Modeling and simulation of FPGA-based variable-speed drives using Simulink

Francesco Ricci, Hoang Le-Huy

Department of Electrical and Computer Engineering, Université Laval, 1127-B Pavillon Adrien-Pouliot, Québec, QC, Canada G1K 7P4

Abstract

This paper presents a novel approach for the design and simulation of variable-speed drives when the control algorithm is to be implemented in a Field Programmable Gate Array (FPGA). The specialized Simulink tools used and the design procedure are presented. An application example using a Direct Torque Control (DTC) induction motor drive is presented and discussed.

© 2003 IMACS. Published by Elsevier B.V. All rights reserved.

Keywords: Simulation; Variable-speed drives; Simulink; FPGA

1. Introduction

Variable-speed drives are complex systems composed of components of different nature: power converter, electrical machine, and control system. The design and simulation of such systems require specialized software and often, the user must develop his own models for the power section [1]. If the developed control algorithm is to be implemented on a prototyping platform, it is desirable to have a simulation tool that can represent exactly the hardware used. It would be even more useful if the simulation software could automatically generate the necessary code from the model of the algorithm. This would avoid the daunting task of rewriting the control algorithm in another language like C for implementation.

If the prototyping platform uses a Field Programmable Gate Array (FPGA) to run the algorithm, a newly created simulation tool can be used not only to simulate exactly the hardware but also to automatically generate the VHDL code needed for the implementation. This software is the Xilinx System Generator (SG), a toolbox working in the MATLAB/Simulink environment. It provides the user with a library of blocks representing functions which can be implemented in an FPGA. It can also convert the created model into VHDL code. The use of the SG has two main advantages over traditional methods. First, the implemented algorithm is guaranteed to function exactly as in the simulation and second, there is no need to create two different models (one for the simulation and one for the implementation).
FPGA are increasingly replacing DSP in many applications due to several advantages they can provide. In an FPGA, multiple operations required by a control system can be executed in parallel so the algorithm can run much faster. Also, their high flexibility permits easy modification of the system functionality.

To simulate the power electronics section in MATLAB/Simulink environment, the Power System Blockset (PSB) is used. It permits accurate and fast simulation of both the power inverter and the motor. It is easy to connect the blocks of the SG and the PSB to simulate the whole system including the power circuit and the controller.

This paper presents a novel approach for the design, simulation and implementation of control algorithms for variable-speed drive systems using an FPGA. First, the PSB and the SG are described. Next, the design procedure using the proposed method is presented to illustrate the important steps. Finally, an application example using a Direct Torque Control (DTC) induction drive is presented and discussed.

2. Power System Blockset description

Simulink is basically a dynamic systems simulator and cannot directly simulate electrical circuits. To overcome this limitation a specialized toolbox is used, the PSB [2]. It consists of libraries of electric blocks and analysis tools that are used to convert electric diagrams into Simulink state–space diagrams. The electric blocks are electrical models such as electrical sources, linear and nonlinear electric elements, power electronic switches, electrical machines, electric connectors, and measurement devices (sensors).

When the simulation is started, Simulink use the PSB to transform the electrical circuit in a state–space representation with the steady state and initial conditions of state variables. After the initial conversion, normal simulation begins. State–space formulation allows the use of a wide variety of fixed step and variable step algorithms available in Simulink. For small- and medium-size systems, variable time step algorithms are usually faster because the number of steps is less than with a fixed time step method. However, for large systems containing a great number of states and/or power switches, it is advantageous to discretize the electrical system and to use a fixed time step algorithm.

Simulation results can be displayed on Simulink scopes or sent to workspace or disk file while the simulation is running. The user can access a variety of MATLAB functions and toolboxes for processing and plotting of waveforms from stored data.

3. Xilinx System Generator description

The Xilinx System Generator is a Simulink blockset developed by Xilinx Inc. [3]. Its purpose is to provide an easy-to-use high-level programming tool for FPGA users. The SG consists of a library of FPGA blocks (for model building) and a code generator which automatically generates VHDL code from the models created. The SG can work only with Xilinx FPGAs.

Additional conversions are required to have the VHDL code implemented in an FPGA. First, the generated VHDL code is transformed into a standard schematic circuit by a synthesizer program. Then, the schematic is transformed by an implementation tool into a configuration file that can be downloaded to the FPGA.
The main advantage of using the SG to simulate the control algorithm is the possibility to generate code that can be used to program an FPGA directly from the simulation model. Furthermore, the implemented code is guaranteed to perform exactly as in the simulation because the SG blocks represent exactly the FPGA hardware. This reduces the implementation time because the algorithm needs to be modeled and simulated only once.

Because the SG library is built to generate VHDL code it differs slightly from other Simulink libraries. First, all the mathematical operations are done in fixed-point format. For each function, a precision must be given in the $MN$ format where $M$ is the number of bits representing the integer and $N$ is the number of bits representing the fraction. Floating point arithmetic in FPGA is possible but too much logic is required. Control algorithms can be readily implemented in fixed-point format by using appropriate scaling.

Another difference is that SG blocks work only in fixed time step. FPGA usually uses registered logic that requires a clock which means that the simulation time step represents one clock cycle in the FPGA. It can be noted that the frequency in the FPGA does not depend on the time step chosen for the simulation.

The SG library contain many blocks that are classified in five sections:

1. **Basic elements**: Contains simple signal processing functions (delay, down sample, multiplexers, registers, ...).
2. **Math**: Contains common mathematical functions (add, multiplication, relational, negate, ...).
3. **MATLAB I/O**: Contains blocks to connect the SG section to the Simulink model and determine the quantization error created by the use of fixed-point values.
4. **Memory**: Contains memory elements found in FPGA (RAM, ROM, FIFO, ...).
5. **DSP**: Contains DSP like math functions (FIR, FFT, ...).

Many of these blocks are replicas of standard Simulink blocks. So it is easy to build a model the same way than with standard Simulink blocks. It is possible to create a particular block with the special block, the “Blackbox” but knowledge of VHDL language is needed. Fig. 1 presents some of System Generator blocks.

Some complex functions like division and inverse trigonometric functions are missing. When absolutely needed, it is possible to use a “Blackbox” to create these functions, but they will take a lot of space on the FPGA and require more than one clock cycle to execute [4].

Fig. 1. Some System Generator blocks.
To connect the SG algorithm to the remainder of the Simulink model, two specials blocks are used. “GatewayIn” transforms an incoming floating point signal to a fixed-point one with a fixed time step. “GatewayOut” is used to return a value to the normal Simulink environment. These two blocks represent the boundaries of the SG model. Everything inside them can be implemented.

A special block (System Generator) is used to generate the implementation VHDL code from the model. The conversion is automatic. The only thing needed to be specified is the FPGA model that will be used. The generated code can be then used to incorporate the control algorithm in an FPGA.

4. Modeling and simulation of a complete drive system

The creation of a Simulink model for the power section and the control algorithm needs a few steps that can be separated in two main parts. First, the system is designed and tested in MATLAB. Second, the VHDL code is generated and implemented in the FPGA with the appropriate tools. The detailed steps are illustrated in the flow diagram in Fig. 2.

The first step is to determine the parameters of the power electronic section and use them to create a model using the PSB blocks. Also the hardware section of the prototyping board like the ADC can be simulated with a mathematical model. In this way, the control algorithm will be simulated as an exact replica of the real system. The next step is the building of the control algorithm with the use of SG blocks only [5]. It is then connected to the simulated power system by “GatewayIn” and “GatewayOut” blocks. Fig. 3 illustrates the interconnections between the controller and the power circuit.
The fourth step is the simulation of the complete system in Simulink. Variable or fixed time step can be used for the PSB section but the SG blocks will always simulate in discrete time. Due to the nature of the SG blocks, the simulation time tends to be higher than with standard blocks. Because each SG block represents an exact replica of an implemented function, many operations are necessary even for a simple addition.

The next step is the VHDL code generation which is done with the “System Generator” block. Only the model of the targeted FPGA needs to be chosen, everything else is automatic. Usually, the FPGA will contain other functions in addition to the algorithm, communication for example. The algorithm must therefore be connected to the remainder of the VHDL code. This code is created only once and comes with the prototyping platform. A standard interface or a VHDL edition software can be used to join all the code parts.

The last step is the synthesis and implementation of the VHDL code. It is done with Xilinx software and needs only a few simple steps. After this step, a configuration file is obtained permitting programming the FPGA. In case modifications of the system are required, not all these steps are to be done again so the process will be much faster.

5. Application example—A DTC controller for induction motor drives

To test the capacity of the SG, a motor control algorithm has been modeled and implemented on an FPGA-based prototyping platform. The Direct Torque Control algorithm [6] has been chosen because it is widely used and offers a good level of complexity. The algorithm model creation was simple because nearly all the blocks needed exist in the SG library. Only one modification has to be done. The arctan function used by the original algorithm to locate the stator flux vector requires too much logic. It was then replaced by a simpler function that determines the sector by looking at the sign and the difference of the two vector components. Fig. 4 shows the model of this modified function.
Fig. 4. System Generator diagram representing the flux vector locator.
Fig. 5. Flux estimator diagram using System Generator blocks.
Fig. 6. DTC controller diagram using System Generator blocks.
The rest of the algorithm needs only simple functions available in the SG library. For example, the flux estimation needs only summers, multipliers and some registers to do the integration. Fig. 5 presents its model.

Fig. 6 presents the complete model of the control algorithm entirely built with SG blocks. It can be noted that each input and output needs a “Gateway” block to connect to standard Simulink blocks.

Fig. 7 shows the complete model of the DTC induction motor drive including the power circuit and the controller.

The simulation of the complete system permitted us to debug and determine the precision needed for the SG blocks to have an acceptable performance. A simulation of the original algorithm side by side with the SG algorithm was made to compare the performances. In Fig. 8, the torque and stator flux estimations

---

**Fig. 7. Simulink diagram representing a DTC induction motor drive.**
Fig. 8. Comparison between Xilinx SG and Simulink simulation results.
are compared. A slight difference was noted which is due to the one time step delay used in the SG algorithm to represent the ADC on the prototyping platform. This difference is too small to affect the performance of the algorithm.

Once the performance in simulation is acceptable, the algorithm can be implemented on a prototyping platform based on an FPGA. Fig. 9 shows a simplified diagram of the proposed prototyping platform. It consists essentially of two FPGA boards. A high-speed optical link assumes the communication between the two boards. The control algorithm is implemented on the master board (next to the computer). The other board controls the power inverter and digitizes the measures. The drive system uses a 2 kW induction motor and an IGBT PWM inverter.

The implemented algorithm was functional at the first try, but the performances were less than expected. The actual motor torque was lower than the simulation value. Inaccurate current and voltages measures of the motor (due to the ADC system) were identified as the cause of this problem. Since the torque and flux of the motor are estimated from voltage and current measures, their values are slightly affected. Nevertheless, the control algorithm was able to provide a static speed error less than 1% in the 400–1800 rpm range.

Fig. 10 shows the experimental DTC controller response to a step change in speed reference (from 1000 to 1100 rpm). Both speed and torque responses are smooth and do not present large oscillations. The small ripple appearing on the speed signal is due to the noisy signal provided by the speed sensor. The controller dynamics agree well with theoretical predictions. The obtained speed control accuracy (1%) is acceptable for industrial applications.

The implementation of the DTC controller using the proposed method was a success but the prototyping board needs some improvement regarding the ADC system. With more accurate measures, the control algorithm performance would be improved. The proposed simulation and design approach using SG has permitted testing the real world capabilities of the control algorithm without building and debugging a second model for implementation.
6. Conclusion

A novel approach to simulating complete variable-speed drive systems using specialized tools (Power System Blockset, System Generator) in Simulink has been proposed. This approach provides many advantages in particular when the control algorithm is intended to be implemented on an FPGA-based system. First, the modeling of the algorithm need to be done only once so a lot of time is saved. Second, the simulation using SG blocks represents exactly the hardware used to run the algorithm. This reduces considerably the time to implement the algorithm because no modifications are needed to make it work on the hardware.

The implementation of the DTC algorithm has proved that the use of the SG in combination with the PSB is powerful and easy. Implementation time was reduced and the control algorithm was working as
expected on the hardware platform on the first try. Minor performance problems were detected but they were not due to the implementation method.

The combination of FPGA and rapid algorithm development will help the future development of complex control algorithms.

References