IJREAT International Journal of Research in Engineering & Advanced Technology, Volume 1, Issue 5, Oct-Nov, 2013
ISSN: 2320 - 8791
www.ijreat.org

Cloud Based Reconfigurable DC Motor Controller Code Design Using IOPT Models

R.Nithya[1], S.Usha[2]

Dept. Of Embedded System Technologies, Sri Sai Ram Engineering College, Chennai, India

Dept. Of Electronics And Communication Engineering, Sri Sai Ram Engineering College, Chennai, India

Abstract: The current integrated circuit technologies are approaching their physical limits in terms of scaling and power consumption, during this context, the electronic design automation trade is pushed towards finding more challenging issues in terms of performance, scalability, adaptability, and reduced development time. The combination of an intuitive graphical modeling language, in conjugation with automatic model checking and code generation tools is proposed. The proposed framework contributes to significantly reduce development time, both during the specification, system design phase and during the test. To demonstrate the proposed approach, the Brushless DC Motor closed-loop speed controller is designed with IOPT Petri Net models. The IOPT Petri Net model is an integrated development environment offered by IOPT-Tools. IOPT- Nets is a Web based Petri net class specifically designed to support the implementation of embedded system controllers without the need to manually write software or hardware description programs. The IOPT Web service includes an interactive graphical editor to plan IOPT models, a model checking framework consisting of a query system and state space generator, and automatic code generation tools to provide software or hardware controller implementations. The BLDC Motor speed controller designed rely on several subsystems, as well as noise filter, Quadrature decoder, PWM generator and a BLDC commutation manager. The sub-systems were modeled by adopting IOPT models, analyzed using the model checking tools, leading to the automatic creation of VHDL modules for every sub-system.

1. INTRODUCTION

The increasing complexity in recent real-time systems coupled with requests for more performance and shorter time to market have produced a high interest to investigate advanced design methods and technologies that could simplify and improve the reliability of embedded software design and implementation, while promoting the reuse of software and hardware components for a co-design system development.

In this complexity scenario on system design, the manual coding is tedious, time consuming and error prone. On the other hand, automatic code generation implement the designers to create changes in the system level model, and produce an HDL implementation in minutes by regenerating the HDL code. In this context, the electronic system design adopts the model-driven engineering to automate the design proposes. This approach solves more challenging problems in terms of performance, scalability, adaptability, and reduced development time on system design.

The Model Driven Architecture (MDA) is intended to support model-driven engineering of software systems. The MDA framework is combination of an intuitive graphical modeling language, in conjugation with automatic model-checking and code generation tools. This framework contributes to significantly reduce development time, both during the specification, system design phase and during the test.

The complex system design requests leads the whole system is integrated into a single chip; the system size is restricted by the chip size. The current integrated circuit technologies are approaching their physical limits in terms of scaling and power consumption. It has been estimated that the number of integrated transistors on chip increases by 50 % per year. Moreover, as the integration level escalates in accordance with Moore’s law, the chip size is getting smaller and smaller. The UML and MATLAB are the most used automatic code generation tools in MDA based ASIC/FPGA system design.

The current trend used for modeling system controller behavior was Petri Nets (PN), which allows the association of its static characteristics (marking of the PN) and its dynamic characteristics (firing of transitions) to the graphical characteristics of the desired synoptic. The developed computer tool interoperates with other tools developed under FORDESIGN project, as the Petri Net graphical editor and automatic VHDL code generator, allowing the automatic generation code from behavioral models.

Model-based design in Petri-Nets environment for FPGA prototyping is very flexible and makes implementation of control algorithms in FPGA for power electronics and motor drives a lot faster with no need of special attention to internal connections in the device prototype. The prototype is used to verify various control functions, modulation methods and power flow regulation algorithms for various tailor made power electronic design and motor control applications in minimum time. Thus, by using an FPGA-based controller, the designer is able to build a fully dedicated digital system that is perfectly adapted to the control algorithm being implemented.

The following section presents the Model Driven Architecture, Petri-Net based design methodology and the development of Petri-Net based FPGA Controller systems development specifications. After, briefly present the tools, which, taken as a whole, provide the support for that methodology and finally this present an example model, and conclude the test results.

2. MODEL DRIVEN ARCHITECTURE

The MDA proposes model transformations to obtain executable model from a platform independent model. MDA is intended to support Model-Driven Engineering (MDE) of software systems. MDE is a software development methodology which focuses on creating and exploiting domain models (that is, abstract representations of the knowledge and activities that govern a particular application domain), rather than on the computing (or algorithmic) concepts.

The MDA is a specification that provides a set of guidelines for structuring specifications expressed as models. Adopting the MDA methodology, system functionality could initial be outlined as a platform-independent model (PIM) through an appropriate Domain Specific Language. Given a Platform Definition Model (PDM) corresponding to C,.Net, the Web, etc., the PIM could then be translated to at least one or a lot of platform-specific models (PSMs) for the particular implementation, using completely different Domain Specific Languages, or a General Purpose Language like VHDL, C,C++,Java, C#, Python, etc. The translations between the PIM and PSMs are usually performed using automated tools like model transformation tools, as an example tools compliant to the new OMG standard.

The principles of MDA may also be applied to other areas like business method modeling wherever the design and technology neutral PIM is mapped onto either system or manual processes. The Model driven architecture model is expounded to multiple standards, like the Unified Modeling Language (UML), Matlab/Simulink, and Petri-Net. Note that the term “architecture” in Model-driven design does not see the design of the system being modeled, but rather to the design of the various standards and model forms that functions the technology basis for MDA.

MDA APPROACH

One of the main aims of the MDA is to separate design from architecture and realization technologies facilitating that design and architecture can alter independently. The design addresses the useful needs whereas design provides the infrastructure through which non-functional requirements like measurability, performance and reliability are realized. MDA visualize that the platform independent model (PIM), which represents a conceptual design realizing the useful needs, will survive changes in realization technologies and software package architectures.

MDA TOOLS

An MDA tool is a tool used to develop, compare, interpret, measure, align, transform, verify, etc. models or Meta models. In the following section "model" is interpreted as any kind of model (e.g. a UML model) or meta-model. In any MDA approach we have two kinds of models: initial models are created manually by human agents while derived models are created automatically by programs. An MDA tool may be one or more of the listed functions say, Creation, Analysis, Transformation, Composition, Test, Simulation, Metadata Management, Reverse Engineering

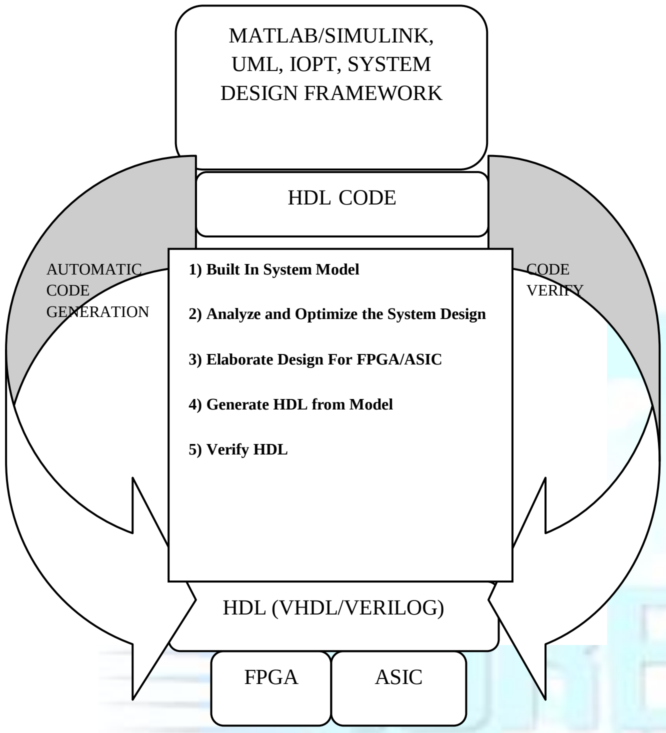
MATLAB/SIMULINK, UML, IOPT, SYSTEM DESIGN FRAMEWORK
HDL CODE
AUTOMATIC CODE GENERATION
CODE VERIFY
1) Built In System Model
2) Analyze and Optimize the System Design
3) Elaborate Design For FPGA/ASIC
4) Generate HDL from Model
5) Verify HDL
HDL (VHDL/VERILOG)
FPGA
ASIC

Fig 1: MDA Tool Framework

UNIFIED MODELING LANGUAGE

UML provides is the ability of modeling at different levels of abstractions. Since C++ is object oriented, a modeling language such as the UML can be used to develop object modeling based systems. Accordingly, UML satisfies the basic requirements to link specifications with implementation: visualization, modularization and abstractions.

The UML provides model image through a variety of various kinds of diagrams. The UML diagrams are divided into three main categories: interaction diagrams, structural diagrams, and behavioural diagrams. Interaction diagrams show the present relationships as well as the exchange of messages inside the system. Structural diagrams show the building blocks of the system. Behavioural diagrams show the system reactions to external and internal events. UML provides thirteen completely different diagrams. The usage of those diagrams depends on what is being visualised. For example, use case diagrams show the services that actors can request from the system, whereas sequence diagrams show the exchange of messages among the various modules within the system, and class diagrams show real-world entities and their relationships.

MATLAB/SIMULINK

MATLAB & Simulink enable an alternative way to automatically generate readable and portable IEEE standards compliant HDL from MATLAB, Simulink and State flow models for a variety of FPGAs. In addition, MATLAB model-based design facilitates creation of FPGA-based prototypes and automates HDL code verification by co-simulating it with Simulink and optimizes the models to meet speed-area-power objectives for the FPGA.

The MATLAB environment provides two
Model based tools for fast system development:

i) Xilinx System Generator and
ii) HDL Coder

Either of these approaches gives a good FPGA design flow when used severally. But some projects take pleasure in a combination of approaches a workflow that combines the native Simulink workflow, device-independent or device-specific code, and code readability offered by Simulink HDL coder, with the Xilinx FPGA-specific options and optimizations offered by Xilinx System Generator.

3. PETRI NETS

Petri nets are graphical and mathematical modeling tool applicable to several systems. They are a promising tool for describing and learning information processing systems that are characterized as being synchronous, asynchronous, distributed, parallel, nondeterministic, and random. As a graphical tool, Petri nets can be used as a visual communication aid nearly like block diagrams, flow charts and networks. In addition, tokens are utilized in these nets to simulate the synchronous and dynamic activities of systems. As a mathematical tool, it is possible to set up state equations, mathematical models and other algebraic equations governing the behavior of the systems.

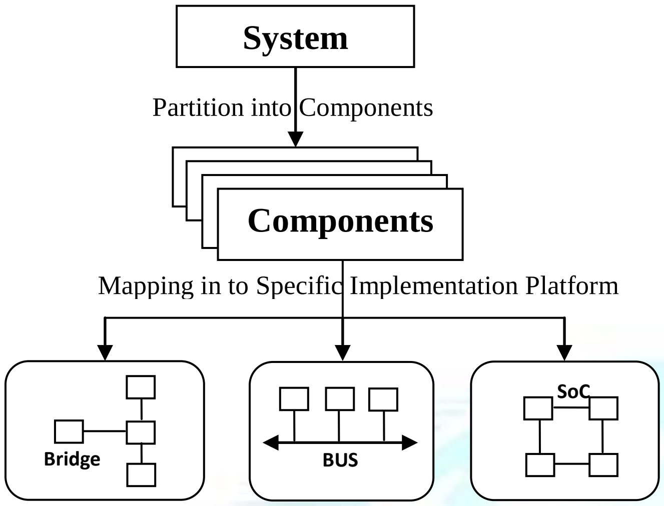
System
Partition into Components
Components
Mapping in to Specific Implementation Platform
Bridge
BUS
SoC

Fig 2: PETRI-NETS Usage Framework

Petri net was introduced by Carl Adam Petri in 1962. Petri net model is a graphical illustration used to design a complex system. A diagrammatical tool to model concurrency and synchronization in distributed systems. It is terribly like State Transition Diagrams. Utilized as a visual communication to model the system behavior. It is based on strong mathematical foundation. Petri net is a bipartite graph. It consists of two parts a net structure and an initial marking. A net (structure) contains two forms of nodes: places and transitions. There are directed arcs from transitions to places and directed arcs from places to transitions in a net. Places are diagrammatically represented by circles and transitions by boxes or bars. A place can hold a token which is denoted by black dots, or a positive integer representing their number. The allocation of tokens over the places of a net is termed a marking that corresponds to a state of the modeled system. The initial token distribution is hence referred to as the initial marking.

4. IOPT PETRI-NETS

IOPT stands for input output place transition tool. IOPT tools are used to generate the code automatically for the model designed. IOPT tools have been developed by many members of the R&D Group on Reconfigurable and Embedded Systems (GRES). The mathematical properties of IOPT Petri nets are applied to notice the design errors during the early design stages, contributing to minimize the time consumed during the check and validation stages and reducing price and time-to market.

The IOPT Petri net class inherits all characteristics from P/T (place-transition) nets, and a collection of non-autonomous extensions planned to the design of embedded system controllers, providing support for communication with physical devices, together with the controlled systems and user interfaces. The Web interface and the automatic code generation provides a easy way to create embedded system controllers, ready to be used even by persons without deep knowledge of hardware and low-level software code.

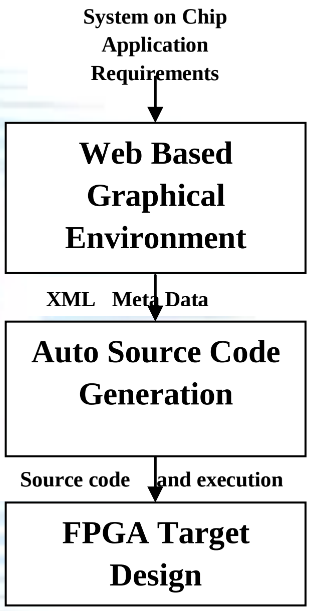
System on Chip
Application
Requirements
Web Based Graphical Environment
XML Meta Data
Auto Source Code Generation
Source code and execution
FPGA Target Design

Fig 3: IOPT-Tools Web User Interface

The Web based new model-checking tools, consist of a state-space generator, a query system and a automatic code generator, were added to associate existing tool framework that already had tools to import and edit automatic code generators, controller models and automatic hardware synthesis tools. The new tools allows the automatic analysis of system properties and the also the quick detection of errors during early design stages, significantly reducing debug and validation time. All tools share the similar Web based user interface, publicly accessible on the research group Web page (http://gres.uninova.pt).

Model definition and specification benefit from the graphical nature of Petri nets, where design often starts from a set of known use-cases. Complex concurrent systems can be easily modelled as a collection of individual sub-systems that are later joined with the support of a net addition operation. To detect design flaws, the framework contains a model checking tool comprehending a state-space generator and a query system used to extract information from the resulting state-space graphs. The query system is necessary to analyze real-world embedded system controllers that frequently lead to very large state-space graphs with millions of states and cannot be easily inspected by human operators. Queries allow the detection of critical situations, as the reachability of undesired states that would cause malfunctions or expose users to physical danger. The reachability of desired states can also be verified using queries. For example, to verify the reachability of ending-states where specific operations should finish or the reachability of the initial state on systems that must exhibit reversible behaviour.

In addition, the model-checking tools always check important system properties, as the existence of deadlocks and conflicts between net transitions. Maximal place bounds are calculated to determine the size of the memory elements, or data types, used in controller implementations. The automatic code generation tools compile IOPT controller models into software source-code implementing the model's execution semantics, minimizing the amount of handwritten code that is reduced to simple interface code, dependent on the target devices. Low level details are hidden from the high-level models and the error-prone and time consuming coding tasks are almost eliminated.

The tools are offered under a Web-based user interface, implementing an integrated development environment, displayed in Figure 4. Users can upload IOPT Petri net model files, perform model visualization and edition, state space generation and execute model-checking using a query editor and a query results filter page. Finally, the automatic code generation tool produces the controller code.

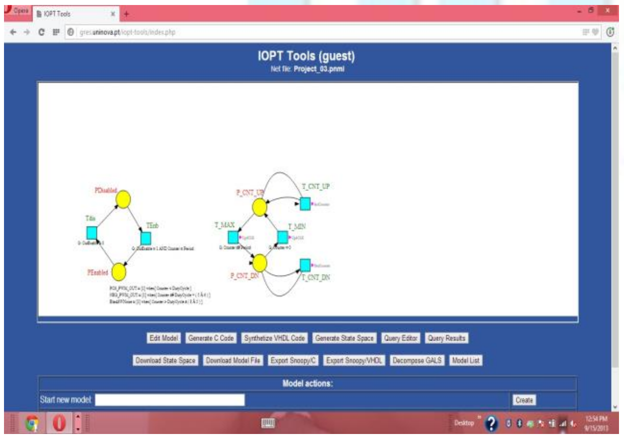

Fig 4: IOPT-Tools Web User Interface

System modeling and controller design is done using the SnoopyIOPT Petri net editor, or other Petri net editor supporting the PNML standard, with capability to support the IOPT extensions. Controller implementation was performed using several automatic code generation tools, generating C software code (PNML2C, IOPT2C) or VHDL hardware descriptions (PNML2VHDL), according to the target device.

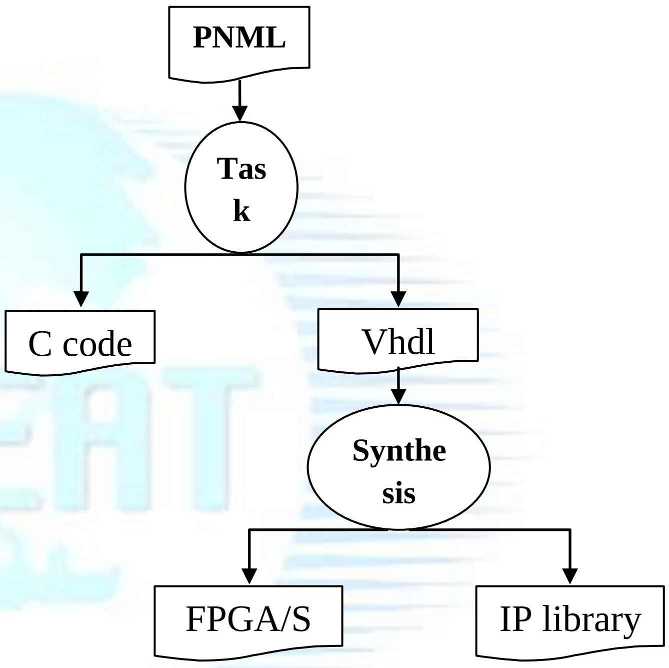

Fig 5: Automatic Code Generation Tool

An Animator tool creates system synoptic and graphical interfaces to interact with users. The interfaces are designed by specifying a set of rules relating the system state and input values with the position and movement of graphical objects on screen, without requiring writing any software code. Another tool, GUIGen4FPGA, implements those graphical user interfaces directly in FPGA hardware, including a high resolution video generator and pointing and touch device interfaces, enabling the automatic creation of full embedded system controllers comprehending the system controller and a user interface.

5. SYSTEM MODEL

Closed-loop controls are utilized in applications that need additional accurate and adaptive control of the system. These controls use feedback to direct the output states of a dynamic system. Closed-loop controls overcome the drawbacks of open loop control to produce compensation for disturbances within the system, reduced sensitivity to parameter variations, and stability in unstable processes.

A PID controller is a closed-loop control implementation that is widely used and is most typically used as a feedback controller. The closed-loop control regulates the speed of the motor by directly dominant the duty cycle of the PWM signals that direct the motor drive circuitry. The main distinction between the two control systems is that the open-loop control considers only the speed control input to update the duty cycle of the PWM, whereas the closed-loop control considers both speed-input control and actual motor speed (feedback to controller) for changing the PWM duty cycle and in turn the motor speed.

The System Architecture of Implementation given in Fig 6, the Speed Control Input unit provides motor speed input to the system. This input can be either digital or analog. The actual motor speed is fed back to the closed-loop controller, which is implemented on an FPGA. The PID controller is used as the closed-loop control algorithm to trace the actual motor speed and also apply the speed control input. Based on speed control input and present and past errors (proportional and integral values), the closed-loop control will either increases or decreases the PWM duty cycle, that successively controls the speed of the motor.

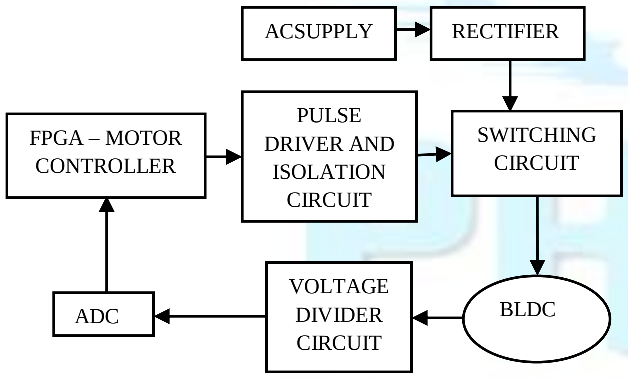

Fig 6: System Model Architecture

The FPGA – Motor Controller module consist of Noise Filter, QE Decoder, PWM Generator, and BLDC Communication Manager.

NOISE FILTER

Filters high frequency glitches from feedback signals.

QE DECODER

Decode the Quadrature signals produced by encoders, used to count pulses produced by the rotary encoder found in the FPGA board. In more advanced control strategies, this module could also be used to read the rotor position from feedback encoders on the motor shaft.

PWM GENERATOR

Output two complementary center aligned PWM signals according to the input period and duty-cycle values, with 500ns dead-time insertion. Another signal is used to blank spurious spikes in the current limit signal, produced during semiconductor switching.

BLDC COMMUTATION MANAGER

Redirects PWM signals to the correct U/V/W motor phases according to the selected commutation sector and current/torque direction.

SAMPLE IOPT MODEL

The sample implementation of PWM model is given in fig 7 using IOPT Petri net.

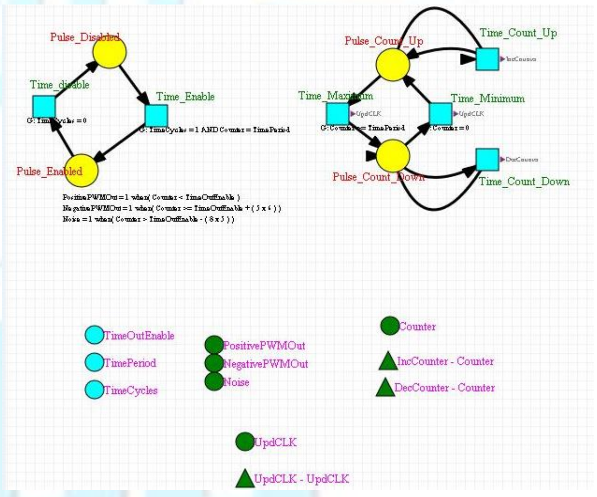

Fig 7: PWM generator

6. PROTOTYPE MODEL

The simple closed loop system BLDC motor control model designed for real-time justification as given in Fig 8. The designed models are inserted in the IOPT-Tools Web service (http://gres.uninova.pt). To examine for deadlocks, unreachable desired states the model checking tools were applied.

The VHDL code synthesis tools were applied to every model, producing VHDL codes for designed modules that were appended to a Xilinx ISE project.

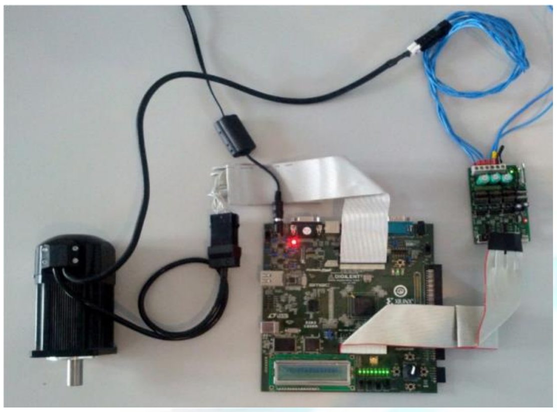

Fig 8: Closed Loop system BLDC Motor Model

The Test FPGA board uses Spartan3AN X700 FPGA with 50MHz clock with required motor driver circuits.

Although the VHDL code created by the automatic code generator is not optimal and includes several dead sections preceded by conditions that are always false or always true, the optimizing tools provided by FPGA vendors can easily remove the dead sections and produce a virtually optimal result. As a consequence, the FPGA resource consumption of the whole project is very small, corresponding to approximately 3% of the resources available on the Spartan3 FPGA.

7. CONCLUSION

The main goal of the work is to control the speed of BLDC motors using IOPT Petri net models. IOPT models provide a very simple and intuitive way to specify the desired system behavior. By dividing entire systems in small components that can later be inter-connected, complexity can be kept at very low levels and sub-systems are easily validated with the model- checking tools. Codes are generated for each subsystem and these codes are interconnected to single module. The dead sections of automatic code generation need to be improved on IOPT tools.

REFERENCES:

[1] FPGA based Speed Control of Brushless DC Motors using IOPT Petri Net models. Pereira.F, Gomes,L. Industrial Technology (ICIT), 2013 IEEE International Conference on Digital Object Identifier: 10.1109/ICIT.2013.6505810 Publication Year: 2013 , Page(s): 1011 - 1016

[2] L.Gomes, J.Barros, A.Costa, and R.Nunes, “The Input-Output Place-Transition Petri Net Class and Associated Tools,” in Proceedings of the 5th IEEE International Conference on Industrial Informatics (INDIN’07), Vienna, Austria, July 2007.

[3] F.Moutinho, L.Gomes, “From models to controllers integrating graphical animation in FPGA through automatic code generation,” in IEEE International Symposium on Industrial Electronics (ISlE 2009), Seoul Olympic Parktel, Seoul, Korea, July 5-8 2009.

[4] Ming-Fa Tsai, Tran Phu Quy, Bo-Feng Wu, Chung-Shi Tseng , "Model construction and verification of a BLDC motor using MATLAB/SIMULINK and FPGA control," Industrial Electronics and Applications (ICIEA), 2011 6th IEEE Conference on, vol., n., pp.1797-1802, 21-23 June 2011doi: 10.1109/ICIEA.2011.5975884

[5] R. Nunes, L.Gomes, J.P.Barros, "A graphical editor for the input-output place-transition petri net class", Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on , vol., no., pp.788-791, 25-28 Sept. 2007

[6] Pereira.F, Moutinho.F, Gomes.L , "Model-checking framework for embedded systems controllers development using IOPT Petri nets," Industrial Electronics (ISIE), 2012 IEEE International Symposium on , vol., no., pp.1399-1404, 28-31 May 2012 doi: 10.1109/ISIE.2012.6237295

[7] W. Reisig, “Petri nets: an introduction.” NY, USA: SpringerVerlag New York, Inc., 1985.