Automatic Generation of Graphical User Interfaces for VHDL based Controllers

Filipe Moutinho
Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologia & UNINOVA - CTS
Portugal
Email: fcm@uninova.pt

Fernando Pereira
Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologia, UNINOVA - CTS &
Instituto Superior de Engenharia de Lisboa
Portugal
Email: fjp@deea.isel.ipl.pt

Luís Gomes
Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologia & UNINOVA - CTS
Portugal
Email: lugo@uninova.pt

***Abstract*—Recent trends in embedded systems can be categorized by high application sophistication, shorter product life cycles and low cost requirements. As system complexity increases, rapid application development platforms and user interface builders became an important factor to reduce design cost and time-to-market. The tools proposed in this paper enable the rapid development of applications and the respective user interfaces, with automatic VHDL code generation for FPGAs, without requiring the need to write a single line of code. With high processing power, capability to generate graphical images and interface with pointing devices, reconfigurable hardware platforms provide a very competitive solution for embedded systems.**

I. Introduction

The range of applications for embedded systems is mainly limited by the availability of very low cost computing devices with increasing processing power and low energy consumption. Fortunately, the capabilities of computing devices have been constantly growing, with a corresponding cost reduction on the previous generation devices, enabling the design of more complex systems to cope with more demanding applications. Even traditional applications have seen a constant growth in complexity due to increased user requirements and the competition with more sophisticated concurrent products, leading to shorter product life cycles.

Under these conditions, embedded systems engineers must focus on two areas: rapid application development frameworks and graphical user interfaces. With lower hardware cost, increased complexity and shorter product life cycles, development has become a very significant part of the total production cost, generating the need for rapid application development frameworks. Higher product complexity and user sophistication places a great focus on user interfaces. This paper addresses both of these problems proposing a tool and a development flow to allow the creation of entire systems with animated Graphical User Interfaces (GUI) without writing code.

The development flow begins with the controller design using Petri net models, followed by the definition of animated GUI rules using an animator tool and finally the code generator tools automatically create the VHDL hardware descriptions for FPGAs or ASICs targeting the final embedded devices.

Petri nets are a proven modeling formalism that has long been accepted as a successful tool in the design, simulation and test of embedded systems, leading to the rapid development of prototypes. The Animator tool [1] previously developed during the FORDESIGN project [2], enables the design of animated Graphical User Interfaces based on a set of rules related to the Petri net model status and input signals, and can generate software code to simulate the system controller and run the animations on a personal computer.

The GUIGen4FPGA tool proposed in this paper enables the generation of animated GUI VHDL code for implementation in FPGA or ASIC platforms, including all the necessary IP modules, as a video frame buffer and pointing device interfaces. Each Graphical User Interface screen is composed by one static background image and many dynamic graphical objects. The graphical objects content can consist of text information, GUI buttons, icons, or general images. Icons and buttons are sensitive to mouse and touch events, and send information to the system controller. Each graphical object has a set of rules associated with the controller state and input signals to define visibility, screen location and movement. Animation sequences can be generated combining the effects of several objects and the respective rules.

II. State of the Art

The application of Petri net models to the design of embedded systems has been the subject of extensive research from several authors, including both low level Petri nets [3], [4] and high level [5], [6], [7].

Several authors have proposed tools to create Graphical User Interfaces and Interactive Animations [8], [9]; those systems generally require the association of function calls with the firing of Petri net transitions, and arc annotations to execute code that trigger actions in the animation during simulation. This solution presents several disadvantages: the user must write new code and make model modifications, with the risk of accidentally changing the system behavior. In contrast, the present solution is based on an external list of rules defined

978-1-4244-9312-8/11/$26.00 ©2011 IEEE

inside a graphical application and does not require writing new code and any modifications in the original model. As the model does not suffer changes, it is possible to create several animations referencing the same model file.

Other authors [10] used the formal theory of typed attributed graph transformation to create animations based on Petri net models; however the framework proposed by these authors only supports simulation. Most tools only support animation during simulation or execution in PC systems, and do not support automatic code generation for embedded devices. These animations are particularly useful to detect model flaws during test and debug, to extract graphical logs from simulation and to interact with users not familiar with Petri net formalisms.

The architecture proposed in this paper offers the advantage of automatic code generation for both the system controller and the animated GUI, enabling the implementation of final production systems on the real embedded devices. Other architectures [11] support automatic code generation but do not support automatic graphical user interfaces and animations.

Traditional frameworks, like LabVIEW and Simulink, include modules [12] to automatically generate controller's VHDL code for FPGAs and tools to build GUIs running on an computer connected to the FPGA device. These solutions have several disadvantages: the GUI does not run in the FPGA platform and require an external computer and expensive software licences.

This work benefits from a previous tool framework developed during the FORDESIGN project [2], reusing the tools for Petri net model edition, definition of animation rules and generation of controller's implementation code. The new tool (GUIGen4FPGA) offers the capability to automatically generate animated GUI VHDL code and all the core IP modules necessary to support the GUI under FPGA architectures, including a VGA video generator, frame buffer and mouse interface. The support for pointing devices introduced by the new tool greatly improves the functionality of the generated user interfaces, providing feedback capabilities to icons and button objects.

III. The IOPT Petri Net Class

The main characteristics of the Input-Output Place-Transition Petri net class (IOPT) are briefly presented in this section. A complete presentation, including the formal semantics and the Petri net type definition (according with the PNML format [13]) was presented elsewhere [3]. Roots from IOPT class come from well-known place-transition Petri nets [14], augmented with the specification of input and output signals and events, allowing explicit representation of the interaction between the environment and the net model, which models the controller behavior. In this sense, the IOPT class is a non-autonomous net class, as the dynamics of the net also depends on characteristics other than the graph and the marking of the Petri net model. Several other existing non-autonomous net classes also targeted for modeling controllers behavior (e.g. [15], [16], [17], [18], [19]). The benefits of selecting IOPT nets are associated with availability of a set of tools, including automatic C and VHDL code generation. The IOPT nets have maximal step semantics, which means that each evolution step includes the firing of all the autonomously enabled transitions whose input conditions and events are true. When compared with place-transition nets, IOPT nets have a few additional characteristics, namely test arcs, test arcs weights, and priorities associated with transitions (which are important for automatic conflict resolution), as well explicit dependency on input signals and input events associated with the enabling of the firing of a transition; finally, output signals can be associated with the marking of places and output events with the firing of transitions.

IV. Proposed Development Framework

The development flow proposed in this paper to create hardware based controllers integrating GUI extends the methodology proposed in [20] with the GUIGen4FPGA tool, and comprises the following steps: (1) collecting controller requirements and its description through UML Use Cases; (2) modeling of UML Use Cases through IOPT Petri nets; (3) automatic VHDL controller code generation from IOPT Petri nets; (4) establishing animation rules; (5) association between controller model inputs/outputs and implementation platform pins or GUI buttons/icons; (6) automatic VHDL GUI code generation from files generated in the previous two steps; (7) the final development step is the configuration of the physical platform.

This development flow uses several tools: (1) SnoopyIOPT, (2) Animator, (3) GUIGen4FPGA, (4) PNML2VHDL and (5) Xilinx ISE (a set of design tools created by Xilinx to allow synthesis, simulation and implementation in FPGA). Fig. 1 illustrates the set of tools and their interaction through files. The GUIGen4FPGA tool was developed in this work and will be presented in section VII.

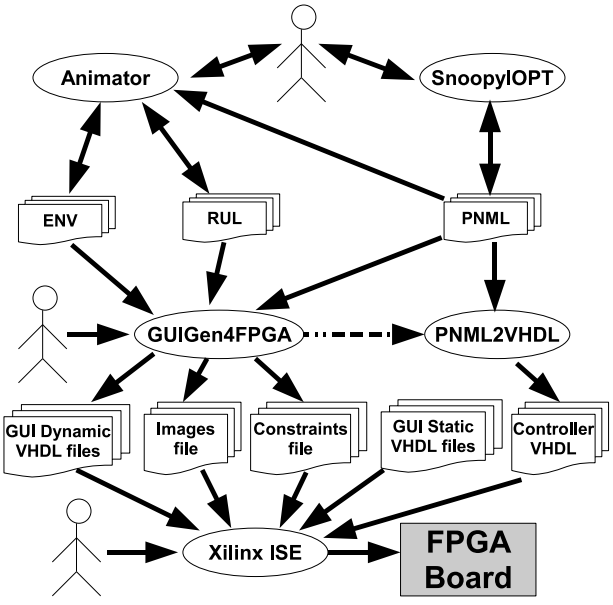

Fig. 1. Tool framework and information flow.

A. SnoopyIOPT

The SnoopyIOPT is an IOPT Petri net graphical editor [2], that among other tasks, allows the representation of IOPT Petri nets in PNML format [21].

B. Animator

The Animator [1] is a graphical editor to design graphical user interfaces associated to controllers modeled with IOPT Petri nets. To achieve this, the Animator supports the following tasks: (1) the definition of rules associating model characteristics to graphical objects, fig. 2; (2) the association between model inputs/outputs and platform inputs/outputs; (3) the generation of an executable application named Synoptic [1], enabling the simulation of the controller (controller code is generated by PNML2C tool) as well as the graphical user interface, on a personal computer.

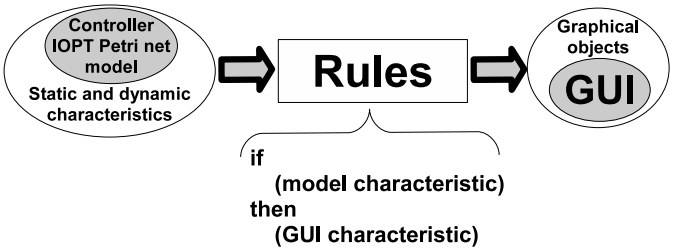

Fig. 2. Animation rules.

To develop FPGA based controllers integrating Graphical User Interfaces, the Animator defines a set of rules associating IOPT Petri net model characteristics (namely place marking and input signals) and synoptic graphical objects. These rules contain information about which images will be shown and in which conditions. Each object can be shown or hidden depending on the rules evaluation, and movement can be assigned to graphical objects along predefined paths.

C. PNML2VHDL

The PNML2VHDL [2] is an automatic controller VHDL code generator, starting from PNML files representing IOPT Petri net models. It is invoked by GUIGen4FPGA to generate the controller VHDL code file.

V. Global System Architecture

The proposed controller architecture is presented on fig. 3, and is composed of two main modules: the system controller (VHDL module code generated by the PNML2VHDL tool) and the GUI. The VHDL modules containing the GUI code are automatically generated by the GUIGen4FPGA tool, based on the files generated by the Animator. The system GUI is composed of several modules: (1) Video Controller; (2) MIG; (3) iMIG; (4) iFLASH; (5) iMouse; (6) GUI Controller; and (7) GUI Data.

The Video Controller module generates a SVGA RGB or LVDS signal to feed the LCD. The MIG module interfaces with the video RAM memory and was generated by the Xilinx

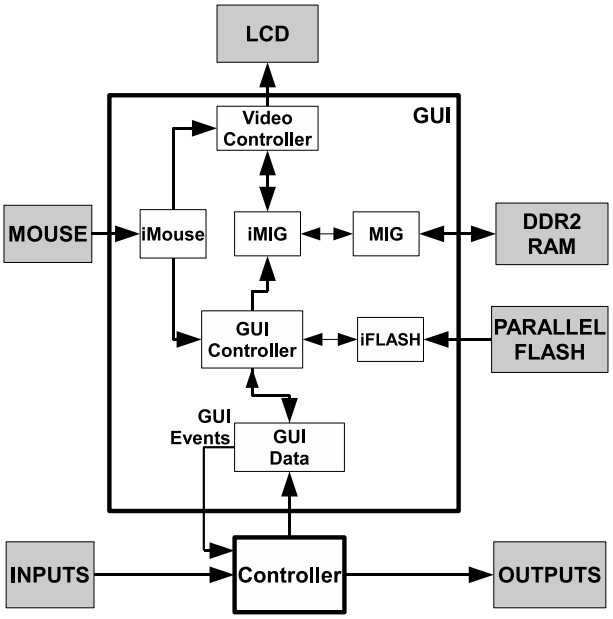

Fig. 3. Global system architecture.

Memory Interface Generator (MIG). The prototype presented in this paper uses an interface with DDR2 RAM memory, but interfaces with other memory types can be easily generated. The iFLASH module interfaces with the FLASH memory where the images are stored.

The iMIG module provides a higher-level interface with the RAM memory, hiding most of the MIG complexity, and offers a dual port interface to allow concurrent read and write operations. The read port implements a high-priority sequential interface to support video image scanning and the write port offers a random access interface. The video frame buffer stores two image frames to allow simultaneous image modifications in one frame while the other is being displayed. To take full advantage from the DDR2 RAM memory performance, the iMIG concatenates single data-byte operations into burst packets and uses a multi-stage pipeline architecture.

The iMouse module interfaces with pointing devices, receives mouse data and keeps track of the current mouse position and pressed buttons. As expected, the Video Controller displays the mouse cursor, and the mouse buttons generate events to the GUI Controller module. The present prototype implements an interface to support mice with the PS2 protocol. However, the iMouse module can be modified to support different pointing device protocols, including touch-screens.

The GUI Data module (fig. 4) contains the following information: (1) the locations where images are stored in the FLASH data memory; (2) images widths; (3) images sizes; and (4) the screen locations where the images will be drawn. It also checks: (1) controller inputs and IOPT model places to inform the GUI controller module if the corresponding images must be drawn; and (2) mouse events to report user actions to the controller module. The picking algorithm implemented in this module supports graphical objects with complex shapes

and holes instead of just rectangles, opening the possibility for advanced graphical applications, for example, a user can point individual countries in a world map.

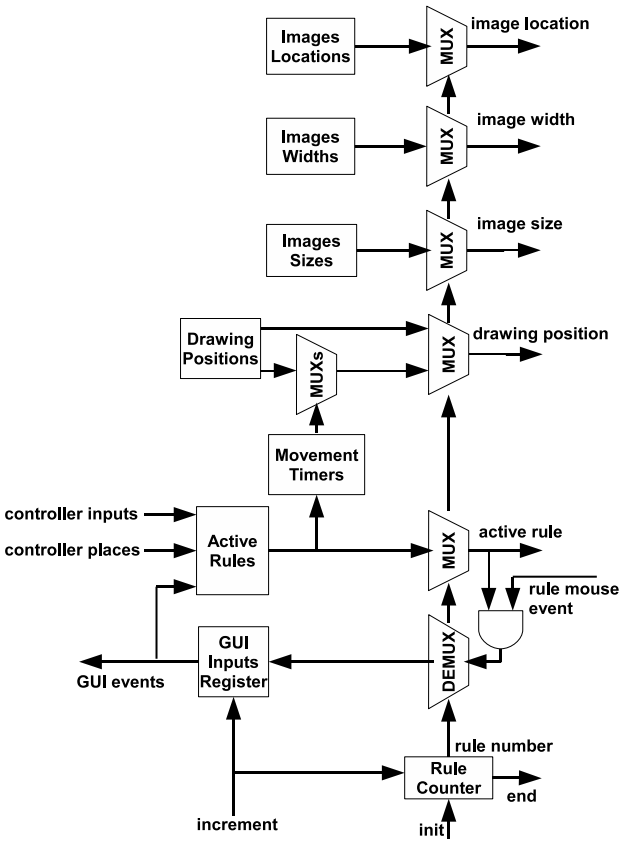

Fig. 4. Architecture of the GUI data module.

The GUI Controller module checks if rules are active and copies the corresponding images to the frame buffer. Fig. 5 displays a simplified IOPT Petri net describing the behaviour of this module. Mouse events are forward to the GUI Data module to check if the user has clicked over any graphical object.

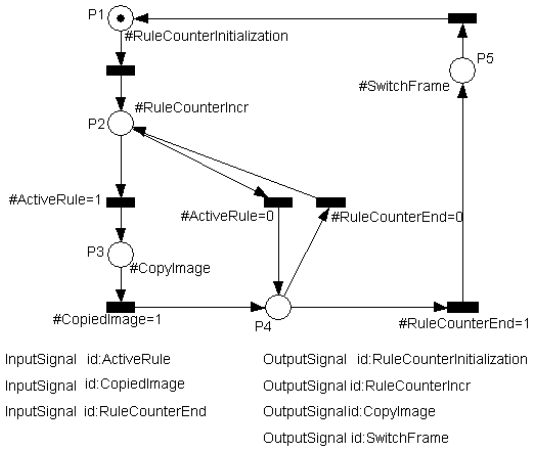

Fig. 5. GUI controller simplified Petri net model.

VI. Prototype Implementation

The implementation board used in this work was an Avnet Spartan-3A DSP 1800A Video Kit [22], including the Xilinx Spartan-3A DSP 1800A Starter Board, the Avnet EXP PS Video Module, and a LCD panel. The Xilinx Spartan-3A DSP 1800A Starter Board contains a 16M x 8 parallel Flash and a 128MB (32M x 32) DDR2 SDRAM. Fig. 10 shows a photo of the demonstration prototype. The Flash memory used to store the images is a JS28F128J3D Intel Embedded Flash Memory and has 128Mbits organized as 16M x 8bits. This memory has capacity to store more than 10 background images plus hundreds of animation images, depending on the dimensions. The LCD used in this prototype has a maximal resolution of 1024x768 pixels, which was the resolution used in this work. Each image pixel uses 8 data bits, representing up to 255 colours plus a special transparency colour.

For development purpose, the prototype was implemented using a standard FPGA development kit, but production embedded systems could be manufactured using much lower cost PCB boards, since many of the development kit sub-systems are not being used in this application. The production boards just require one FPGA or ASIC chip, less than 2Mb of RAM, one EEPROM memory to store images, and the remaining support electronic and passive components.

VII. GUI Generator Tool

The GUI generator tool (GUIGen4FPGA), displayed in fig. 6, was developed in this work and automatically generates VHDL GUI modules (see section V). As can be seen in fig. 1, GUIGen4FPGA receives input from the following files: (1) PNML controller model; (2) ENV animator environment file containing image file path names; and (3) RUL file containing animation rules.

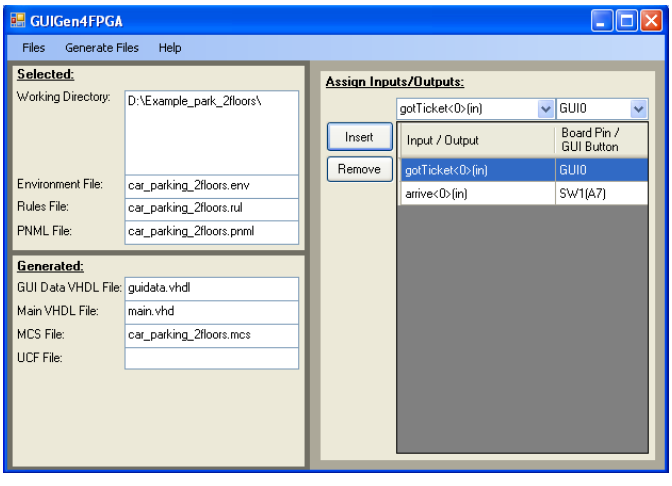

Fig. 6. The user interface of the GUI generator tool.

The tool generates the following output files:

- Images file (MCS file in fig. 6), used to write the images to FLASH memory;
- GUI Static VHDL files, independent of the controller and the GUI. These files implement the GUI Controller and the hardware interface modules (mentioned in section V);
- GUI Dynamic VHDL files dependent on each GUI animation: the GUI Data file and the Main VHDL file (fig. 6). The GUI Data is generated from the information present in the ENV and RUL files, and defines the GUI behaviour. The Main VHDL file is the top level module and encapsulates all other modules;
- Constraints file (UCF file in fig. 6), associates model inputs/outputs to platform pins.

The GUIGen4FPGA tool was developed using Microsoft Visual Studio 2005 and the dotNET Framework 2.0, and the programming language used was C#.NET.

VIII. Test and Validation

This section presents an application example consisting of a car parking lot with two entrances, one exit, two floors and one ramp between the floors. The ramp has only one route, and can be used in two directions, go up or down. Each entrance and exit has a gate and a sensor. The ramp has tree sensors and two gates for car access control. Fig. 7 displays an image of the car parking lot and fig. 8 an image of the ramp.

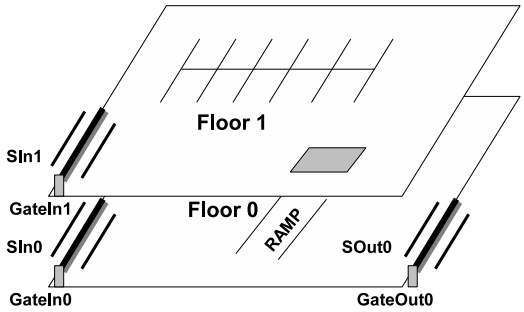

Fig. 7. Two floor car parking lot.

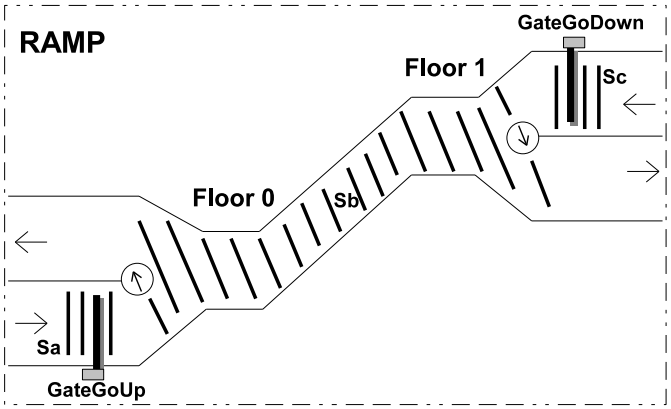

Fig. 8. Two ways ramp.

After collecting the controller requirements and its description through UML Use Cases, the controller was modeled through the IOPT Petri net model presented in fig. 9. After editing the model using SnoopyIOPT, the Animator tool was used to define GUI rules. The GUIGen4FPGA tool was then used to automatically generate the GUI code, and invoked the PNML2VHDL tool to automatically generate the controller code. The last action in the development process was the implementation platform configuration, using Xilinx ISE.

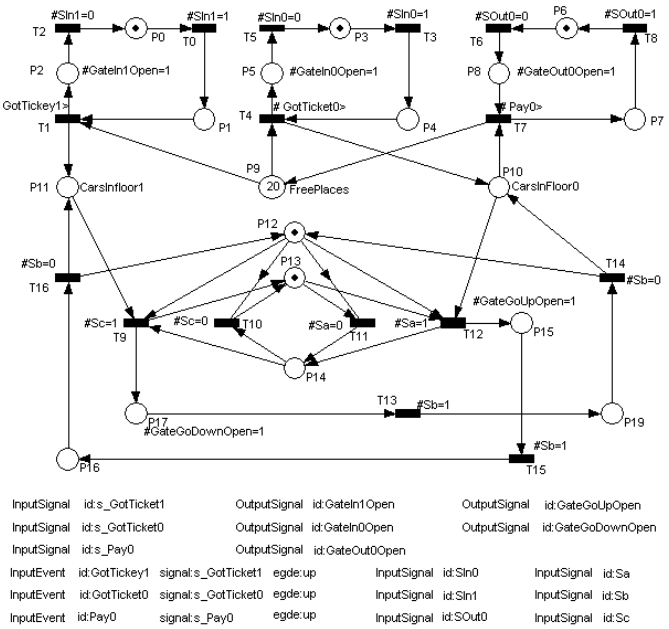

Fig. 9. IOPT Petri net model of example's controller.

Although the example corresponds to a simple automation system controller, it exhibits several interesting characteristics: ability to accommodate physical input and output signals (coming from the plant and associated with car detection, for instance), interaction with the car parking operator through mouse events pointing specific areas in the GUI (allowing opening gates after customer payment, for instance), presentation of the parking lot current status (showing and hiding cars, showing cars moving along a trajectory), as well as system variables (ex: the number of free places inside the parking lot). Additionally, fig. 10 also shows debugging info at the bottom. As fig. 10 shows, the example was then deployed and successfully tested.

The complexity of the graphical user interface that can be generated is limited by the implementation platform resources, where the Flash Memory used to store images could be the most important limit, although it can be easily expanded as necessary, in order to accommodate new requirements.

IX. Conclusion and Future Work

The tools proposed have been implemented and tested on a Spartan 3A development board connected to a high resolution LCD display and a pointer device. Several example application controllers and the respective animated user interfaces were

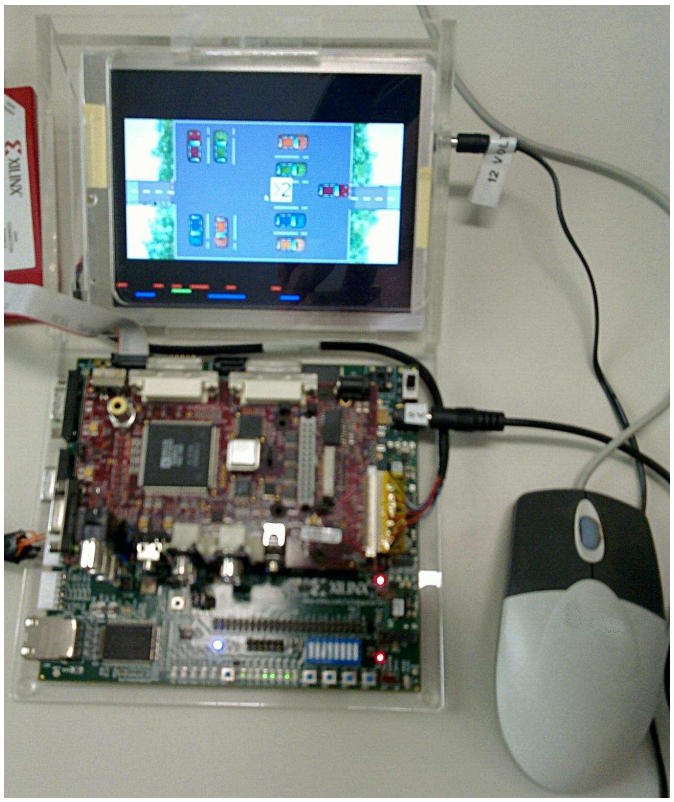

Fig. 10. Prototype photo.

developed using the new toolchain and demonstrated that it is possible to create full applications without the need to write software code or low-level hardware design.

During tests, the new tools exhibited a significant reduction in development time compared to direct VHDL coding, but more importantly, do not require developers with deep hardware knowledge. With the proposed toolchain, embedded system designers only need to receive training in Petri net modeling and may have reduced experience in hardware or software development, thus bringing FPGA-based embedded system design to a much broader audience.

The whole tool framework will be publicly available at institution website for non-commercial use in the near future.

Future work includes porting the tools to other FPGA architectures, support for touch-screens and other pointing devices and the addition of audio and other media types. In the same way as the current version uses rules to display images, the architecture is ready to be expanded with rules to play sound samples, opening the way to create richer multimedia applications.

ACKNOWLEDGMENT

The first author work is supported by a Portuguese FCT grant, ref. SFRH/BD /62171/2009.

REFERENCES

[1] L. Gomes and J. Lourenço, "Rapid Prototyping of Graphical User Interfaces for Petri-Net-Based Controllers," in *IEEE Transactions on Industrial Electronics*, vol. 57, May 2010, pp. 1806 – 1813.

[2] "FORDESIGN project home page," Available at http://www.uninova.pt/fordesign, accessed on March 30, 2011.

[3] 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.

[4] J. Coolahan and N. Roussopoulos, "Timing Requirements for Time-Driven Systems Using Augmented Petri Nets," in *IEEE Transactions on Software Engineering*, September 1983, pp. 603 – 616.

[5] R. Esser, "An Object Oriented Petri Net Language for Embedded System Design," in *STEP '97: Proceedings of the 8th International Workshop on Software Technology and Engineering Practice (including CASE '97)*. Washington, DC, USA: IEEE Computer Society, 1997, p. 216.

[6] S. Chachkov and D. Buchs, "From an Abstract Object-Oriented Model to a Ready-to-Use Embedded System Controller," in *Rapid System Prototyping, 12th International Workshop on, 2001*, Monterey, CA, June 2001, pp. 142 – 148.

[7] K. Jensen, *Coloured Petri Nets. Basic Concepts, Analysis Methods and Pratical Use - Volume 1 Basic Concepts*. Berlin. Germany.: Springer-Verlag., 1997.

[8] M. Westergaard and K. B. Lassen, "The BRITNeY Suite Animation Tool," in *S. Donatelli and P.S. Thiagarajan (Eds.): ICATPN 2006*. Springer-Verlag Berlin, Heidelberg, 2006, pp. 431–440.

[9] J. Jorgensen, "Addressing Problem Frame Concerns via Coloured Petri Nets and Graphical Animation," in *2006 international Workshop on Advances and Applications of Problem Frames*, May 2006, pp. 49–58.

[10] H. Ehrig, C. Ermel, and G. Taentzer, "Simulation and Animation of Visual Models of Embedded Systems," in *7th International Workshop on 'Embedded Systems Modeling Technology, and Applications*, June 2006, pp. 11–20.

[11] P. Nascimento, P. Maciel, M. Lima, R. Santana, and A. Filho, "A Partial Reconfigurable Architecture for Controllers based on Petri Nets," in *17th Symposium on integrated Circuits and System Design*, September 2004, pp. 16 – 21.

[12] LabVIEW FPGA, "NI LabVIEW FPGA Module," Available at http://www.ni.com/fpga/, accessed March 30, 2011.

[13] J. Billington, S. Christensen, K. van Hee, E. Kindler, O. Kummer, L. Petrucci, R. Post, C. Stehno, and M. Weber, "The Petri Net Markup Language: Concepts, Technology, and Tools," in *Proceeding of the 24th International Conference on Application and Theory of Petri Nets*, ser. LNCS, W. van der Aalst and E. Best, Eds., vol. 2679. Eindhoven, Holland: Springer-Verlag, June 2003, pp. 483–505.

[14] W. Reisig, *Petri nets: an introduction*. New York, NY, USA: Springer-Verlag New York, Inc., 1985.

[15] R. David and H. Alla, *Petri Nets and Grafcet: Tools for Modelling Discrete Event Systems*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1992.

[16] M. Silva, *Las Redes de Petri: en la Automática y la Informática*. Ed. AC., 1985.

[17] R. C. K. Venkatesh, MengChu Zhou, "Comparing ladder logic diagrams and Petri nets for sequence controller design through a discrete manufacturing system," *IEEE Transactions on Industrial Electronics*, vol. 41, no. 6, pp. 611–619, 1994.

[18] G. Frey and M. Minas, "Editing, Visualizing, and Implementing Signal Interpreted Petri Nets," in *Proceedings of the AWPN 2000*, Koblenz, October 2000, pp. 57–62.

[19] Hanisch, H.-M. and Lüder, A., "A signal extension for Petri nets and its use in controller," *Fundamenta Informaticae*, vol. 41, no. 4, pp. 415–431, 2000.

[20] L. Gomes, A. Costa, J. P. Barros, R. Pais, T. Rodrigues, and R. Ferreira, "Petri Net based Building Automation and Monitoring System," in *5th IEEE International Conference on Industrial Informatics*, Vienna, 23-25 June 2007, pp. 57–62.

[21] "PNML Framework's site," Available at http://pnml.lip6.fr, accessed on March 30, 2011.

[22] "Avnet Spartan-3A DSP 1800A Video Kit," Available at www.em.avnet.com, accessed on March 30, 2011.