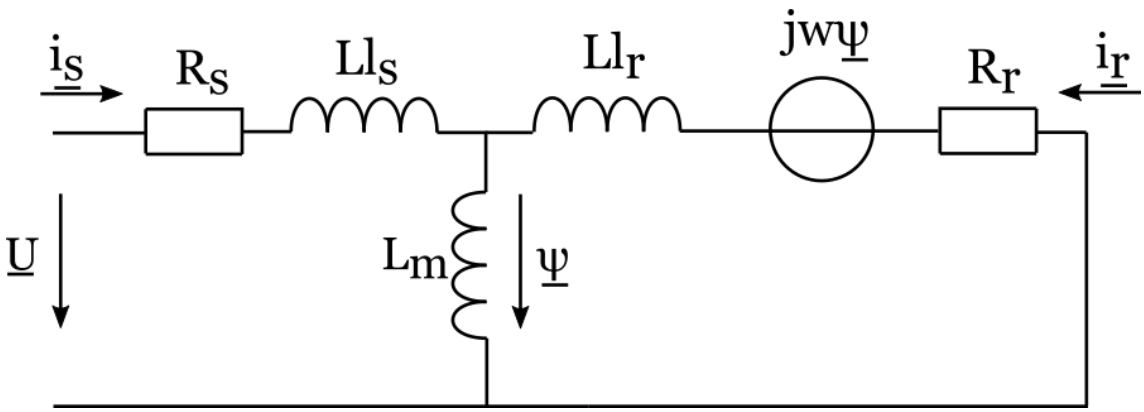
The deadtime block can functionally operate when the control signals change their value back while the counters are still counting, and the change of the output signal is declined.

The inverter block calculates the DC current (i_{dc}) for the DC link. If it would simply add all the motor phase currents, the DC current would be just be zero (if the control signals are correct). Instead, it adds the currents flowing through the high side of the transistor, which will result in the DC current. Each branch has a "phase DC current" as its output, which is the phase current if the high transistor or diode conducts (T_{high} =1, or T_{low}=0 and i_{ph}<0), and 0 otherwise. The inverter model adds these up and gets the DC current.

3.5 Asynchronous machine

The asynchronous machine model operates in standing Descartes coordinates transforms to rotating coordinates in the end. It uses a reduced parameter equivalent circuit. Its inputs are the three phase voltages (U_1, U_2, U_3) and the load torque (. Its outputs are the three phase currents, the rotor angular speed and rotor position, and every electrical parameter in either x-y or d-q coordinates, which can be used for testing.

The schematic of the equivalent circuit of the squirrel-cage asynchronous machine:



14. Figure: The schematic of the asynchronous machine

The parameters and the variables are the following:

$\underline{U}$	Stator voltage vector
$\underline{\Psi}$	Inducted flux vector
$\underline{i}_s$, $\underline{i}_r$	Stator and rotor currents
R_s, R_r	Stator and rotor resistances
Ll_s, Ll_r	Stator and rotor leakage inductances
L_m	Magnetizing inductance
w	Electric angular speed of the rotor
$L_s = Ll_s + L_m$	Stator net inductance
$L_r = Ll_r + L_m$	Rotor net inductance

3.5.1 The Clarke and inverse Clarke transformations

The Clarke transformations are used to transform the motor phase voltages (U_1, U_2, U_3) to standing Descartes coordinates (U_x, U_y). The Clarke transformation in matrix format:

$$\begin{bmatrix} U_x \\ U_y \\ U_z \end{bmatrix} = \frac{2}{3} \begin{bmatrix} 1 & -0.5 & -0.5 \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ 0.5 & 0.5 & 0.5 \end{bmatrix} \cdot \begin{bmatrix} U_a \\ U_b \\ U_c \end{bmatrix}$$

The U_z component is the zero-sequential voltage component and does not influence the motor calculations, so the model does not calculate it.

The inverse Clarke transformations are used to transform the x-y coordinate currents back (i_x, i_y) to phase currents (i_a, i_b, i_c). The inverse Clarke transformation in matrix format:

$$\begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ -0.5 & \frac{\sqrt{3}}{2} & 1 \\ -0.5 & -\frac{\sqrt{3}}{2} & 1 \end{bmatrix} \cdot \begin{bmatrix} i_x \\ i_y \\ i_z \end{bmatrix}$$

The i_z current is assumed to be 0.

3.5.2 Parameter reduction

The equivalent circuit seen above would be hard to model, because one inductivity would need to be driven by current, which cannot be modelled via Euler method functionally. Therefore, reduced parameters are introduced, so the model does not need to model the inductivity driven by current.

The reduction starts from the basic flux equations of the motor:

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

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

I want the member with the $\underline{i_r}$ to disappear, so I introduce “a” reduction factor.

$$a = \frac{L_m}{L_r}$$

The new reduced values, L_{mr} and i_r^*:

$$L_m \cdot \underline{i_r} = (L_m \cdot a) \cdot \left(\frac{\underline{i_r}}{a}\right) = L_{mr} \cdot \underline{i_r}^*$$

The stator flux with the reduced values:

$$\underline{\psi_s} = L_s \cdot \underline{i_s} + L_{mr} \cdot \underline{i_r}^*$$

The stator current must remain the same through the reduction, so the reduced magnetizing current has to change:

$$\underline{i_s} = \underline{i_m} - \underline{i_r} = \underline{i_m}^* - \underline{i_r}^*$$

The flux equation reduced, with the reduced stator leakage inductivity, Ll_{sr}:

$$\underline{\psi_s} = L_s \cdot \underline{i_s} + L_{mr} \cdot \underline{i_r}^* = L_s \cdot \underline{i_s} + L_{mr} \cdot \left(\underline{i_m}^* - \underline{i_s}\right)$$

$$\underline{\psi_s} = (L_s - L_{mr}) \cdot \underline{i_s} + L_{mr} \cdot \underline{i_m}^* = Ll_{sr} \cdot \underline{i_s} + L_{mr} \cdot \underline{i_m}^*$$

The rotor flux with the reduced values:

$$\underline{\psi_r} = L_m \cdot \underline{i_s} + L_r \cdot \underline{i_r} = L_m \cdot \left(\underline{i_m}^* - \underline{i_r}^*\right) + L_r \cdot \underline{i_r}^* \cdot a$$

$$\underline{\psi_r} = L_m \cdot \underline{i_m}^* + \left(L_m - L_r \cdot \frac{L_m}{L_r}\right) \cdot \underline{i_r}^* = L_m \cdot \underline{i_m}^*$$

As we can see, the i_r successfully disappeared, so we don't have to model the additional inductivity. Additionally, I introduce the reduced rotor flux because I want to use the reduced L_{mr} parameter instead of the original L_m.

$$\underline{\psi_r}^* = \underline{\psi_r} \cdot a = L_m \cdot \underline{i_m}^* \cdot a = L_{mr} \cdot \underline{i_m}^*$$

Now the flux equations work in the reduced model. Let's see the voltage equations. The stator voltage equation for the squirrel-cage asynchronous machine, from a standing coordinate system:

$$\underline{U_s} = R_s \cdot \underline{i_s} + \frac{\partial \underline{\psi_s}}{\partial t}$$

The stator current and flux are the same in the reduced parameter model, so the stator resistance can remain the same as well.

The rotor voltage equation for the squirrel-cage asynchronous machine, from a standing coordinate system:

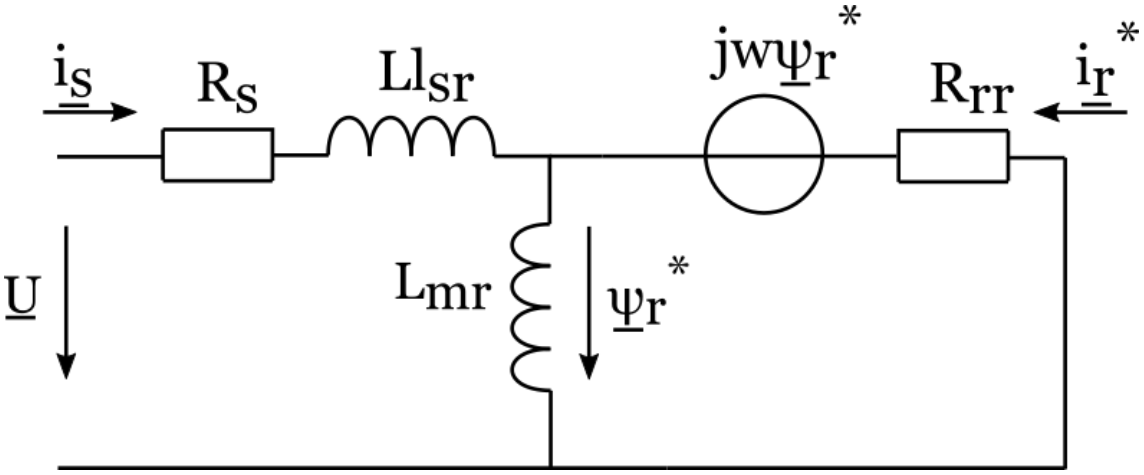
$$0 = R_r \cdot \underline{i_r} + \frac{\partial \psi_r}{\partial t} - j\omega \underline{\psi_r}$$

Now with the reduced parameters, introducing the reduced rotor resistance, R_{rr}:

$$0 = R_r \cdot \underline{i_r}^* \cdot a + \frac{\left(\frac{\partial \psi_r}{\partial t}^* - j\omega \underline{\psi_r}^*\right)}{a} = R_r \cdot \underline{i_r}^* \cdot a^2 + \left(\frac{\partial \psi_r}{\partial t}^* - j\omega \underline{\psi_r}^*\right)$$

$$0 = R_{rr} \cdot \underline{i_r}^* + \frac{\partial \psi_r}{\partial t}^* - j\omega \underline{\psi_r}^*$$

The reduction is complete. The equivalent schematic circuit reduced:



15. Figure: Schematic of the reduced equivalent circuit

Where the reduced parameters can be calculated from the original ones, via the eqations above. Summarizing the reduced parameters:

$$a = \frac{L_m}{L_r}$$

$$L_{mr} = L_m \cdot a$$

$$Ll_{sr} = L_s - L_{mr}$$

$$R_{rr} = R_r \cdot a^2$$

3.5.3 Model calculations

The model is built up from resistance and inductivity blocks, that compute with the same Euler method as presented earlier. The only new addition is how the model computes the fluxes from voltage, an integration with the Euler method:

$$\psi_L[n+1] = \psi_L[n] + U_L[n+1] \cdot T_s$$

The equation between current and flux, which is also used by the model:

$$\psi_L[n] = i_L[n] \cdot L$$

With all the blocks in mind let's see the equations of the equivalent circuit, that connect the blocks:

$$U_x - {U_{r,x}}^* = U_{Rs,x} + {U_{Llsr,x}}^*$$

$$U_y - {U_{r,y}}^* = U_{Rs,y} + {U_{Llsr,y}}^*$$

$${i_{r,x}}^* = {i_{m,x}}^* - i_{s,x}$$

$${i_{r,y}}^* = {i_{m,y}}^* - i_{s,y}$$

$${U_{r,x}}^* = -{i_{r,x}}^* \cdot R_{rr} - \omega {\psi_{r,y}}^*$$

$${U_{ryx}}^* = -{i_{r,y}}^* \cdot R_{rr} + \omega {\psi_{r,x}}^*$$

$$\omega = \omega_{mot} \cdot p_{mot}$$

Where w_{mot} is the motor angular speed and p_{mot} is the number of pole pairs in the motor.

The equivalent circuit model also calculates the motor electric torque (M_e). The torque equation in vectoral form, then in x-y coordinates:

$$M_e = \frac{3}{2} \cdot p_{mot} \cdot (\underline{{\psi_r}^*} \times \underline{i_s})$$

$$M_e = \frac{3}{2} \cdot p_{mot} \cdot ({\psi_{r,x}}^* \cdot i_{s,y} - {\psi_{r,y}}^* \cdot i_{s,x})$$

3.5.4 Saturation calculation

The magnetizing inductance saturates, so the on higher currents it will decrease substantially. At the current state of the model, only the magnetizing inductance saturates. If the leakage inductances saturate due to certain motor construction, it can be added to the model very easily, because the reduced parameters are coming from a lookup-table anyway. The saturation is approximated well with the following equation:

$$i_m = i_{m\,nom} \cdot \left(k \cdot \frac{\Psi}{\Psi_{nom}} + (1-k) \cdot \left(\frac{\Psi}{\Psi_{nom}} \right)^{p2} \right)$$

This is an empirical equation, where the variables are the following, and “nom” stands for nominal

$$i_{m\,nom} \approx \sqrt{2} \cdot i_{nom} \cdot \sin(\varphi)$$

Where φ is the phase angle and sin(φ) is proportional to the reactive power.

$$\Psi_{nom} = \frac{\sqrt{2} \cdot U_{nom}}{\sqrt{3} \cdot 2\pi \cdot f_{nom}}$$

U_{nom} is the nominal effective line voltage, f_{nom} is the nominal frequency.

$$k \in [0 \ldots 1]$$

$$p2 = 5 \; or \; 7$$

“k” is an empirical constant. If “k” is one, the magnetizing inductivity is linear and does not saturate. p2 determines how the equation approximates the saturation. These two constants define the nature of the saturation and can be measured for each motor.

The magnetizing inductance:

$$L_m = \frac{\Psi}{i_m}$$

Expressing the magnetizing inductance via the two equations:

$$L_m = \frac{1}{i_{m\,nom}} \left(\frac{\Psi_{nom}}{k} + \frac{1}{(1-k)} \cdot \frac{{\Psi_{nom}}^{p2}}{\Psi^{(p2-1)}} \right)$$

This equation is performed by a Matlab script. The script calculates the L_m for 128 flux value. L_m determines all the reduced parameters, which are used in the model, so the model calculates the reduced parameters by lookup tables. These lookup-tables have the flux square as their input, because the model calculates in x-y coordinates, and this way the FPGA does not have to perform a square root.

Each lookup has 128 breakpoints and values. The model wants to divide by the L_{mr} reduced inductance. A division takes muck more resources in the FPGA than multiplication, so I rather calculate the reciprocal of L_{mr} in the lookup-table to save resources. Regarding the Ll_s, the model wants to use its reciprocal multiplied by the step size. I rather do this all in the script, so I fill up the lookup in the model with the Ll_s reciprocal multiplied by the step size.

3.5.5 Normal circuit variable calculations

Throughout the reduction, most inner variables change, and they have to be transformed back to the normal circuit values, if I want to have them as outputs. The stator values, stator flux and stator current are invariant. The calculations of the variant ones:

$$i_r = i_r^* \cdot a$$

$$\Psi_r = \frac{\Psi_r^*}{a}$$

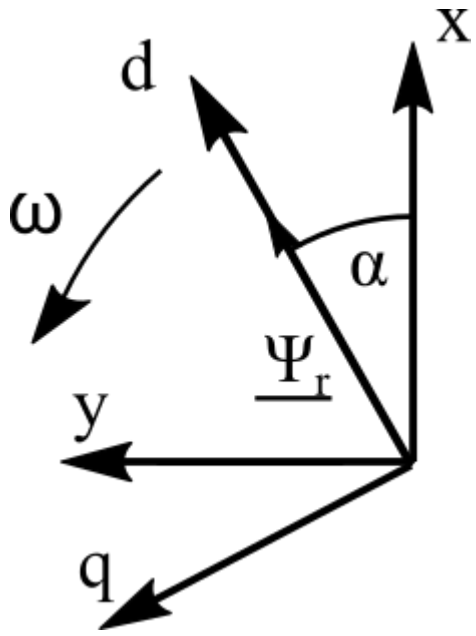
$$i_m = i_s + i_r$$

$$\Psi = \Psi_r - \mathrm{i}_r \cdot Ll_r$$

The calculation of the inducted flux is necessary even if we don't want to have it as an output, because it is the input of the parameter lookup-tables for the saturation.

3.5.6 Transformations to rotating field coordinates (d-q)

The rotating field coordinate transformations are based on the rotor flux angle (α). The figure of the x-y and d-q coordinates, including the rotor flux angle:



16. Figure: d-q transformation vectors

Its calculated by the rotating field x-y coordinates:

$$\alpha = \mathrm{atan}\left(\frac{\Psi_{r,y}}{\Psi_{r,x}}\right)$$

Each variable in x-y coordinates are transformed to d-q coordinates via the following equations which are implemented in the model's d-q transformation blocks. The v stands for the transformed variable.

$$v_d = \cos(\alpha) \cdot v_x + \sin(\alpha) \cdot v_y$$

$$v_q = \cos(\alpha) \cdot v_y - \sin(\alpha) \cdot v_x$$

3.5.7 Mechanical model

The model calculates a basic inertia differential equation, which calculates the angular speed from the motor torque and the inertia:

$$\frac{d\omega_{mot}}{dt} = \frac{1}{\vartheta} \cdot M$$

Where the motor torque is the load torque and the friction torque subtracted from the electrical torque.

$$M = M_e - M_l - M_f$$

$$M_f = \omega \cdot ff$$

M_f is the friction torque, which is proportional to the angular speed and the friction factor (ff).

The Euler equation of the mechanical model:

$$\omega_{mot}[n+1] = \omega_{mot}[n] + \frac{M[n+1]}{\vartheta} \cdot T_s$$

3.5.8 Encoder modelling

The rotary encoder converts the angular position of the rotor to a digital signal. I model an incremental encoder, which has two digital outputs. These outputs are called "a" and "b" incremental encoder outputs. Each is a square wave, which's frequency is determined by the angular speed of the rotor (f_{enc}), and the resolution (res) of the encoder:

$$f_{enc} = \frac{\omega_{mot}}{2\pi} \cdot res$$

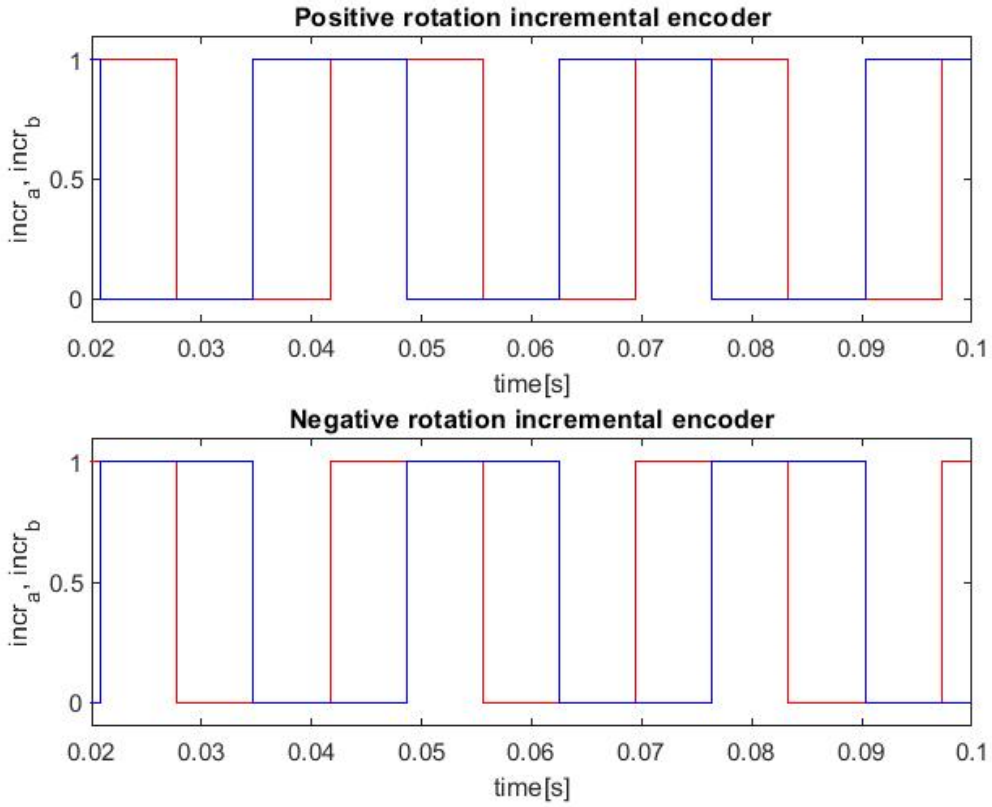
The "b" output is offset by 90° compared to the "a" output, so the observer can determine the direction of the rotor speed.

I modelled the incremental encoder with a counter, that adds the current angular speed to the counter at every clock cycle, until it reaches its limit ($incr_{limit}$), then resets to zero:

$$incr_{limit} = \frac{\frac{2\pi}{T_s \cdot res}}{2}$$

The output of the incremental encoder changes each time the counter reaches its limit. The limit is divided by two because the counter needs to have twice the encoder frequency.

“a” encoder (red) and “b” encoder (blue) outputs on a 2π and -2π angular speed, with a resolution of 36 per turn, plotted via Matlab.



17. Figure: Incremental encoder outputs

4 Design of the FPGA environment

In this chapter I will describe what changes I made to the model to make it HDL coder compatible and functional on the FPGA.

4.1 Fixed points

As default, the Simulink uses floating-point arithmetic, double. Since I want to use fixed-point number representation, I have to determine the fixed-point types of the variables.

The model has a global fixed-point data type for each physical quantity. A script calculates the fixed-point number's integer and fractional bits by the following variables: width and range.

$$integer = floor(log_2(range)) + 1$$

I calculate it this way instead of simply rounding it upwards, because that would give incorrect value if the logarithm gave an integer value.

$$fraction = width - integer - sign$$

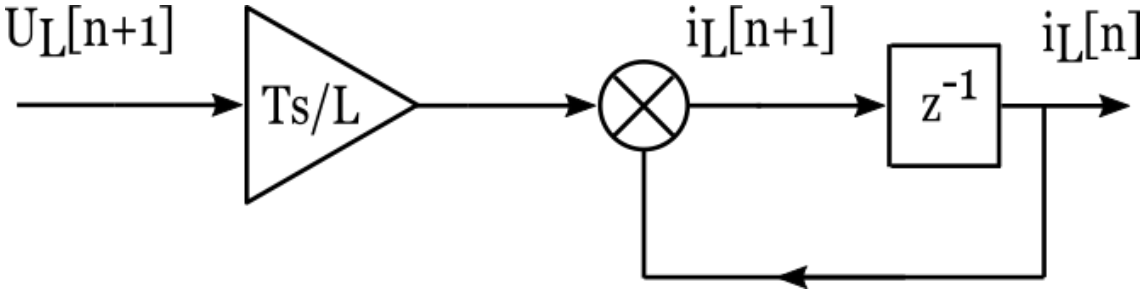
Most of the variables calculated via script are signed, so the sign bit is 1. My model has two typical number widths, 18 and 25 bits. It is because as I wrote it in a previous topic, the FPGA has 18x25 bit multipliers, and I want to optimize the model for these multipliers.

The fixed-point numbers are given in the following format:

$$fixdt(sign, width, fraction)$$

Fraction bits can be higher than the width, that representable number does not contain the binary point, it is "over" it. If the fraction bits are negative, the binary point is "under" the number.

The blocks that perform Euler equations have different fixed points and are converted back to the global ones after. Let us take a look at the signal flow of the inductivity model from a previous chapter again:



18. Figure: Signal flow of the inductivity Euler

The incoming signal is a 18 bit wide voltage value. It is multiplied by the step size divided by the inductivity, ending up as a really small number. This number will be added to the integrator, changing its previous value. This change is pretty small, and it has to be representable by the fixed point. The full current value on the other hand is a much bigger number. The data type of the adder has the fraction length of its input, and has the integer length of the output. this means that it has to be a really wide variable. This is also calculated by the script.

4.2 Clock management

20 MHz is the fastest clock cycle in the model. The most important calculations run at the 20 MHz clock, like the Euler blocks, the converter and motor calculations.

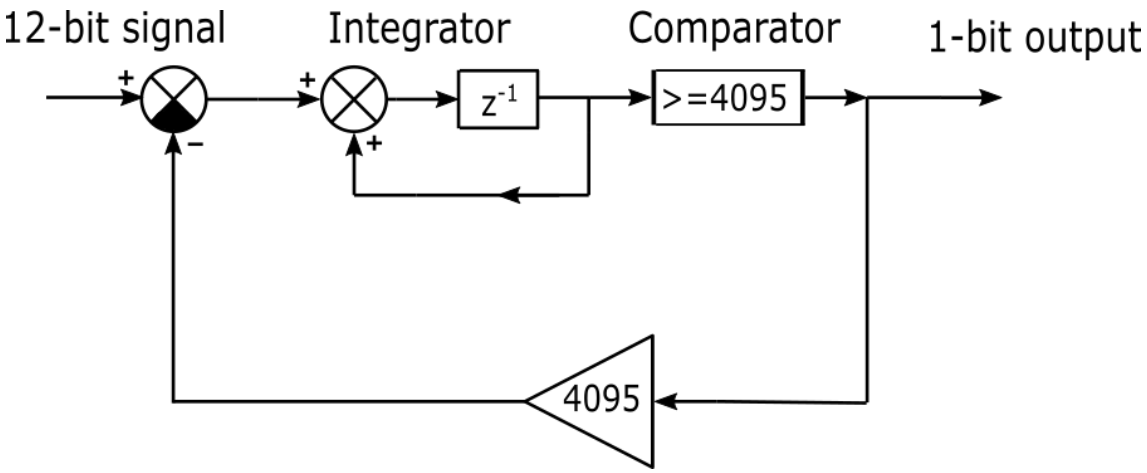
10 MHz is the other major clock cycle. Some motor calculations run in the 10 MHz clock, like the reduced to normal circuit block and the d-q transformation block which are not in feedback path. The saturation parameter calculation block also runs on the 10 MHz clock, which is in feedback path, but it is an approximation and is not important to run on the faster clock.

Sine wave generator lookup-tables run on a 500 kHz, because they do not have enough breakpoints and values to run on a faster one at 50 Hz.

4.3 Sigma-delta conversion

The output signals of the model are digital number signals, which I want to convert to analogue signals. Since the FPGA does not have analogue signals, a sigma-delta conversion is need. This S-D modulated signals feeds an RC low pass filter, which filters the clock cycle of the FPGA, resulting in an analogue signal.

The schematic of the S-D converter:



19. Figure: Schematic of the S-D converter

The input of the S-D is scaled to a 12-bit number (max 4095) via the reference value. The model subtracts 4095 from this scaled input, if the previous output was a 1. The subtracted value goes into an integrator, which is followed by a comparator, which leads to the 1-bit output of the S-D converter.

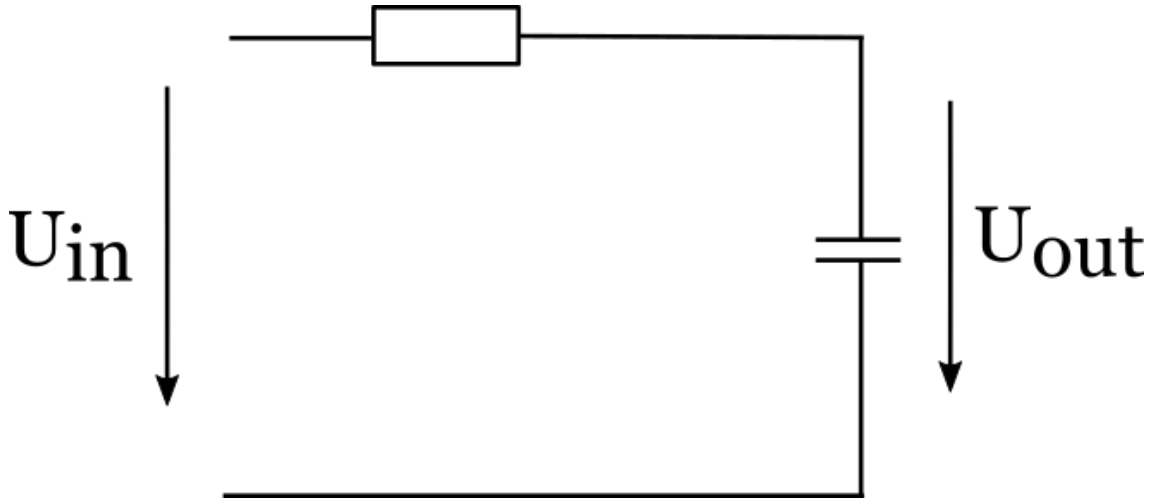
Basically, the S-D sums up the average difference of the input and the reference value, hence the name. The ratio of the input (V_{in}) and the reference (V_{ref}) value will determine the interval ratio (IR) of the S-D modulated output:

$$if\ \frac{V_{in}}{V_{ref}} \leq 0, \qquad IR = 0$$

$$if\ \frac{V_{in}}{V_{ref}} \geq 1, \qquad IR = 1$$

$$if\ 0 < \frac{V_{in}}{V_{ref}} < 1\ \ IR = \frac{V_{in}}{V_{ref}}$$

The Sigma-Delta conversion takes place inside of the FPGA. The S-D modulated outputs have to be filtered for an analogue signal. The RC filter schematic:



20. Figure: The schematic of the RC filter

I selected the values of the resistor and the capacitor. The first determined value was the resistor via the current limit of the Zedboard's PMODs, which is 1.2 mA. The PMOD outputs are 3.3 V. According to Ohm's law, the resistor has to be at least 2750 Ω. I put a standard 3 kΩ in the filter. The Zedboard has a 200 Ω serial resistance built in, meaning the capacitor has to be chosen next to a 3200 Ω resistor.

I chose a 1 nF capacitor for the filter. This way the RC filter's cutoff frequency is around 50 kHz. This means it won't filter the switch frequency effects, which can be around 30 kHz maximum. The point of the RC is to filter the FPGA clock frequency, that is 20 or 10 MHz. The RC with these parameters reduces the 10 MHz component to 0.5% and the 20 MHz component to 0.25%.

5 Verification

I based the verification on the Simulink Simpower systems toolbox. I built a Simulink model of a converter and an asynchronous machine. This machine is an ABB 18.5 kW asynchronous machine. I took the parameters of the ABB asynchronous machine and implemented my real-time model with those parameters. The semiconductor and DC link parameters can be found in the references [2,3]

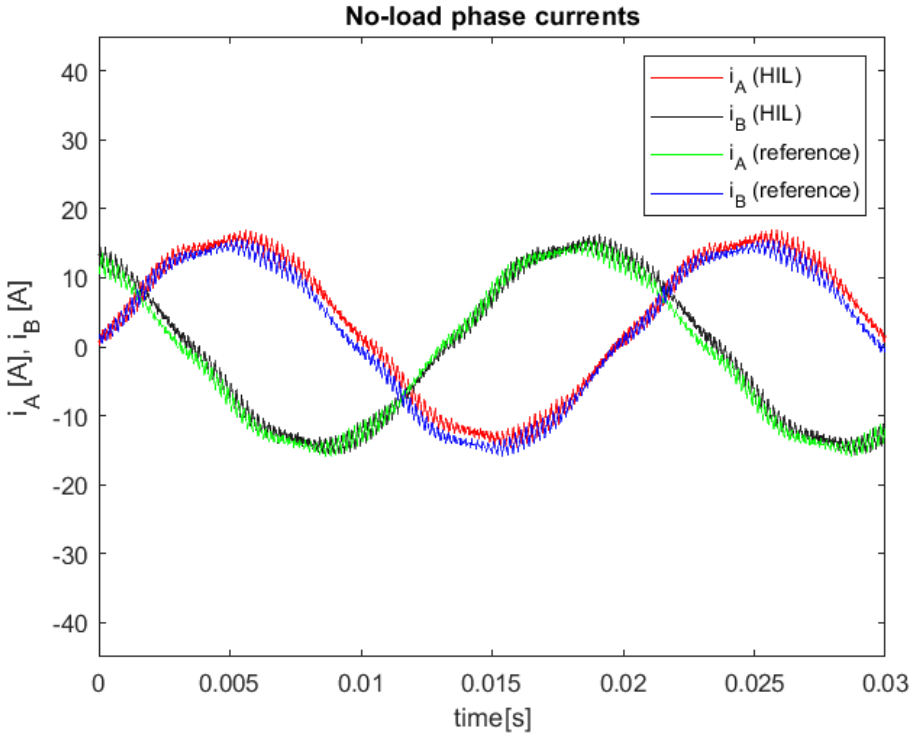
The real-time model's time functions were captured by a Digilent Analog Discovery 2, which is a PC based oscilloscope and logic analyser. The measurements were put into a .csv file and imported by Matlab. It measures the FPGA signals after S-D conversion, and an RC filter.

The inverter was driven by a basic, "dummy" control unit that I made for a 50 Hz voltage and ~ 10 kHz switching frequency.

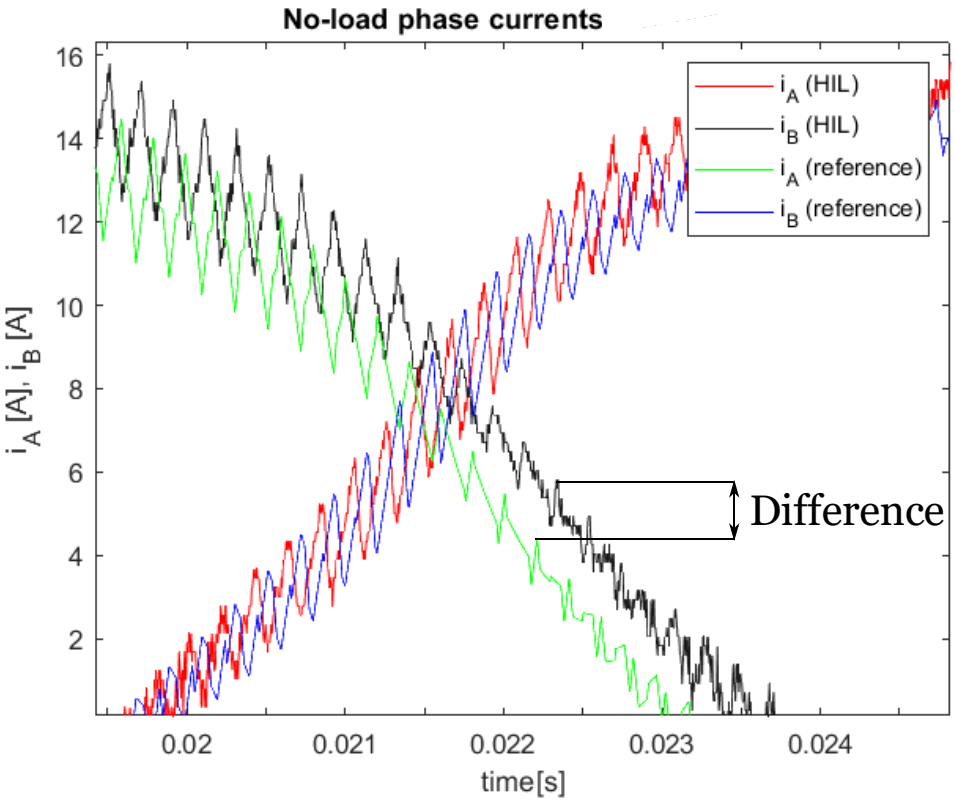
5.1 Stationary tests

I made some no-load stationary tests, and load tests with 40 NM load torque.

The phase currents from the HIL model and the reference:

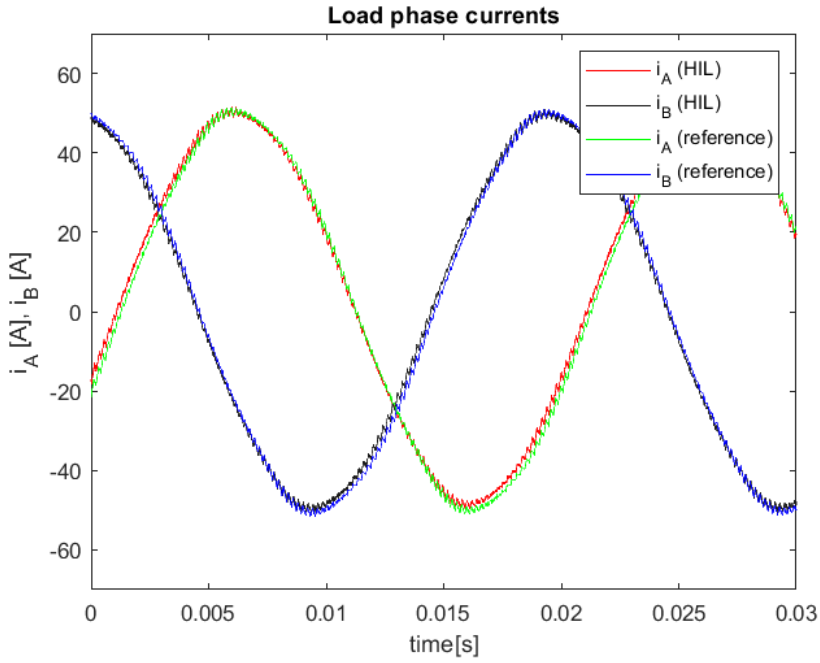


21. Figure: No-load phase currents



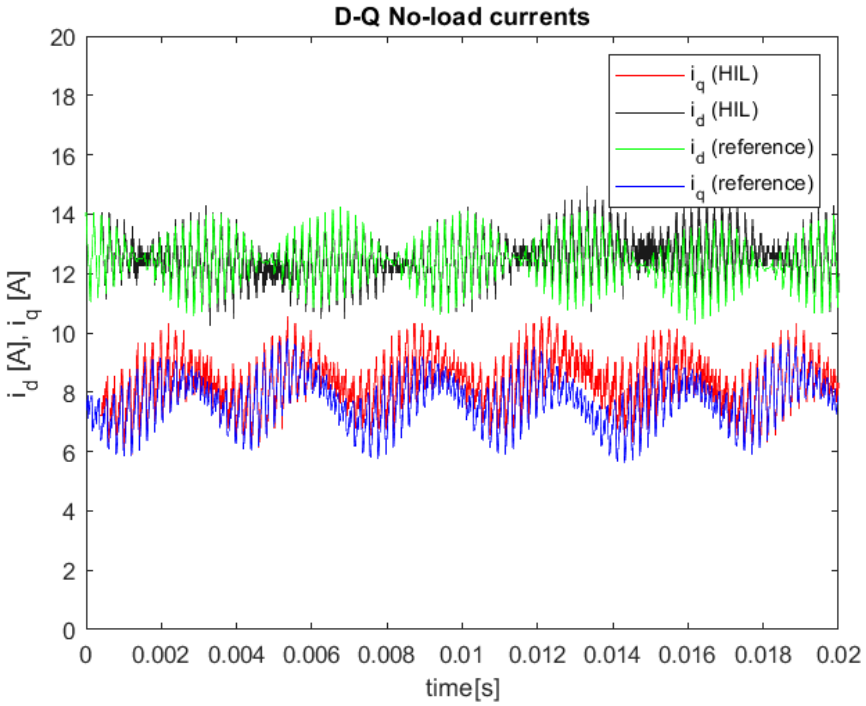
22. Figure: No-load phase current zoomed in

They are similar, both for nature and amplitude, the amplitude difference is shown in the zoomed in figure. The transistor switch frequency current ripple can be observed cleanly. The load currents will be higher, as the next figure shows:

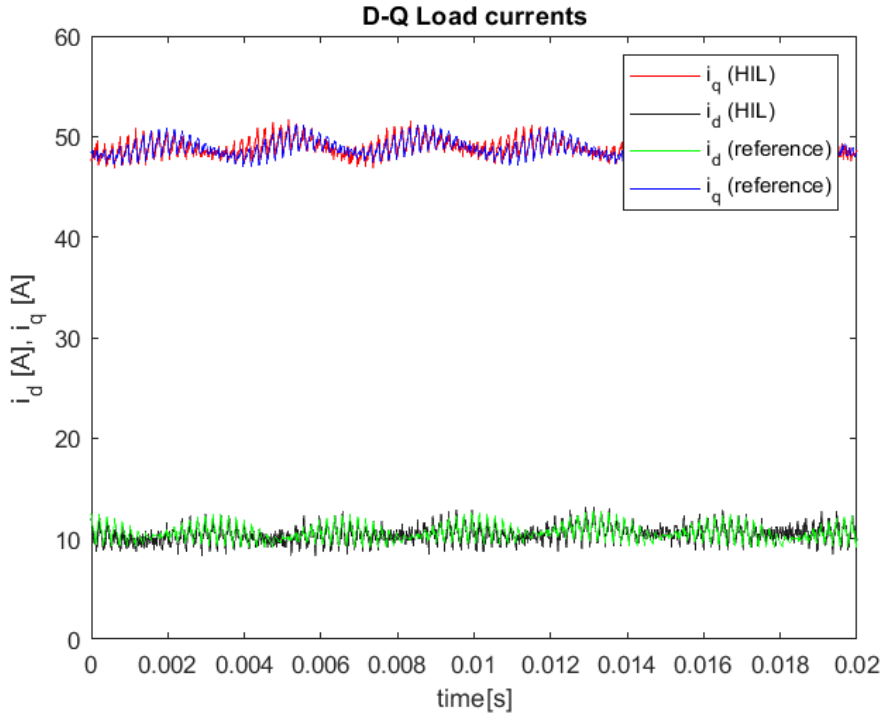


23. Figure: Load phase currents

The comparison of the d-q currents could be informative, as we know that the d current component generates flux, and the q component generates torque. The plotted values:



24. Figure: No-load d-q currents

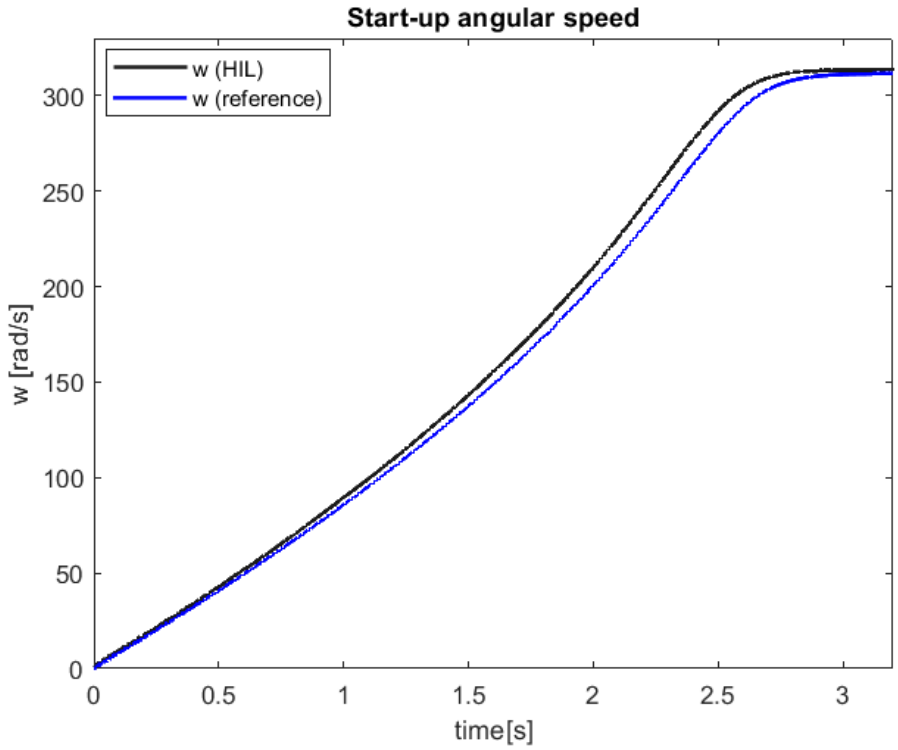


25. Figure: Load d-q currents

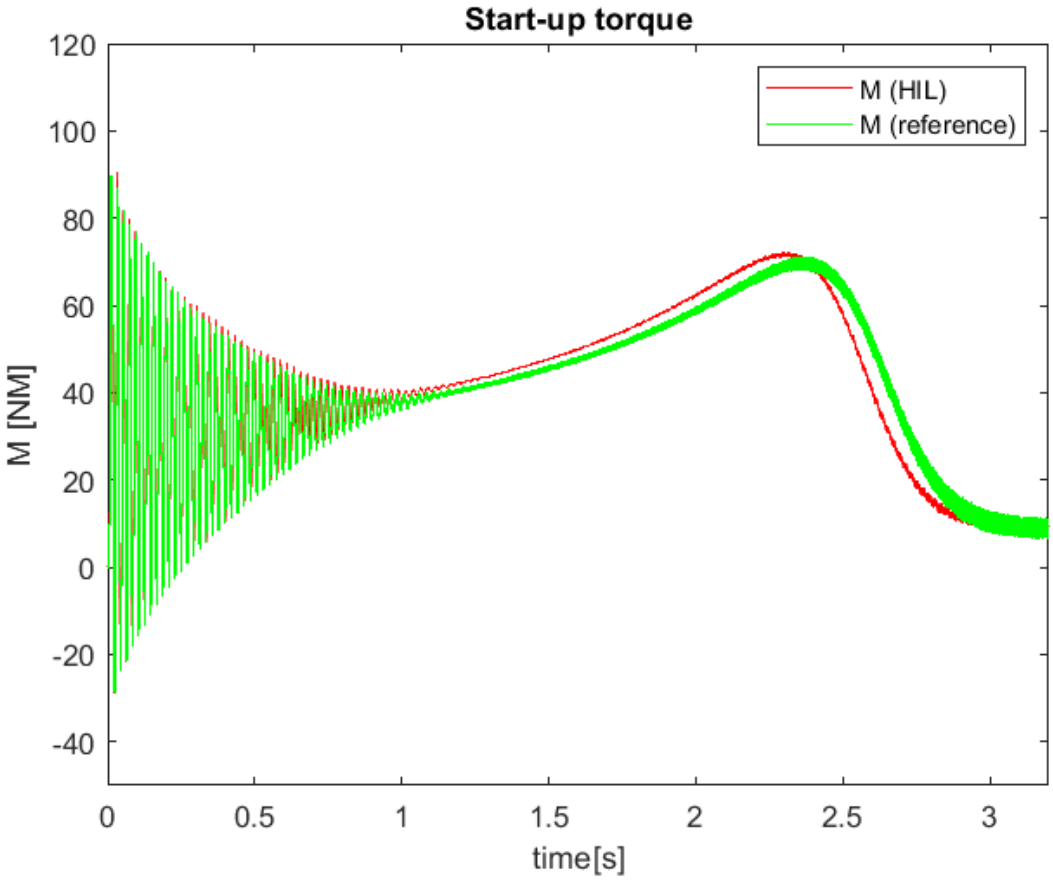
As we can see, the load q current is much higher. The q component decreased a little bit as the load is increased.

5.2 Dynamic tests

I want to show the start-up transients of the HIL model and compare them to the Simpower System reference. First, let us take a look at the torque and angular speed transient:

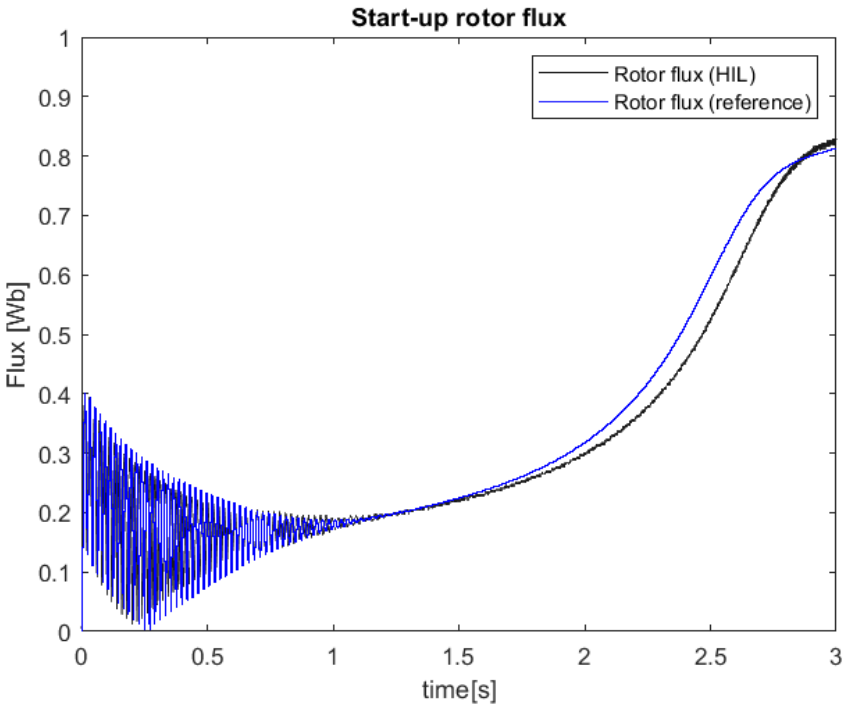


26. Figure: Start-up angular speed

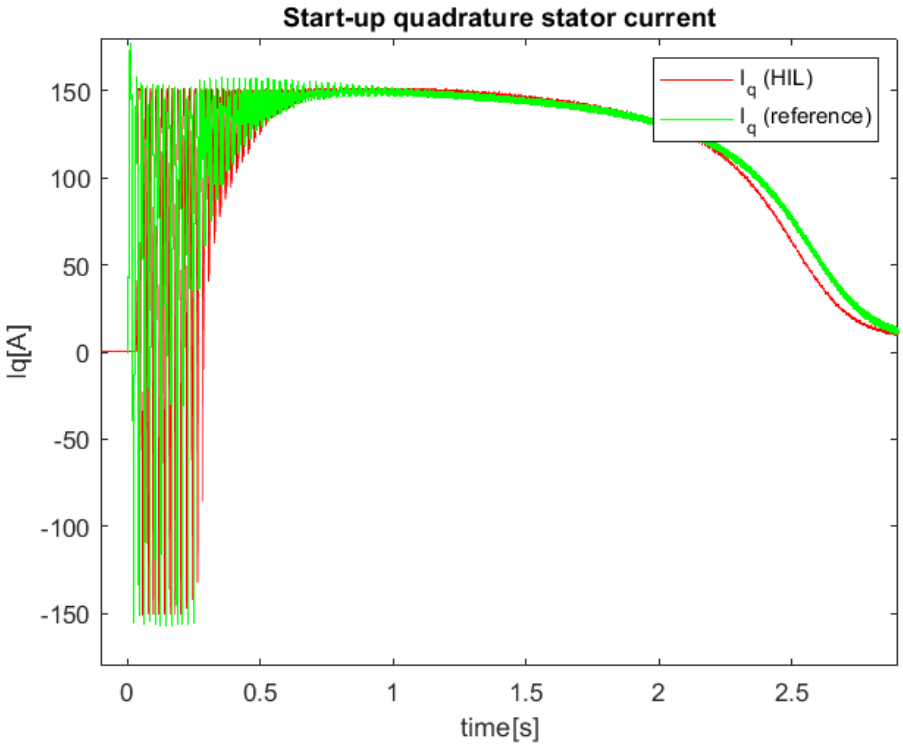


27. Figure: Startup torque

The initial torque of the asyncronous machine is low and has big ripples. Since the torque is made by the flux and the q component of the current, their startup characterisctics may tell us why:



28. Figure: Start up rotor flux



29. Figure: Start-up stator "q" current

We can see that the rotor flux takes some time to be created. It becomes stable similarly to the q current and they make the toque stable and bigger as well, a little bit after the initation.

References

[1] https://www.xilinx.com/support/documentation/sw_manuals/ug998-vivado-intro-fpga-design-hls.pdf

[2] https://www.infineon.com/dgdl/Infineon-FF650R17IE4-DS-v03_03-EN.pdf?fileId=db3a30431ff9881501201dcfe2a54986

[3] https://hu.mouser.com/ProductDetail/KEMET/C44HLGR6400AASJ?qs=sGAEpiMZZMsh%252b1woXyUXjzRu9w46NtHr%252bEdhPtGVC0k%3d

https://hu.mouser.com/ProductDetail/KEMET/SS26V-300028?qs=sGAEpiMZZMsVJzu5wKIZCRKkZmKdbMQOky%252b7rTdJ2X4%3d

[4] http://zedboard.org/product/zedboard