Real-time step motor emulation for hardware in the loop simulation

Oceguera Valenzuela, A.

Award date:
2009

Technische Universiteit Eindhoven

Master of Science in Embedded Systems
Master's Thesis

Real-Time Step Motor Emulation for Hardware in the Loop Simulation

by
Alvaro Oceguera Valenzuela

Department of Mathematics and Computer Science
and Department of Electrical Engineering

Advisor: prof.dr.ir. Twan Basten
Tutors: dr. Lou Somers
ing. Sander Hulsenboom
Eindhoven, Netherlands
July 2009

Abstract

Testing is one of the compulsory steps in developing software. Tools like a Hardware in the Loop (HIL) simulator are used when testing real-time embedded software. Océ Technologies, a company that manufactures and sells production printing and copying systems, wants to test among others the drivers of step motors used in their systems. For instance, they want to test error handling code for the event of a motor breaking down. With real motors, the cables of the step motors need to be physically disconnected to emulate such an error, making the test slow and tedious. Furthermore, continuous automatic testing is not feasible because the step motors may be overheated. For these reasons, Océ wants to investigate the emulation of a step motor in an FPGA. Such a solution allows continuous automatic testing and it allows fault injection when testing step motors through a HIL simulator.

Schulte et al. [39] develop a *load inductive simulation*, as a solution for emulating a brushless direct current (BLDC) motor. Adapting their work was considered for creating a step motor emulator and a fixed point differential equation solver was created using VHDL. The solver is needed to calculate the currents that should be flowing through the motor and consequently detect the number of steps the motor is moving. This solution has the disadvantages of being computationally expensive due to the operations needed for the differential equation solver and the big number of I/O ports needed for generating the inputs to the solver and the outputs of the emulator; on the other hand, the solution can emulate theoretically all kinds of inductive loads by parametrization and hardware changes.

A real time embedded system was eventually chosen as solution for the step motor emulator, where jointly electronics components and FPGA embedded components interact for emulating the step motor. The electronic components, an inductor and a resistor, help to recreate the behavior of the step motor, while the FPGA helps to identify the steps the motor should be moving, and to transmit the steps to the HIL simulator in the form of encoder signals. Less I/O port utilization and less FPGA processing are needed when comparing it to the load inductive simulation approach, making the parallel emulation of more than one step motor potentially easier in one FPGA.

The step motor emulator presented in this work is able to:

- generate a voltage sine wave signal similar to the generated current signal flowing through the coils when controlling the step motor with a driver using pulse wide modulated (PWM) voltages;
- identify the steps the motor driver requested; the FPGA performs parallel readings of the analog to digital converters, synchronizes them and processes them for the identification of the steps;
- generate an encoder signal as feedback for the HIL simulator; two signals simulating an encoder come out of the FPGA with the maximum resolution of one step;
- perform fault injection by allowing the simulation of the motor breaking down and the motor skipping steps; commands are read via the serial port from the HIL simulator or from a PC and executed by the FPGA.

The results show that the generated voltage sine waves of the emulator are analog to the current signals flowing through a step motor, and that the FPGA is able to recognize and distinguish the steps requested by the driver. There is still a challenge in the emulator when testing with higher frequencies; the coils generate a lot of spikes in the current signal that are reflected in the voltages read by the FPGA, giving as a consequence errors during the identification of the steps. Nevertheless, the step motor emulation turns to be a valuable tool by allowing continuous automatic testing and fault injection when testing step motors through HIL simulators when comparing it to traditional and manual testing.

Mater perfectam habebit
curam et victoriam!
P. Josef Kentenich

Acknowledgements

A lot of things happened before having me finishing this project and a lot of those things happened thanks to my advisor Twan Basten, who I would like to thank sincerely for helping me in finding this project, for his support in the moments where things were not clear, for his ideas, comments and questions that make me realize the things I was doing wrong and the things that I was doing right; I want to thank him specially for his valuable comments and his willingness to helping me in finishing this master project.

Also my sincere thanks to Océ, for adopting me while I was developing the project, to Lou Somers and Sander Hulsenboom who were the link between me and Océ, for giving me the opportunity to see how the things are being developed inside the R&D department, for their guidance, project proposal and for being there any time I needed; to Marcel van der Veen, for his time and support while coaching me with VHDL and an special recognition to Theo Pubben, for his ideas and solutions while I was developing the electronics circuits of my thesis.

To the TU/e and the great people working there, for accepting me in its great family and for sharing me all the knowledge that I have acquired.

To Shell, for their economic support, because without it I would have never been able to be here, writing these lines, for their help to make one my dreams comes true.

To my colleagues in Océ, for helping me to enjoy my time in Venlo, it is the best to go to work when you enjoy the company and friendship of the people next to you.

To you Ely thanks for walking next to me during these two years; to all my friends in Netherlands, for being one of the reasons of making me love this country so much.

And to the most important part in my life, Alvaro and Herlinda, my parents, and Yadira and Sergio, my sister and brother, with whom I have lived so many things, for their support even if we are so far away from each other, for their love and understanding; and finally to the rest of my family … just thank you for being there.

Alvaro Oceguera Valenzuela
July 2009

Contents

List of Figures

List of Tables

Abbreviations

HIL	Hardware in the Loop
HILs	Hardware in the Loop Simulator
I/O	Input/Output
BLDC	Brushless DC
EC	Electronic Control board
SIL	Software in the Loop
IC	Integrated Circuit
ECU	Electronic Control Unit in a car
VR	Variable Reluctance
PM	Permanent Magnet
EMF	Electromotive Force

Chapter 1

Introduction

1.1 Motivation

Most of the products used in our daily lives related with entertainment, communications, and office appliances among others, have electronics in them, and the more complex they get, the more common that embedded software will play an important role in their functioning. Therefore, developing good software is a goal for companies developing professional high-tech systems or consumer electronics, and testing the software is a compulsory step in any software development process they use.

Océ Technologies is part of these companies, it is a company that manufactures and sells production printing and copying systems. It has facilities in Venlo where the prototypes for new products are being developed. Software, mechanical and electrical engineers work together to the design of these products, and the software group in Océ has simulators for the printers to be developed to test new software packages without the need of the final copier hardware.

One of these simulators is a *Hardware in the Loop (HIL)* simulator; HIL simulators (HILs) provide a powerful tool to simulate complex systems. According to [19], there exist three main differences between HIL simulations and normal computer simulations: the outputs in HIL simulators are real hardware signals not squiggly lines plotted on a graph, the HILs runs in real time and in a HILs the embedded software to be tested runs in the hardware that will be built into the products.

HIL simulators are used in Océ as test facilities to test the complete embedded control platform including target hardware and software, using simulated sheet behavior and simulated or real I/O without changing the target hardware or software. Step Motors contribute to these I/O signals by receiving voltages from the hardware under test, voltages that produce a movement by a certain number of steps in the motor shaft.

Automatic testing of step motors is a hard task; physical disconnection is the only solution for simulating a motor breaking down, an annoying noise is produced when running the steppers at frequencies higher than 2 KHz, a high risk of fire occurs when the motor gets overheated by performing continuous testing; all of these issues only increase when testing up to 10 step motors at the same time, as happens in a normal HIL simulator for printers.

The solution for these issues is a Step Motor Emulator, developed during this work, with the following goals:

- A step motor will be substituted from the HILs and an emulated step motor version on an FPGA will be placed instead.
- The emulated step motor in the FPGA will simulate the behavior of the motor and it will also allow simulation of errors of the step motor, for automatic and continued tests.

1.2 HIL Simulator for Testing Embedded Systems Applications

In the late '40s Jon Von Neumann conceived the idea of running multiple repetitions of a model, gathering statistical data to derive behaviors of the real system, an activity which can be considered according to [40] as one of the first main approaches for simulation. Nowadays simulations are everywhere, from computer to medical simulations; simulations are helping humanity to substitute physical objects with virtual, cheaper and/or smaller objects compared to the actual cost or size, and are helping to predict the behavior of the real systems.

Simulations are also widely used for model validation, performance prediction, bottleneck detection and more. Two types of simulators can be defined: *Self-driven simulators* where the goal is to predict the system performance or verify the system layout, which rarely define sensors or actuators and *Software-driven simulators* where the goal is to replace the actual environment and where sensors and actuators are driven by the simulator [21].

Both simulation types are used in Océ under the terms: Software in the Loop (SIL) and Hardware in the Loop (HIL) simulators. In SIL (figure 1), the target hardware is simulated and the software under test runs on that simulated hardware in a normal PC, while in HIL (figure 2), the software is tested in real-time using the real embedded hardware.

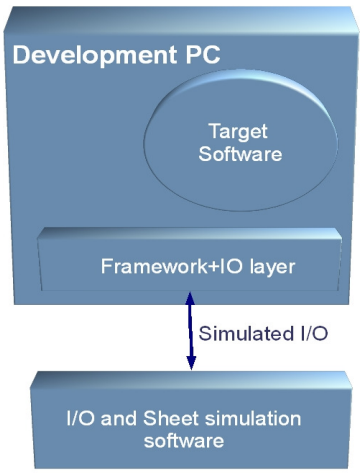


Figure 1. SIL Block Diagram

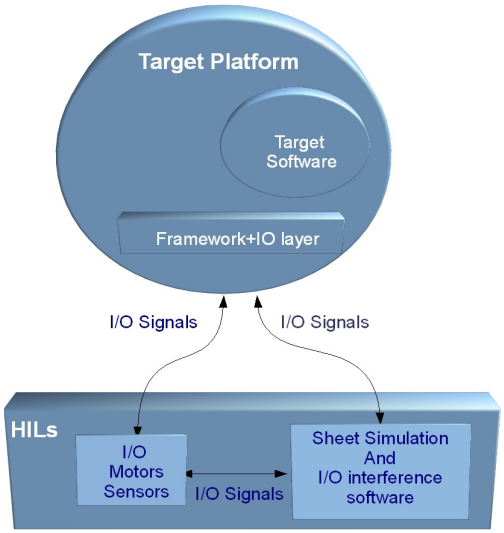


Figure 2. HIL Block Diagram

Considering the automotive industry to illustrate the concept of the HIL simulator vehicle testing can be expensive, time consuming and the results subject to poor repeatability [28]; for instance, if the control system for traction and braking needs to be tested, the complete functionality test will include testing on ice and snow, available only on certain times of the year. A big reason of why the manufacturers are turning to HIL as a technique is the ability to test in a laboratory environment under a full range of conditions.

The SIL simulator has been a great success for Océ but the need for a real-time simulator made it necessary to create HIL simulators, where flexibility, test repetitiveness, and visualization are characteristics inherited from the SIL simulators while real-time behavior with a real embedded platform are added.

The HILs in Océ is used to verify the correct functionality of the electronic control board (EC). This board is present in all the printers and it is where signals are read, processed and sent to the different modules of the printers (as shown in figure 3). These signals are used in the modules for activating tasks needed for copying, scanning or printing documents and images.

Figure 3. Electronic Control board and its interaction with the modules of the Printer [31]

The HILs is generating all the I/O signals needed by the EC to verify its functionality, and step motors form part of these signals, as can be seen in figure 4; testing the hardware and software of the EC using simulated or real I/O, satisfying real-time constraints, allowing flexibility, project independency, and maintainability form part of the requirements for the HILs in Océ.

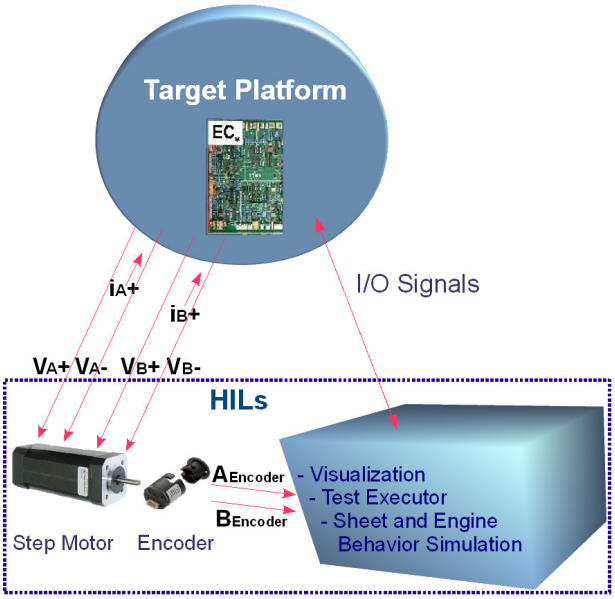


Figure 4. EC and HILs Interaction

1.3 Step Motors

Direct Current (DC) motors are used everywhere: white goods, toys, cars; their most important parts are the stator and the rotor [23]. The stator is stationary and usually fastened to the frame of the motor, while the rotor is the revolving member used to move the motor load, using the output shaft.

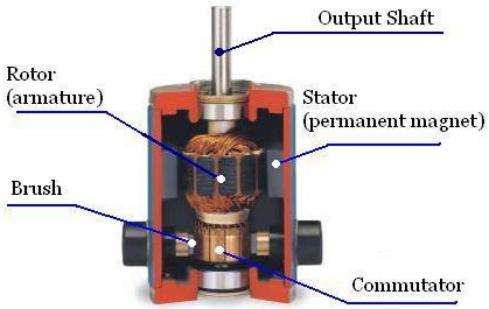


Figure 5. Cut Away View of a DC motor [27]

Figure 5 shows the main components in a permanent magnet motor are presented; the stator has a housing and two permanent magnets. The rotor is equipped with copper windings uniformly placed on a cylindrical iron core. Current is supplied to the rotor windings via the commutator and two brushes. When a current is passed through the rotor, the current gives rise to a circular magnetic field around the rotor, and consequently, the rotor will rotate around the magnets (stator).

Step motors are considered a digital version of the DC motors; these motors can divide a full rotation into a large number of steps and the motor's position can be controlled precisely, without any feedback mechanism.

Three types of step motors exist based on their complexity: variable reluctance (VR), permanent magnet (PM) and hybrid. As the name suggests, a PM motor consist of a permanent magnet rotor, while the VR consist on a toothed rotor made from a soft iron material. The hybrid step motor incorporates physical properties of both the VR and PM motors, hence the name hybrid; it combines the permanent magnet rotor from the PM step motor and the teeth in the rotor from the VR motor.

Step motors can also be grouped based on their winding arrangement: unipolar and bipolar step motors. Unipolar motors are composed of two windings, each one with a center tap that can be tied together or independently go out of the motor, giving as a result five or six wires. Current direction in unipolar motors depends on which half of a winding is energized, and based on the winding that is powered, they will be acting as North or South Pole. Controlling these motors is easy because the direction of the current through the windings will determine the movement of the step motors; there is no need to reverse the current through the windings, just energize the correct winding at the correct time [12].

In Océ the step motor of choice is the bipolar hybrid PM motor, so this motor is explained in more detail below.

In bipolar motors a single winding per phase is found. They are composed of two windings, consequently having two phases and have four wires going out of the motor. The current runs through the entire winding, producing more torque than unipolar motors of the same size. But the circuitry involved to control these motors is more complex than the one in unipolar motors [12]. The current needs to be reversed in order to reverse the magnetic pole, consequently changing the direction of the current and finally create a movement in the rotor. Movement in the rotor can only be done by changing the polarity in its windings.

In figure 6, a hybrid bipolar step motor is presented. Current will flow from left to right when V_A+ is positive and V_A- is negative. Current will flow in the opposite direction when changing the polarity using an "H-Bridge circuit" (see below).

Hybrid PM Step motors have a permanent magnet rotor, but like the VR motor, the rotor is toothed. With a magnet along the rotor axle, a mix of the standard PM motor and the VR motor is created.

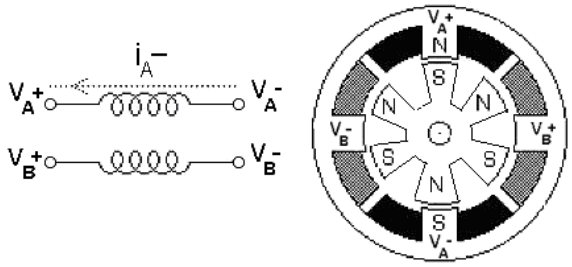


Figure 6.Hybrid Bipolar Step Motor [12]

Winding 1 with terminals V_A+ and V_A- in the step hybrid bipolar motor in figure 6 is distributed between top and bottom stator poles, while winding 2 with terminals V_B+ and V_B- is distributed along left and right stator poles; the rotor is a permanent magnet with 3 souths and 3 norths around its circumference.

Hybrid Step motors can be single stepped with two different control sequences. Using + and - to indicate the power polarity applied to each terminal and 0 to indicate no power; the sequences are showed in table 1 below.

Table 1. Control sequences [12]

	1	2	3	4	5	6	7	8	9	10	11	12
Terminal V_A+	+	0	-	0	+	0	-	0	+	0	-	0
Terminal V_A-	-	0	+	0	-	0	+	0	-	0	+	0
Terminal V_B+	0	+	0	-	0	+	0	-	0	+	0	-
Terminal V_B-	0	-	0	+	0	-	0	+	0	-	0	+
Time ==>												
Terminal V_A+	+	+	-	-	+	+	-	-	+	+	-	-
Terminal V_A-	-	-	+	+	-	-	+	+	-	-	+	+
Terminal V_B+	-	+	+	-	-	+	+	-	-	+	+	-
Terminal V_B-	+	-	-	+	+	-	-	+	+	-	-	+
Time ==>												

The first sequence only supplies current to one winding minimizing the power consumption, and the second sequence maximizes torque since both of the windings are energized at the same time [12].

Stepper motors can be rotated to a specific angle with ease, and hence step motors were used in the pre-gigabyte era computer disk drives, where the precision they offered was adequate for the correct positioning of the read/write head of a hard disk drive; printers, computer numerical controlled machines and volumetric pumps are other examples of their application.

1.4 Step Motor Drivers

Commercial step motor drivers use microstepping to increase the performance and limit noise and resonance problems. Microstepping is a common technique when working with step motors. It divides a motor step in substeps since this allows smoother transitions between steps and a better step resolution.

With sine-cosine microstepping a constant torque is produced by controlling the current in the windings. The torque produced by both windings can be added linearly to calculate the net torque. As a result, to set the motor to a angle θ, the currents through the winding of the motor need to have according to [12] the values given by equations (1) and (2).

$$I_1 = I_{\max} \cos(((\pi/2)/S)/\theta) \quad (1)$$

$$I_2 = I_{\max} \sin(((\pi/2)/S)/\theta) \quad (2)$$

Where $I_{1,2}$ current flowing through winding 1 or 2

$I_{\max}$ = Maximum current allowed to flow in the winding

S = Step angle, in radians

θ = Shaft angle, in radians

The basic circuit for driving the currents that will flow through the windings of a bipolar step motor is the H-Bridge. An H-Bridge is a circuit that allows a current to flow in either different direction across a winding, as can be seen in figure 7. When transistors T1 and T4 are turned on and T2 and T3 are off, current will flow from left to right in winding 1; while when T3 and T2 are turned on and T1 and T4 are off, the current will flow in the opposite direction [12].

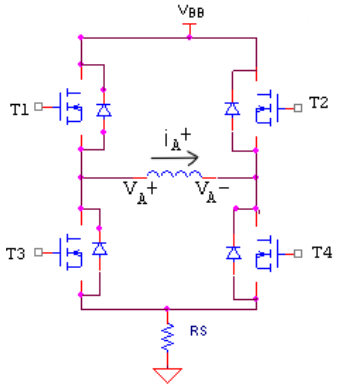


Figure 7. Bipolar Step Motor control circuit

Instead of using a fixed voltage to create the currents that will flow through the windings, a technique called chopper is a way to limit the current in the winding of a step motor when using a voltage supply higher than the motor rated voltage**.**

The chopper technique uses a high voltage supply (V_{BB} in figure 7) to bring the current up to $I_{\max}$ very quickly and when $I_{\max}$ is flowing through the windings the voltage will be chopped (switched off). The result is a pulse-wide modulated waveform that is used to create an average voltage and consequently an average current equal to the voltage and current necessary to the windings in the step motor.

When a supply voltage is applied to the windings of the step motor the current rises exponentially until $I_{\max}$ is reached, modeling the winding as an inductive-resistive circuit (figure 16, considering $V_{EMF} = 0$) the current as a function of time is represented by

$$I(t) = (V/R)\cdot\left(1 - e^{-(L/R)\cdot t}\right) = (V/R)\cdot\left(1 - e^{-\tau\cdot t}\right) \qquad (3)$$

where according to [12], V is the voltage applied to the winding, R is the resistance of the winding, L is the inductance of the winding and $\tau = L/R$ is the time constant of the motor, representing the time it takes the step motor to reach approximately 63.2% of its final value after a step input (the voltage supply change from zero to a one in a very short time) is applied to the windings of the motor, as can be seen in figure 8.

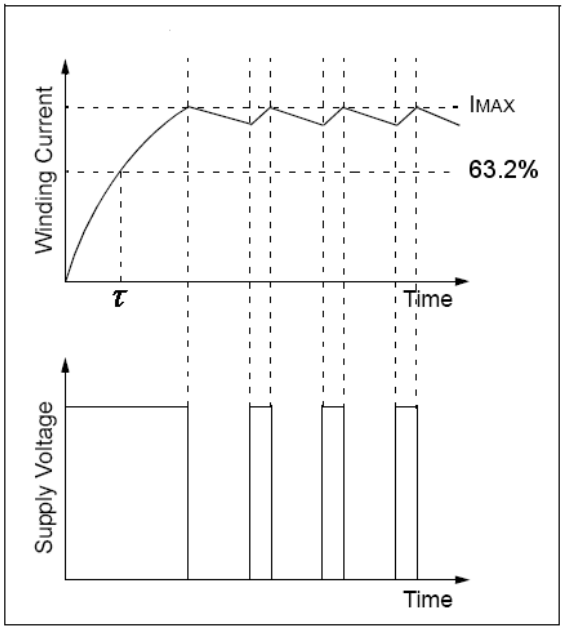


Figure 8. Current and Supply voltage comparison [12]

Figure 8 presents also the current generated to flow in the winding based in the supplied PWM voltage over time. The voltage supply is switched off (all transistors are off) when I_{max} is reached. The voltage is modulated (PWM) to limit the current flowing through the winding to I_{max}.

Microstepping drivers exist in the form of integrated circuits (IC). These circuits consist of circuitry allowing an easy implementation for applications where a complex microprocessor is unavailable or is overburdened. A simple input of one pulse (STEP) into a pin makes the motor to move one microstep, which can be either a full, half, quarter or sixteenth step, depending of the configured settings.

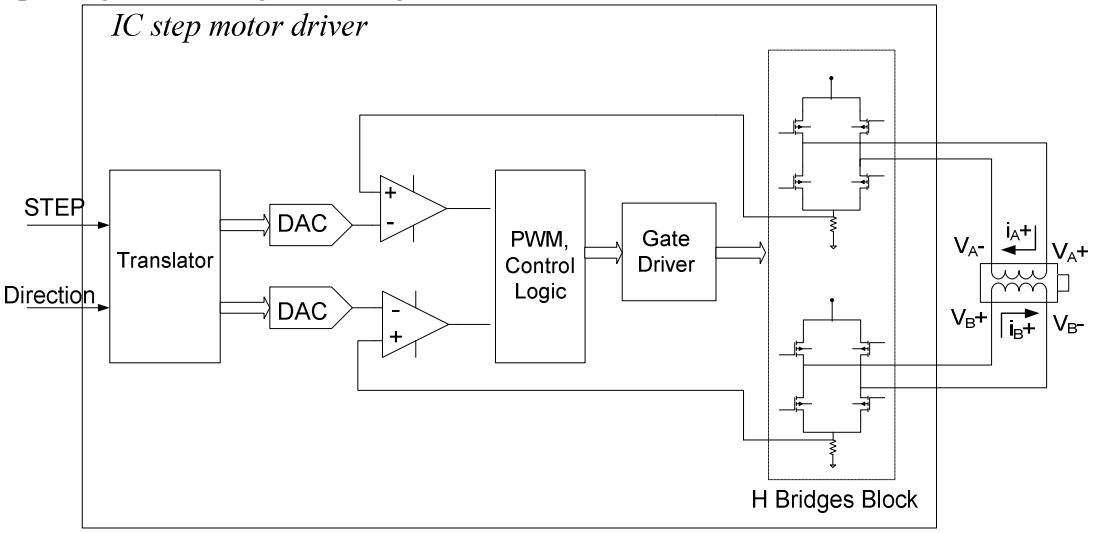


Figure 9. Functional Block Diagram of an IC step motor

A coarse block diagram of an IC step motor driver can be seen in figure 9. When a step command signal occurs on the STEP input, a built-in translator sequences the Digital to Analog converters (DACs) to the next level and current polarity based on the direction input pin voltage and the current-level sequence shown in figure 10.

For instance, when the motor is in the home position (figure 10) the current flowing through the winding 1 (I_{OUT1A}) is 70.71% and the current flowing through winding 2 (I_{OUT1B}) is 70.71% and a pulse is detected on the STEP input pin, the translator will modify the current in both phases to move the step motor one microstep forward (to move the step motor forward, the input pin "Direction" shown in figure 9 should be high) and the current in the phases will change to $I_{OUT1A} = 38.27\%$, $I_{OUT1B} = 92.39\%$ (figure 10).

The outputs values of the DACs with the information of the current values needed to move one microstep are compared with the current flowing through the windings of the step motor, the differences between these values are the inputs to the "PWM, Control Logic" block, where the values are processed and control signals are sent to the "Gate Driver" block, in charge of controlling the transistors in the H-Bridges, turning them on and off, increasing or decreasing the current flowing through the windings, to match the required values and move one microstep.

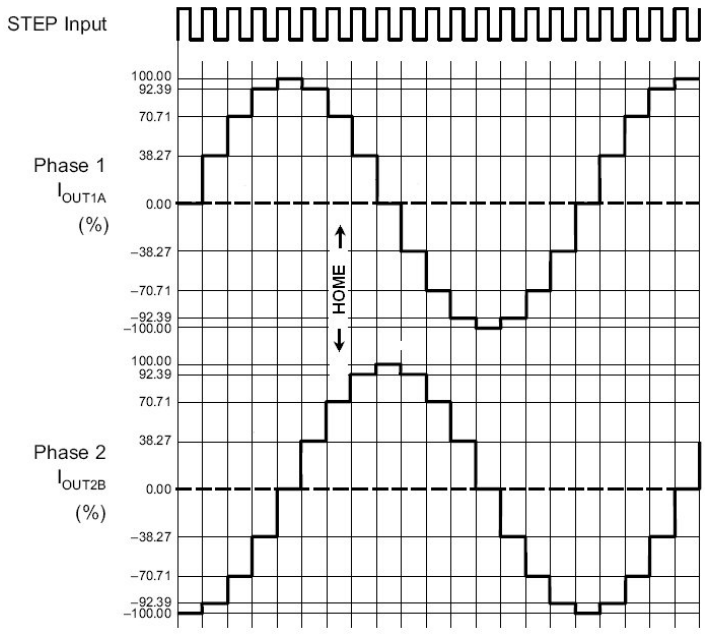


Figure 10. Current-level sequence

1.5 Incremental Rotary Encoders

Encoders are used in many applications, including controls, robotics, photographic lenses, mice, trackballs, etc., to convert the angular position of a shaft or axle to an analog or digital code, i.e. an angle transducer [49].
An incremental or quadrature encoder can be either mechanical or optical and are the most widely used of all rotary encoders due to its low cost since only two sensors are needed. They are used to determine position and velocity through two outputs signal called $A_{Encoder}$ and $B_{Encoder}$, called quadrature outputs as they are 90 degrees out of phase as can be seen in figure 11.

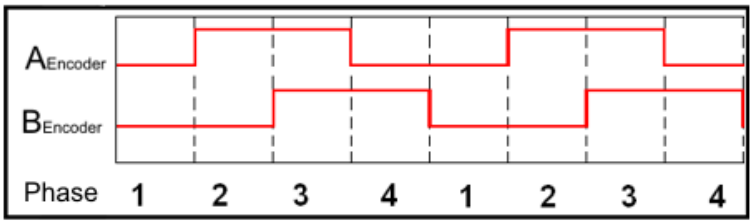


Figure 11. Quadrature diagram

These signals are used to encode the shaft position based on table 2 for the clockwise direction and table 3 for the counter-clockwise direction. For instance, if the last value was 00 and the current value is 01, the shaft has moved one step in the clockwise direction, while if the current value is 10 after being in 00 the shaft has moved one step counter-clockwise.

Table 2. Clockwise rotary encoder

Gray Coding

Phase	$A_{Encoder}$	$B_{Encoder}$
1	0	0
2	0	1
3	1	1
4	1	0

Table 3.Counter-clockwise rotary encoder

Gray Coding

Phase	$A_{Encoder}$	$B_{Encoder}$
1	1	0
2	1	1
3	0	1
4	0	0

In the HIL simulator of Océ, encoders are used as generators of feedback signals to verify that the movement of the motor shaft is the one requested by the control drivers.

1.6 Problem Statement

Exhaustive testing of a control system is one of the main reasons for the HIL simulator in Océ and it provides the context for the project described in this thesis. The HIL simulator available in Océ has some step motors in it, allowing the test of some of the control signals from the EC. It is not suitable for automatic verification because if the motors are tested for long periods, they produce heat, which may lead to overheating.

This problem together with the need for testing the motor for different faults, presents an opportunity to improve the current HIL simulators. The graduation project targets the development of a step motor emulator to be integrated in the printer HIL simulator, and, two main requirements can be specified:

1.6.1 Requirement 1

The purpose of this project is to substitute a step motor from the simulator and replace it with an emulated version implemented in an FPGA. In figure 12, the block diagram of this replacement can be found. The simulation of the behavior of the motor can be done because of the FPGAs natural characteristics of parallelism and high throughput programmable logic [36].
The emulated step motor in an FPGA, will not present overheating and testing different failures should be easy; for instance, instead of having to physically disconnect the power cable from the motor, the emulated version of it in the FPGA will do it virtually. Through serial communication commands, the HILs will tell the emulator which errors to simulate.

The step motors in the printers developed by Océ, do not have any feedback signal with the information about the position of the step motor; an encoder was placed in the HIL to verify the position of the shaft and whether the motor is moving the requested number of steps. This encoder will also be emulated in the FPGA represented as the output signal of the emulator.

When substituting the motors with the step motor emulator, the emulator will have to receive the voltage signals (which are the input signals to the step motor emulator and request a maximum step frequency of 9306 Hz from the step motor driver) as if the real motor was there and it will send outputs as if an encoder was there too.

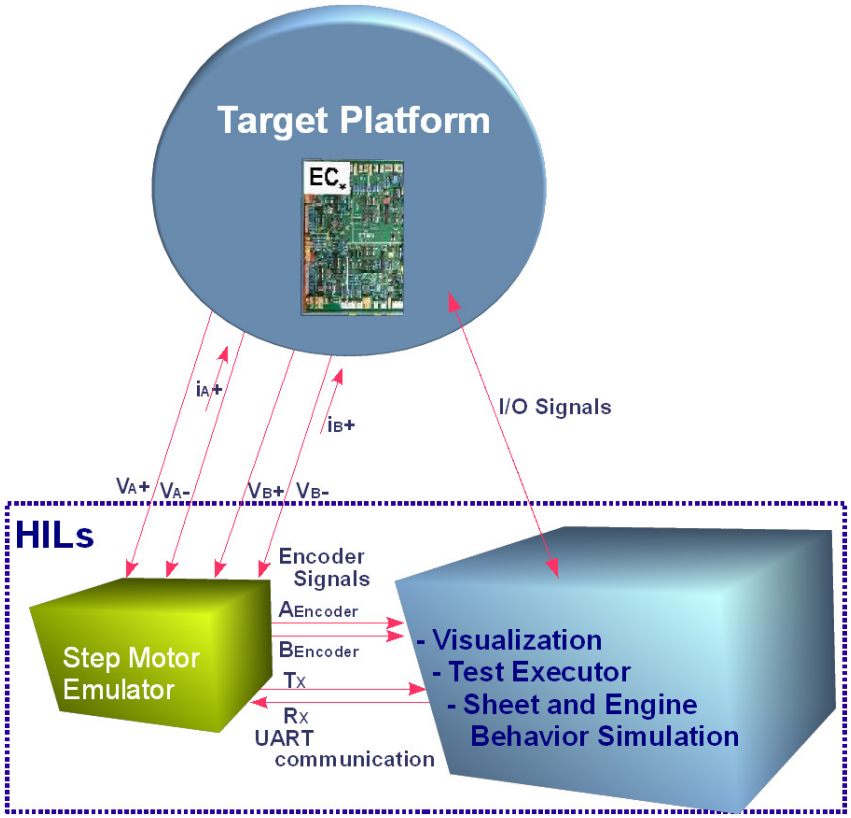


Figure 12. Block Diagram of the EC and the Emulated Motor in the HILs.

1.6.2. Requirement 2

Océ wants the following errors to be simulated by the step motor emulator:

- *Motor breaks down*. For this fault, the emulator will stop sending the simulated signals of the encoder with the position of the shaft (representing the number of steps the motor has moved).
- *Skipping steps*. In this fault, the emulator will send the encoder signal that indicates, based on the actual simulated position of the motor, that it has moved less steps than required by the EC.

1.7 Thesis Overview and Contributions

The remainder of this report details this problem statement and sketches the approaches followed to achieve these goals. Chapter 2 presents the state of the art methods for HIL simulation, motor emulation and fault injection testing; Chapter 3 presents three different approaches to design a step motor emulator where only the last one is completely implemented and is satisfying all the requirements from Océ; Chapter 4 presents partial results of the first two approaches and the complete results of the third approach and Chapter 5 presents the conclusions and future work for improving the step motor emulator.

Chapter 2

State of the Art

This chapter present the state of the art for the different components used for this graduation project. The HILs is helping software engineers to test their designs while the real hardware is being developed, or when it is difficult to do testing using the real hardware; the relation between the step motor emulator in the HILs in Océ with the HILs in the literature can be found in section 2.1. The state of the art techniques for emulating motors are presented in section 2.2, although, no work was found about emulating step motors. On the other hand emulating BLDC motors is an extensively researched topic due the extensive use of these motors in the automotive industry. Section 2.2.1 presents the design and implementation of a BLDC motor simulation for a HIL simulator used in the automotive industry, section 2.2.2 presents the work done for implementing and solving the state space representation of motors in FPGAs, with the objective of using it later in HIL simulators and section 2.2.3 presents the work available in the literature for the state space representation of step motors. Section 2.3 presents companies developing software and also devices for fault injection, that are used as guidelines for the fault injection in the step motor emulator. Finally section 2.4 presents how to use the state of the art techniques for the creation of the step motor emulator, identifying missing elements and open issues that are addressed in this work.

2.1 Hardware in the Loop Simulators

HIL simulation is not new; the aerospace industry has been using this technique ever since software first became a safety-critical aspect of flight control systems [28]. However, recently it has seen increasing its use because of:

- the intense pressure to reduce development cycles,
- safety requirements which mandate exhaustive testing of a control system prior to use on the real system,
- the need to prevent costly failures, either in-service or late in the design cycle,
- reduced cost and greater availability of off-the-shelf products.

Océ uses the HIL simulator for testing the final software in the EC board. To be able to do this a big number of I\O pins is required. Similar problems are present when testing the Electronic Control Units (ECU) in cars, where one of the tests includes the disconnection of

each one of its pins and the verification of the triggering of its corresponding diagnostic. This is a time consuming task when modules containing over 135 pins are verified. The task is greatly improved by automatic tests generated by the HIL simulators. In [25], King et al, present the HIL simulator as a flexible solution for these problems, an environment test in the automotive industry, that can be updated and modified as the vehicle and corresponding systems evolve throughout the development cycle.
Temperature, pressure and physical links between the HIL system and ECU are issues needed to be considered while testing with HIL simulators and are discussed also in [25]; for instance, communication bus lengths and termination will be different when testing in HIL simulators and when testing in cars, ground shift effects, back-feeding voltages to the ECU via sensor paths are overlooked issues while testing with HIL simulators, but that are not present in the step motor emulator developed in this graduation project.

The HILs in Océ can also be used to test software while the mechanical and electrical teams are developing the new hardware; a situation similar to the one presented by Maclay in [28]: the Gemini Telescopes project, of which the goal was the construction of two high-performance, 8 meter aperture telescopes and where the actual telescope was not available during the controller development, so a HIL simulator was used for helping in its design.

The step motor emulator needed by Océ will substitute the step motor (actuator) and will simulate the encoder (sensor) signals for feedback to the HILs; Schuette et al present in [46] the sensor and actuator signals needed for testing the ECU for automotive applications through HIL simulators, the calculation of the sample time, the models for the I/O ports and a classification of different types of HIL simulators based on the interface with the ECU. For instance, the electronic throttle control unit controls the desired throttle angle by means of a DC motor, and Schuette et al, decided to use in the HIL simulator the real throttle system for testing the signals coming from the ECU, being able in this way to verify the correct positioning of the throttle after receiving the control signals; a similar approach is used in Océ where the real step and BLDC motors are used in the HILs; this situation that will change after this graduation project because the step motors will be emulated, adding flexibility and simplicity to automatic tests.

2.2 Real-Time Step Motor Emulation

Emulating a step motor in real-time includes reading the input signals (analog voltages) from the real world, calculating the number of steps the motor should have moved based on these voltages and generate the output signals (digital encoder signals) specifying this movement. All of these actions need to be finished in a specified and short time. No work was found in the literature about emulating step motors but literature exists about emulating brushless direct current (BLDC) motors. An overview will be given in the rest of this section because the ideas can also be applied for emulating step motors.

Emulation of BLDC motors is done by solving the state space equations of the model of the motor in real-time with the help of an FPGA. Two different approaches can be followed here. Both of the approaches solve the state space equations of the step motor in the FPGA but one of them uses Matlab to create auto-generated VHDL code and the other one uses normal VHDL coding.

Furthermore, only one BLDC motor emulator has been completely developed and it is done by using normal VHDL coding presented in section 2.2.1. The rest of the approaches have only reached the stage of solving the differential equations in the FPGA and through simulations the time to solve the differential equations is verified to satisfy the real-time constraints (section 2.2.2); but no further work has been done about reading the analog voltage signals, generating outputs, etc., where big issues are present as well.

2.2.1 Inductive Load Simulation

In [7] Bracker et al, performed the simulation of an inductive load. This simulation includes the realistic current waveforms of AC motors for the automotive industry. In this work they developed the simulation of the load for HIL simulators, for testing the Electronic Control Unit (ECU) in cars, by using the normal plug used in any real environment. They develop the simulation of the inductive load by measuring the voltages at the output stage of the controller, calculating the current in real time through an FPGA, and by generating a current in the output stage (in case the motor is working in generator mode) with DAC converters.

In [39], Schulte et al, who work in the same company as Bracker, present the simulation of a BLDC motor, a solution where only the model of the three-phase windings is implemented in an FPGA, while the remaining part of the electric motor is simulated on a conventional real-time processor (calculation of the torque and the back-EMF). It is claimed that their emulator allows testing, by connecting the electronic units for cars as they are, without manipulations. Furthermore it is stated that it is also suitable for testing control units using sensorless control techniques.

2.2.2 FPGA Induction Motor Simulation

Solving the state space model of the motor in FPGAs is presented in this section as a solution for emulating motors in HILs. Simulation time constraints are verified to satisfy real-time constraints allowing these solutions to be used in HIL simulators.

In [36] a real-time emulation of an induction motor is done in an FPGA; in this work Jaztrzebski after having the state space model of the motor, uses integral methods programmed by a fixed point representation in the FPGA, to solve the differential equations in less than $1\mu s$, allowing testing and evaluation of very fast motor controllers in real time.

In [15] Duman et al, present a real-time implementation of a three-phase induction machine for HIL simulators. In this work, the state space representation model of the system is represented by matrices using floating point numbers. The model is solved in less than $1\mu s$ by using matrix multiplication algorithms studied by Prakhya [37] and the design is downloaded into a PCI FPGA development kit using Quartus-II.

The approaches by Jaztrzebski and Duman are intended to work for HIL simulators. In both works the model of the motors is implemented in the FPGA and in both works it is verified that the FPGA is solving the model equations in less than $1\mu s$, but no work is done for reading the signals and to generate the output signals to close the loop with the HIL and have a real-time motor emulator.

If thinking about auto-generating code in Matlab for implementing the state space model of the motor in an FPGA, Delli Colli et al. [13], present a speed and position observer for a non-salient permanent magnet synchronous motor; three different simulations were done in their work with successful results. The work is developed by using the Altera DSP builder Simulink library allowing them to download VHDL code in an FPGA. After finishing their work they realized the importance of checking the code for bugs, after it is generated by Matlab.

2.2.3 Step Motor State Space Representation

The state space representation of motor is needed when testing control laws. Several models are available in the literature for this purpose. The most common model consist of four state variables: the currents in the phases represent two state variables, the angle of the rotor is the third state variable while the angular speed is the fourth one. Different motors are able to be

simulated with the same state space equations by only replacing constant parameters, like the inductance in the windings, the resistance, the inertia of the load, etc.

Most of the research papers related with to step motors are talking about the description of and methods to control the steppers and not about how to emulate them; [1], [47], [41] have a description of step motors, their functioning, their design and types. In [24] modeling and calculating the nonlinear magnetic fields in linear step motors is discussed for step motors that control the rod of nuclear reactors. The magnetic fields that are found here can be useful if the state model of the step motor has as parameters these magnetic fields, but based on the requirements by Océ, the measurements of the electromagnetic field will not be used; consequently no further information is presented in this document.

In Océ there are already studies about the behavior of the step motor, the state space model, its basic behavior, the behavior under friction and different load, etc.

2.3 Fault Injection Testing

While developing software systems, the necessity for testing them also appears, and throughout the years different techniques have been appearing to help software engineers to test the software that they are developing. While testing is being performed several questions pop up automatically: what are the causes of defects? how to avoid them in the future? when is testing enough?. Solving these questions is a decision needed to be made in every company developing software.

Fault injection is a technique for improving the coverage of a test by introducing faults in order to test code paths. Particularly three benefits can be found according to [9]: an understanding of the effects of real faults, feedback for system correction or enhancement, and a forecast of expected system behavior.

The step motor emulator developed in this project needs to be able to perform fault injection of errors; these errors are defined based on previous experience when testing the motor, for instance if the step motor is skipping steps then a lower frequency will be inferred from the encoder and the code will erroneously calculate the new speed for moving the paper. This kind of test fits in the “Experienced-based techniques” for testing defined by Müller et al in [29], where a standardized qualification for software testers is presented. This standardization defines the “Experienced-based techniques” as a class of test, where experienced testers design tests based on their skills, intuition and experience with the software. A list of possible errors is enumerated and design tests are designed to attack these errors and use them for software testing.

Software of third party companies like DevPartner [11] helps to test error handling routines by simulating application errors in software. This software tool helps developers and quality assurance engineers to write, test and debug the software responsible for handling fault situations. DevPartner allows developers to work in a predictable and repeatable environment to analyze and debug application error-handling code; characteristics that with the step motor emulator will be available when testing the EC board in Océ. The EC board can then also be tested using fault injection testing, allowing repeatable tests, helping the engineers to test and to debug the software.

As mentioned before, the automotive industry counts a large number of HILs and companies different from DevPartner, of which solutions are more software oriented, are developing hardware solutions for doing fault injection; ETAS, for instance, is a company headquartered in Germany [17] that offers integrated tools and tool solutions for the development and service of automotive ECUs. Some of these solutions are devices for fault injection; these devices have the ability of doing simulation of failures in real time through current or voltage

channels. The failures that the devices can simulate are: cable breaks, short circuits (to ground, to the battery and pin-to-pin), leakage currents, contact corrosion, bouncing or a combination of all these failures.

Vector is another company providing fault injection devices, it is a company developing software components and engineering services for the networking of electronic systems in the automobile and related industries. Figure 13 shows how the faults are injected: switches are placed between the sensor, the actuator, or both, and the ECU for making short circuits to ground, to battery or to open the lines coming and leaving the ECU ports.

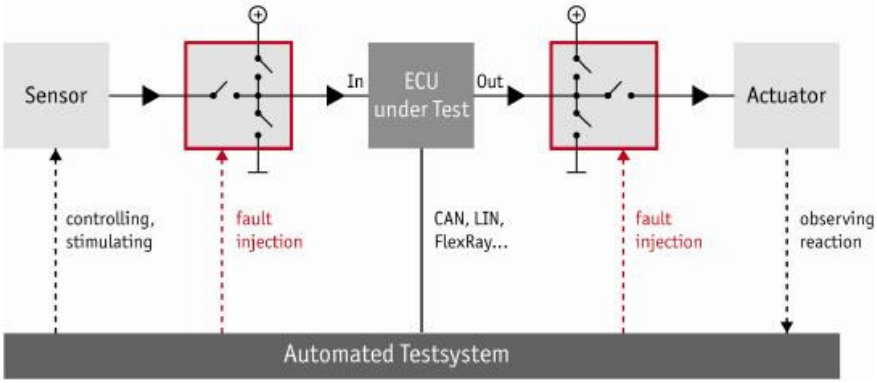


Figure 13. Fault Injection by Vector [43].

National Instruments also provides devices for testing the response of systems tested under HIL simulators. These devices are switching modules that insert faults and are able to vary the load for the systems under test [30]. Similarly, Pickering Interface [33] and dSPACE [14] are also companies developing devices for failure simulation through switches, for the automotive industry; in [22], Karpenko et al present a HIL simulator of an F-16 aircraft. An experimental hydraulic system with two independent fluid power circuits was developed. The first hydraulic is the flight control actuator, while the second one enables failures modes (fluid leakages, changes in actuator friction and hydraulic supply pressure, among others). The HIL simulation used here support future experimentation in the presence of flight control actuator faults giving an example of HIL fault injections out of the automotive area.

The step motor emulator developed for the HILs in Océ can be classified among all these fault injection tools, since the types of errors that Océ needs to simulate are errors between the EC and the motors. However, because of the step motor driver behavior (that needs to read the current flowing through the windings of the motor as a feedback signal) it is not possible to just use switches for introducing errors as most of the companies presented in this section do.

Océ is not interested in the efficiency, the temperature, the rotor voltages with load or without load that are the tests defined using IEEE standards [16] for induction motor testing. Neither is Océ interested in inductor motor fault diagnosis, where tests are performed to obtain reduction in maintenance costs and to prevent unscheduled downtimes of the motor [26], [38]; this is an area of intensive research since induction motors are widely used in industrial processes in industry presenting a significant cost in some cases. Océ's interest is the effect of motor faults on the printer behavior, in particular on the embedded control software.

2.4 Step Motor Emulation and the State of the Art

In the aerospace and the automotive industry HIL simulators are the preferred option when testing software embedded in electronic control units; they are the preferred option because with HIL simulators it is possible to test the software when the final hardware is still not available, it is possible to test the software when testing conditions are not easily reproduced or when it is needed to test the software in the target platform as Océ is doing it, increasing with it the coverage of the test in HIL simulators.

The need to generate in real-time voltages or currents to interact with the control unit present a lot of difficulties when trying to generate emulators of valves and motors in HIL simulators, this is why using real valves and/or of real motors is still a common practice in the HIL simulators in the automotive industry. However, because automatic testing and fault injection are greatly improved when using emulators in HIL simulators it is worth it to invest in their creation.

Automatic testing and fault injection are the reasons for Océ to generate a step motor emulator; these motors are used in all the processes in the printers and developing an emulator for them represents a great improvement for testing the software using HIL simulators; similarly to the automotive industry where BLDC motors are used in big amounts in cars, and where BLDC motor emulators already exist to help in the process of testing the software using HIL simulators.

The BLDC motor emulator generated for the automotive industry reads the analog voltages coming from the EC in the cars, solves the state space equations of the motor in real-time with the help of an FPGA, and with these results generates the currents that should be flowing through the motor. The procedure for the creation of this BLDC motor emulator represents one of the possible solutions for developing the step motor emulator requested by Océ.

How to solve the state space model of a motor in real time is also a topic of research for creating motor emulators for HIL simulators. If the state space equations are solved fast enough to simulate the behavior of the motor then it is assumed that they can be used in HIL simulators. Different solutions exist for solving the state space models of motors in FPGA: integral methods programmed by fixed point representations, matrix multiplications using floating point numbers or through Matlab auto generated VHDL code after developing algorithms in Simulink to solve the state space model of the motors; solutions intended to create motor emulators and use them in HIL simulators but besides the BLDC motor emulator for the automotive industry there is no other complete motor emulator available in the literature.

Fault injection as stated before is one of the reasons for generating the step motor emulator in Océ; fault injection is done in the automotive industry for improving the test coverage for the EC and different alternatives are available to select the faults to be introduced in the system; several third party companies are developing fault injection software for testing error handler routines and some others are developing fault injection through hardware devices. The step motor emulator in Océ belongs to these kinds of fault injection devices, since the faulty signals are generated in the emulator and then sent to the HIL simulator.

To sum up, no literature was found about emulating step motors but ideas where taken from the BLDC motors emulators where a lot of work has been done since they are widely used in different industries and fault injection that will be used in the step motor emulated is presented as an alternative for increasing the tests coverage using software or hardware devices.

Chapter 3

Design

This chapter starts by describing the input and outputs signals in the step motor emulator by considering it as a black box. The chapter continues with three different approaches for generating the step motor emulator. In section 3.2 the "Constant Frequency step motor emulator" presents a solution for emulating step motors when only constant frequencies are requested from it; section 3.3 presents an adaptation of the work by Schulte et al. [39] about emulating BLDC motors and a fixed point differential equation solver programmed in VHDL is described for the creation of the step motor emulator; finally section 3.4 presents an easier and cheaper solution where by reproducing the electrical model of the step motor the emulator was generated.

3.1 General Description

Seeing the step motor emulator as a black box helps to get an overview of the signals that will be read and the signals that will be generated when designing the emulator; figure 14 shows this black box scheme for the step motor emulator, the input signals being:

- four PWM voltage signals (V_A+, V_A-, V_B+ and V_B-) coming from the EC board; these signals are generated by the H-bridges and they will generate the average currents flowing through the windings of the step motor,
- one serial line (R_X) coming from the HILs for reading commands and activating the fault injection

and the output signals being:

- two currents flowing through the windings of the step motor one for A and one for B; in figure 14 the current in A is flowing in the positive direction while the current in B is flowing in the negative direction; with dotted arrows the currents are shown when flowing in the opposite directions,
- two square wave signals ($A_{Encoder}$, $B_{Encoder}$) simulating the behavior of an encoder,
- and one serial line (T_X) for sending information to the HILs.

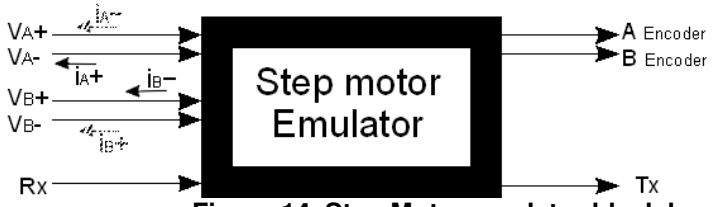


Figure 14. Step Motor emulator black box.

After the definition of the input and output signals, the rest of the chapter presents two different approaches for the creation of the emulator that lead in section 3.4 to the final step motor implementation.

3.2 Constant Frequency Step Motor Emulator

3.2.1 Overview

The first approach for the creation of the emulator was to reproduce the time constant of the step motor with an RC filter, but after implementing the idea the result was a step motor emulator for only constant frequencies that is not good for Océ since changing frequencies are needed while testing.

A modular design was considered for the emulator structure to have a distinction between electronics components and FPGA components; each one of the blocks performs self-related and independent tasks allowing high flexibility and maintenance. Figure 15 presents the different blocks of the emulator design and they are:

1. The *Motor Behavioral Simulation* block in charge of simulating the sinusoidal current behavior of the step motor (consequently the change in direction of the currents, in figure 32 the currents flowing in the positive direction are shown). The nominal value of the time constant τ per phase of the step motor is reproduced in this block with the help of RC filters as an attempt to reproduce the behavior of the step motor.
2. The *Signal Conditioning and Data Acquisition* block in charge of transforming the analog voltages coming from the EC board into digital values for interaction with the FPGA (where the data will be processed).
3. The *Step Detection and Fault Injection* block in charge of reading the digital signals in the FPGA from the A/D converters; of detecting the steps requested by the driver and of injecting the faults asked by the HILs.
4. The *UART Controller* block in charge of controlling the serial communication between the HILs or a computer and the emulator for receiving the fault injection commands.
5. The *Encoder Signal* block in charge of generating the encoder signals based on the information coming from the Step Detection and Fault Injection block.

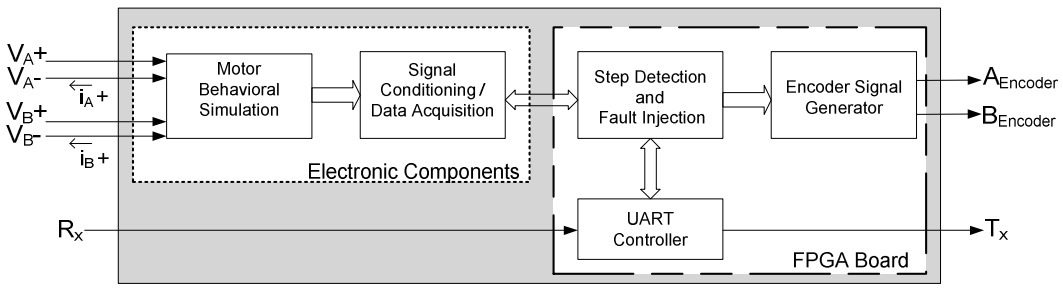


Figure 15. Step Motor Emulator Block Diagram

Electronics and FPGA components can now be distinguished from figure 15. The motor behavioral simulation and the signal conditioning/data acquisition blocks are electronics

components while the step detection and fault injection, the UART controller and the encoder signal generator are generated in the FPGA development board.

3.2.2 Motor Behavioral Simulation

This block is in charge of generating a current (i_A) with a similar sinusoidal (positive or negative currents) behavior to the one in the step motor. The behavior is expected to be reproduced by having a time constant in the emulator equal to the one in the step motors; an RC filter is used in the emulator to reproduce the time constant in the step motor. This block needs to convert also the generated current into voltages, with the purpose of use them in the next blocks of the emulator.

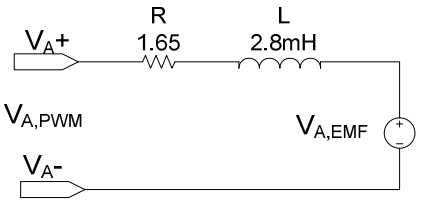


Figure 16. Electrical motor model phase A

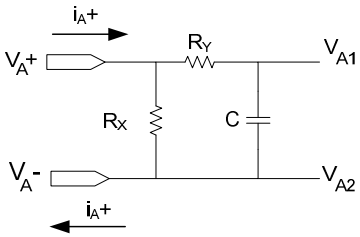


Figure 17. Resistor and RC filter

The time constant of the step motor is calculated with equation (4), which is derived from the electrical model of the motor presented in figure 16; the values of the circuit components are taken from the specification of the step motor.

$$\tau = \frac{L}{R} = \frac{2.8mH}{1.65\Omega} = 1.7ms \qquad (4)$$

Once that the time constant of the motor is known, formula 5 presents the calculation of the RC filter with a time constant equal to the one of the electrical model of the step motor.

$$R_Y = \frac{\tau}{C} = \frac{1.7ms}{0.22\mu F} = 7.73\Omega \qquad (5)$$

The step motors are controlled by the current flowing through them but the currents can not be read in the FPGA; a current to voltage converter is needed to transform currents into voltages, the voltages are then converted in digital values that can be read by the FPGA. The current to voltage converter is done with R_X, in figure 17; using $R_X = 1\Omega$ allows the driver to change the current value from 0 until $I_{\max}$ with the help of the PWM voltages.

3.2.3 Signal Conditioning\Data Acquisition

This block is divided into three parts needed to transform the voltages representing the currents requested by the driver, into digital values that can be read by the FPGA.

The first part is a voltage divider used to reduce the input voltages into smaller values so they can be used in the instrumentation amplifier. The instrumentation amplifier is the second block in charge of merging the two voltages controlling the current flowing through the motor and the third block is used to convert the analog voltages into digital signals to be read in the FPGA board.

The electronics components of the circuit are energized by the FPGA with a positive power supply with a maximum voltage of $3.3V$. Creating a constraint on the electronics circuit of using voltages only between $0V$ and $3.3V$.

The voltage divider is used to reduce the voltages coming from the step motor driver ($V_{A1} \leq 24V$) to a lower level ($V_{Ao1} \leq 1.6V$) to be able to use them in the rest of the electronics circuits. The voltage divider is shown in figure 18 and it has to be placed in all the inputs coming from the step motor driver, for instance V_{A1} and V_{A2} in figure 19.

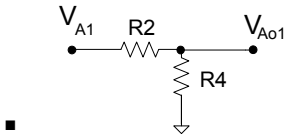


Figure 18. Voltage Divider

Having a low current flowing through this branch of the circuit is the reason of choosing $R2$ to be $10K\Omega$ in the RC filter; considering $V_{A1} = 24V$ as the maximum voltage applied to the circuit (Océ specifications) and $V_{Ao1} = V_{R4} = 1.6V$ as the maximum output voltage (for connection with the next stage), R4 can be calculated with formula 6, giving as a result $R4 = 714.29\Omega$ and $R4 = 680\Omega$ by choosing the closest commercial value.

$$V_{R4} = \frac{V_{A1}}{R2 + R4}(R4) \qquad (6)$$

The instrumentation amplifier (figure 19) used to merge the two input voltages into one, computes the difference between the reduced PWM voltage signals coming from the driver

$V_{out1} = V_{Ao1} - V_{Ao2}$ and $V_{out2} = V_{Bo1} - V_{Bo2}$

V_{out1} the output of the instrumentation amplifier will be changing between $-V_{Ao2}$ and V_{Ao1}, by choosing these values to be $1.6V$ V_{out1} is changing between $-1.6V$ and $1.6V$.

The electronics components are not able to handle negative voltages due to the power supply used by them and an offset is necessary to shift the negative voltages into positives values. If the offset voltage is $1.6V$, the output of the instrumentation amplifier will be shifted to the positive quadrant and the range of V_{out1} will be $0V \leq V_{out1} \leq 3.2V$ that can be used in all the electronic components.

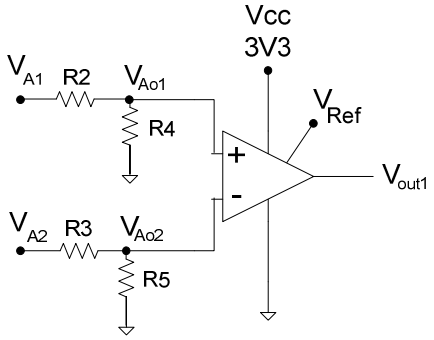


Figure 19. Instrumentation Amplifier

The analog to digital converter is the next stage in this block and it will convert the analog voltage values of V_{out1} and V_{out2} varying from $0V \leq V_{out1} \leq 3.2V$ and from $0V \leq V_{out2} \leq 3.2V$ to digital values using 8 bits that will be read with FPGA for the detection of the steps.

The output of these first two blocks is not the expected sinusoidal voltage signal similar to figure 10, instead as can be seen in section 4.1 only under constant frequencies the steps can be detected. This approach was stopped after implementing these two first blocks since the

step motors in Océ are varying its frequency, making this approach not good for the creation of the emulator.

3.2.4 Step Detection and Fault Injection

The two 8 bit digital signals values representing the voltages in the windings will be compared by using the current level percentages in figure 10 for the identification of the steps requested by the driver. This block is also in charge of reading the commands coming from the UART controller and to execute the faults received through the serial ports.

3.2.5 Encoder Signal Generator

This block will receive a pulse signal for every step the motor is requested to move and will receive commands from the serial port for the fault injection operation. The encoder signals will be generated based on the pulse signal and on the fault injection commands.

3.2.6 UART Controller

This block is in charge of the communications with the HILs or a PC through the serial port. It will read commands to execute fault injection and it will return the number of steps detected by the emulator.

3.3 Load Inductive Simulation for Step Motor

3.3.1 Overview

Adapting the solution by Schulte et al. [39] for a load inductive simulation for BLDC motors was the second approach to the creation of the Océ step motor emulator. In their work Schulte et al [39] created a simulator for different electric motors: permanent-magnet synchronous motors, BLDC motors and induction motors, where all can be simulated in the same electronic test bench, but the BLDC motor was the only one tested.

An introduction to their approach is given now to understand the construction of the BLDC emulator to be able to modify and to adapt this solution for creating the step motor emulator. Different from step motors to rotate the BLDC motor only 2 windings need to be energized at any time in a predefined sequence, leaving the third winding with a non-energized condition (floating phase). During the floating phase a voltage is induced to this line by the other two energized windings and by measuring this induced voltage sensorless control techniques can be applied to the motor.

The BLDC motor simulation can be explained coarsely with figure 20 and the following steps:

1. *PWM voltages conversion* situated in the converters module. Receives the PWM voltages from the driver and with sampling frequencies of 10 MHz convert them into digital values with the help of fast analog to digital (A/D) converters.
2. *Current calculation* situated in the FPGA block. Using the Forward-Euler integration method a 2 degree-of-freedom stator-oriented state space of the three-phase windings is solved in the FPGA. Parameters needed to solve the model like the inertia of the load, the coulomb friction, the torque, the back-EMF etc., are calculated or read from a conventional processor (processor board).
3. *Torque and back-EMF calculation* situated in the processor board. Based on the drawback that tool chains for implementing FPGA-based models are less flexible and maintainable compared with tools for normal processor based-implementations like Simulink. Schulte et al., decide to calculate the torque and the back-EMF in a conventional processor with a much lower sampling frequency comparing it with the one

in the FPGA, having as a result a simulator with a compromise between flexibility (in the processor board) and performance (FPGA board).

4. *Electronic Load Control/Current Sensor* situated in the voltage/current sources and switches block. To use sensorless control, the driver needs to read the back-EMF in one of the lines and currents in the other two lines; a state machine controlled by the generated currents will switch each one of the current lines between current source mode (when current is flowing through the winding) and voltage source mode (when there is no current flowing through the winding and the induced voltage needs to be generated (floating phase).

The simulation of the BLDC motors is still more complicated:

- the PWM phase voltages need to be read and the state space model of the motor needs to be solved to produce the output currents. A voltage needs to be generated in the floating phase to be able to close the loop every $1\mu s$. These calculations implied fast A/D and D/A converters per voltage line, switches and a state machine for control,
- the BLDC simulation needs to work sometimes as a current sink, current source or as a voltage supply,
- the switching is done with the help of state machines that are difficult to design because of the difficulty to define appropriate transition conditions,
- fixed or floating point operations inside the FPGA needs to be used for solving the state space model in real-time,
- the current supply needs to be faster than the PWM signals to accurately simulate the alternation of the current and a slew rate (current delivered by the power supply per unit of time) of $4A/\mu s$ is needed in the current supply for the BLDC emulator,

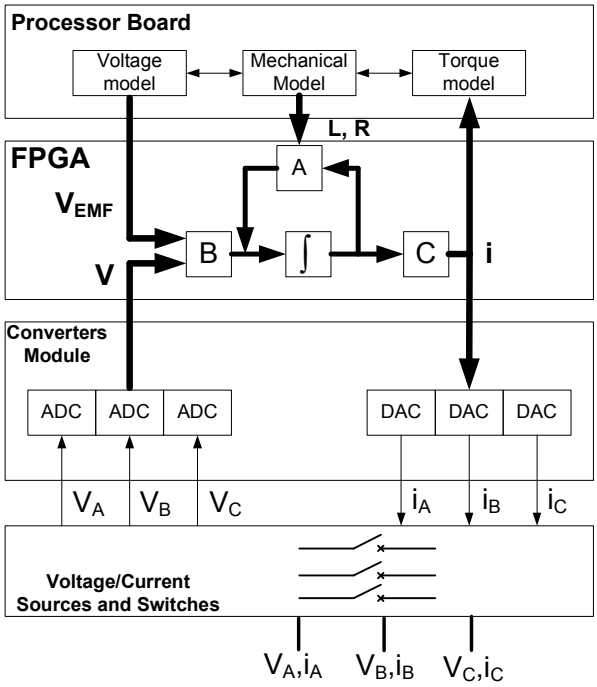


Figure 20. BLDC Simulator [39]

- auxiliary power supplies are used to maintain the current flowing during the freewheeling phase (transistors are off),
- a heat sink for dissipating up to 350 W when working in high current mode.

are issues that need to be solved when using this approach for emulating BLDC motors.

Adapting this solution for a step motor emulator was one of the ideas of this project and section 3.3 shows how to derive the state space design for a step motor and presents a

differential equation solver using the Euler method for solving the equations representing the model of the step motor.

3.3.2 Tools Solving the State Space Model of the Step Motor

Solving the state space model of the step motor will give as output the simulated currents flowing through the windings of the step motor, making it possible to identify the steps by using figure 10. Different from the design of Schulte, the solution of the state space model of the step motor was designed to be calculated completely in the FPGA.

3.3.2.1 Step Motor State Space Modeling

A schematic representation of a step motor connected to a mechanical load is presented in figure 21 to derive the dynamics of the step motor according to [6]; equations (7) and (8) represent the relation between current and voltage in the windings based on the electrical model of the motor in figure 16.

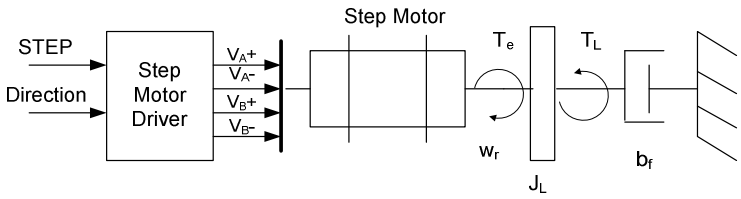


Figure 21. Step motor connected to a load

Considering w_r the rotor angular speed, J_L is the inertia of the load, T_L the dynamic load torque and b_f the viscous friction we have,

$$\frac{di_A(t)}{dt} = \left(V_{A,PWM} - V_{A,EMF} - Ri_A(t)\right)\frac{1}{L} \quad (7)$$

$$\frac{di_B(t)}{dt} = \left(V_{B,PWM} - V_{B,EMF} - Ri_B(t)\right)\frac{1}{L} \quad (8)$$

$$V_{A,EMF} = -k\sin\left(p\theta_r(t)\right)w_r(t) \quad (9)$$

$$V_{B,EMF} = k\cos\left(p\theta_r(t)\right)w_r(t) \quad (10)$$

Where θ_r represents the rotor angle, J_R the rotor inertia of the step motor, T_f the friction torque, c_f the coulomb friction, i_a and i_b represent the currents flowing through winding A and B respectively and $V_{A,PWM}$,$V_{B,PWM}$, are the PWM voltages coming from the driver; these voltages are generated in the step motor emulator by the "Signal Conditioning" block in figure 15, two instrumentation amplifiers (one per phase) are the ones in charge of generating these two voltages.

Using the values of the holding torque (T_{pk}) of $4.4kg \cdot cm$ given by the supplier and I_o representing the rated current per phase equal to $1.68A$; it is possible to calculate the value of k used in equations (9) and (10).

$$k = \frac{T_{pk}}{I_O} = 26.190476 \cdot 10^{-3} \quad (11)$$

Equations (9) and (10) need the value of p, which represents the number of rotor teeth and can be found from equation (13).

$$step\_size = \frac{360°}{200\,full\_steps} = 1.8° \tag{12}$$

$$step\_size = \frac{360°}{4 \cdot p} \Rightarrow p = 50 \tag{13}$$

The electrical model of the motor (equations (7) and (8)) is related with the mechanical part of the motor through the electric torque T_e produced by the step motor,

$$T_e = -k\left(i_a \sin\left(p\theta_r(t)\right) - i_b \cos\left(p\theta_r(t)\right)\right) \tag{14}$$

The relation between the electric torque T_e and the mechanical dynamics of the load are given by,

$$\frac{dw_r(t)}{dt} = \frac{1}{J_L + J_R}\left(T_e(t) - T_f(t) - T_L\right) \tag{15}$$

$$\frac{d\theta_r(t)}{dt} = w_r(t) \tag{16}$$

$$T_f = b_f w_r(t) + c_f sign(w_r(t)) \tag{17}$$

The model of the motor represented by equations (7), (8), (15) and (16) will be solved in the FPGA in less than $1\mu s$ to be able to use it in the HIL simulator. The Euler method was chosen for this purpose based on its simplicity, the lack of accuracy of the Euler method is compensated by oversampling the PWM voltage input signals.

3.3.2.2 Euler Method

Using numerical methods for calculating the solutions of differential equations has been studied for a long time, and the Euler method presents the easiest implementation algorithm while showing a small error if choosing a small step size.

To explain the Euler Method consider the ordinary differential equation,

$$y'(t) = f(t, y(t)), \quad 0 \le t \le b, \tag{18}$$

with $y(t_0) = y_0$ representing the initial condition. Formula 18 represents the change of function f in time.

In [50] the Euler method is presented as a method to solve numerically differential equations using the knowledge that any point in a curve can be calculated by having the equation of that curve and another point on it. Considering that formula 18 represents the tangent of function f and the initial condition represents the point in the curve, it is then possible to calculate another point in the tangent curve that can be considered to be also in f if the distance between the points (h) is small.

Rewriting equation (18) as

$$y'(t) \approx \frac{y(t+h) - y(t)}{h},$$

$y(t+h) \approx y(t) + hy'(t)$ and by using (18)

$$y(t+h) \approx y(t) + hf(t, y(t))$$

The step size h indicates how far we want that point to be calculated and by using the initial conditions as starting point, the rest of the solution can be calculated with:

$$y_{n+1} = y_n + hf(t_n, y_n) \quad (19)$$

$$t_{n+1} = t_n + h \quad (20)$$

To use these equations in the FPGA t_n is replaced by k to represent discrete time, and the equations of the step motor to be used with the Euler method are,

$$\frac{i_A[k+h]-i_A[k]}{\Delta k} = L^{-1}U_{A,PWM} - L^{-1}U_{A,EMF} - R\cdot L^{-1}i_A[k]$$

$$\frac{i_B[k+h]-i_B[k]}{\Delta k} = L^{-1}U_{B,PWM} - L^{-1}U_{B,EMF} - R\cdot L^{-1}i_B[k]$$

$$\frac{\theta[k+h]-\theta[k]}{\Delta k} = w_r[k]$$

$$\frac{w_r[k+h]-w_r[k]}{\Delta k} = (J_L+J_R)^{-1}T_e[k] - (J_L+J_R)^{-1}T_f[k]$$

where $T_L = 0$ since there is no load in the motor when using the HILs; from these equations solving them for the $[k+h]$ terms, we have,

$$i_A[k+h] = i_A[k] + \left(L^{-1}U_{A,PWM} - L^{-1}U_{A,EMF} - R\cdot L^{-1}i_A\right)\cdot\Delta k \quad (21)$$

$$i_B[k+h] = i_B[k] + \left(L^{-1}U_{B,PWM} - L^{-1}U_{B,EMF} - R\cdot L^{-1}i_B\right)\cdot\Delta k \quad (22)$$

$$\theta[k+h] = \theta[k] + w_r\cdot\Delta k \quad (23)$$

$$w_r[k+h] = w_r[k] + \left((J_L+J_R)^{-1}T_e[k] - (J_L+J_R)^{-1}T_f[k]\right)\cdot\Delta k \quad (24)$$

Equations (21) to (24) are the equations to be programmed in the FPGA. These equations will give as a result the currents flowing through the windings, the angle and the speed of the rotor.

There are still some challenges after finding all the constant values for the step motor: the FPGA can not use real numbers. So an algorithm needs to be developed to calculate the addition and multiplication for fixed point numbers in the FPGA as well as an algorithm to calculate the sine and cosine values for the arguments generated while solving the Euler equations. Difficulties are solved and described in the next sections.

3.3.2.3 Fixed Point Numbers Representation

There are already libraries by the IEEE to work with fixed point or with floating point [4]. Unfortunately these libraries work only under version 8.1 of the ISE XST software from Xilinx that comes with the FPGA and is used to codify applications for the FPGA, and only if using Synplicity to synthesize code, which was the reason for developing our own algorithms.

Using the Euler method for solving differential equations means implicitly the use of real numbers. Using VHDL to program the FPGA means describing hardware, making it impossible to describe real numbers with hardware elements. Two solutions exist for this problem: using floating or fixed point numbers. Fixed point numbers were chosen for being easier to implement than floating point numbers and because less computational effort is required when comparing them with floating point operations.

A fixed point number representation means that numbers will be represented with a certain number of bits and a virtual radix point that will be fixed and virtually placed somewhere in this number. For instance considering the 4 bit binary number $0b1111$ without fixed point representation its value is,

$$0b1111 = 1\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0 = 15$$

When using a fixed point representation, the radix point can be placed anywhere in the number but the interpretation given to the number is different. For instance, considering the

same 4 bit binary number, placing the virtual radix point to have two bits for the integer part and two bits for the fractional, we have a different interpretation in its value,

$$0b11.11 \Rightarrow 1\cdot 2^{1}+1\cdot 2^{0}+1\cdot 2^{-1}+1\cdot 2^{-2}=3.75$$

The model of the step motor described with equations (21) to (24) is represented with a 32 bit number, having 1 bit for the sign, 6 bits for the integer part and 25 bits for the fractional part.

The 6 bits (maximum value of 63 in decimal) for the integer part are enough to represent the values of the currents flowing through the step motor since they will never exceeds 4 amperes in the step motor, the angle of the shaft is as maximum $360° \rightarrow 2\pi = 6.2832$ radians, and based on the simulation of the motor by Océ the angular speed of the motor oscillates between 1 and -1. None of these values exceed 63. In other words because of **physical restrictions** the numbers representing the integer part will not exceed the 6 bits assigned to them and 6 bits were chosen to use powers of two terms.

On the other hand, 25 bits are necessary in the fractional part since the sampling time is fixed to $5\mu s$ (the time to read the digital values from the A/D converter) causing results of some of the multiplications to be numbers with a factor of 10^{-9}; these numbers can be represented using 25 bits in the fractional part, having as a minimum value to represent $29.8023\cdot 10^{-9}$.

A function in Matlab that reads decimal numbers and convert them to hexadecimal 32 bits fixed point representation is shown in figure 22. This function is called "dec2hxfp" and it is used to represent the numbers in hexadecimal form and use them in the Euler method.

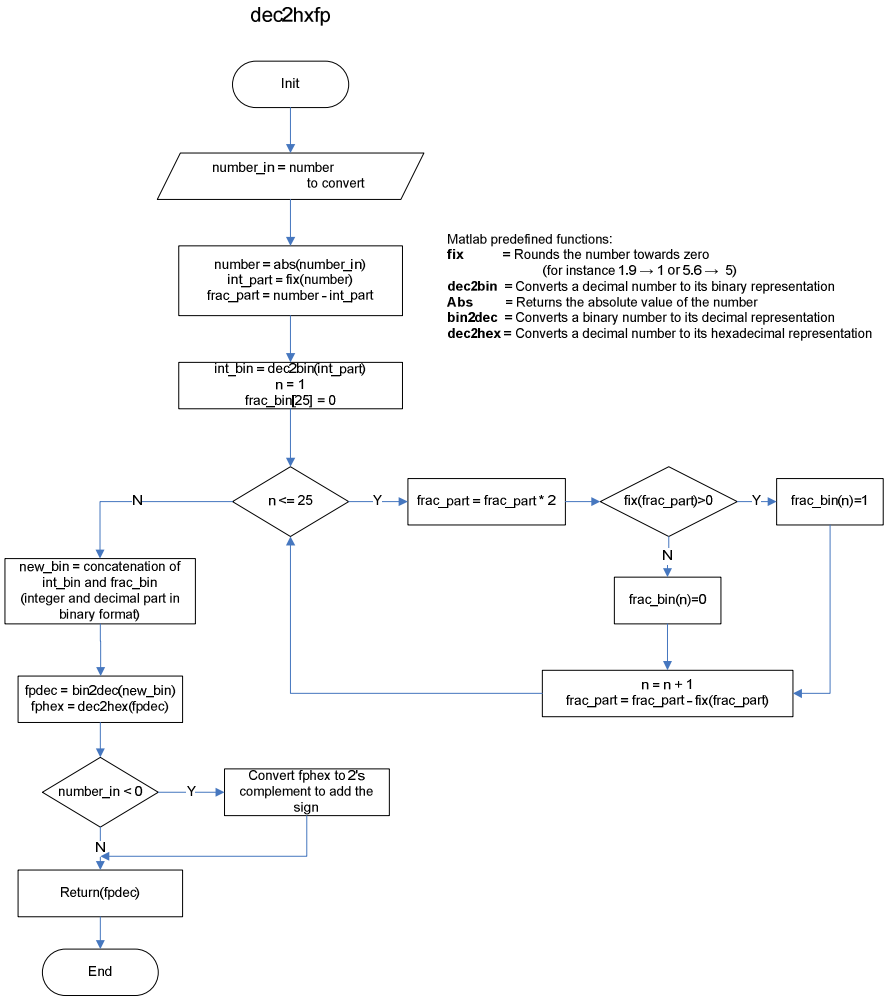


Figure 22. Decimal to Hexadecimal Fixed Point

A function to generate decimal numbers from a hexadecimal 32 bits fixed point number is shown in figure 23 and it is designed to transform the hexadecimal result values of the Euler method into decimal numbers.

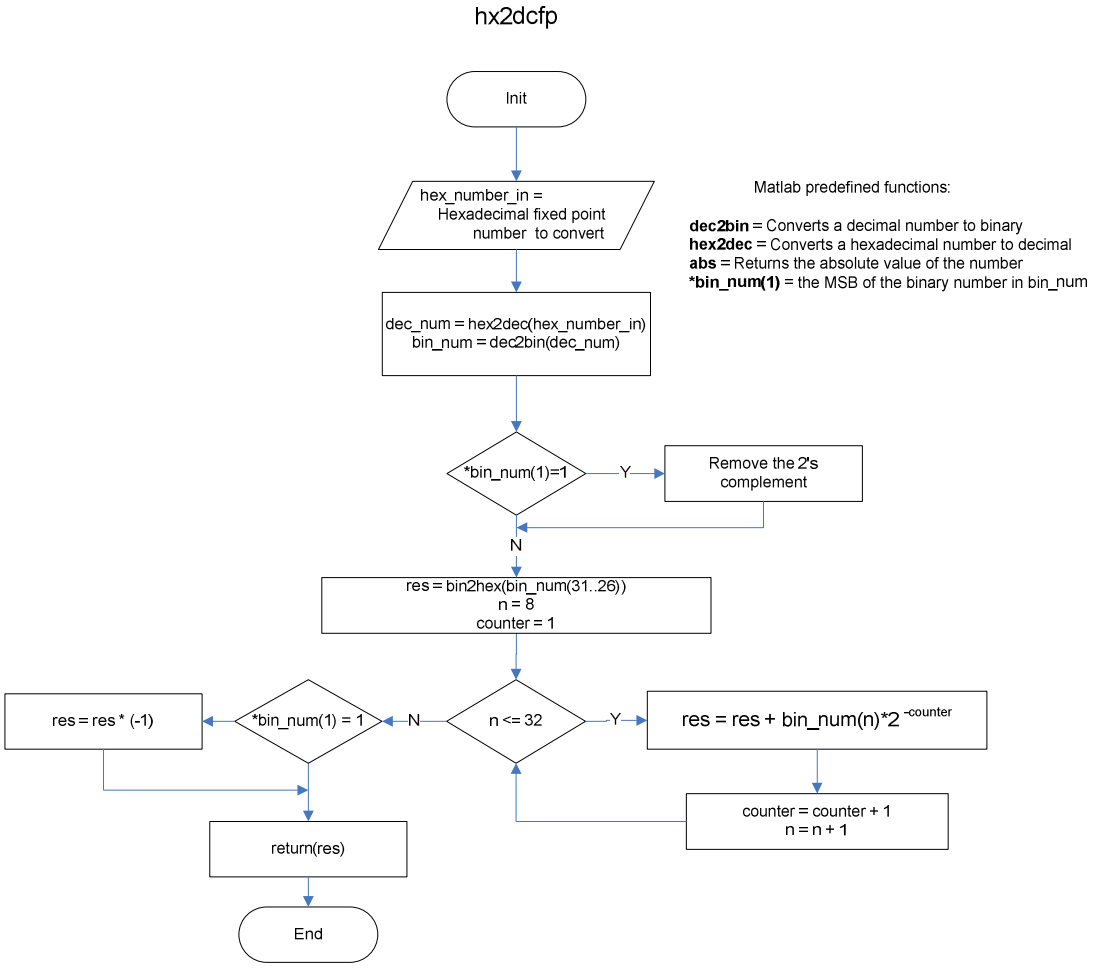


Figure 23. Hexadecimal Fixed Point to Decimal.

The parameters and constant values for the equations (21) to (24) are taken from the motor specification and will be substituted in the equations (21) to (24). After having the equations with decimal numbers they will be transformed into hexadecimal 32 bits fixed point numbers for solving the Euler method into the FPGA.

After having the equations in this format the following sections will indicate the procedure to calculate the solution of the Euler equations where two packages in VHDL were created, one for the addition and multiplication and the other one for the sine/cosine calculation.

3.3.2.4 Addition of Fixed Point Numbers

After converting the decimal numbers to hexadecimal 32 bits fixed point numbers, a normal addition operation with the standard IEEE libraries can be performed, since the numbers are hexadecimal values. The difference will be in the interpretation of the results, the 32 bit hexadecimal numbers represent a decimal number when the virtual radix point is placed on them.

When doing addition operations of hexadecimal numbers the results can be 1 bit more than the maximum length of the numbers being added. In the present algorithm the result is stored

in a 32 bit variable, while the summands are also 32 bit, this is done because the values will not exceed the 6 bits assigned to the integer part due to the physical restrictions presented in section 3.3.4.

$$result = summand1 + summand2$$
$$result = add(summand1, summand2)$$

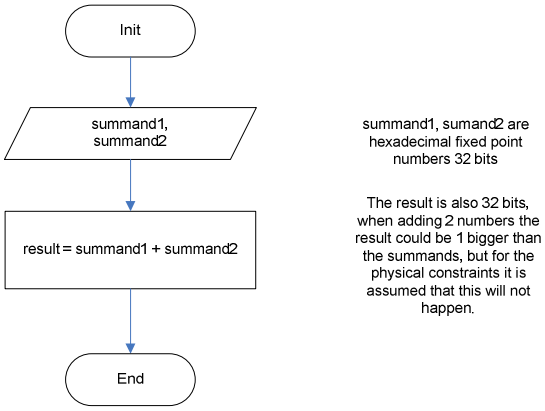


Figure 24. Addition

3.3.2.5 Multiplication of Fixed Point Numbers

Three different algorithms for multiplying two numbers are introduced; the three algorithms are to simplify the operations for solving the state space model of the step motor with the Euler method.

The first algorithm does the multiplication of two 32 bits numbers:

$$result = factor1 \times factor2$$
$$result = multi(factor1, factor2)$$

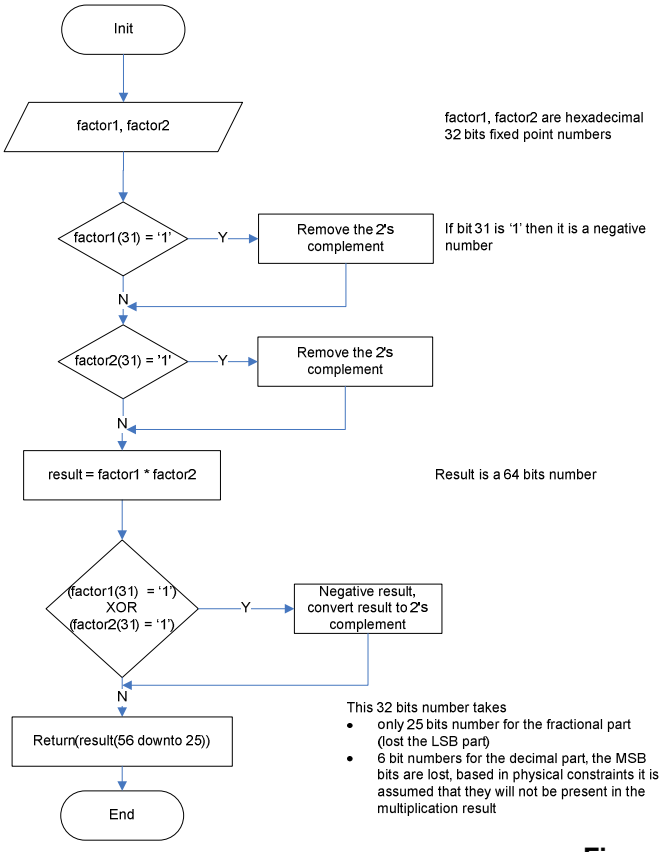
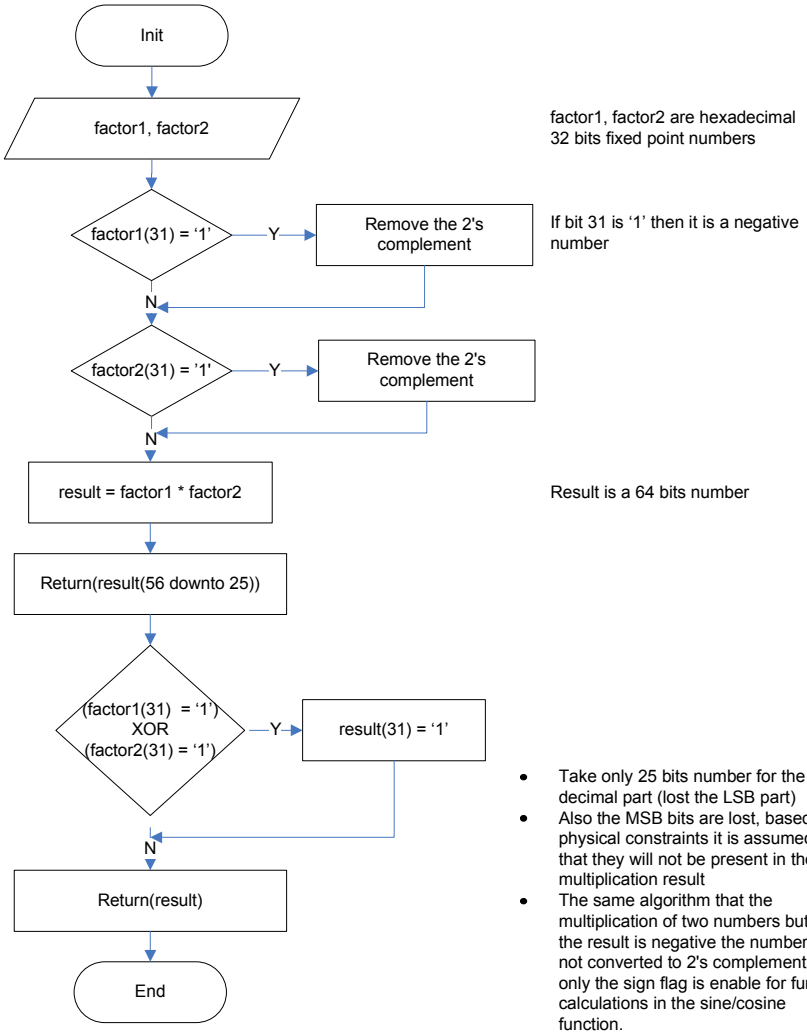


Figure 25. Multiplication

The second algorithm gives the result of the multiplication between p and θ, this result is used as argument for the sine and cosine functions.
If the argument is negative, then the number will not be transformed to 2's complement only the bit sign will be set and the sine/cosine function will calculate the correct value of the functions using the identities:

$$\sin(-\alpha) = -\sin(\alpha)$$
$$\cos(-\alpha) = \cos(\alpha)$$

$$result = p \times \theta$$
$$result = p\_times\_theta(factor1, factor2)$$



- Take only 25 bits number for the decimal part (lost the LSB part)
- Also the MSB bits are lost, based in physical constraints it is assumed that they will not be present in the multiplication result
- The same algorithm that the multiplication of two numbers but if the result is negative the number is not converted to 2's complement, only the sign flag is enable for further calculations in the sine/cosine function.

Figure 26. Argument calculation

Finally the third multiplication algorithm is for the multiplication of a number times the sign of another one.

$$result = factor1 \times sign(factor2)$$
$$result = mult\_sign(factor1, factor2)$$

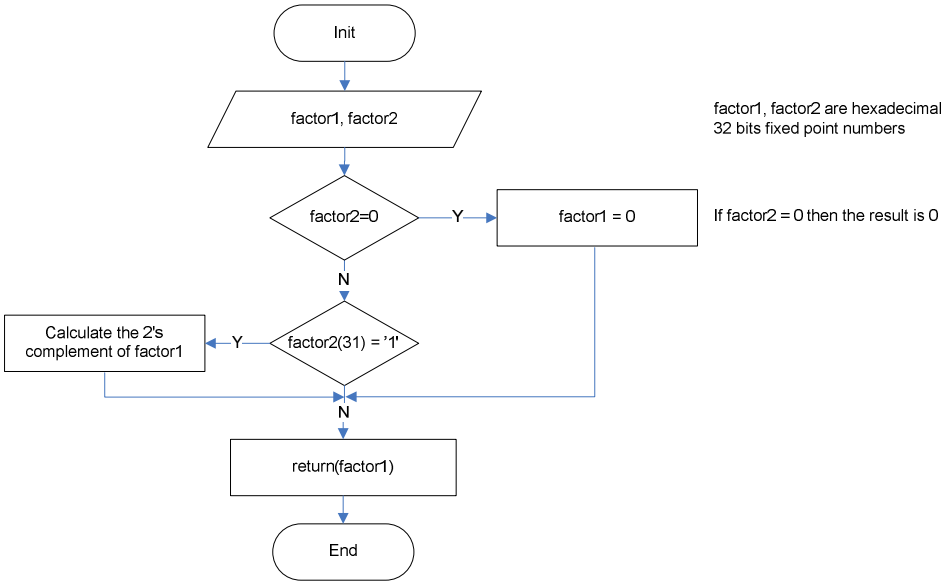


Figure 27. Multiplication of a number times the sign of another one.

3.3.2.6 Sine/Cosine Function Generator

This function generator will give the result of the sine and cosine mathematical functions A ROM of 800 positions is created to store 800 values for the sine and for the cosine mathematical functions. 800 values were chosen because the motor has 800 steps when working in a ¼ step mode. Using quarter steps for microstepping means that every step will be subdivided in four microsteps, the step motor moves a total of 200 steps per revolution, this is 800 steps after dividing the steps into microsteps.

Addressing the 800 values is done by splitting the calculated argument in 2 parts the "high argument" ranging from 0 to 200 and the "low argument" ranging from 0 to 3; as shown in figure 29, by multiplying the high argument by 4 and then adding the lower argument it is possible to address 804 values.

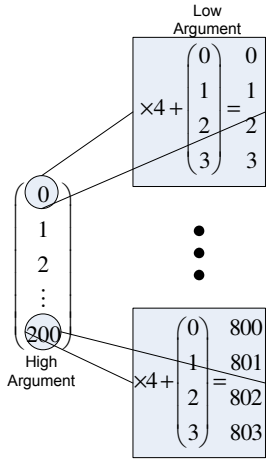


Figure 28. Addressing algorithm

Table 4 shows how to derive the high and low argument from 32 bit fixed point numbers representing angles. For each one of the angle values in table 4 the corresponding sine and cosine value is stored in the ROM. Looking at bits 16 to 19 the hex values $0x0, 0x4, 0x8, 0xC$

are repeated through the entire table. From this series bits 18 and 19 will form the low argument as shown in table 5, while the high argument will be formed from bit 20 to bit 27 as showed in table 6.

Table 4. Sine/Cosine addressing

	Degrees	**Radians**	**32 bit Fixed Point HEX value**								
			bits	31-28	*27-24*	*23-20*	**19-16**	15-12	11-8	7-4	3-0
0	0.00	0	0x	0	*0*	*0*	**0**	0	0	0	0
1	0.45	0.00785	0x	0	*0*	*0*	**4**	0	5	6	F
2	0.90	0.01571	0x	0	*0*	*0*	**8**	0	-	-	-
3	1.35	0.2356	0x	0	*0*	*0*	**C**	1	0	0	0
.	.	.		.	.	.	.	.	.	.	.
.	.	.		.	.	.	.	.	.	.	.
.	.	.		.	.	.	.	.	.	.	.
796	358.20	6.251784	0x	0	*C*	*8*	**0**	D	3	E	C
797	358.65	6.259638	0x	0	*C*	*8*	**4**	D	9	5	5
798	359.10	6.267492	0x	0	*C*	*8*	**8**	D	E	B	F
799	359.55	6.275346	0x	0	*C*	*8*	**C**	E	4	2	8
800	360	6.2832	0x	0	*C*	*9*	**0**	E	9	9	1

Table 5. Low Argument values for addressing, bits 18 and 19 form the Low Argument

Hex	19	18	17	16		19	18	decimal
0	0	0	0	0	→	0	0	**0**
4	0	1	0	0	→	0	1	**1**
8	1	0	0	0	→	1	0	**2**
C	1	1	0	0	→	1	1	**3**

Table 6. High Argument values for addressing, bits 20 to 27 form the High Argument

	27-24	*23-20*		decimal
0x	*0*	*0*	→	0
0x	*0*	*0*	→	0
0x	*0*	*0*	→	0
0x	*0*	*0*	→	0
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
0x	*C*	*8*	→	200
0x	*C*	*8*	→	200
0x	*C*	*8*	→	200
0x	*C*	*8*	→	200

Using the fact that $\sin(-\alpha) = -\sin(\alpha)$ and $\cos(-\alpha) = \cos(\alpha)$ the sine/cosine function will only look at the sign bit for the sine algorithm to transform it to negative using 2's complement representation.

In the ROM memory of the FPGA eight hundred values for the sine and eight hundred values for the cosine are stored for calculating the sine and cosine functions needed in the solution of the differential equations.

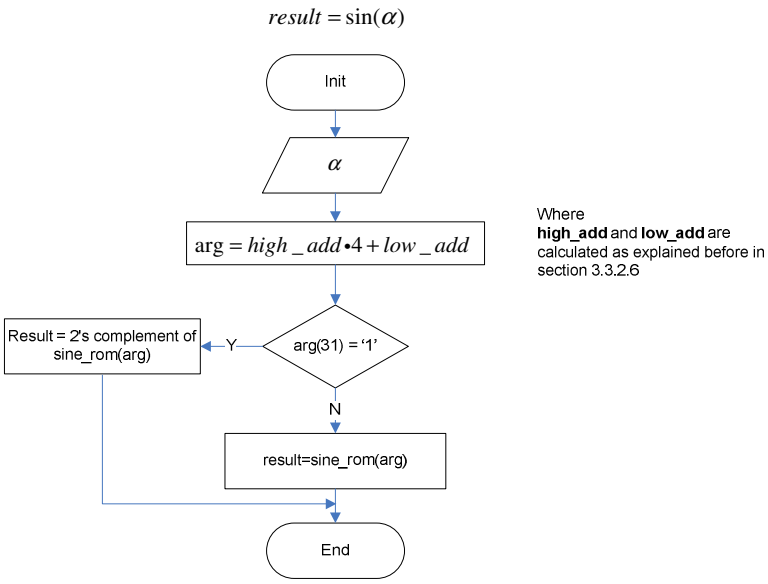


Figure 29. Sine calculation

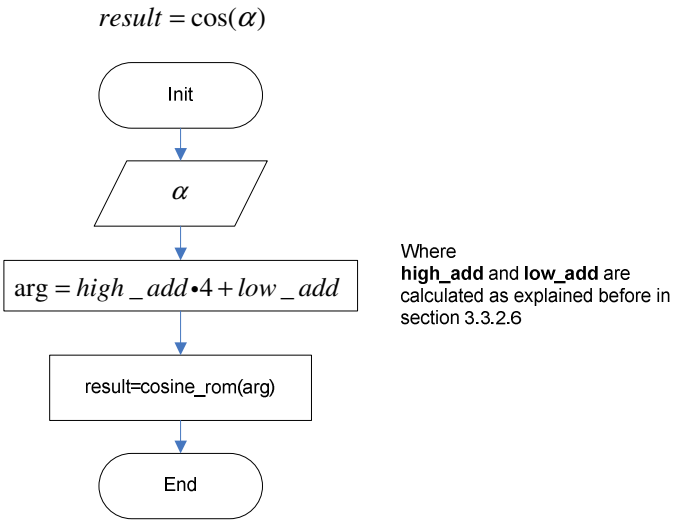


Figure 30.Cosine Calculation

.

3.3.3 Solving the State Space Equations of the Step Motor

Equations (21) to (24) will be solved in the FPGA with the Euler method, and to have a better performance they are divided in eight states (see figure 31). Each one of the states is computing independent arithmetic operations the results of these operations are needed in the next state and the results of the new state are needed in the next state and so on until reaching the last state where the final values are calculated.

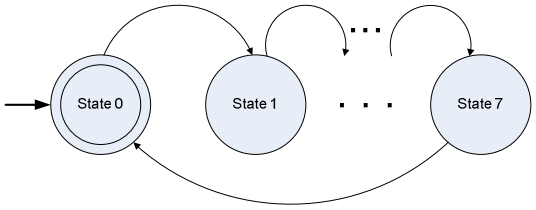


Figure 31. Solving the State Space Equations

The operations in each state are independent and can be executed in parallel while the states need to be executed sequentially due to data dependencies. For instance in state 0 the operations calculated are,

- $iaterm1 = \left(R \times L^{-1}\right) \times i_A$, where $\left(R \times L^{-1}\right)$ is calculated with the help of Matlab and it is introduced in the multiplication as a constant value,
- $iaterm2 = L^{-1} \times U_{A,PWM}$,
- $argument = p \times \theta$,
- $ibterm1 = \left(R \times L^{-1}\right) \times i_B$, where $\left(R \times L^{-1}\right)$ is the constant value calculated before,
- $ibterm2 = L^{-1} \times U_{B,EMF}$

All of these operations can be executed in parallel since their parameters are either constant values or input values, all the operations calculated per state are independent from each other, and their results are needed in the next states. For instance, state 1 needs the result of $argument$ calculated in state 0 to calculate the $high\_arg$ and low_arg (see below) needed to find the sine and cosine values of the argument in state 2.

- $high\_arg = argument(27 \text{ downto } 20)$;
- $low\_arg = argument(19 \text{ downto } 18)$;

After eight states all the operations needed to solve the equations are calculated and the currents flowing through the windings, the angle and the speed of the rotor results are available. The state machine is receiving as input values the PWM voltages every $5\mu s$, the time it takes to the A/D converter to convert new values.

3.3.4 Current Generation

After the currents are calculated in the FPGA a switching block needs to be designed. This switching block will change between current sink and current source mode.

- Current Sink mode, the current is flowing to the step motor emulator and the identification of the steps will take place in the FPGA.
- In Current Source mode, the calculated currents need to be fed back to the step motor driver. This is necessary since the step motor driver has the current flowing through the step motor as a feedback signal. Based on the current in the step motor the driver knows if it has to increase or decrease the current flowing through each one of the windings and move the step motor one more step (the current values are shown in figure 10).

In the present work after finishing the design and the implementation of the differential equation solver and while designing the switching block and the current sources, this approach was stopped since the approach shown in the next chapter presented a breakthrough to the creation of the step motor emulation, since needs less electronic components and it is not necessary to solve the differential equations as the Load Inductive method making it an easier alternative for the development of the step motor emulator.

This section is presented as reference for future work to someone who would like to continue implementing the step motor emulator or another inductive load emulator with Schulte's approach; since by using Schulte's approach is possible to emulate all kind of inductive loads not only step motors and where emulated real currents and voltages are generated for testing all kind of motor controllers in HIL simulators.

3.4. Real-Time Step Motor Emulator

3.4.1 Overview

In section 3.2 an RC filter was placed in the output lines coming from the EC, this filter was unable to reproduce the sinusoidal behavior of the current flowing through the step motor (figure 10). An inductor is present in the electrical model of the motor shown in figure 16; the value of the inductor affects the current flowing through the motor. Therefore, placing an inductor across the outputs of the voltage signals coming from the EC is presented as the third approach for simulating the sinusoidal behavior of the current in the step motors.

A modular design similar to the one proposed in section 3.2 is used also in this approach; inheriting the advantages of a modular design by implementing independent tasks in each block results in high flexibility and maintenance for the emulator.

The block diagram is presented in figure 32 and it consists of the following components:

1. the *Motor Behavioral Simulation* block in charge of simulating the behavior of the step motor; it includes two inductors of $1mH$ to generate the sinusoidal current similar to the one flowing through the phases in a step motor; in figure 32 the currents are flowing in the positive direction, when they flow in the negative direction the arrows will be in the positive voltage lines with a negative sign; the currents need to flow in both ways to simulate the sinusoidal current wave; this block also has a resistor to convert the currents into voltages and together with the inductors form the load for the voltage signals coming from the EC.
2. the *Signal Conditioning and Data Acquisition* block, which is in charge of transforming the voltages coming from the EC board into smaller signals to use the A/D converters and for processing them into the FPGA. This block is necessary to read the analog PWM voltages from the EC into the FPGA where they will be processed.
3. the *Step Detection* block in charge of reading the digital signals from the A/D converter; these voltage levels are compared with the current levels in figure 10, the maximum voltage read by the FPGA will represent $I_{\max} = 100\%$ from where the rest of the voltages will be calculated, this block will generate 2 signals, a pulse signal that with a rising edge will annunciate a one step change and the second signal that will be high when the step motor is moving forward and will be low when the motor is moving backwards.
4. the *Encoder Signal Generator* block in charge of generating the two encoder signals that will be seen in the output based on the information coming from the Step Detection block.
5. the *Fault Injection* block receives the encoder signals from the encoder signal generator block and receives commands from the UART controller with the faults to be injected, based in these commands this block will modified or not the encoder signals before sending them out of the emulator.
6. the *UART controller* block in charge of managing the serial communications between a computer or the HILs and the emulator. This block will send commands to the fault injection block requesting the injection of faults.

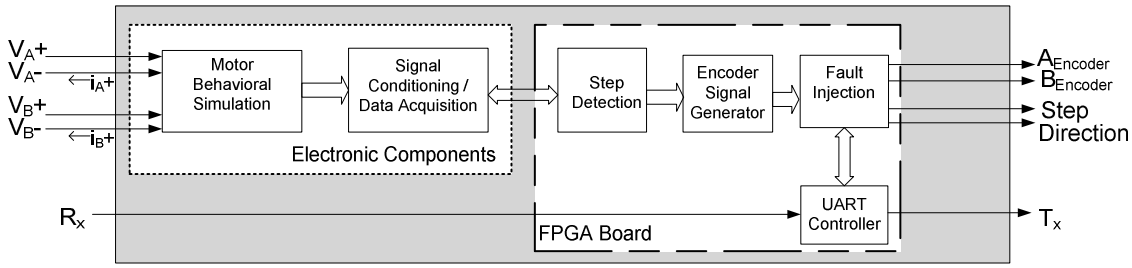


Figure 32. Step Motor Emulator Block Diagram

The Electronic and the FPGA components can also be distinguish from figure 32, the motor behavioral simulation and the signal conditioning/data acquisition blocks are electronic components while the step detection, the fault injection, the UART controller and the encoder signal generator are generated by the FPGA development board.

The rest of this section will explain the purpose and the behavior of each one of the blocks for the creation of the step motor emulator.

3.4.2 Electronics Design

3.4.2.1 Motor Behavioral Simulation

This block is reproducing the sinusoidal behavior of the current through the windings of the step motor. Based on the electrical model of the step motor (figure 16) the circuit shown in figure 33 is composed of two $1mH$ inductors and a 1.2Ω resistor in series. The inductors are reproducing the sinusoidal behavior of the current flowing through the windings of the step motor and the resistor is used to transform the current into voltages to be read in the next blocks.

When a positive voltage is applied to the terminal V_A+ the current i_A will flow to the terminal V_A- and it is called i_A+; current i_A- (dotted line in figure 33) will flow when the positive voltage is applied to terminal V_A-; making the change in the direction of the current. The change in direction of the currents is needed in the step motor to move step by step and it is handled by the step motor driver.

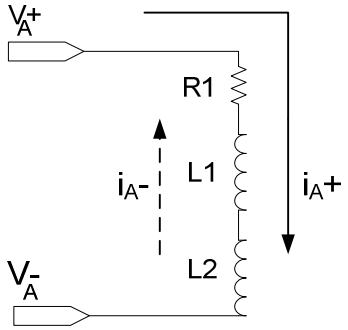


Figure 33. Electronic circuit for the behavioral simulation of the Step Motor

The nominal inductance according to the step motor specification per phase of the motor is $2.8mH$. Based on this nominal value three $1mH$ inductors with a total inductance of $3mH$, were placed in the circuit. After increasing the frequency requested by the step driver to $9KHz$ (the maximum frequency asked by Océ) a steep current is seen flowing through the circuit a better performance for high frequencies is obtained by using only two $1mH$ inductors.

Having a better performance with the two $1mH$ inductors could be caused because the change in the current is faster for smaller inductance values as can be deducted from equation (25). If the current is changing fast (lower inductance values) then the voltage levels at the output of the instrumentation amplifier also are changing fast, making it easy to distinguish between voltage levels representing the steps. On the other hand, when two consecutives voltage levels, representing two consecutive steps are close to each other and the change from one step to the other is slow, then the emulator step identification algorithm will be detecting the two consecutive steps alternatively until the voltages are different enough from each other.

$$\frac{di}{dt} = \frac{1}{L}V(t) \qquad (25)$$

The resistor R1 was chosen to be 1.2Ω to let the driver swing along all the currents and to have less power consumption while working in high current mode (in consequence less heat is produced by the resistor). Calculating the power for R1 during high current mode ($i_{peak} = 1.62A$) we have,

$$P_{R1.2\Omega} = i_{RMS}^2 R = 1.5746W$$

Since heat was one of the main concerns for the step motor emulator, a resistor of $11W$ is used in the design, because it satisfies the power requirements and because it was available at Océ.

3.4.2.2 Signal Conditioning / Data Acquisition

This block is similar to the one presented for the constant step motor emulator in section 3.2.3; a voltage divider to reduce the voltages coming from the step motor driver and being able to use them in the electronics components part; an instrumentation amplifier to calculate the difference between the voltages per phase; and the digital to analog converter to allow the FPGA to read the digital voltage values representing the current flowing through the behavioral simulation circuit.

The voltage divider shown in figure 18 will also be used in this module and will also be placed in all the inputs coming from the step motor driver (V_A+, V_A-, V_B+, V_B-) and it has the same task, to compute the difference between the voltage signals coming from the driver.

$V_{out1} = V_{Ao1} - V_{Ao2}$ and $V_{out2} = V_{Bo1} - V_{Bo2}$

The value of $1.6V$ for V_{Ao2} and V_{Ao1} is also used because it is also needed to have the same maximum variations for V_{out1} and V_{out2} ($0V \le V_{out1} \le 3.2V$ and $0V \le V_{out2} \le 3.2V$) since the rest of the electronics components are powered up by a $3.3V$ positive power supply from the FPGA development board.

The output voltage V_{out1} of the instrumentation amplifier has the sinusoidal behavior of the current flowing through phase A of the step motor (figure 34) but V_{out1} change based on the current selection from the EC (high or low). When the EC is in low current mode (appendix A) V_{out1} never reaches $0V$ nor $3.3V$. For this reason a variable resistor of $100k\Omega$ (chosen from the specification datasheet of the instrumentation amplifier) is used in the gain selection pins of the instrumentation amplifier (R6). This variable resistor allows to change the gain of the amplifier and have V_{out1} varying through the entire voltage range $0V \le V_{out1} \le 3.3V$.

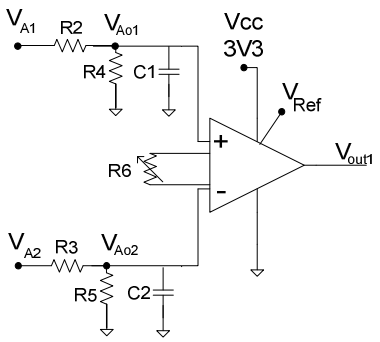


Figure 34. Signal conditioning circuit.

An RC filter is placed in the input pins of the amplifier to reduce the spikes produced by frequencies higher than $10KHz$, since the maximum frequency requested by Océ is $9KHz$. Calculating the constant time τ of the RC filter, the input pin of the instrumentation amplifier sees a resistance value of R_2 in parallel with R_4 so,

$$\tau = \left(R_2 \| R_4\right) \cdot C_1 \qquad (26)$$

The maximum frequency requested by the driver is $9KHz$, which means a period of $100\mu s$ making this value 4 times smaller we have $\tau = 25\mu s$ and the value of C_1 is calculated using equation (25), giving as a result $C_1 = 39.26nF$. A capacitor of $47nF$ was chosen from commercial values for these filters.

The next step is the analog to digital converter. The IC chosen for this purpose is the AD7819 from Analog Devices. This circuit has a conversion time of $4.5\mu s$, enough for an oversampling factor of 23 times when the maximum frequency $9.306KHz$ is requested from the step motor driver ($107.46\mu s$ seconds per step).

The analog to digital converter has four control signals that are handled by the FPGA to do the analog to digital conversions and to read the generated digital value from the conversion. Information about the state machine to control this IC is found in section 3.4.3.2.2 and the specification of the IC is found in [3].

It is important to notice that the CONVST pin, which is the pin to control the start of a conversion, needs to be low when powering up the IC. This can not be done in the FPGA because powering up the FPGA means powering up the rest of the electronic components since the power supply for the electronic components is taken from the FPGA development board. The output ports of the FPGA are in their initial state (generally the initial state of the ports is high impedance or a logic one) when powering up the board, causing the CONVST pin of the converter to have an incorrect initial state.

To satisfy the requirement of having the CONVST pin low when powering up a switch is placed on this pin; the switch will change between ground and the FPGA port dedicated to it. Care should be taken to have the switch connected to ground (switch off) when powering up the converter and to turn the switch on (FPGA port) before starting to use the emulator.

3.4.3 FPGA Components Design

3.4.3.1 Top-Down design

A **top-down** approach was used for designing the components that will be placed in the FPGA. In figure 35 the top level is presented.

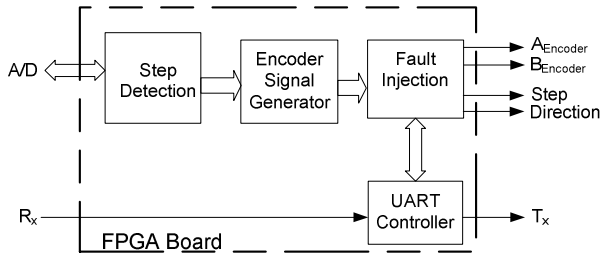


Figure 35. General Block Diagram with the FPGA Blocks

The inputs of the top level block come from the *Signal Conditioning and Data Acquisition* block. These input signals are the I/O lines to control the A/D converters; the signals to the HIL simulator and for the communication through the serial port.

Figure 35 shows the FPGA block top level design and figure 36 presents one level below of the top one. Each block has a specific task for helping in the creation of the emulator: one for detecting the steps requested by the step motor driver, one for communication with the HIL simulator through the UART port and one for doing the fault injection.

The blocks in figure 36 are explained in more detail in the following sections, but a general description is:

- Two *A/D controllers (A/D_1 and A/D_2*) for controlling the two external analog to digital converters. These blocks are running at $12.5MHz$. This clock signal is received from the *Clk12.5Mhz* block.
- A *Passivator* block [18] (*Passivator_1*) used to synchronize the A/D control blocks that provide digital values at a slower frequency than the *Identify Steps* block (all the blocks but the A/D control ones are running at $100MHz$, the FPGA development board frequency). The passivator uses the 4-phase handshake protocol explained below.
- The *Identify Steps* block where the steps are identified based on the digital values coming from the A/D converters. It will send a pulse every time a step is detected and a signal with the movement direction of the step motor.
- The *Encoder* block where the encoder signals are generated based in the pulse coming from the block making the identification of the steps.
  This block sends the number of steps counted to the fault injection block to transmit them through the UART port and the generated encoder signals to the fault execution block.

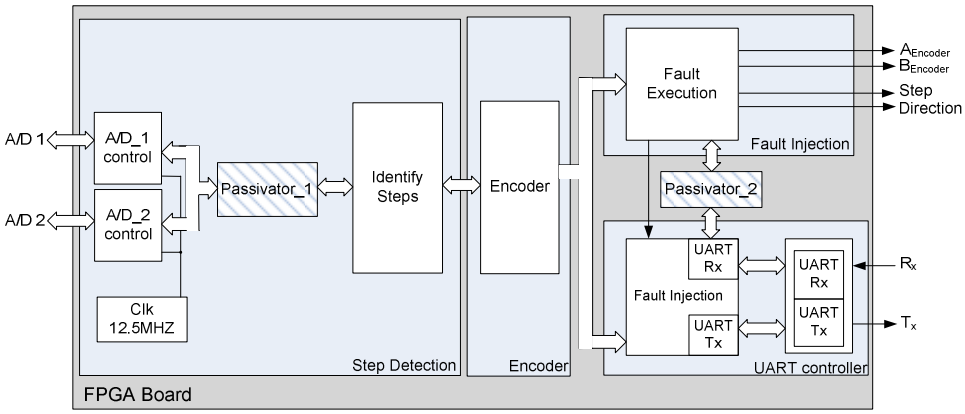


Figure 36. FPGA Block Design

- The *Fault Injection* block is in charge of controlling the *UART* blocks developed at Océ for using the UART communication protocol, it is in charge of the input and output communication with the HILs through the serial port.
- The *Fault Execution* block is in charge of sending the output signals out of the FPGA development board. This block receives the correct encoder signals and also the fault injection commands from the fault injection block; in here is where the fault injection will be executed.
- Another *Passivator* [18] (*Passivator_2)* is used for the synchronization of the UART received commands and the fault execution block.
- Finally reused code for the UART communication is used for read/transmit serial data.

The passivators shown in figure 36 are needed to solve the problem when two hardware components connected together with a single wire want to communicate between them, the

problem of knowing when one of them is ready to transmit and when the other one is ready to receive is solved with the passivators.
To understand better the problem, lets consider an active component (producer) and a passive one (consumer), with the help of hand-shake lines to control the transmission (figure 38) the problem can be solved [53].

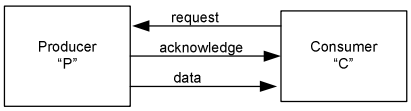


Figure 37. 4-phase hand-shake protocol

Four phases can be distinguished in this hand-shake protocol,

1. When C is ready to receive a value, it will send a request(request='1')
2. If C has sent the request and P received it, it will send an acknowledge signal back to C (acknowledge = '1')
3. When C receives the acknowledge, it will read the data from P and it will make the request line '0' to indicate that the communication was done.
4. When P detects the change of the request line from high to low it will change the signal acknowledge to '0' finishing the communication between the blocks.

Using this protocol, there is no need of a global clock to synchronize the elements, allowing synchronization of modules working at different speeds.
In the presented configuration, one of the sides is active and the other one is passive. By adding another element between two components (a passivator), both components may become active. The new block diagram is presented in figure 39 making both the producer and consumer active and the passivator controlling the hand-shake lines for both sides.

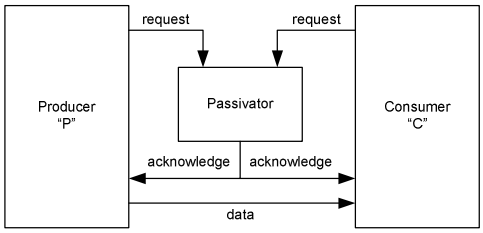


Figure 38. 4-phase hand-shake protocol

The 4-phases hand-shake protocol is used in the passivator modules explained before to help in the synchronization between the elements.

3.4.3.2 Analog/Digital circuit controller

3.4.3.2.1 Block Diagram

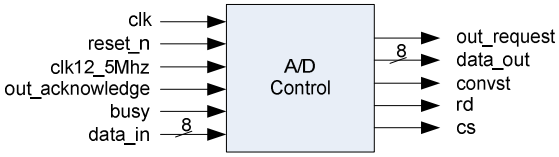


Figure 39.A/D Control Block Diagram

3.4.3.2.2 Description

This block controls the A/D circuit, after receiving data from the A/D converter; it will transfer the data to the next block with the help of the passivator. After the read data is sent to the passivator, it will request a new digital value from the A/D converter.

3.4.3.2.3 State Machine

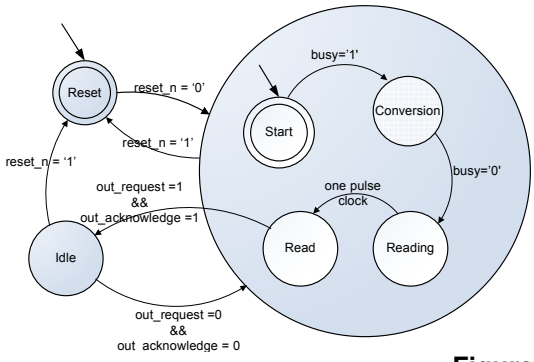


Figure 40. A/D State Machine

3.4.3.3 Clock 12.5 MHz

3.4.3.3.1 Block Diagram

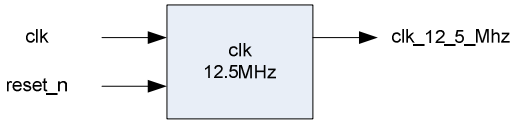


Figure 41.Clock 12.5 MHz Block Diagram

3.4.3.3.2 Description

This block generates a 12.5*MHz* clock signal using the clock in the FPGA development board of 100*MHz* . This clock signal is only used in the A/D converter blocks since using this slower frequency helps to facilitate the control of the A/D circuits.

3.4.3.3.3 Flow Diagram

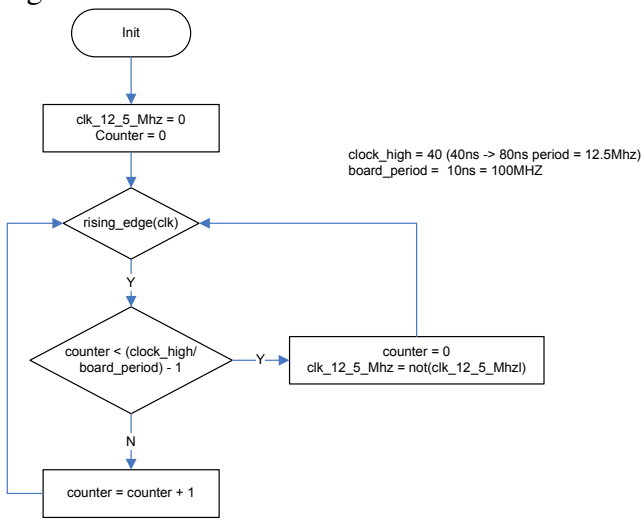


Figure 42. Clock 12.5MHz Flow Chart

3.4.3.4 Passivator 1

3.4.3.4.1 Block Diagram

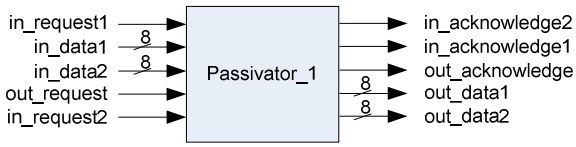


Figure 43.Passivator 1 Block Diagram

3.4.3.4.2 Description

This block helps to synchronize the A/D control blocks and the Identify Steps block since they are running at a different clock frequency.

3.4.3.4.3 Logic Block Diagram

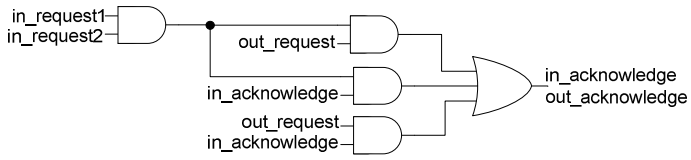


Figure 44. Passivator Block Diagram

3.4.3.5 Step Detection

3.4.3.5.1 Block Diagram

Figure 45. Identify Steps Block Diagram

3.4.3.5.2 Description

This block reads the digital values of the voltages and based on figure 10 compare these voltages to distinguish the steps. Every time a step is detected a pulse will be generated in the step_out pin. The step_direction pin will indicate with a '1' that the motor is moving forward, while with a '0' it will indicate that the step motor is requested to move backwards.

3.4.3.5.3 State Machine/Flow Diagrams

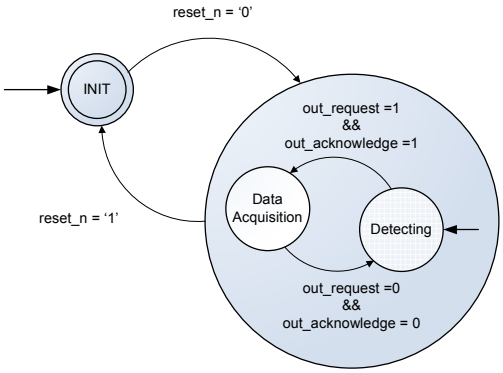


Figure 46. State Space diagram

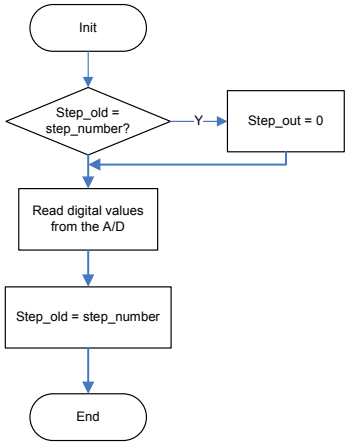


Figure 47. Flow chart diagram for the "Data Acquisition" state.

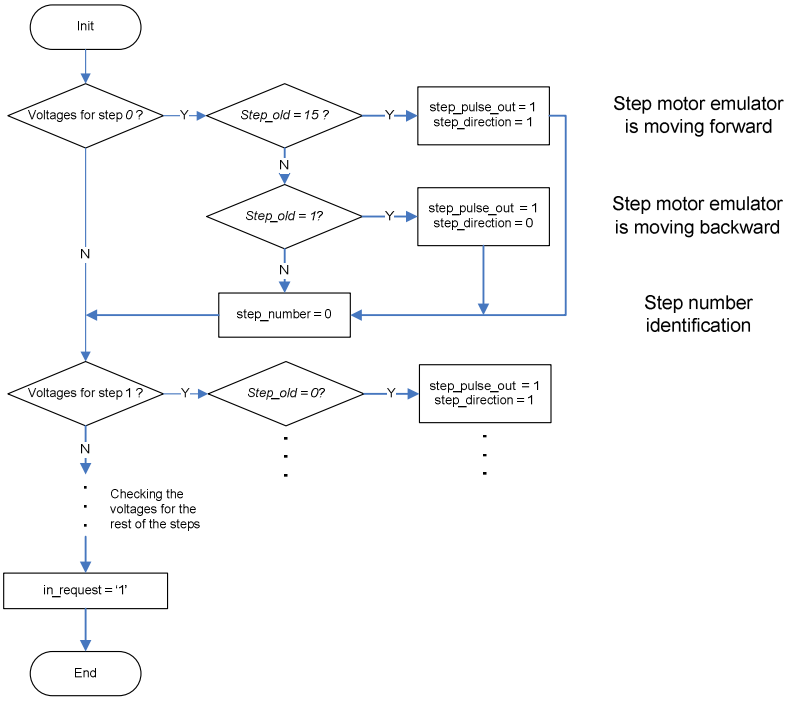


Figure 48. Flow chart of the detecting state from figure 46.

3.4.3.6 Encoder Signal Generator

3.4.3.6.1 Block Diagram

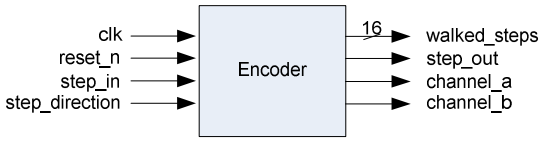


Figure 49. Encoder Block Diagram

3.4.3.6.2 Description

This block will generate the encoder signals based in the step_in pulse and the step_direction input generated in the step detection block. The encoder signal has a resolution lower than the encoder used now in the HIL simulator, but the encoder resolution is one step, causing no difference in the accuracy.

The resolution in the encoder used in the HIL simulator is of 300 counts per revolution (CPR) with a resolution of $1.2°$ or $0.3°$ if counting all the rising edges of the quadrate signals.

$$\frac{360°}{300} = 1.2°, \qquad \frac{1.2°}{4} = 0.3°$$

While the resolution of the encoder signal generated in the emulator can be either of $1.8°$ or of $0.45°$ if counting all the rising edges of the quadrature signals.

$\frac{360°}{800} = 0.45°$ but the encoder signal is composed of 4 steps giving us $(4)0.45° = 1.8°$

One period of the encoder signal is generated as shown in figure 50 by four steps; every two steps there is a change in the encoder signal.

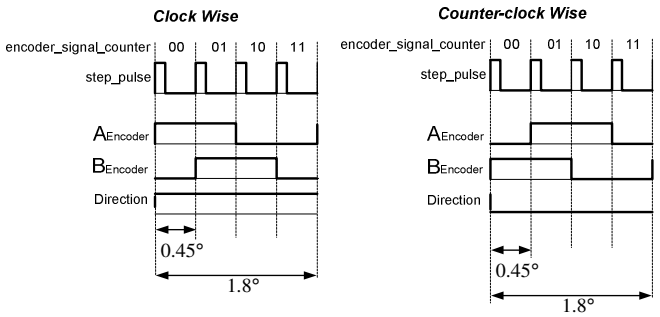


Figure 50. Encoder signal generator description.

This block is also counting the number of steps the emulator has moved, and how many steps the step driver has requested. The counter is implemented with a 32 bit variable representing the maximum value for the counter.

3.4.3.6.3 Flow Diagram

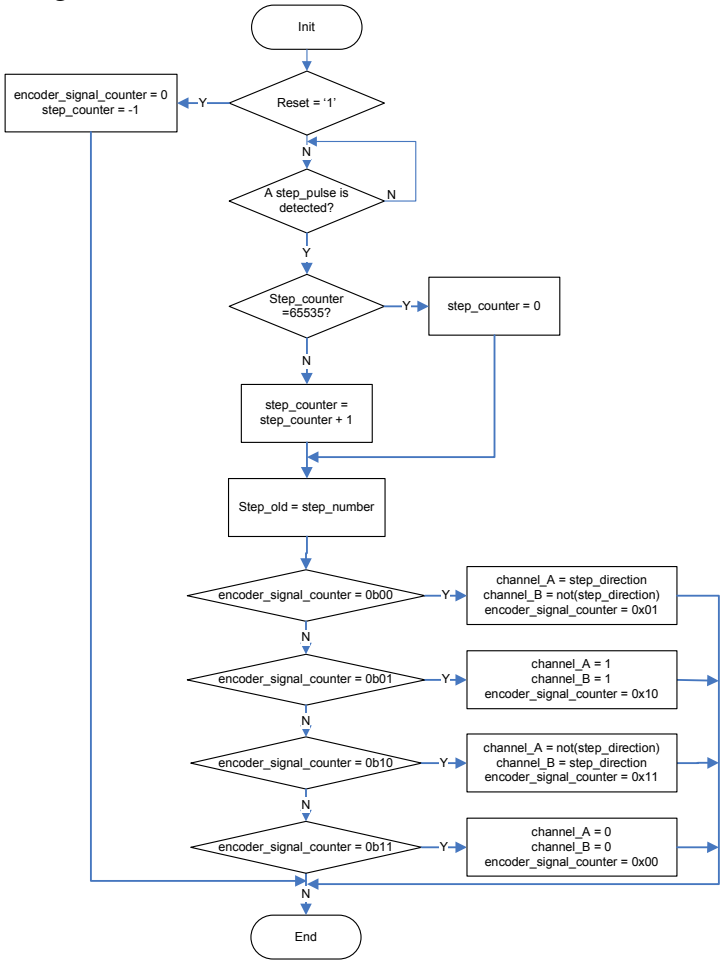


Figure 51. Encoder signal generator

3.4.3.7 Fault Injection

3.4.3.7.1 Block Diagram

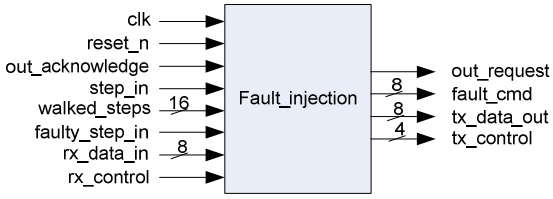


Figure 52. Fault Injection Block Diagram

3.4.3.7.2 Description

This block is controlling the UART re-used code from Océ for using the serial communication with the HIL simulator or with the PC. It will receive fault injection commands and it will send through the serial port the total number of steps the emulator has moved with and without faults when the "East" or the "West" switches in the FPGA development board are pressed.

3.4.3.7.3 Flow/State Machine Diagrams

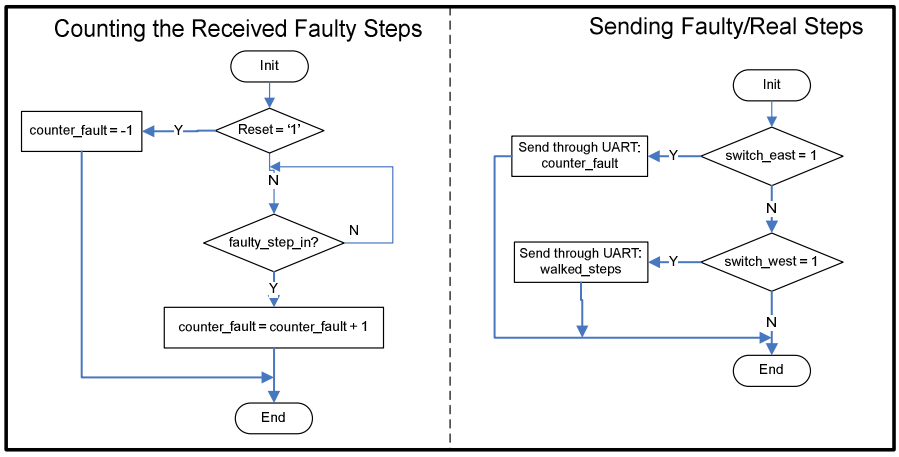


Figure 53. Parallel fault injection transmit block

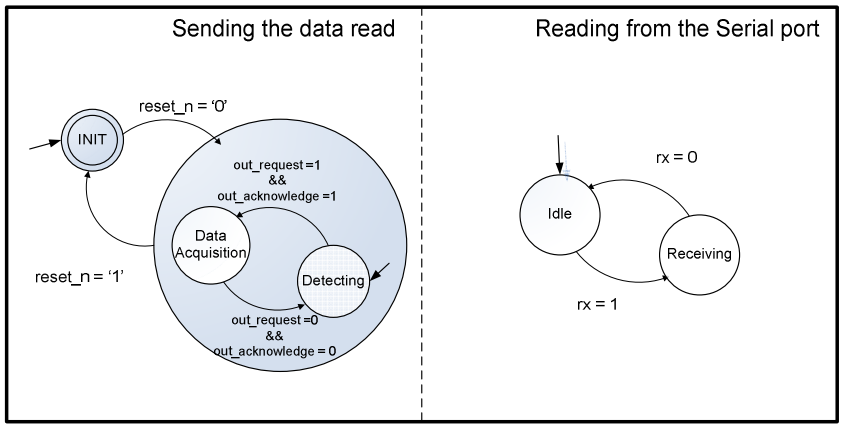


Figure 54. Parallel fault injection receives block

3.4.3.8 UART Controller

3.4.3.3.2 Description

Reusable code from Océ is used in this block; it is the implementation of the UART communication protocol.

3.4.3.9 Passivator 2

3.4.3.9.1 Block Diagram

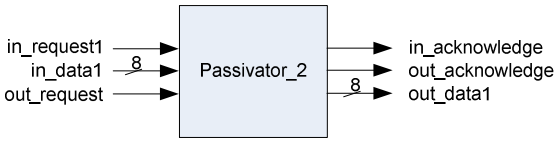


Figure 55. Passivator 2 Block Diagram

3.4.3.9.2 Description

It is used for synchronizing the fault injection block with the fault execution block, using the 4-phase protocol explained before.

3.4.3.9.3 Logic Block Diagram

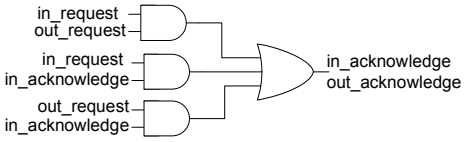


Figure 56. Passivator 2

3.4.3.10 Fault Execution

3.4.3.10.1 Block Diagram

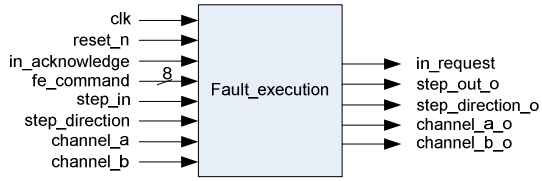


Figure 57. Fault Execution Block Diagram

3.4.3.10.2 Description

This block will receive the encoder signals and the step pulse and direction signals from the encoder block. If a fault injection command is requested by the user, this block will execute the fault injection and faulty encoders and step pulse signals will be sent out.

3.4.3.10.3 State Machine/ Flow Diagrams

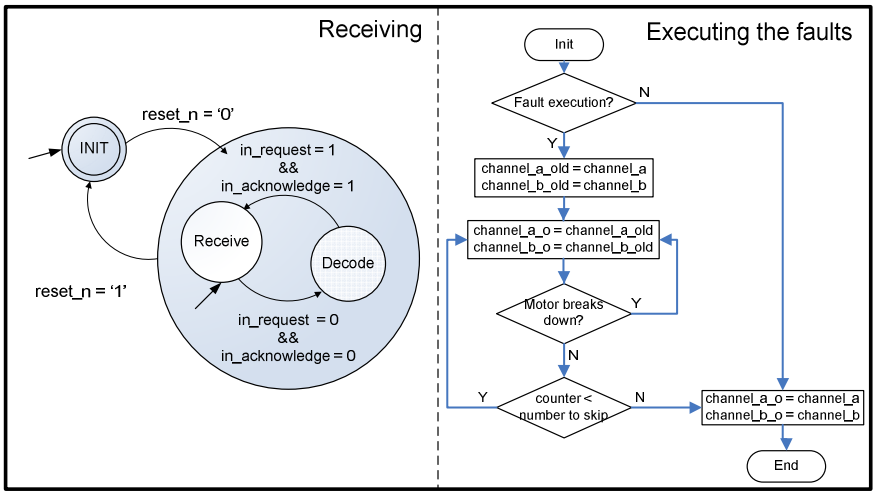


Figure 58. Fault Execution

Chapter 4

Results

This chapter presents the results for each one of the approaches explained in chapter 3. The results of the third approach include also a section with the device utilization parameters of the FPGA, helping us to see how many step motor emulators can be placed be in the same FPGA; finally this chapter finishes with a comparison between the three different approaches.

4.1 Constant Frequency Step Motor Emulator

- V_A and V_B were expected to have the sinusoidal behavior that the current has when flowing through the step motors (figure 10), but after testing the output voltages of the instrumentation amplifier, the output signal was not good enough to identify all the steps (figure 59). Furthermore, the frequency of the steps is only visible at 2 points per period of the signal, as can be seen in figure 59 where the small vertical lines represent the frequency requested by the driver; in this case 1 ms, i.e. 1 KHz.

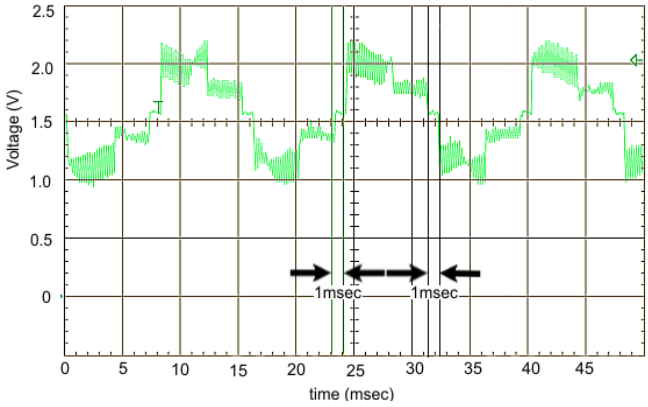


Figure 59. Output signal, for $R_X = 1.5\Omega$

The RC filter helps to reproduce the time constant of the motor but it does not help to reproduce the behavior of the current flowing through the inductor in the step motor, making

it impossible to reproduce the sinusoidal behavior in the current of the step motor with only the RC filter.

To see all the steps it was considered to have four different resistor values to generate all the current levels ($\pm 38.27\%, \pm 70.71\%, \pm 92.39\%, \pm 100\%$)from the step motor driver. The problem with this idea is that by changing the resistors with the help of the FPGA all current levels will be generated, but there is no possibility to detect a change in the frequency of the steps; the same result occur when having only one resistor value demanding only two current levels, as in figure 60 where the nominal value of the frequency is 2 KHz, and 0.5ms can be read from the graph.

Figure 60 presents the voltage levels representing the current levels after the instrumentation amplifier. The dotted line in the figure represents the zero of the graph that was moved to 1.6V due the offset included in the instrumentation amplifier.

As a result this approach works only for constant frequencies since interpolating all the steps can be done by using the only two steps that can be detected.

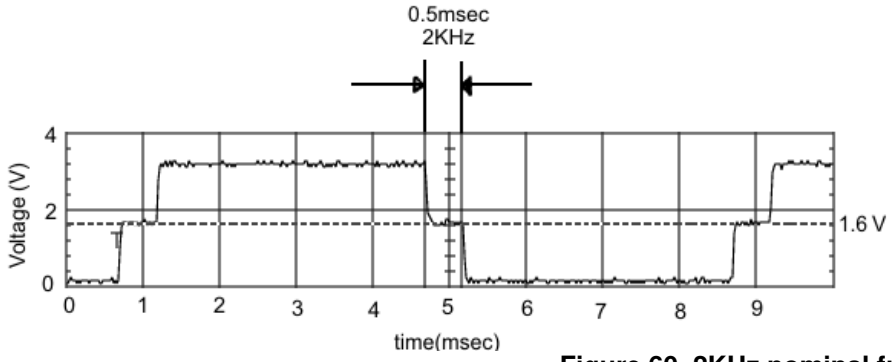


Figure 60. 2KHz nominal frequency

4.2 Load Inductive Simulator for Step Motors

A script in Matlab was done to verify the results of the differential equation solver in VHDL. The Euler method is applied in Matlab to the converted numbers and the results compared with the results of the simulation using VHDL code.

The test was done by changing the voltages during a period of 0.004705sec, a really short time but having a sampling time of $5\mu s$ (the step size $h = 5\mu s$) gives a total of 941 different values. Figure 61 shows the result of the Matlab simulation with the operations calculated using the 32 bits fixed point numbers. The y axis is the decimal value corresponding to this binary representation of the current I_a. This value needs to be converted to the real current value following 2 steps. The first one is to convert this number to a hexadecimal form and then from the hexadecimal form to a decimal representation using the "hx2dcfp" function.
For instance the number 100000000 in the graph using the "dec2hex" predefined Matlab function to transform it into hexadecimal,

$$dec2hex(100000000) = 0x5F5E100,$$

And then using the hx2dcfp function for finding the real current value,

$$hx2dcfp('5F5E100') = 2.98A$$

The x axis represents the number of samples used in the simulation, 941 samples in this case.

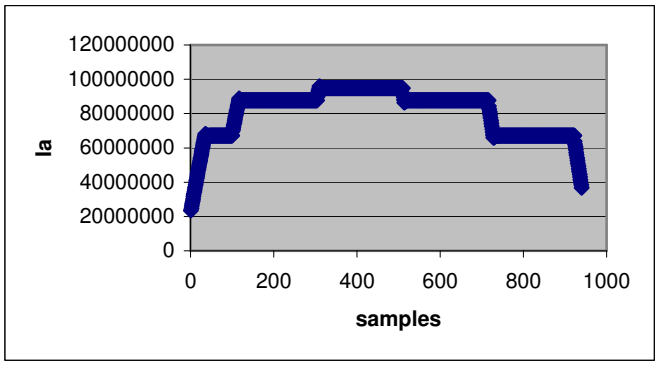


Figure 61. I_a calculated in Matlab using fixed point numbers.

When comparing the VHDL results with the Matlab fixed point results, a difference of 0.66% is present in the initial values but at the end of the calculations after the 941 values the difference between them is 4.5%. The conclusion is that the error is increasing in the calculations done in VHDL can be found out as a conclusion.

The Matlab solution using fixed point numbers was then compared with a step motor model simulation available in Océ using Simulink shown in figure 62. The difference between the step motor model in Simulink and the Matlab fixed point number results is of 0.5%, and this error percentage remains during all the test.

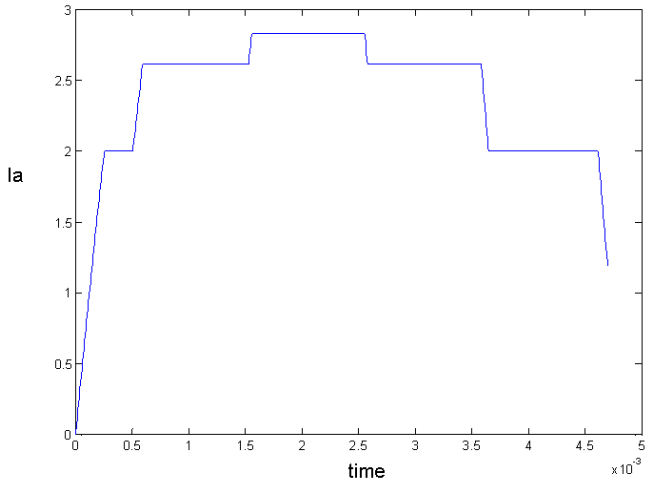


Figure 62. I_a calculated with Matlab

The calculation of the input vector containing the PWM voltages is a difficult task for feeding the VHDL simulation since the PWM voltages in the step motor change based on the currents flowing through the step motor and the desired currents (based on the sine wave in figure 10).

To calculate the PWM voltages to test the VHDL design it is needed to simulate also the driver of the step motor to generate these PWM voltages. Instead of also simulating the step motor driver, input voltages from the step motor simulation available in Océ were used instead of generating our own input voltages. A total of 941 values were used for the VHDL simulation, the results were then compared with the simulation results for the step motor emulation in Matlab.

That is why only 941 values were calculated using the voltages generated for the simulation in Matlab.

Further work is needed here to correct the error of the VHDL calculation of i_A against the Matlab simulation; for i_B and consequently for w_r and θ there is even a bigger difference between the values. This work was stopped after having these results because the approach presented in section 3.4 started to work.

The difference in the values between the VHDL solution and the Matlab simulation can be related in the use of signed numbers. A solution for the difference in the values could be to increase the 32 bits variable to bigger number to verify if the problem is in the bits assigned to the integer part; other solution is to use the fixed point libraries already available by the IEEE. These IEEE libraries works not only for 32 bits numbers with 25 bits as the presented algorithm does, it is possible to use numbers with different sizes, signed or unsigned and the main difference between the libraries and the algorithms presented in this work is that they have been tested already a lot.

4.3 Real-Time Step Motor Emulator

4.3.1 Motor Behavioral Simulation Block Results

After the motor behavioral simulation block the current has a sinusoidal behavior thanks to the effect the inductors cause in the currents. High or low current modes are available in the EC for controlling the step motor; using higher currents means increasing the torque of the step motors.

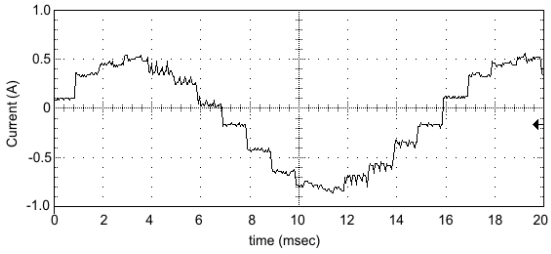


Figure 63. Low current through the circuit for 1 KHz

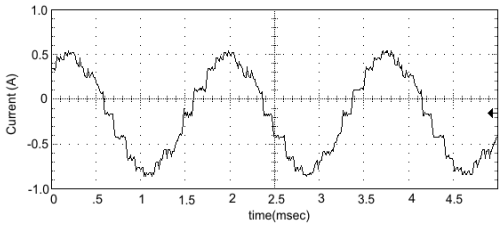


Figure 64. Low current through the circuit for 9 KHz

Figures 62 and 63 show the current flowing through the circuit when frequencies of 1 and 9 KHz are requested by the step motor driver. The distance between steps is consequently 1 milliseconds for 1 KHz and 0.111 milliseconds for 9 KHz; in both graphs the current is varying between $-0.8A$ and $0.5A$ because the EC is in low current mode.

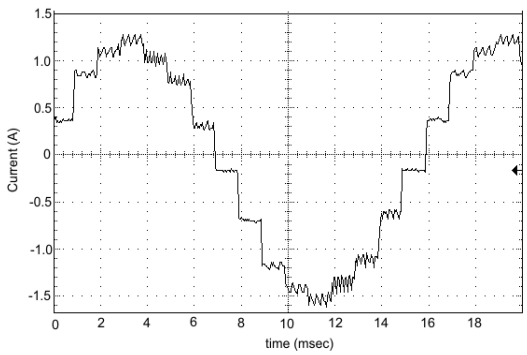


Figure 65. High current through the circuit for 1KHz

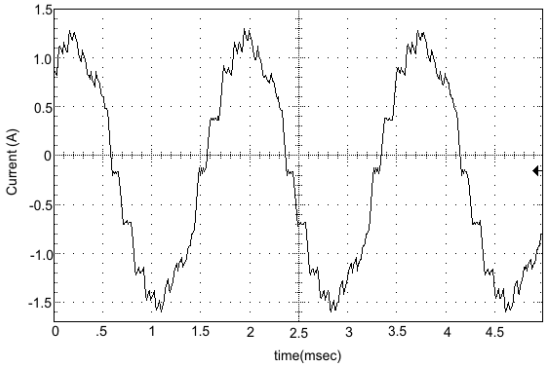


Figure 66. High current through the circuit for 9 KHz

Having a high current flowing through the behavioral simulation circuit means a higher torque in the motor. Figures 64 and 65 show the high current flowing through the circuit when frequencies of 1 and 9 KHz are requested by the step motor driver.

4.3.2 Signal Conditioning Block Results

The current flowing through the behavioral simulation circuit is formed by the interaction o the voltages V_A+ and V_A- as explained before. These two voltages are reduced to lower values with the voltage divider and then combined in the instrumentation amplifier into one voltage signal; the offset added in the instrumentation amplifier to this signal makes the voltage signals to have only a positive variation.

The design of the step motor emulator was implemented by considering that it should always work in low current mode. This decision was made because the commands to accelerate the step motor work always in low current mode. Even if the EC is working in high current mode when commands to accelerate the step motor are requested from the EC, the EC will change to low current mode before accelerating (this is not a correct behavior of the EC board, but the EC used for the development of this project has old software and old hardware causing probably this erroneous behavior).

The output voltage in the instrumentation amplifier using a unitary gain in the amplifier will not vary through all the voltage range (0 to 3.3V) when the EC is in low current mode. To make the sinusoidal voltage output of the instrumentation amplifier vary through the entire voltage range, the potentiometer (R6, in figure 34) is used to increase the gain of the amplifier and have a better voltage range for the analog to digital conversion.

The gain used when working in low current mode is such that the low part of the voltage signals is chopped because the amplifier makes the signal go below 0 and saturation occurs in the amplifier (0V in this case), making it impossible to distinguish the steps in that part of the signal, but making the voltage levels before reaching saturation easily distinguishable (figure 67 and 68).

Using the voltage signals in both phases to do the step identification makes the identification of the steps an easier tasks, for instance, if the voltage in phase A is in saturation (0 volts) the voltage of phase B will be used to make the distinction between steps and when phase B is in saturation then the voltage level in phase A will make the step distinction.

Figure 66 and 67 shows the output of the instrumentation amplifier using the necessary gain to chop the lower voltage levels of the voltage signals. The steps are easily distinguishable when working with frequencies of 1 KHz as can be seen in figure 67 but the step identification gets complicated when identifying steps at frequencies higher than 7 KHz. This happens because of the steep sinusoidal voltage signals when working under higher frequencies. The step motor emulator works for frequencies below 7 KHz but starts to miss steps if higher frequencies are requested.

A solution for missing the steps in higher frequencies could be to increase the voltage range in the analog to digital conversion, using a voltage range of 0V to 5V instead of the one using now of 0 to 3.3 V.

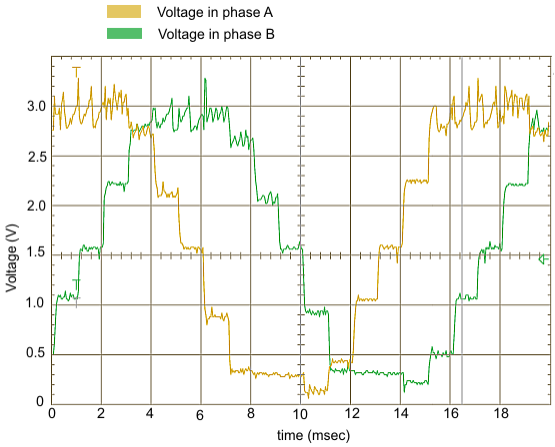


Figure 67. Voltages in the instrumentation amplifier for 1 KHz

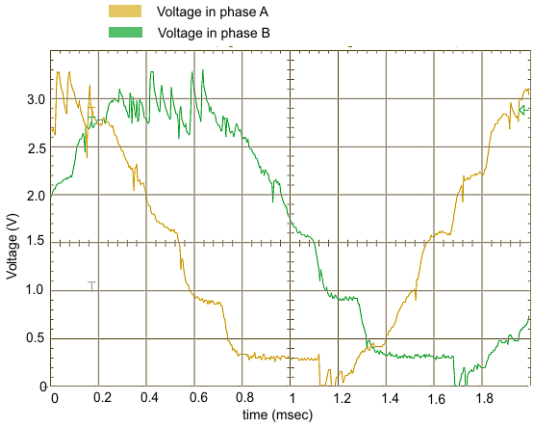


Figure 68. Voltages after the instrumentation amplifier for 7 KHz

If tests are needed in high current mode, the first thing to do is to make sure that the current asked by the EC when accelerating is indeed high current.

Having a high current flowing through the behavioral simulation block will be reflected as a higher voltage signal in the output of the instrumentation amplifier, and consequently a smaller gain (the value of the resistance in R6 figure 34) will be needed to make the variations of this output voltage being between 0V and 3.3V (to use all the voltage range in the A/D converter and have a better step identification).

After the signal conditioning function generates the sinusoidal output voltages to vary between all the voltage range (0V to 3.3V) this voltage is sent to the analog to digital converter for converting the voltage levels into digital values that are read by the FPGA.

4.3.3 Step Detection Block Results

In figure 10 the current levels for each step were presented as a percentage of the maximum current ($I_{\max}$) flowing through the step motor. After the instrumentation amplifier the voltage levels can also be identified as a percentage of the maximum voltage. These voltages levels are read by the FPGA from the A/D converter to perform the step identification.

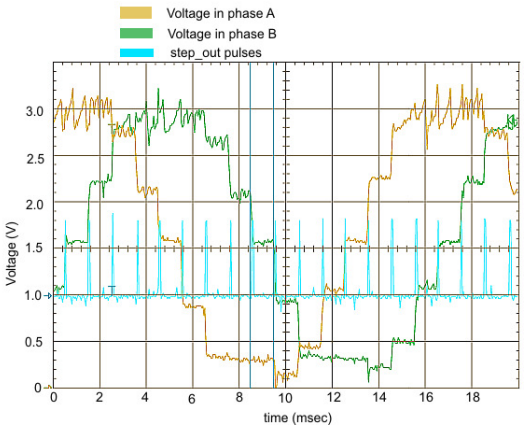


Figure 69. 1KHz Step Pulse and Voltages

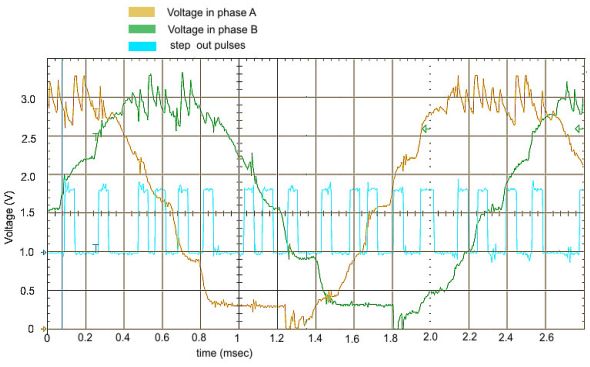


Figure 70. 7Khz Step Pulse and Voltages

For instance in figure 69 when the voltage in phase A is above 2.5 V the voltage in phase B will be used for identifying a change in step since it is changing from 2.6 V to 2.1 V to 1.6V and so on facilitating the step identification.

Figure 69 also shows the step out spiky signal. Every time a new step is requested by the driver the step_out will show a pulse in the signal. The step_out signal is the output of the *step detection block.* This block is generating a spike every time the voltage levels change to a new step and this spiky signal is used in the *encoder signal generator* block for the creation of the encoder signals.

Figure 70 shows the same spiky signal but when the step motor driver is requesting 7 KHz, the detection of the steps becomes harder due to the steep voltage signals.

4.3.4 Encoder Signal Generator Block Results

The outputs of the encoder signal generator block are two lines simulating the encoder signals as presented in figure 71. These signals are generated based on the spiky step signal generated in the step detection block, where as explained before (section 3.4.3.6) every two steps represents a change in the encoder signals as can be seen in figure 71 or in figure 72.

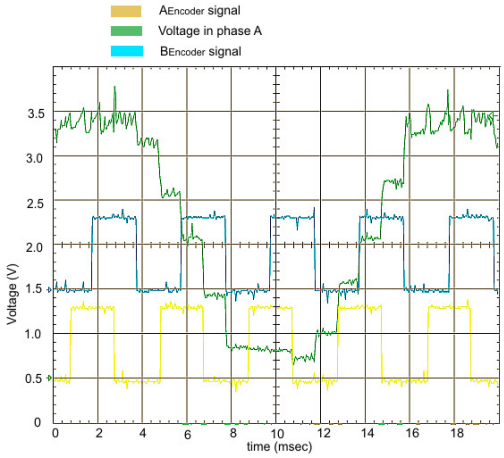


Figure 71. 1KHz Encoder Signal and Voltage

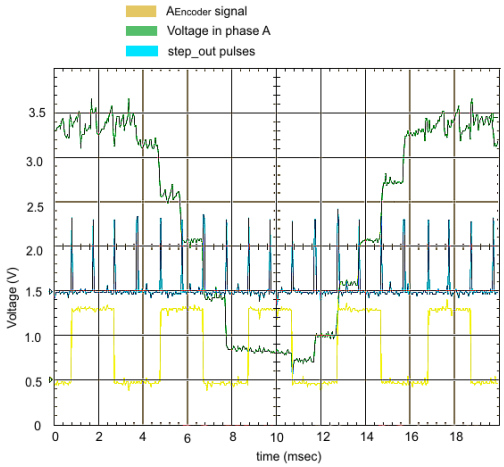


Figure 72. 1Khz Encoder, Step Pulse and Voltage

4.3.5 Fault Injection Block Results

The encoder signals pass through the fault execution block (figure 36) where they will be modified if a fault injection command has been received from the HIL simulator or a PC through the serial port.

Figures 73 and 74 show the result of the fault injection in the generated encoder signals; in figure 73 both encoder signals were high when the fault injection command to skip steps was read, between 4 and 5 milliseconds the motor goes to normal function and the encoder signals starts to run again until the time 12 milliseconds when the command to simulate the step motor breaking down is received, leaving the encoder signals in the last position they were when the fault injection command was received. In this case, one of the encoder signals is high and the other one is low.

Figure 74 presents another example of the behavior of the encoder signals when doing fault injection in the step motor emulator.

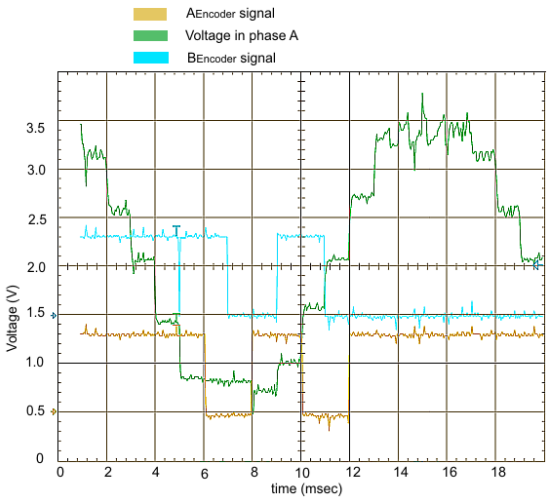


Figure 73. Fault Injection

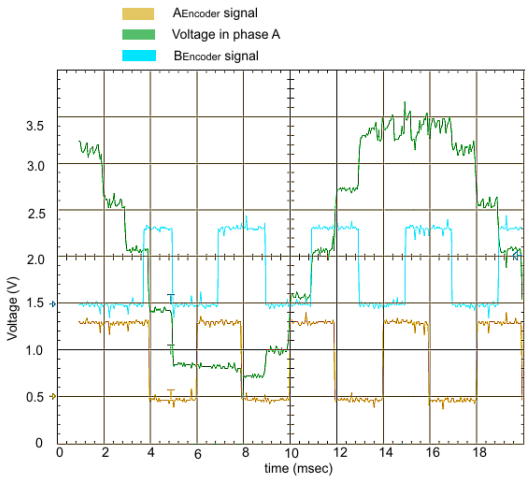


Figure 74. Fault Injection

4.3.5 Device Utilization

This section will help us to determine how many resources the step motor emulator is using in the FPGA to know how many step motor emulators can be placed in the same FPGA.
Table 7 shows the summary of resources used by the step motor emulator in the FPGA and figure 75 shows a graph based in table.

Table 7. Device Utilization for the step motor emulator

Step Motor Emulator.				
Number of Slices:	427	out of	5472	7.8%
Number of Slice Flip Flops:	419	out of	10944	3.82%
Number of 4 input LUTs:	813	out of	10944	7.42%
Number of bonded IOBs:	46	out of	320	14.37%
Number of GCLKs:	2	out of	32	6.25%

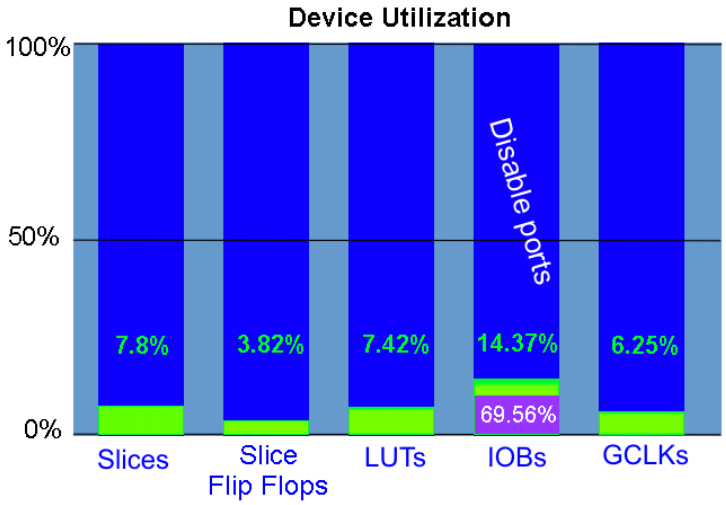


Figure 75. Device Utilization

Considering that the number of resources has a linear increase if more step motor emulators are emulated in the same FPGA, we can conclude from table 7 and figure 75 that there is enough number of slices and LUTs for creating up to 10 step motors in the same FPGA.

The limitation is the number of ports needed for the emulation of the step motor (IOBs); the FPGA development board only allows the use of 32 single ended ports plus another 14 ports

(assigned to switches and LEDs but that can be used for the step motor emulator), the rest of the pins are assigned to other components like VGA outputs, mouse and keyboard connections, etc. The number of available ports represents 14.37% of the entire number of ports as can be seen in figure 75. From those 14.37% available ports only 28 of them are used for the step motor emulator (representing 69.56% of the available ports) making us conclude that with this approach only one step motor can be emulated with the current design.

A solution for this problem is to use serial A/D converters that will reduce enormously the usage of I/O pins, since the data is read through only one communication line instead of using the eight lines in the parallel case used in this emulator. Also if not using a development board but wiring the FPGA by ourselves will give us the flexibility of use the ports available in the FPGA as we want for emulating a higher number of step motor emulators.

4.4 Comparison between the three approaches

In this section a comparison between the three presented approaches will be presented, during this section the first approach, the constant frequency step motor emulator will be denoted as CFSME_1, the second approach which is the load inductive simulation for step motors will be denoted as LISSM_2 and the third approach the real-time step motor emulator will be denoted as RTSME_3.

Varying frequencies. CFSME_1 is not able to emulate step motors under varying frequencies, while LISSM_2 and RTSME_3 are. LISSM_2 has an advantage over RTSME_3, since it is able to emulate motors running at frequencies that RTSME_3 can not emulate. (RTSME_3 can emulate until 7 KHz)

Simplicity. CFSME_1 and RTSME_3 are easier to implement than LISSM_2 because they do not need to solve the state space equation of the step motor model in the FPGA, do not need to generate the currents in an external power supply to feed them back to the step motor driver, less number of ports are necessary and no switching electronic controls need to be implemented.

Generalization. LISSM_2 will be the approach with the more generality since it can emulate all kind of inductive loads running at all kind of frequencies, RTSME_3 can emulate different step motors by only changing the gain in the instrumentation amplifier and probably the number of inductors in the motor behavioral simulation block. CFSME_1 is able to simulate different step motors but with the big constraint of not allowing changes in the frequencies.

Number of I/O ports needed. RTSME_3 and CFSME_1 need less number of ports to emulate one step motor than LISSM_2, since there are less electronic components to control.

Working for low/high current modes. The EC is able to work under high or low currents mode (appendix A). RTSME_3 and CFSME_1 will need to change only the gain in the instrumentation amplifier to match the voltage sine waves in figure 67. LISSM_2 needs that the external power supply in charge of generating the current can generate these higher currents.

Fault Injection. The three approaches are able to generate fault injection into the step motor emulator.

Accuracy. RTSME_3 is sometimes one step of difference when comparing with the counting from the EC, the other two approaches were not finished but theoretically both of them should have a good accuracy.

Table 8. Comparison between approaches

	Constant Frequency Step Motor Emulator	Load Inductive Simulator for Step Motors	Real-Time Step Motor Emulator
Varying frequencies	0	+	+
Simplicity	+	0	+
Generalization	0	+	0
Number of I/O ports	+	0	+
Low/High current nodes	0	+	0
Fault Injection	+	+	+
Accuracy	0	+	+

Table 8 presents a graphical interpretation of the comparisons explained above between the three different approaches, where "+" represents the highest option, "0" is a medium high and "-" represents the low option.

Chapter 5

Conclusions

In the literature as far as we know, there is no research about step motor emulation. A step motor emulator will be of great help for the simulators at Océ, and at other companies using step motors, because of their capability to do fault injection.

The step motor emulator developed in this graduation project is a new asset for the HIL simulator in Océ. It represents a new option for automatic testing (there is no need of physical interaction); it is able to do fault injection by receiving commands through the serial port, increasing the test coverage by simulating the motor breaking down and the motor skipping steps faults.

The step motor emulator is also an easier solution when comparing it with the Load Inductive simulation approach for step motors, since there is no need to solve the differential equations representing the state space model of the step motor, there is also no need to generate externally the currents that the driver needs and less electronics components are needed; but there is the problem of detecting the steps at frequencies higher than 7 KHz.

Theoretically more than one step motor can be emulated in the same FPGA if there are enough ports. Furthermore, emulation is not noisy at all when compared to a normal step motor and it does not produce the same amount of heat than a normal step motor when testing for a continuous and long period of time.

Future Work

Testing error handling code is included in future work, with the help of the fault injection capability of the step motor emulator this task will be easy to perform.

There is an issue with the emulator of counting one more step when comparing the counter in the emulator with the counter from the EC (Section 5.1). The board used for the development of this project counts with an old hardware and with old software revisions; the error was investigated in the emulator and there is no reason to have a different count; so the emulator needs to be tested using a newer board and software to verify whether the error is still present, to see whether it is coming from the emulator or from the EC.

The resolution of the encoder signal in the emulator is less than the one from the encoder used now in the HIL simulator, but it is a maximum resolution of one step, being accurate enough.

The tables with the walked distance based in the encoder values need to be change in the HIL simulator code to have the resolution of the encoder from the emulator.

There are problems when the emulator is used to emulate the step motor at frequencies higher than 7 KHz, The problem can be solved by increasing the voltage range in the A/D converter from 3.3V to 5V, having a bigger range, making it easier to distinguish between the voltage levels of the steps.

And finally to emulate a different step motor there are some changes needed in the step motor emulator it is presented in section 5.2.

5.1 One more step error analysis

This section will explain more in detail the difference of one step between the number of steps counted in the emulator and the numbers of steps counted in the EC. This one step difference does not occur always, moreover, if the counter is 1000 steps or if the counter is 20000 steps the difference of one step remains to be only one step, when it occurs.

The EC board is not capable of asking the motor to move only one step or only a couple of steps. The tests consist in asking the motor to move from a velocity *k* to a final velocity *l* using an acceleration *m.* (where *k*, *l* represent velocities in Hz and *m* represents acceleration in Hz/sec).

Because there is no possibility to ask the motor to move only a couple of steps it is difficult to debug the emulator. A test consisted in reading the initial conditions of the motor (the voltages) and the final conditions of the motor (the voltages) allows us to see how many steps the driver has requested.

The voltages at the output of the instrumentation amplifier are read when the motor is not moving (0 Hz) before doing a test, these voltages represents the current levels presented in figure 10 and allow us to see which is the step number corresponding to those voltages. After the initial step number is identified the command to move from 0 Hz to 2 KHz with an acceleration of 5 KHz is sent to the EC and also a command to move from 2 KHz to 0 Hz with the same acceleration of 5 KHz; after these commands the motor is moving from 0 Hz to 2 KHz and it is going back to 0 Hz through a number of steps. The voltages at the output of the instrumentation amplifier are read again and the step number that corresponds to those voltages is identified.

A program in Matlab with input parameters the initial step number, the counter from the EC and the counter from the emulator was programmed to give as results the final step numbers after adding to the initial step number the steps counted.

Once the final step number is given by the Matlab program, the voltages corresponding to this final step number are identified and compared with the voltages in the output of the instrumentation amplifier (final conditions of the motor after the test).

The initial conditions for each test are the final conditions of the previous test and the results shown that:

1. When the counters are the same in the EC and in the emulator: The final voltages (the final conditions) read in the oscilloscope match with the final step calculated in Matlab.
2. When the counters are different. The final read voltages correspond to the counter in the emulator and the counter from the EC is one step less than the voltages read.

Drawing as a conclusion that the counter in the emulator is counting correctly the steps by reading the voltages and the EC is counting sometimes one step less based in the same voltages.

5.2 Emulating different step motors

In this project only one step motor was emulated, to emulate a different step motor there are some changes needed in the step motor emulator as explained below.

The inductance value of the step motor to be emulated needs to be reproduced physically in the motor behavioral simulation block, but as explained in section 3.4.2.1 there exist the probability that there is no need to add more inductance to the emulator if the steps can be recognized with the 2mH inductance value used during this project; if needed it is possible to do so even for the higher inductance values in the windings considering the specifications of the step motors in Océ by adding more 1mH inductors in the circuit.

If voltages higher than 24V are supplied by the EC, the voltage divider will need to be changed and new resistors values will have to be calculated to reduce the new voltage supply values to a lower voltage levels to be able to use them in the instrumentation amplifier (section 3.4.2.2).

The instrumentation amplifier is one of the most important components in the development of this project, the sinusoidal voltage signals generated in its output need to reproduce the voltage levels from figure 67, avoiding in this way any change in the VHDL code if a quarter step increment step motor is being emulated. By modifying the resistor R6 in figure 34 the gain of the instrumentation amplifier can be change and the voltage levels can be adjusted to handle low or high currents.

In the FPGA components, the Identify Steps block will have to be change if the step motor to be emulated does not have quarter step increments; the values for the comparison conditions will have to be changed if emulating full, half or sixteenth step increments to match the new voltage levels.

Finally because there are so many comparisons to detect the steps, the use of a normal microcontroller is also suggested as future work, the maximum requested frequency of 9.3 KHz can be handled with a fast microcontroller, but a trade off between the number of step motors to be emulated and the use of a normal microcontroller will have to be evaluated, because probably the lack of parallelism in a normal microcontroller could limit the number of emulated step motors in the same microcontroller.

References

[1] Akdogan, E.; Topuz, V.; Akbas, A. “An education tool study on mechatronics: emulation of stepper motor driving systems by using a microcontroller based system interface”; Proceedings of the IEEE International Conference on Mechatronics ICM 2004,Volume 3, Issue 5; June 2004; Pages 509- 512

[2] Allegro; “Part Number: A3979, Microstepping DMOS Driver with Translator” Datasheet.

[3] Analog Devices; “Part Number: AD7819 High speed, 8 bit analog to digital converter” Datasheet.

[4] Ascher, Uri; Petzold, Linda, “Computer methods for ordinary differential equations and differential algebraic equations”. Philadelphia : SIAM, 1998.

[5] Baldursson, Stefan: "BLDC Motor Modelling and Control - A Matlab/Simulink Implementation" Göteborg, May 2005; Chalmers University of Technology.

[6] Bishop, David; “Fixed and Floating point support package”. URL: http://www.eda.org/fhpd/vhdl.html

[7] Bracker, J.; Dolle, M; "Simulation of Inductive Loads" IEEE International Symposium on Industrial Electronics, 2007. ISIE 2007. Volume 4, Issue 7, June 2007; Pages:461 - 466

[8] C. Dufour, S. Abourida, J. Bélanger, V. Lapointe; "FPGA-based Ultra-Low Latency HIL Fault Testing of a Permanent Magnet Motor Drive using RT-LAB-XSG" SIMULATION, Vol. 84, No. 2-3, 161-171 (2008)

[9] Carreira, J.V.; Costa, D.; Silva, J.G.; "Fault injection spot-checks computer system dependability". Spectrum, IEEE. Volume 36, Issue 8, Aug. 1999 Page(s):50 - 55

[10] CEO Power, Inc. “TurboFault, Fast Concurrent Fault Simulator with SDF Timing Support”.

[11] Compuware Corporation, “DevPartner Fault Simulator" Software

[12] Condit, Reston; “AN907, Stepping Motors Fundamentals”; MICROCHIP; 2004

[13] Delli Colli, V.; Di Stefano, R.; Marignetti, F.; Scarano, M.; "Hardware in the Loop Simulation of a FPGA-based Speed and Position Observer for non-Salient Permanent Magnet Synchronous Motors" Industrial Electronics Society, 2007. IECON 2007. Nov. 2007 Volume 5, Issue 8. Pages: 992 - 997

[14] dSPACE. “” URL: http://www.dspaceinc.com/ww/en/inc/home.cfm

[15] Duman, E. Can, H. Akin, E. “Real time FPGA implementation of induction machine model - a novel approach” International Aegean Conference on Electrical Machines and Power Electronics, 2007. ACEMP '07. Volume 10, Issue 12. Sept. 2007. Pages: 603-606

[16] Electric Machinery Comittee of the IEEE Power Engineering Society ; “IEEE standard test procedure for polyphase induction motors and generators” IEEE Std 112-2004 (Revision of IEEE Std 112-1996). Pages" 1-79

[17] Etas. URL: http://www.etas.com

[18] Gabriëls, René, “Communication protocols: a synchronous and asynchronous one”. Literature material for the VLSI programming course at the TU/e. April, 2008.

[19] Gomez, Martin; “Hardware-in-the-loop simulation”; Embedded Systems Programming. Vol. 14; No. 13; December 2001.

[20] Jan M. Rabaey, Anantha P. Chandrakasan, Borivoje Nikolic; “Digital Integrated Circuits: A Design Perspective” Pearson Education, 2003; ISBN-10: 0130909963

[21] Jo, J.-Y.; Kim, Y.-W.; Ameduri, S.A.; Podgurski, A.; “A new role of graphical simulation: Software testing” Simulation Symposium, 1997. Proceedings. 30th Annual. Page: 216

[22] Karpenko, M.; Sepehri, N. “Hardware-in-the-loop simulator for research on fault tolerant control of electrohydraulic flight control systems”. American Control Conference, June 2006.
[23] Kenjo, Tak; “Electronic Motors and their Controls”. Oxford Science Publications. 1991. ISBN. 0-19-856235-7
[24] Khan, S.H.; Ivanov, A.A.”Modelling and computation of nonlinear magnetic fields in linear step motors by finite element method”. Magnetics, IEEE Transactions on; Volume 30, Issue 6, Nov 1994
[25] King, P.J.; Copp, D.G.; “Hardware in the loop for automotive vehicle control systems development”. UKACC Control 2004 Mini Symposia. Volume 8, Issue 8, Sept 2004. Pages: 75-78
[26] Kyusung Kim Parlos, A.G. Mohan Bharadwaj, R; "Sensorless fault diagnosis of induction motors"; Transactions on Industrial Electronics, IEEE, 2003. Volume 50, Issue 5. Pages: 1038-1051.
[27] Lesson Electric, “Bulletin 1600: Permanent Magnet DC Motors & Gearmotors, Custom DC Motors & Gearmotors” URL:http://www.leeson.com/cgi-bin/fetchpdf.cgi?/literature/bulletins/pdf/1600/19-26_custom.pdf
[28] Maclay, D.; ” Simulation gets into the loop” IEE Review Volume 43, Issue 3, 15 May 1997 Page(s):109 - 112
[29] Müller, Thomas; Graham, Dorothy; Friedenberg, Debra and Van Veendendal, Erik; “Certified Tester Foundation Level Syllabus”; version 2007. International Software Testing Qualifications Board. (ISTQB).
[30] National Instruments. “Hardware-in-the-Loop Test System”. URL: http://www.ni.com/embedded/hil.htm
[31] Océ Document Systems URL: http://www.oce.com/nl/default.htm
[32] Phillips, Charles; Nagle, H. Troy; “Digital Control System Analysis and Design”, Prentice Hall 1995; ISBN 0-13-309832-X
[33] Pickering Interfaces “Fault Simulation Applications”. URL. http://www.pickeringtest.com/index.html
[34] PWB Technologies. “Part Number ME22. Miniature Encoder” Datasheet.
[35] Das, S.R.; Assaf, M.H.; Petriu, E.M.; Jone, W.-B, “Fault simulation and response compaction in full scan circuits using HOPE”, IEEE Trans. Instrum. Meas., Volume 1. 2002, Pages: 607-612
[36] R. Jastrzebski, O. Laakkonen, K. Rauma, J. Luukko, H. Sarén, and O. Pyrhönen (Finland). ”Real-time Emulation of Induction Motor in FPGA using Floating Point Representation” Proceeding (443) Applied Simulation and Modelling – 2004.
[37] S. Prakhya, “Real-time matrix multiplication in FPGA”, Madras. University, India, 2005.
[38] Sang-Hyuk Lee, Sungshin Kim, Jang Mok Kim, Changho Choi, Jaesig Kim, Sanghoon Lee, Yongmin Oh; "Extraction of induction motor fault characteristics in frequency domain and fuzzy entropy"; IEEE International Conference Electric Machines and Drives, 2005. Pages: 35-40
[39] Schulte, T; Bracker, J; “Real-time simulation of BLDC motors for hardware-in-the-loop applications incorporating sensorless control”. IEEE International Symposium on Industrial Electronics, 2008; Pages: 2195-2200
[40] Smith, Roger, “Simulation Article”; Encyclopedia of Computer Science URL: http://www.modelbenders.com/encyclopedia/encyclopedia.html
[41] T. Kenjo, A. Sugawara, “Stepping Motors and Their Microprocessor Controls”, 2nd Edition, Oxford University Press, Oxford, 2003.
[42] ThomasNet. Industrial NewsRoom. URL: http://news.thomasnet.com/fullstory/451126
[43] Vector. URL: http://www.vector.com
[44] Verhoeff, Tom, “Passivator”., Encyclopedia of Delay Insensitive Systems (EDIS). URL.http://edis.win.tue.nl/sys/passivator/index.html
[45] W. Hong, W. Lee, B.K. Lee, "Dynamic Simulation of Brushless DC Motor Drives Considering Phase Commutation for Automotive Applications"; IEEE International

Electric Machines & Drives Conference, 2007. IEMDC '07. Volume 2, Pages:1377-1383

[46] Wagener, A; Schulte, T; Waeltermann, P; Schuette, H; "Hardware-in-the-Loop Test Systems for Electric Motors in Advanced Powertrain Applications" SAE, 2007.

[47] Wale, J.D. Pollock, C. Stebon Ltd., Burntwood; "Hybrid stepping motors and drives" Power Engineering Journal 2001. Volume 15, Issue 1. Pages: 5-12

[48] Weijie Lin, Zhuo Zheng; "Simulation and Experiment of Sensorless Direct Torque Control of Hybrid Stepping Motor Based on DSP", Proceedings of the 2006 IEEE International Conference on Mechatronics and Automation. Pages. 2133-2138.

[49] Wikipedia, "Rotary Encoder". URL. http://en.wikipedia.org/wiki/Rotary_encoder

[50] Xilink System Generator TM for DSP. URL: http://www.mathworks.com/products/connections/product_main.html?prod_id=304

[51] Yedamale, Padmaraja; MICROCHIP, "AN885, Brushless DC Motor Fundamentals". 2003

[52] Zabalawi, S.A.Nasiri, A; "State Space Modeling and Simulation of Sensorless Control of Brushless DC Motors Using Instantaneous Rotor Position Tracking". Vehicle Power and Propulsion Conference, 2007. Pages: 90-94

Appendix A

Step Motor Specifications

Rated voltage	2.8 V
Current/phase	1.68 A
Resistance/phase	1.65 Ω
Inductance/phase	2.8 mH
Holding Torque (T_{pk})	4.4 $kg \cdot cm$
# of Leads	4
Rotor Inertia (J_R)	68 $g \cdot cm^2$
Viscous Friction (b_f)	$5 \cdot 10^{-6}$ $N \cdot m \cdot s/rad$
Coulomb Friction (c_f)	0.007 N $N \cdot m$
Number of steps per revolution	200 steps
Number of steps per revolution in ¼ microstep	800 steps

EC Specifications

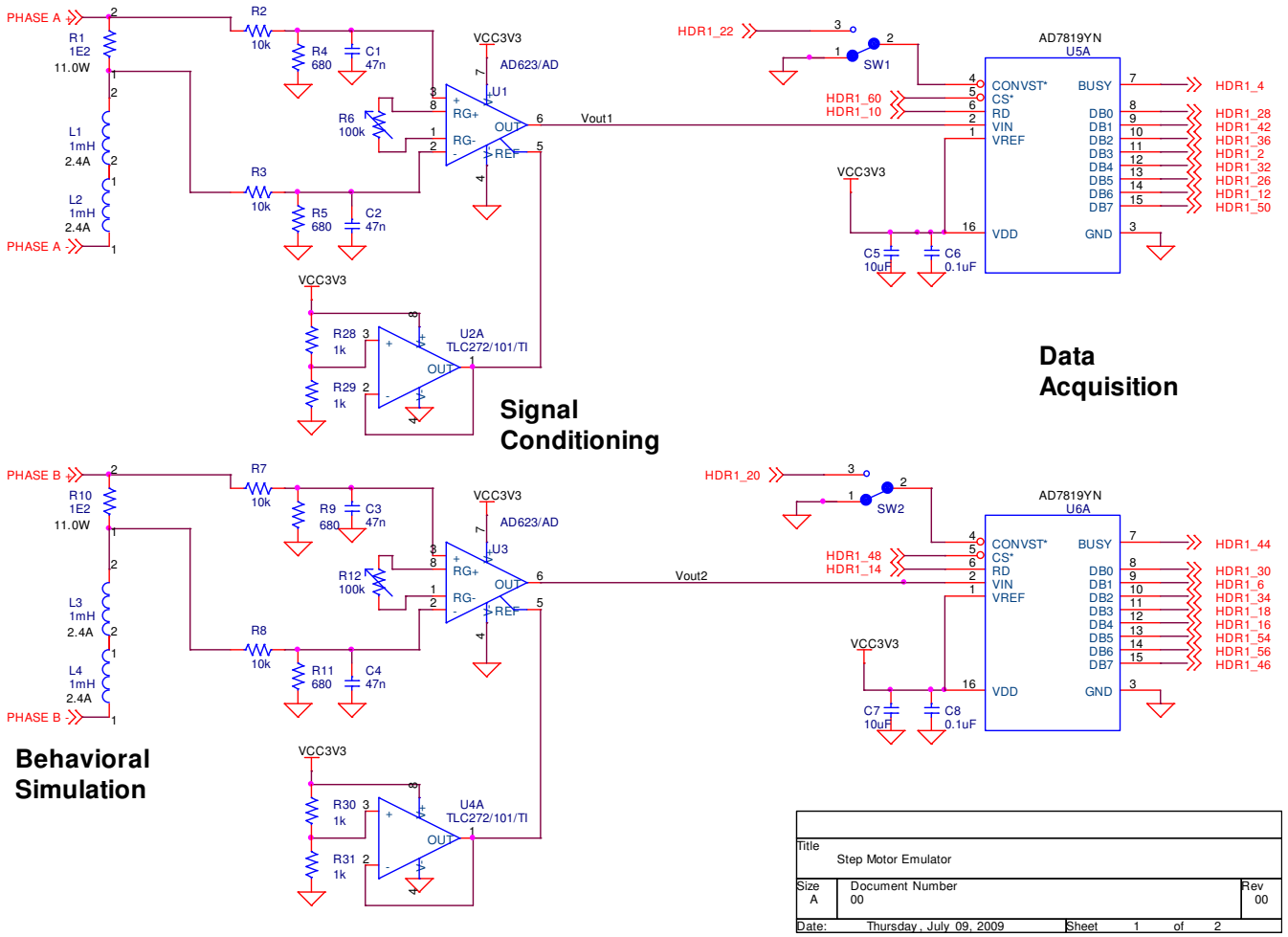
- The maximum frequency requested from the EC to the step motor is of 9306 KHz
- The EC has two current modes for driving the step motors, can be working in low or high current modes.

  High Current: $I_{\max} = 1.69A \rightarrow I_{RMS} = 1.2A$

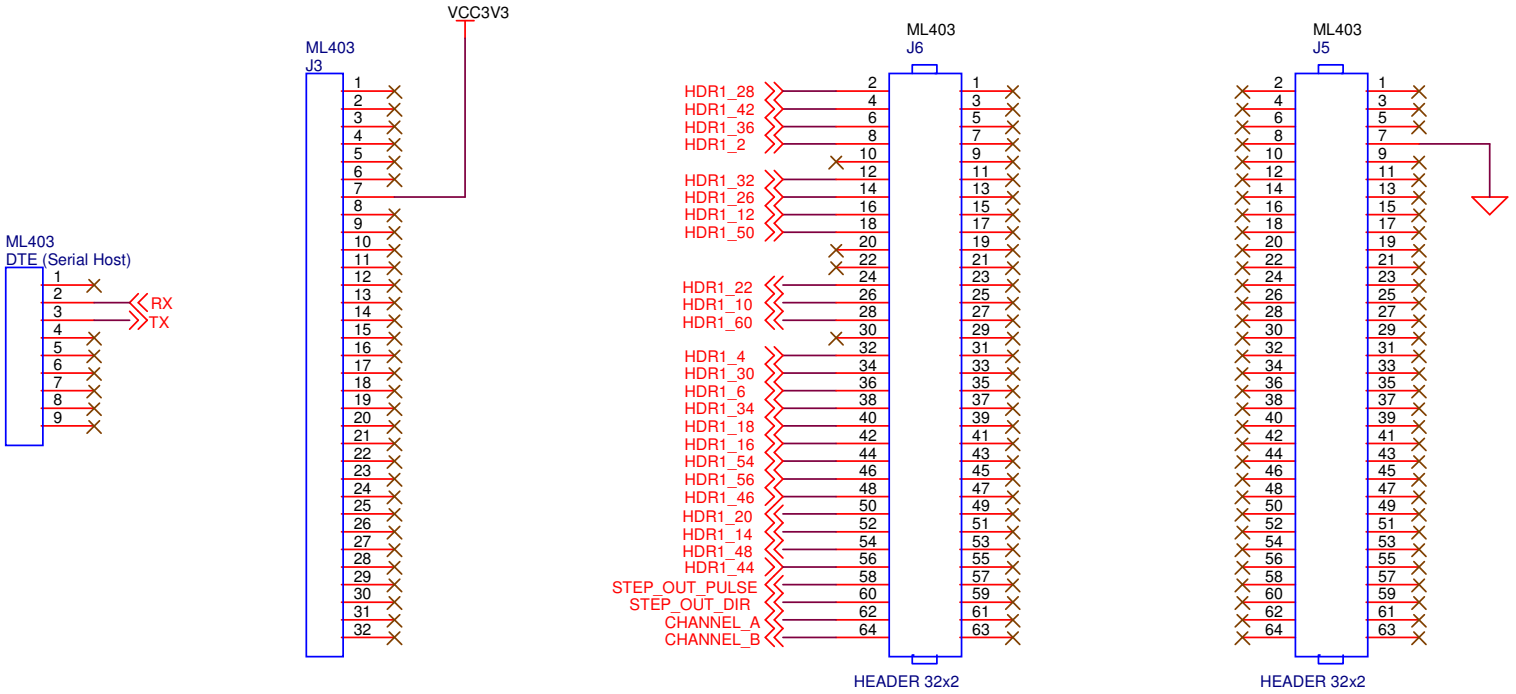
  Low Current: $I_{\max} = 0.6A \rightarrow I_{RMS} = 0.42A$

  Higher current flowing through the motor windings will develop higher torque in the step motor.

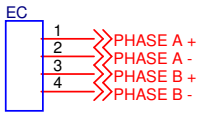
Appendix B


PHASE A
R1
1E2
11.0W
L1
1mH
2.4A
L2
1mH
2.4A
PHASE A
R2
10k
R3
10k
R4
680
C1
47n
R5
680
C2
47n
R6
100k
VCC3V3
AD623/AD
U1
RG+
RG-
OUT
REF
Vout1
VCC3V3
R28
1k
R29
1k
U2A
TLC272/101/TI
OUT
Signal
Conditioning
HDR1_22
SW1
HDR1_60
HDR1_10
AD7819YN
U5A
CONVST*
CS*
RD
VIN
VREF
VDD
BUSY
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
GND
HDR1_4
HDR1_28
HDR1_42
HDR1_36
HDR1_2
HDR1_32
HDR1_26
HDR1_12
HDR1_50
VCC3V3
C5
10uF
C6
0.1uF
Data
Acquisition
PHASE B
R10
1E2
11.0W
L3
1mH
2.4A
L4
1mH
2.4A
PHASE B
R7
10k
R8
10k
R9
680
C3
47n
R11
680
C4
47n
R12
100k
VCC3V3
AD623/AD
U3
RG+
RG-
OUT
REF
Vout2
VCC3V3
R30
1k
R31
1k
U4A
TLC272/101/TI
OUT
Behavioral
Simulation
HDR1_20
SW2
HDR1_48
HDR1_14
AD7819YN
U6A
CONVST*
CS*
RD
VIN
VREF
VDD
BUSY
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
GND
HDR1_44
HDR1_30
HDR1_6
HDR1_34
HDR1_18
HDR1_16
HDR1_54
HDR1_56
HDR1_46
VCC3V3
C7
10uF
C8
0.1uF
Title
Step Motor Emulator
Size
A
Document Number
00
Rev
00
Date:
Thursday, July 09, 2009
Sheet
1
of
2

VIRTEX 4 , ML403 - BOARD



EC Board



Title: Step Motor Emulator		
Size: A	Document Number: 00	Rev: 00
Date: Tuesday, June 16, 2009	Sheet 2 of 2	