

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

A. Oceguera¹, T. Basten^{1,2}, L. Somers^{1,3}, S. Hulsenboom³

¹Eindhoven University of Technology, ²Embedded Systems Institute, ³Océ Technologies
alvaro.oceguera@gmail.com, a.a.basten@tue.nl, lou.somers@oce.com, sander.hulsenboom@oce.com

Keywords: Step Motor Emulator, HIL simulation, Fault Injection, Emulator in an FPGA

Abstract

It is not possible to do automatic testing with step motors in Hardware-in-the-Loop (HIL) simulators because motors may be overheated if working for a long time. Furthermore, possibilities for fault injection are limited and physical interaction is needed to inject faults like a breakdown that can actually be mimicked. We present an emulator in an FPGA that allows to emulate a step motor for automatic testing as well as fault injection for a variety of faults, with the option of emulating more than one step motor in the same FPGA.

1. INTRODUCTION

Testing is one of the compulsory steps in developing software. Tools like a Hardware-in-the-Loop (HIL) simulator are used for testing real-time embedded software. Océ Technologies uses HIL simulators (HILs) to test the correct functionality of the electronic control board (EC) in printers (Figure 1). The EC is in charge of reading, processing and sending signals to the different modules of a printer for activating tasks needed for copying, scanning or printing documents and images.

Océ uses a HIL simulator to test among others the drivers of step motors. For instance, error handling code can be tested for the event of a motor breaking down. With real motors, the cables of the step motor need to be physically disconnected to emulate such an error, making the test slow and tedious. Other failures like missed steps cannot be emulated with real motors. Furthermore, a high risk of fire occurs when the step motors get overheated by performing continuous testing.

The solution for these issues is to have a step motor emulator; this paper presents a real time step motor emulator of a hybrid bipolar step motor, using a combination of electronics components and components embedded in an FPGA. The electronic components, an inductor and a resistor, help to recreate the behavior of the step motor. The FPGA helps to identify the steps the motor should be moving, to transmit the steps to the HIL simulator in the form of encoder signals and to perform fault injection.

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 windings 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 a resolution of one step;
- perform fault injection by allowing the simulation of the motor breaking down and skipping steps; commands are read via the serial port from the HIL simulator or from a PC and executed by the FPGA.

Compared to traditional and manual testing, this solution allows continuous automatic testing and fault injection when testing step motors through a HIL simulator.

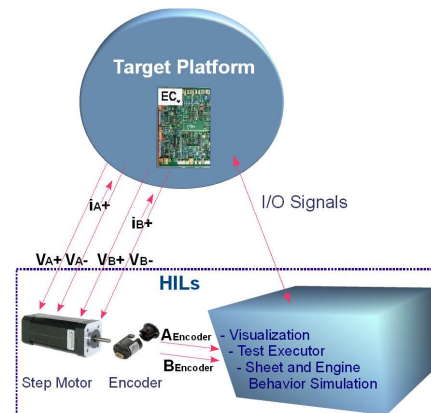


Fig. 1. Electronic Control board and its interaction with the HIL simulator (HILs).

Section 2 presents related work. Section 3 introduces step motor drivers. Section 4 discusses different approaches to realize a step motor emulator. The experimental evaluation is given in section 5. Section 6 concludes.

2. RELATED WORK

The standard method to create motor emulators works by reading the analog signals coming from the Electronic Control Unit, process them and use them as inputs to solve the state-space equations representing the model of the

motor. The hardest part, due to its computational cost, is solving the differential equations fast enough to simulate the behavior of the motor in real time. Most research focuses on this aspect. In [7], a real-time emulation of an induction motor is done in an FPGA; in this work Jastrzebski uses integral methods programmed by a fixed point representation in the FPGA to solve the differential equations of the state-space model in less than $1\mu s$, allowing testing and evaluation of very fast motor controllers in real time. In [6], Duman presents 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 the matrix multiplication algorithms studied by Prakhya [8] and the design is downloaded into a PCI FPGA development kit using Quartus-II.

The approaches by Jastrzebski and Duman are intended to work for HIL simulators. Both approaches implement the motor model in an FPGA and both verify that the FPGA is solving the model equations in less than $1\mu s$. However, no further work is done for reading the signals and for generating the output signals to close the loop with the HIL simulator. Closing the loop presents more challenges than solving the model equations in real time.

A complete BLDC motor emulator is presented in [3]. Bracker simulates an inductive load that includes the realistic current waveforms of AC motors for the automotive industry. The work develops 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. The simulation measures the voltages at the output stage of the controller, calculates the current in real time through an FPGA, and generates a current in the output stage (in case the motor is working in generator mode) with the help of DA converters. In [9], Schulte presents the simulation of a BLDC motor, in which 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). This emulator allows testing, by connecting the electronic units for cars as they are, without manipulations and it is also suitable for testing control units using sensorless control techniques. In [4,5], Dufour presents an FPGA simulator of a Permanent Magnet Synchronous (PMS) Motor, using the Xilinx System Generator (XSG) from Simulink and the RT-LAB platform from Opal-RT. This simulator runs the model of the motor on an FPGA while the RT-LAB platform reads the analog input signals and generates the currents to feed them back into the controller, closing the loop and simulating the entire motor. The simulator allows the injection of open-phase and short-circuit faults.

Unlike all previous approaches, we propose an emulator for step motors where there is no need to solve the model of the step motor in the FPGA. Similar to [1], we use high power inductors and resistances. Implementation details are not provided in [1] though. Our implementation can be used to realize the functionality provided by [1], while adding the features of calculating the number of steps the motor should have moved according to the input signals, the generation of encoder signals representing these steps and fault injection, increasing test coverage of the embedded software in the EC.

3. STEP MOTOR DRIVERS

There exist commercial step motor drivers that facilitate the job of design engineers. In some of these drivers, when a pulse occurs on the STEP input pin, some internal activity modifies the output currents to the next level and polarity. These step motor drivers use microstepping to increase the performance and to limit noise and resonance problems. Microstepping divides a motor step in substeps allowing smoother transitions between steps and a better step resolution. With the sine-cosine microstepping technique used in the drivers for this work a constant torque is produced by controlling the current in the windings. Figure 2 (taken from [2]) shows the current-level sequence needed in each phase to move the step motor one microstep at a time after the pulse signal on the STEP input occurs. The length of this signal will determine the frequency of the step motor.

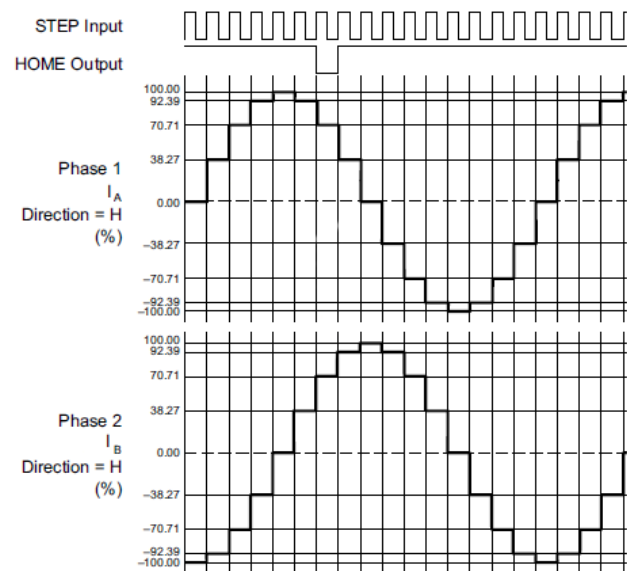


Fig.2. Current-level sequence to move the stepper microstep by microstep. [2].

These step motor drivers use the chopper technique instead of using a fixed voltage to create the currents that will flow through the windings; the result is a pulse-wide

modulated waveform that is used to create an average voltage and consequently an average current to move the step motor microstep by microstep according to Figure 2.

4. STEP MOTOR EMULATOR

Given how a step motor is controlled, we can now see the step motor emulator as a black box to get an overview of the signals that will be read and the signals that will be generated when designing the emulator. Figure 3 shows this black box scheme for the step motor emulator. The input signals are:

- four PWM voltage signals (V_{A+} , V_{A-} , V_{B+} and V_{B-}) coming from the EC board; these signals are generated by H-bridges and in the motor they generate the average currents flowing through the windings of the step motor to make it move microstep by microstep,
- one serial line (R_x) coming from the HIL simulator for reading commands and activating fault injection.

The output signals are:

- two currents flowing through the windings of the step motor (the value for the peak currents 1.62A and 0.8A were calculated using the loads to be present in the printers for the studied step motor), one for winding A and one for B; in Figure 3, 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 (the changing currents move the step motor microstep by microstep),
- 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 HIL simulator or a PC.



Fig. 3. Step Motor Emulator Black Box.

We consider three different methods to generate the step motor emulator. The third one is the most effective. We present the other two for completeness, and to illustrate the design decisions that play a role in developing a step motor emulator.

4.1. Constant Frequency Step Motor Emulator

The idea of this method is to reproduce the time constant τ of the step motor with an RC filter, where τ is the time it takes the current flowing in the winding to reach

approximately 63.2% of its final value after a step input (the voltage supply changes from zero to a one in a very short time). Figure 4 presents the electrical model of one of the two identical phases in the studied step motor, this model was used to calculate the time constant to be reproduced by the RC filter; $V_{A,PWM}$ is the voltage applied to the winding, R is the resistance of the winding, L is the inductance of the winding and $V_{A,EMF}$ is the back-electromotive force (back-EMF); $\tau = L/R$ is the time constant reproduced by the RC

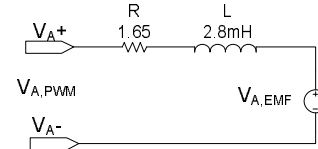


Fig. 4. Electrical model representing phase A in a step motor.

filter.

After building the RC filter for the step motor emulator, signal conditioning and data acquisition stages were built to read the signals in the FPGA. The result is not a sine wave similar to the one in Figure 2 but a square wave signal. This square wave signal is good enough for detecting the steps for constant frequencies, since it is possible to interpolate the steps that are not visible, but it does not work for varying frequencies. Section 5.1 provides the details.

4.2. Load Inductive Simulation for a Step Motor

Adapting the solution by Schulte et al. [9] for a load inductive simulation for BLDC motors, and a similar solution by Dufour et al [4,5] for a PMS motor, is a second approach to the creation of a step motor emulator.

The load inductive simulation developed of [4,5,9] reads the control voltages from the drivers, calculates the current that should be running through the motors by solving the differential equation model, and finally generates and sends the currents back to the driver, closing the loop. The adapted solution to the step motor driver is illustrated with Figure 5. It contains the following components:

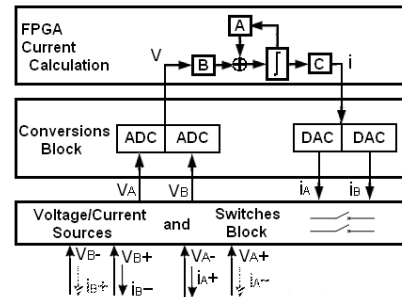


Fig. 5. Step Motor Emulator Block Diagram.

Voltage/Current Sources and Switches Block. This block is in charge of switching the emulator between reading voltages and supplying currents. Is the interface connected to the step motor driver. The emulator switches states based on the results of the differential equation solver,

and works as a voltage sink (when reading the PWM control voltages from the step motor driver) and as a current supply (when generating the currents the step motor should be generating as feedback to the step motor driver), through the same wires. This switching must be controlled by a state machine based on the results of the equation solver and the current level sequence in Figure 2.

Conversions Block. This block is in charge of converting the analog voltage signals to digital values to be processed in the FPGA and of converting the digital current values to analog ones for the current supplies. The analog to digital converters have a $5\mu\text{s}$ sampling rate (ten times more than the 10Khz maximum frequency demanded by the step motors used in this application) causing an oversampling of the voltages. This increases the accuracy of the Euler method we used for solving the state-space equations. The digital to analog converters, convert the currents calculated in the FPGA to analog values that are fed into the "Voltage/Current Sources and Switches Block", where they are generated to be supplied back to the step motor driver.

FPGA current calculation. This block calculates the currents that should be flowing through the step motor, by solving the state-space model of the motor in real-time using the voltages from the driver as input signals.

We chose a four-state state-space model of the step motor to be solved in the FPGA. The states of the model are the currents flowing through each one of the windings, the angular velocity and the rotor angle; while the inputs to this model are the voltages coming from the step motor driver in the EC. Based on these inputs, the currents can be calculated and the steps can be detected by comparing the calculated versus the predefined current levels shown in Figure 2.

To solve the model of the step motor we chose the Euler method since it presents the easiest implementation algorithm while showing a small error when choosing a small step size. Fixed point numbers are used to implement the Euler method for being easier to implement than floating point numbers and because less computational effort is required for operations on fixed point numbers, in comparison to floating point operations. The model of the step motor 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 for the integer part are enough to represent the values of the currents flowing through the step motor since they will never exceed 4 amperes in the step motor, the angle of the shaft is maximally

$360^\circ \rightarrow 2\pi = 6.2832$ radians, and based on simulations the angular speed of the motor oscillates between 1 and -1; in other words because of physical restrictions the numbers representing the integer part will not exceed the 6 bits assigned to them. On the other hand, 25 bits are necessary in the fractional part since the sampling time is fixed to $5\mu\text{s}$ (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. Using the fixed point notation and based on the Euler method, algorithms for addition, multiplication and for sine/cosine functions were created and programmed in VHDL.

The four state-space equations are transformed to be used by the Euler Method, and these transformed equations are divided into eight independent states. Each one of these states is computing independent arithmetic operations and the results of each state are needed by the next state. This was done to have independent operations in each state that can be executed in parallel, while every state needs to be executed sequentially due to data dependencies.

Simulation results show that this approach is sufficiently fast for real-time emulation, confirming earlier results described in the literature for other motor types.

This approach was not further developed, because of the following complexities and disadvantages compared to the approach described in the next subsection: (i) 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. These calculations imply a high computational complexity, an FPGA I/O port cost, fast A/D and D/A converters, switches and a state machine for control, (ii) the step motor simulation needs to work sometimes as a voltage sink and sometimes as a current source (implying the need for external power supplies), (iii) the switching between voltage sink and current supply needs to be done with the help of state machines that are difficult to design because of the difficulty to define appropriate transition conditions, (iv) fixed point operations inside the FPGA need to be used for solving the state space model in real-time, and consequently converting the data from the A/D converters to fixed point to be processed in the FPGA is required, and back again from fixed point to the physical representation, for generating the currents in the current source, and (v) the current supply needs to be faster than the PWM signals to accurately simulate the alternation of the current.

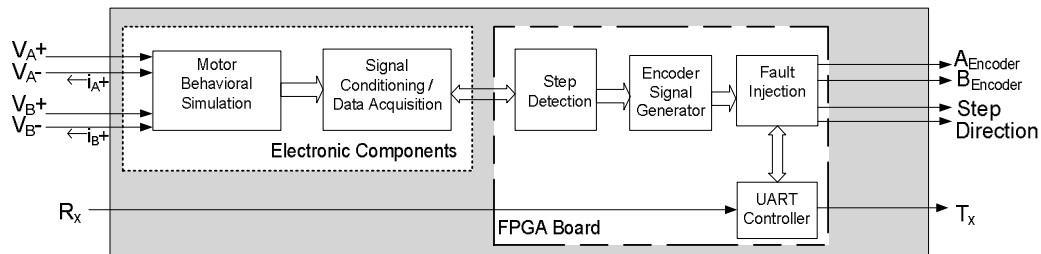


Fig. 6. Step Motor Emulator block diagram.

4.3. A Coil-based Real-Time Step Motor Emulator

An inductor is present in the electrical model of the motor as shown in Figure 4; 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.

Figure 6 presents the block diagram of this method while Figure 7 shows the implementation. The block diagram consists of:

A *Motor Behavioral Simulation* block in charge of simulating the behavior of the step motor that includes two inductors (coils) in series with a resistor. The coils have a value of 1mH and are used to generate the sinusoidal current similar to the one flowing through the phases in a step motor; the nominal inductance value of the motor is 2.8mH, which made us decide to have in the first design three 1mH inductors in series. After increasing the frequency requested by the driver, however, a better performance is obtained by using only two 1mH inductors in series.

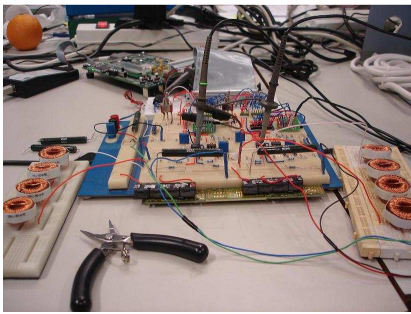


Fig. 7. Coil-based real-time step motor emulator.

A *Step Detection* block that is in charge of reading the digital signals from the A/D converter; these voltage levels are compared with the current levels in Figure 2 and a pulse is generated every time a new microstep is detected.

An *Encoder Signal Generator* block that is in charge of generating the two encoder signals from the information from the Step Detection block for feedback to the HIL simulator.

A *Fault Injection* block that receives the encoder signals from the Encoder Signal Generator block and commands from the UART controller with the faults to be injected. Based on these commands this block will potentially modify the encoder signals before sending them out of the emulator.

An *UART controller* block that is in charge of managing the serial communications between a computer or the HIL simulator and the emulator. This block will send commands to the fault injection block requesting the injection of faults.

5. RESULTS

This section discusses the experimental evaluation of the three approaches.

5.1. Constant Frequency Step Motor Emulator

Figure 8 shows the voltage signals reflecting the current flowing through the RC filter. The RC filter helps to reproduce the time constant of the motor but it does not help to reproduce the behavior of the current in the step motor.

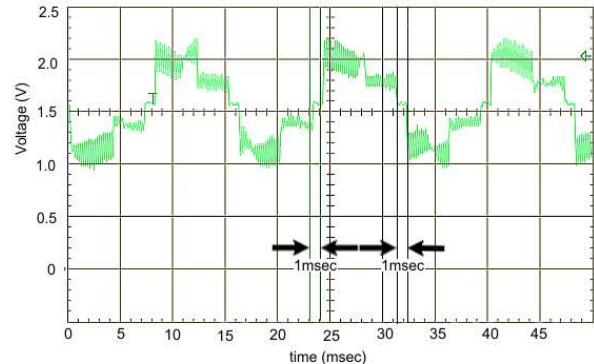


Fig. 8. Output Signal.

The square output signal does not allow us to identify all the steps. It does allow us to interpolate missing steps, since two steps can be distinguished, as seen in Figure 8, where we see the 2 steps of 1 milliseconds representing steps of 1 KHz.

Even when using 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, a varying frequency poses problems, since it is not possible to detect a frequency change of the steps; a similar result occurs when having only one resistor value demanding only two current levels, as in Figure 9 where the nominal value of the frequency is 2 KHz, and 0.5ms can be read from the graph detecting the step.

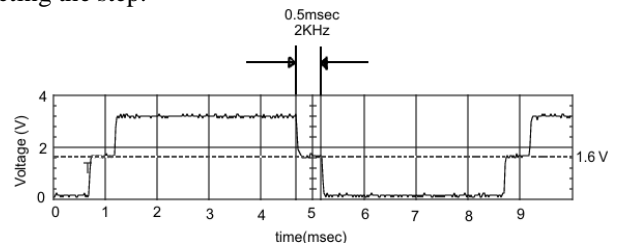


Fig. 9. Output Signal with a nominal frequency of 2Khz.

5.2. Load Inductive Simulator for a Step Motor

A script in Matlab was created to verify the results of the differential equation solver in VHDL. The Euler method was implemented in Matlab using fixed point numbers and the results were compared with the results of an already tested Matlab simulation of the step motor using Simulink.

Figure 10 shows the result of applying the Euler method to the fixed point numbers implemented in Matlab,

the currents in the vertical axis are in fixed point notation and need to be converted to the physical representation (using the decimal point). The VHDL simulation results show that the equations can be solved within 1 μ s, making it suitable for creating the emulator.

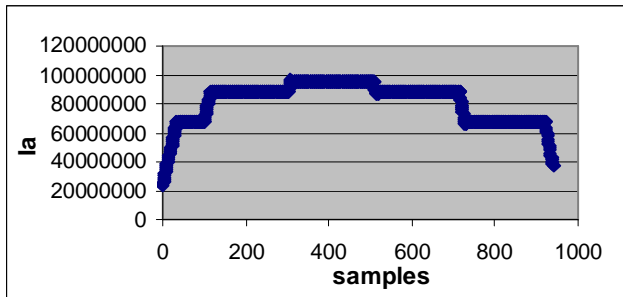


Fig. 10. Ia calculated in Matlab using fixed point numbers.

5.3. Coil-based Real-Time Step Motor Emulator

We consider the blocks of Figure 6 to present the results for the coil-based emulator. At the end of this section, we consider how well the approach scales when we want to realize multiple emulators in a single FPGA.

5.3.1. Motor Behavioral Simulation Block Results

After the motor behavioral simulation block the current generated with the PWM inputs from the EC has a sinusoidal behavior thanks to the effect the inductors cause in the currents. High (1.62A peak) or low current modes (0.8A peak) are available in the EC for controlling the step motors; using higher currents means increasing the torque of the step motor. Figure 11 shows the current flowing through one of the coils when 1 KHz is requested by the step motor driver. The distance between microsteps is 1 ms for 1 KHz.

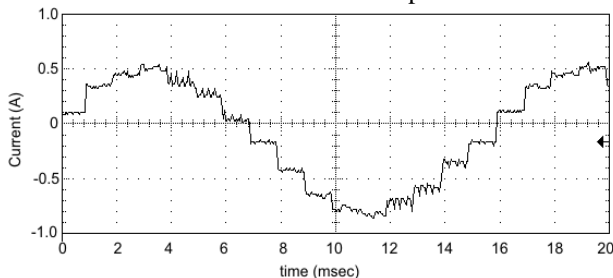


Fig. 11. Low current through the circuit for 1 KHz.

5.3.2. Signal Conditioning Block Results

After the Motor Behavioral Simulation block the current formed by the interaction of the voltages V_{A+} and V_{A-} has a sinusoidal behavior; these two voltages are reduced to lower values with a voltage divider and then combined in an instrumentation amplifier into one voltage signal (signal conditioning) finalizing the conversion from current to voltage; an offset is also added to force the signal to vary only between positive values. The results are the sinusoidal voltage waves shown in Figure 12 ("Voltage in

phase A" and "Voltage in phase B" waves). These voltages can now be sent to the A/D converter and their digital values to the FPGA.

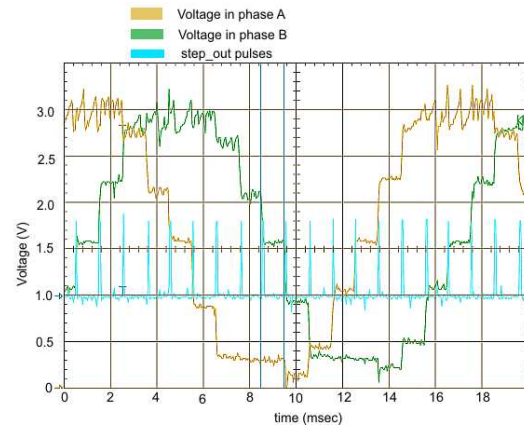


Fig. 12. 1KHz Step Pulse and Output voltages of the Instrumentation Amplifiers.

5.3.3. Step Detection Block Results

In Figure 2, the current levels for each step were presented as a percentage of the maximum current (I_{max}) flowing through the step motor. After the Signal Conditioning block, the voltage levels can similarly be identified as a percentage of the maximum voltage. These voltage percentage levels are compared against the current-level sequence from Figure 2 and a step pulse is generated every time a new step is detected as shown in Figure 12 with the step_out pulses wave.

5.3.4. Encoder Signal Generator Block Results

The outputs of the Encoder Signal Generator block are two waves simulating the encoder signals as presented in Figure 13 ("A_{Encoder} Signal" and "B_{Encoder} Signal"). While one encoder signal is sufficient to count the number of steps (4 steps per cycle), two encoder signals are used to indicate also the direction of the step motor. These signals are generated based on the spiky signal generated in the Step Detection block.

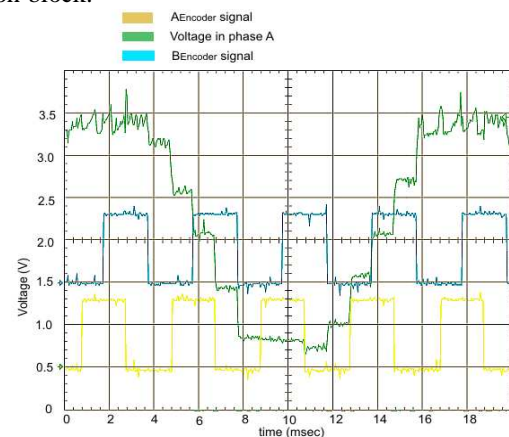


Fig. 13. 1KHz Encoder Signal and Voltage signals

5.3.5. Fault Injection Block Results

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

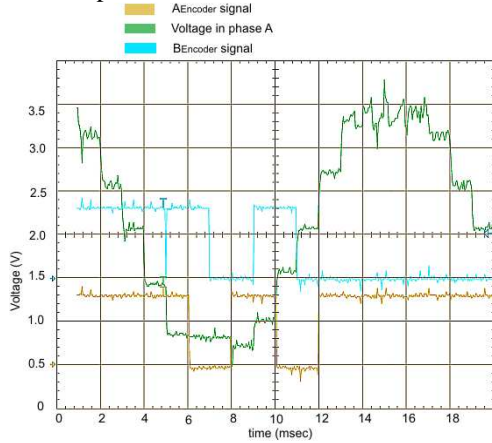


Fig. 14. 1 KHz Step Pulse and Voltages.

Figure 14 shows the result of the fault injection in the generated encoder signals; in Figure 14 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 reaches 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.

5.3.6. Device Utilization

Considering a linear increase in the number of resources in the FPGA when emulating multiple step motors, we can conclude from Figure 15 that the used FPGA (a Virtex 4 in the ML403 Embedded Platform) has enough 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 (IOBs): the development board assigns only 14.37% of the ports of the FPGA for general purpose, and 69.56% of these ports are used for the stepper emulator.

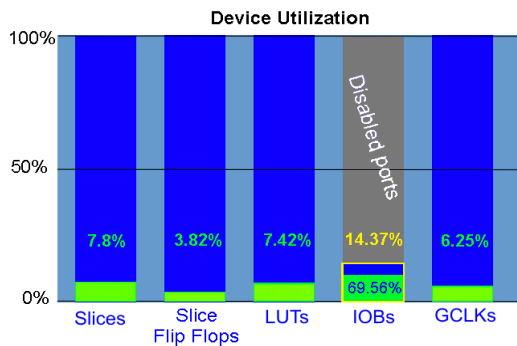


Fig. 15. Resource usage of one step motor emulator.

The rest of the ports (disabled ports) are used for other components like VGA outputs, mouse and keyboard connections, etc. Therefore, an FPGA development kit is not the best choice for emulating more than one step motor. Choosing an FPGA solution for the complete HIL simulator would make more ports available for step motor emulators, and it would have the additional advantage that the encoder signal generation can be omitted.

6. CONCLUSIONS

Table 1 presents a comparison between the three different approaches, where “+” represents the best performing option, “0” represents medium performance and “-” the lowest performance.

The Load Inductive simulator is the best option for *varying frequencies* since the emulator presented in this work starts to miss steps with frequencies above 7KHz; for the *simplicity* criterion, the coil-based emulator is a much easier solution compared to the Load Inductive simulator, since there is no need to solve the differential equations, there is also no need to externally generate the currents that the driver needs and less electronics components are needed. For *generality*, the Load Inductive simulator is the best option since it can emulate all kinds of inductive loads, while the coil-based emulator presented in this work is only for step motors, due to the constraints of finding coils with a high inductance or with a high current. For *number of I/O ports*, the emulator presented in this work needs less ports than the Load Inductive simulator since there are less electronics components to control.

TABLE I
COMPARISON BETWEEN THE THREE METHODS

	Constant Frequency	Load Inductive Simulator	Coil-Based Real-Time Step Motor Emulator
Varying Frequencies	-	+	0
Simplicity	+	0	+
Generality	-	+	0
Number of I/O ports	+	0	+
Low/High Current modes	0	+	0
Fault Injection	+	+	+
Accuracy	0	+	+

For *Low/High current modes*, the Load Inductive simulator can emulate all kinds of currents since it uses external power supplies while the emulator presented in this work is limited to the coils used. All of the approaches allow fault injection. The Constant Frequency emulator is of little value, because it is dominated in all aspects by the emulator using coils. If the constraints for the coil-based emulator are met, we believe that this emulator provides the best compromise between all the criteria. The emulator will be of great help for Hardware-in-the-Loop simulators. It represents a new option for automatic testing (there is no need for physical interaction); it is able to do fault injection

by receiving commands through the serial port, increasing the test coverage of embedded software by simulating the motor breaking down and the motor skipping steps; it is not noisy at all when compared to a normal step motor, and it does not produce the same amount of heat as a normal step motor when testing for a continuous and long period of time.

References

- [1] add2; "Stepper Motor Simulator card for two 4-phase stepper motors"; Genix I/O Cards. <http://www.addtwo.com/add2/genixcards.html>
- [2] Allegro; "Part Number: A3979, Microstepping DMOS Driver with Translator" Datasheet.
- [3] 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.
- [4] Dufour Ch., Bélanger J., Abourida S. "Real-Time Simulation of Permanent Magnet Motor Drive on FPGA Chip for High-Bandwidth Controller Tests and Validation" International Symposium on Industrial Electronics 2006. ISIE 2006. Pages: 2591-2596.
- [5] Dufour Ch., Bélanger J., Abourida S., Lapointe V. "FPGA-Based Real-Time Simulation of Finite-Element Analysis Permanent Magnet Synchronous Machine Drives" Annual IEEE Power Electronics Specialist Conference 2007. PESC '07. Pages: 909-915.
- [6] 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.
- [7] Jastrzebski, R. Laakkonen, O. Rauma, K. Luukko, J. Sarén, H. and Pyrhönen, O. (Finland). "Real-time Emulation of Induction Motor in FPGA using Floating Point Representation" Proceeding (443) Applied Simulation and Modelling – 2004.
- [8] Prakhya, S. "Real-time matrix multiplication in FPGA", Madras. University, India, 2005.
- [9] 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; ISIE 2008, Pages: 2195-2200.

Biography

Alvaro Ocegüera-Valenzuela received the B. Sc. degree in electronics engineering from the Instituto Tecnológico de Querétaro, México, in 2003, and the M.Sc. degree in Embedded Systems from the Eindhoven University of Technology, the Netherlands, in 2009. Currently, he works as Software Engineer for Continental Automotive in Guadalajara, México. His interests include microprocessors, FPGA and control systems.

Twan Basten received the MSc (1993) and PhD (1998) degrees in computing science from Eindhoven University of Technology. He is a professor of computational models in the Electrical Engineering Department at Eindhoven University of Technology and a research fellow of the Embedded Systems Institute, both in the Netherlands. His research interests include the design of resource-constrained embedded systems, multiprocessor systems, and computational models. He is a senior member of the IEEE and a life member of the ACM.

Lou Somers received a PhD in theoretical physics from the Radboud University Nijmegen in 1984. He currently works as senior staff member in the field of embedded software at Océ Technologies and is also part time associate professor at Eindhoven University of Technology.

Sander Hulsenboom, received the B. Sc. Degree in 2001. Currently he works at Océ Technologies as a coach and team leader of the Embedded Development Services FPGA group. This group develops FPGA components and designs for all Océ R&D projects. His interests include FPGA's, reuse, motion control, hardware/software co-design and embedded control architecture.