On the Evolution of Remote Laboratories for Prototyping Digital Electronic Systems

Leandro Soares Indrusiak, *Member, IEEE*, Manfred Glesner, *Fellow, IEEE*, and Ricardo Reis, *Senior Member, IEEE*

***Abstract*—The design of digital electronic systems for industrial applications can benefit in many ways from the prototyping capabilities of field-programmable gate array (FPGA) platforms. This paper presents three evolutionary releases of an FPGA-based remote laboratory and discusses the didactical and technical motivations behind each release, aiming to reduce the overhead of setting up and operate a laboratory environment where designers and students can use FPGA prototyping to validate their designs. To achieve that, a number of abstraction layers were introduced, allowing configuration and data processing in remote FPGA platforms, as well as integrating such platforms within a simulation environment. The proposed approach supported a number of projects where groups of designers and students could specify, refine, and prototype electronic systems using a pool of remotely available FPGA platforms.**

***Index Terms*—Design automation, field-programmable gate arrays (FPGAs), integrated circuit design, remote laboratories.**

I. Introduction

THE DESIGN of electronic systems for industrial applications has become increasingly complex. The advances on silicon technologies have allowed the periodic release of new generations of integrated electronic components with higher logic density than their predecessors, which, in turn, enabled the design of more sophisticated industrial systems that were not feasible before. The increasing complexity of the design process has been tackled by increasing the sophistication of the computer-aided design (CAD) automation techniques that support the specification, verification, and manufacture of such systems.

Recent trends on design automation point to a system-level approach, which uses a single model to specify an electronic system. Such model includes the digital hardware platform (custom processor, digital signal processing (DSP), memory, interconnects, etc.), the software subsystems (operating system, middleware, application programming interfaces, etc.), and the deployment scenario (for instance, a model of the moving parts of an ac motor in a control system or a model of lighting patterns in a computer vision application) and allows for the evaluation of different implementation alternatives. Several research groups addressed such approach under different points of view, such as the hardware/software codesign [1] and the platform-based design [2] methodologies.

The design of industrial electronic systems benefits greatly from such approaches, as they allow the full exploration of the hardware/software tradeoff regarding performance and costs. For instance, most companies are more likely to add value to an electronic product by customizing its software components and using off-the-shelf electronic components, because the manufacture costs of custom hardware are extremely high. On the other hand, the performance that was provided by off-the-shelf microcontrollers and DSP processors is often not enough for state-of-the-art applications (such as the adaptive control system based on a self-tuning regulator shown in [3]), so the design of a custom hardware platform is needed.

To address such wide spectrum of hardware/software solutions, it is necessary to define a complex design flow that is able to integrate multiple abstraction layers—for instance, discrete-event systems, state machines, concurrent processes, dynamic systems, etc.—and handle the inherent complexity of each of them within design entry, simulation, validation, and synthesis tools. The task of preparing engineers that are able to handle such design flow is not less complex. The high degree of automation in such flows requires educational and training activities addressing the fundamental engineering concepts, the usage of the tools, and the development of prototypes to validate successive design refinements that are performed by the tools.

This paper presents three evolutionary releases of a remote prototyping laboratory supporting the design of electronic systems for industrial applications. This approach has the potential to improve the design of educational and training activities, because it allows their close integration with prototyping activities using the same design environment. The major goals of the presented approach are given as follows: 1) to support the successive prototyping of system modules—both hardware and software—into platforms that resemble the target implementation and 2) to allow the validation of such prototypes under realistic operation conditions by simulating or emulating the deployment scenario.

In order to better justify the need for prototyping—and thus prototyping using remote laboratories—we describe in Section II the major steps of the design flow for integrated digital electronic systems. Then, we present in Section III the state of the art in remote laboratories and the successive attempts we did over the last seven years to extend it, aiming to properly integrate and abstract remote laboratories within the electronic systems design flow. In Section IV, we present

Manuscript received March 30, 2007; revised August 16, 2007.

L. S. Indrusiak and M. Glesner are with the Microelectronic Systems Institute, Darmstadt University of Technology, 64283 Darmstadt, Germany (e-mail: indrusiak@mes.tu-darmstadt.de).

R. Reis is with Informatics Institute, Federal University of Rio Grande do Sul, 91501-970 Porto Alegre, Brazil.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIE.2007.907010

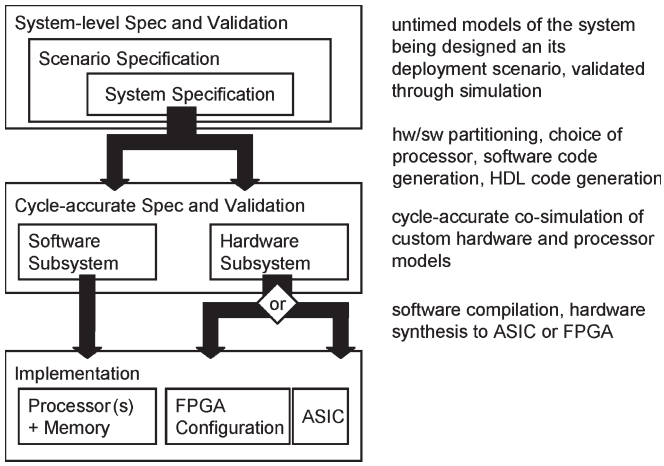


Fig. 1. Simplified representation of the integrated electronic systems design flow.

the advantages of our approach on the system-level design of industrial electronic systems, as well as its impact on related education and training activities.

II. Integrated Electronic Systems Design Flow

Over the last 30 years, the specification of integrated digital electronic systems became increasingly abstract, from the direct specification of the circuit layout to logic gate netlists, then to cycle-accurate models that are described using hardware description languages (HDLs), up to untimed system-level models that do not discriminate hardware and software subsystems. Nowadays, most design flows still use all those specification styles, but only the most abstract of them are directly manipulated by designers, and all the others are automatically generated by design automation tools.

Fig. 1 shows a simplified view of the design flow that is considered in this paper. The most abstract styles of specification (represented as boxes) appear on top, and model transformation and synthesis techniques (represented as arrows) successively convert abstract specification models into the ones closer to the final implementation that is shown on the bottom.

The top level specification describes the system as a whole and does not yet discriminate which modules are going to be implemented as hardware or software. It should also include a model of the final system's deployment scenario—the industrial setup on which the final system will be embedded—so that the functionality of the system can be validated at early stages of the design flow. Such specification models are validated by simulation, so that design alternatives can be compared using application-specific figures of merit (for instance, bit error rate versus signal-to-noise ratio for wireless communication systems, or the number of correctly recognized features per frame in a computer vision application).

The next specification style, as depicted at level 2 in Fig. 1, already discriminates hardware and software subsystems, and is cycle accurate. This means that the double-headed arrow leading to it comprehends the following: 1) a hardware-software partitioning; 2) a choice of one or more microprocessors that execute the software subsystems; 3) software code generation; and 4) (semi-) automatic implementation of the hardware subsystems. In most practical cases, the hardware subsystems are modeled using HDLs such as Very High Speed Integrated Circuit HDL (VHDL) or Verilog, and the microprocessors are abstracted by instruction set simulators or also by HDL-based models.

Cycle-accurate models of the hardware subsystems are synthesized into logic netlists, which are in turn input to technology mapping, placement, and routing tools, so that the final implementation is produced automatically with little assistance by the designer. The final implementation can be an application-specific integrated circuit or a configuration for a field-programmable gate array (FPGA). A given logic netlist can be mapped to either of the implementation alternatives, so the upper levels of the design flow is common for both of them.

It must be clear at this point that the design flow of integrated electronic systems spans several abstraction layers and usually relies on simulation to validate the system models on each of those layers. As it is not always feasible to transform the complete system specification from one abstraction layer to the lower one, it is often necessary to cosimulate subsystems that are described using different specification styles at different levels of abstraction. For instance, while the model of a wireless receiver can be described in HDL at the cycle-accurate level, the transmitter and the wireless channel model may still be described at very abstract untimed models, and to validate the whole communication system, one must simulate them all together.

A problem arises when the final implementation of a given subsystem is achieved: how to validate the functionality of that subsystem together with other subsystems that are still modeled using more abstract styles, or even with the model of the deployment scenario. One way is to fabricate the actual hardware components, connect them to a printed circuit board, and build a prototype. However, if the system comprehends custom hardware components, this approach is unfeasible because of the fabrication time and costs: If a problem is found in the fabricated circuit, the costs of redesign and the new fabrication are usually too high to be recovered, and the product will probably miss its time-to-market window.

This problem has been solved by exploring the capabilities of FPGAs. As explained earlier, FPGAs can be configured to implement logic netlists that are generated out of cycle-accurate models of custom hardware subsystems or microprocessor soft cores. Thus, they can be easily customized to be used as prototypes for such subsystems, allowing the validation of the final implementation of a system without requiring the fabrication of custom hardware. Many state-of-the-art industrial electronic systems are known to rely on FPGA prototyping, such as [3] and [4]. Dubey *et al.* [5] reviewed the application of FPGAs and other programmable logic devices in power electronics and motion control not only as prototyping platforms but also as final implementation platforms. Such acceptance is due to the higher performance of FPGA implementations when compared with solutions based on microcontroller or DSP processors, or the lower cost compared with the fabrication of custom hardware.

In the next sections, we explore FPGA platforms as remote prototyping laboratories supporting the design of integrated electronic systems for industrial applications.

III. Remote Prototyping Laboratories

A. Related Work

The development of remote laboratories flourished during the last ten years with the popularization of the Internet. Applications of remote laboratories were reported on microelectronics [6], real-time systems [7], control systems [8], chemistry [9], and physics [10], among many others. Most of them were aimed at education and training activities, and were developed by university research and teaching staff members. Some of them were supported by commercial tools, such as LabView [11] or Microsoft NetMeeting [8].

Being strongly influenced by the Internet infrastructure and the World Wide Web, most of the reported remote laboratories inherit one or more of its features.

1) Client–server architecture, where a remote laboratory is a server and the users' access is through the client applications (usually a web browser).
2) The remote laboratory is identified and located by its Internet Protocol (IP) address or a Domain Name System (DNS)-resolved server name.
3) User interface materialized as hypertext and/or forms within a web browser.
4) The interaction between client and server is done through stateless protocols, such as Hypertext Transfer Protocol: All the information that a server receives from the client is the set of parameters that are passed within a service request.

Relying on such features usually simplifies the development of remote laboratories, but on the other hand, it may limit its usability or require additional developments.

1) Identification by IP address or DNS-resolved server name makes it hard to assign multiple laboratory units to multiple users (users must choose a specific laboratory to access, even if there are many available).
2) The interaction between user and laboratory should be partitioned in many short-lived transactions.
3) Interactivity among laboratory users is limited, because Internet protocols completely isolate different service requests.

A specific analysis of remote laboratories addressing electronics shows that most of the existing work addresses the experimentation with electronic devices and not complete systems. The pioneer work within the MIT Microelectronics WebLab [6] concentrated on measurements that are performed on individual devices, such as transistors. A similar approach was presented by Salaverría *et al.* [12] but for a wider range of analog electronic devices. On digital electronics, the scenario is not different, even though a number of remote laboratories based on coarse grain components such as microprocessors [13] or programmable logic [14] were reported. In all cases, a number of significant shortcomings prevented the analysis of complete electronic systems and their deployment environment, and thus restricted the usage of such laboratories within the area of industrial electronics. Limitations include the lack of support for realistic test benches and application scenarios, as well as the difficulties to integrate remote laboratories with additional resources that are used in design and education activities (such as simulators or CAD tools). In the next section, we present an evolutionary approach that aims to overcome such limitations in the case of FPGA-based remote laboratories.

B. Remotely Accessing FPGAs

In the last seven years, our group addressed the problem of introducing FPGA-based remote prototyping laboratories into the design flow of industrial electronic systems. The fundamental approach was to increasingly abstract the features of a remote laboratory that hinder the experience of designers, students, and trainees: handling network connections, managing file transfers, and setting up graphical interfaces to analyze the results of experiments, for instance. Over the years, three stable releases of the remote laboratory infrastructure can be identified, with each one built on top of its predecessor and with increased abstraction features.

Our *first release* regarded the encapsulation of an FPGA board, so that it can be remotely configured. The usage scenario of this approach is given here.

1) The designer uses locally installed automation tools to specify a given hardware subsystem (as in levels 1 and 2 of the design flow that was depicted in Fig. 1), synthesize it into a netlist, and generate an FPGA configuration [as in Fig. 1 (level 3, center)]; the FPGA boards are connected to a computer that runs a back-end server, which 1) allows connections through Transmission Control Protocol (TCP)/IP sockets in order to receive the configuration file from the client side and 2) is able to load the configuration file into the FPGA (which is connected to the computer via a Universal Serial Bus or parallel port) by running vendor-specific configuration tools in batch mode.
2) The designer uploads the configuration to the remote FPGA using a client application, which opens a TCP/IP connection to the remote FPGA back-end server (either the designer or the tool must know the IP address of the computer running the server as well as the TCP/IP port of the server).
3) The designer can observe the results by visually inspecting special components of the FPGA board or attached displays through a video streaming application, which runs independently of the back-end server (designer must have a client for the video streaming application and know the IP address of the video server, as well as its TCP/IP port).

Fig. 2 depicts the setup of one of the remotely accessible FPGA boards that we developed using this first approach. The laptop on the left side is the client, where the FPGA configuration was designed and uploaded to the back-end server, which is running on the laptop on the right side. The server laptop is

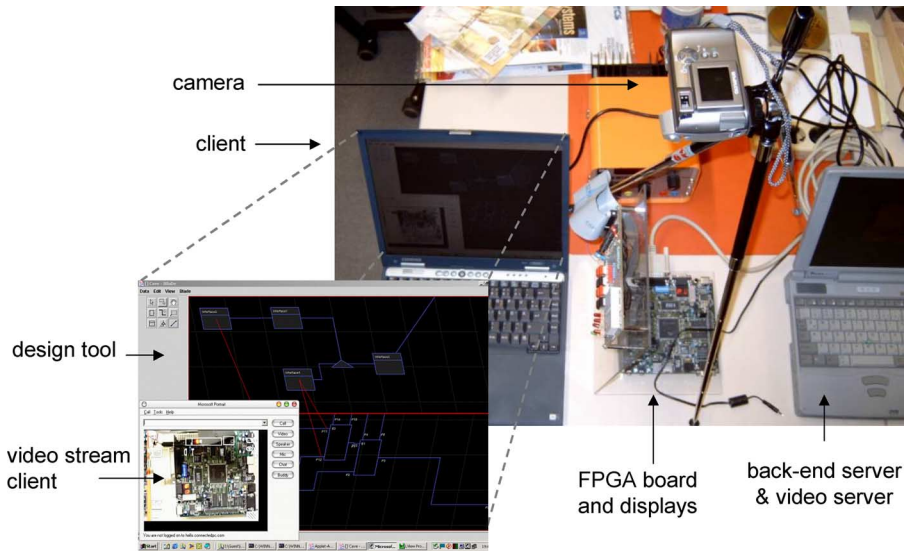


Fig. 2. Remote prototyping laboratory setup.

connected to a Xilinx XCV800 FPGA board (in the middle) and to a camera. It configures the FPGA every time that a new configuration file is received, and it continuously streams the video information captured by the camera. The lower left side of the figure also shows a detail of the screen of the client computer to highlight the interface that was used by the designer to interact with the remote prototyping laboratory.

Our *second release* aimed to increase the abstraction of the FPGA prototyping platforms by using an object-oriented approach, so that the FPGA is "encapsulated" by a proxy that can control reconfiguration and data processing through method calls. Such proxy must be powerful enough to do the following:

1) handle the distribution of FPGA platforms over the network, so that clients can locally interact with it without having any knowledge of the actual network address of the back-end server;
2) abstract the reconfiguration process, so that it can be done through a method call that receives an FPGA configuration as argument;
3) abstract the data processing, so that a client can send data to be processed by the FPGA through a method call that returns the processed data.

To decouple the client–server connection (which was previously point-to-point using sockets) and abstract the network address of the remote prototyping platforms, we decided for a middleware-based approach using Jini [15]. This granted us the possibility of using registry and lookup servers that allow a given client to dynamically query for available prototyping platforms over the network. This meant that, once a designer is ready to prototype her design, she can connect to a lookup server and check whether a suitable prototyping platform is available. The benefits of such approach for supporting multisite design and learning activities will be further reviewed on the next section.

Aside from the network address, the second release also abstracted the reconfiguration process and the data processing on the FPGA platform. The reconfiguration process was already abstracted on the socket-based approach, so it was easily ported to the new approach by simply encapsulating the batch execution of the vendor-specific configuration tool as a remotely accessible method on the back-end server. The data processing capabilities, on the other hand, were not supported in the previous version (the results could only be inspected through the video stream). To allow data processing on the FPGA, we had to define a memory space within the FPGA board (either an external memory on the circuit board or an internal BlockRAM) that could be read and written by both the back-end server and the hardware system, which is implemented as an FPGA configuration. The data processing scenario is then divided in three parts: 1) The client sends data to be processed by the system, which is presently configured in the FPGA, by calling a method of the back end's proxy; the back end, in turn, receives the data, writes it into a predefined memory area, and sets a "ready" flag to the FPGA. 2) The FPGA, which monitors the flag, starts reading, processing the data, and writing results on another predefined area of the memory, and then sets another flag when ready. 3) The back end reads the processed data and sends it back to the client via its proxy.

A number of issues had to be solved to bring such scenario into practice. First, we added the data type abstraction in order to support data processing based on different data types, such as integers and floating points with different precisions (such abstraction defined how each value passed on a method call would be mapped to a bit array to be stored on the FPGA memory). Second, it was necessary to address the data reading and writing that were done by the system, which is configured on the FPGA. For instance, if the FPGA is configured to perform an encryption algorithm, the designer of the configuration must know how to map the input signals of the algorithm implementation to the signals controlling the memory access. To solve this problem, we implemented in VHDL a reusable memory interface that follows our predefined memory organization scheme. By reusing such interface together with

the HDL code that describes the system configuration (in the aforementioned example, the encryption algorithm), designers can easily create configurations that can be used within this approach.

Both the client software and the back-end server had to be reimplemented to incorporate the Jini protocols for communicating with registry and lookup servers, as well as the data processing abstractions. Further details on the implementation of this release were published in [16], and the same approach was also used to implement networks of reconfigurable embedded systems [17].

The *third release* aimed to further abstract the data processing capabilities of the remote prototyping platforms. While allowing data processing on the FPGA, the second release required sending the data to be processed as a file to the back-end server within a method call. This allows designers to test the system that was prototyped on the FPGA by sending test data to be processed and analyzing the results. This process, however, could be tedious and error prone due to the need to package and send the data as a file (particularly, if the system had feedback loops, so the test data had to be repeatedly generated out of the results of the previous cycle). In order to have a better validation environment, we decided to support the integration of the remote prototyping laboratories within a modeling and simulation environment.

The chosen environment was Ptolemy II [18], which is a fully featured open source framework for modeling and simulation of concurrent systems that were developed by the University of California, Berkeley. The core of its simulation models are actors, which communicate with other actors by receiving and sending data through their interface, which is a set of ports. Tokens encapsulate the data that were sent through ports, and directors govern the interaction between actors. To take advantage of the features that are present in Ptolemy II, the encapsulation of the remote FPGA platform into a new Ptolemy II actor is necessary. By doing so, the remote prototype becomes an active part of the simulation environment in a "hardware-in-the-loop" fashion. The overall organization of this release is shown in Fig. 3.

The encapsulation of the remote FPGA as an actor was relatively simple, as it reused the data processing capabilities of the previous release. For every token that is received at the input ports of the FPGA actor, the following are done: 1) data are unpacked; 2) the data processing routine is executed; and 3) the results are packed in a new token, which is sent to the output port. On the implementation, a few changes were needed to support the packing and unpacking of the data as tokens, as well as some performance improvements at the back-end side in order not to degrade simulation speed.

A snapshot of Ptolemy II depicting an actor encapsulating a remote FPGA is shown in Fig. 4. By setting the properties of such actor, designers can choose a configuration file to be uploaded and define the criteria for looking up for a suitable prototyping platform (such as FPGA vendor and family, and gate count) Thus, Ptolemy II becomes the main interface to modeling, simulation, and prototyping.

In Table I, a comparison of the advantages and disadvantages of all three releases is given.

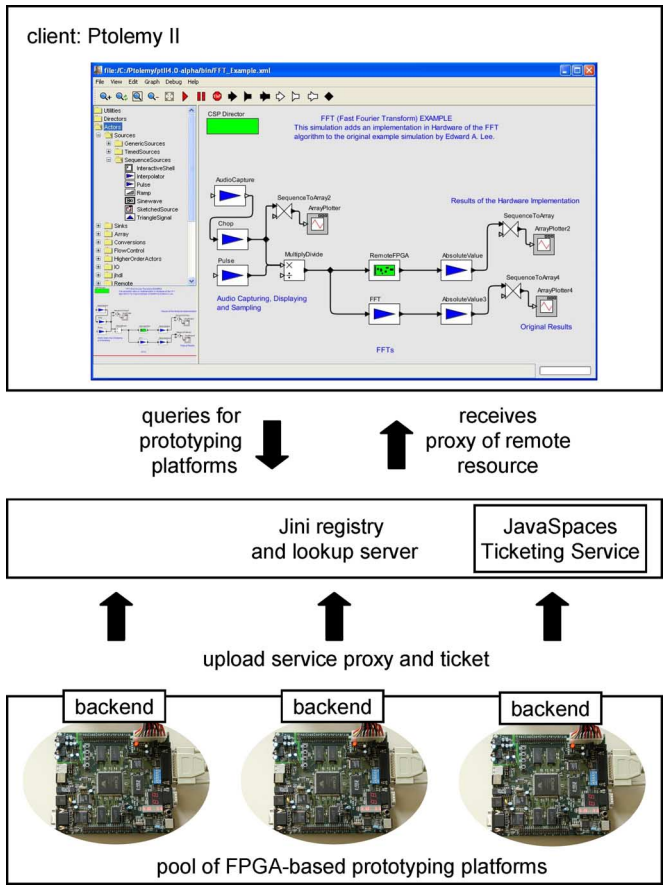


Fig. 3. Integration of remote prototyping platforms and Ptolemy II modeling and simulation environment.

IV. Obtained Results and Experiences

A. Concept Evolution

All three releases that were described in the previous section were used within design and educational activities. The limitations of each release were the actual motivations for the further development of the remote prototyping environment. For instance, the first release allowed visual inspection of the results only, limiting the types of applications that could be used and making the design of the FPGA configuration more complex (as it should include also the test generation and the control of the peripherals providing the visual output, such as light-emitting diodes and displays). The second release solved that problem by allowing data processing on the FPGA via method calls, so that the test data could be generated and analyzed at the client side. While cumbersome because of its file-based approach, this release hosted more complex projects that could start analyzing the tradeoffs between pure software and pure hardware implementations of a given system (transition from level 1 to level 2 in Fig. 1). Table II presents the results of such tradeoff analysis for an implementation of the Data Encryption Standard cryptography algorithm, which was performed as a student project in our university. The software-only implementation was done in Java and was profiled through the execution of the compiled bytecode by a virtual machine running on top of two different processors, while the hardware-only implementation

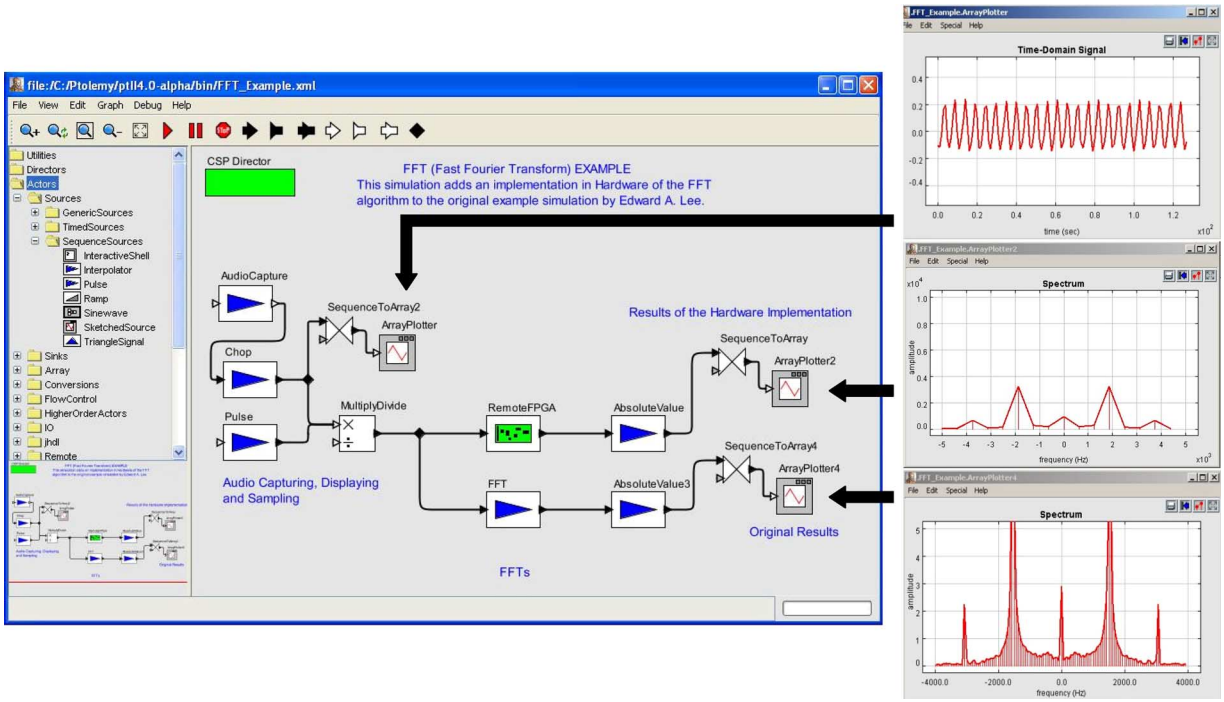


Fig. 4. Ptolemy II GUI displaying simulation results and data processed at the remote prototyping laboratory.

TABLE I
COMPARATIVE ANALYSIS OF REMOTE FPGA ACCESS STRATEGIES

Strategy	Advantages	Disadvantages
Socket-based	- supports remote reconfiguration of FPGA-based prototyping platforms	- clients must know the network address of backend servers - limitations on the evaluation of results (must use visual features of the FPGA board or peripherals)
Jini-based	- same as Socket-based, plus: - allows dynamic allocation of prototyping platforms to clients - adding and removing prototyping platforms is transparent to the client - supports the verification of the designed system through the processing of test data	- requires a middleware infrastructure (lookup server, proxies, remote method invocation, etc.) - designed system must include the provided memory interface
Ptolemy-based	- same as Jini-based, plus: - complete integration with modeling and simulation environment, simplifying generation and analysis of test data	- same as Jini-based

TABLE II
TRADEOFF ANALYSIS OF A DES IMPLEMENTATION

Processor	Max. clock frequency (MHz)	Max. data rate (Mbits/s)
Sun JVM over Intel Pentium MMX	133	0.162
Sun JVM over AMD Athlon XP-1700	1467	1.295
Xilinx XCV800 FPGA	27.23	108.93

was done in VHDL and configured within the Xilinx XCV800 remote prototyping platform that was shown in Fig. 2.

In most of the state-of-the-art designs, however, such tradeoff analysis between pure hardware and pure software solutions is insufficient, because the optimal solution usually lies in between, as a mix of hardware and software subsystems. We tried to do such analysis using the second release, but the fact that we could not easily simulate/emulate the hardware and software subsystems together forced us to introduce inaccuracies and made the setup task very difficult. This was the major motivation for the third release, which allowed us to successively refine the system implementation while still being able to validate it under realistic execution scenarios within Ptolemy II. Still, the Jini-based solution that was introduced on the second release was kept, so the support for dynamic matching of multiple users to multiple remote prototyping platforms could also be used. Those were the most appreciated features of the system during the design and learning activities that we performed, so we describe the experiences and feedback that we obtained on the next sections.

B. *Design by Successive Refinement*

The efficient exploration of different hardware/software partitions is still an open problem in the design of integrated electronic systems. As mentioned previously, one can benefit from cosimulation capabilities in order to evaluate alternatives for the partition as well as the communication among hardware and software subsystems. In the third release that was presented on Section III, it is possible to use the encapsulation of FPGA prototyping platforms to support the evaluation of such

alternatives. We explored such possibilities in a number of case studies and student projects.

The first step is to create a system-level model, as shown in Fig. 1 (upper layer). Such model can be profiled through simulation in order to have a rough idea of the complexity and the performance requirements of each subsystem. In our third release, this is done within Ptolemy II. The subsystems with stringent performance requirements are the usual candidates to be implemented in custom hardware, but other aspects may also affect this decision (power consumption, reuse of legacy subsystems, etc.) Once selected to be implemented in hardware, the subsystem is specified using HDL (either manually or using code generators when available) and simulated in a cycle-accurate manner to verify its functionality and performance, as shown in layer 2 of Fig. 1. Finally, the HDL code is synthesized, and an FPGA configuration is generated. At this point, the approach that is presented here can play a critical role: Once the configuration is generated, it can be uploaded to a remote prototyping platform and executed within a system-level simulation model. This means that one can create the system-level model, profile it, choose one of its components to be implemented in hardware, generate the configuration for an FPGA, and then substitute the original component in the model by the encapsulated FPGA in order to verify if the designed hardware implements all the desired functionality and performance.

Fig. 4 depicts such a case within Ptolemy II. On the left side of the system model, a number of actors model the sampling of an audio signal coming from a microphone. At the center, there are two actors performing a fast Fourier Transform (FFT) calculation—one in software and one in hardware—with the latter being prototyped on a remote FPGA. On the right side, there are actors supporting the result analysis. In this example, the system was completely modeled using software components, and once a hardware implementation of the FFT was obtained, it was uploaded to a remote prototyping platform, so that the functionality of both can be compared. Had the example entail other actors to be implemented in the hardware, a successive prototyping could be done as long as there are further prototyping platforms that are available at the lookup server. The described example, which outputs the spectral information of an audio signal through the application of the FFT, was implemented within a student project. A video showing the joint execution of such Ptolemy II simulation model and the hardware implementation of the FFT prototyped in a remote FPGA platform can be downloaded from [19].

C. Handling Multiple Users

By introducing a lookup-based system, remote FPGA platforms can be dynamically located when needed. Every request for a prototyping laboratory is not sent to the backed servers encapsulating the FPGA boards but to the Jini lookup server introduced on our second release, which in turn will return a proxy to the first suitable board that is available. This allows for a more efficient usage of the FPGA resources, particularly, when they must be shared by groups of designers or students: All FPGA platforms can be made available at any time to every student/designer, so every one can be served as long as there is at least one board available. Since the network address of the back end of each FPGA device is transparent to the user, devices can be safely removed from the network when they are not in use (no user will get a "error 404: device unavailable" message, as the binding between the user and the board is dynamically resolved). Devices can also be added dynamically, without any notification to the users. The newly added devices will be available immediately after their proxy is uploaded to the lookup server.

In order to guarantee that the prototyping resources are properly managed under multiuser access, we implemented a ticketing service that ensures mutual exclusion for each board (otherwise, one user could reconfigure the device, while another is trying to process data, or other similar conflicts). It uses the JavaSpaces storage facilities within Jini, so it can be accessed via the same lookup server as the prototyping platforms themselves.

D. Advantages and Disadvantages on Design and Education

The feedback from the laboratory users—designers, instructors, students, and trainees—can also provide valuable hints about its advantages and disadvantages over conventional hands-on laboratories. On the general analysis, our users provided similar feedback as those quoted in most of the publications that were referenced in Section III: All appreciated the lack of complex setup and tricky cabling, the attractive cost–benefit ratio, and the freedom to access the laboratories anytime and anywhere. Because of the fact that each of the laboratory releases was deployed to different classes of students over the last seven years, it is not possible to have a direct comparison between them. A number of research assistants and designers doing project work at the university had the opportunity to use two or three releases though. The statements listed here are a combination of the feedback received from all of them.

1) The overhead that are required by the first release regarding FPGA design (test generators, display controllers, etc.) overweighed the advantages that are brought by the remote access of the laboratory (students/designers had to do too much additional work in order to have visible results from the remote board).
2) While providing the best user experience because of its simplicity of test data generation and analysis, the third release can hide the remote prototyping board too much, to the point that the student has the impression that he is dealing with a simulator. Thus, even if it is unnecessary to the proper execution of the experiment, the use of a video stream showing the board and its displays can contribute to the "credibility" of the whole setup.
3) Due to the batch mode operation, the second release is more suitable to the integration with other remote laboratories or to workflow systems in general. Ongoing work addresses the encapsulation of that release as a component on a network of remote laboratories that are integrated using Web services.

4) The maintenance of such a remote laboratory is not negligible, as many of the prototyping boards are expensive and often used for other purposes. The middleware approach that was introduced on the second release is very useful in such conditions, as it can hide from the users the fact that a given board was removed temporarily from the system to serve other purposes.

V. Conclusion

This paper described the evolution of an approach for simplifying the setup of remote prototyping laboratories supporting the design of industrial electronic systems. Initially, the approach regarded only the possibility of remotely prototyping electronic systems using FPGAs. Improvements were put in place, so that the validation of the prototyped systems could be done from within feature-rich modeling and simulation environments, allowing for complex testing scenarios and comparative analysis. The possibility of jointly executing simulation models and the prototyping platforms in a "hardware-in-the-loop" fashion allowed for the successive refinement of system-level models toward optimal hardware/software partitions.

The proposed approach was used as a design and prototyping environment in a number of industrial and educational projects, in areas such as wireless communications, cryptography, automation, and multimedia. However, unlike most related work, our approach for remote laboratories is not only tailored for measurements or predefined experiments. It relies on the flexibility of FPGAs to allow designers and students to completely configure the remote laboratory as a prototype of the system that they are designing. Furthermore, such prototype is rendered completely interactive, so its execution can be controlled and monitored, which is ideal for both design and education purposes. The fact that our approach simplified the integration of remote prototyping laboratories and design flows also increased its educational potential, because training and education activities in the area of electrical, electronic, and computer engineering tend to strongly rely on hands-on experience with design tools and flows. Finally, its Jini-based approach for the middleware provided a simple yet efficient way to assign remote prototyping platforms (from a limited pool) to a group of students or designers.

Acknowledgment

The authors would like to thank Prof. J. Becker, F. Lubitz, and D. Jimenez for their valuable contributions to the concepts, methods, and implementation work leading to this paper.

References

[1] A. Kalavade and E. A. Lee, "A hardware-software codesign methodology for DSP applications," *IEEE Des. Test Comput.*, vol. 10, no. 3, pp. 16–28, Sep. 1993.

[2] A. L. Sangiovanni-Vincentelli and G. Martin, "Platform-based design and software design methodology for embedded systems," *IEEE Des. Test Comput.*, vol. 18, no. 6, pp. 23–33, Nov./Dec. 2001.

[3] Z. Salcic, J. Cao, and S. K. Nguang, "A floating-point FPGA-based self-tuning regulator," *IEEE Trans. Ind. Electron.*, vol. 53, no. 2, pp. 693–704, Apr. 2006.

[4] C. F. Juang and J. S. Chen, "Water bath temperature control by a recurrent fuzzy controller and its FPGA implementation," *IEEE Trans. Ind. Electron.*, vol. 53, no. 3, pp. 941–949, Jun. 2006.

[5] R. Dubey, P. Agarwal, and M. K. Vasantha, "Programmable logic devices for motion control—A review," *IEEE Trans. Ind. Electron.*, vol. 54, no. 1, pp. 559–566, Feb. 2007.

[6] J. A. del Alamo *et al.*, "MIT microelectronics WebLab," in *Lab on the Web*, T. A. Fjeldly and M. S. Shur, Eds. Hoboken, NJ: Wiley, 2003, pp. 49–87.

[7] A. Rasche *et al.*, "Real-time robotics and process control experiments in the distributed control lab," *Proc. Inst. Electr. Eng.—Software*, vol. 152, no. 5, pp. 229–235, Oct. 2005.

[8] N. Swamy, O. Kuljaca, and F. L. Lewis, "Internet-based educational control systems lab using NetMeeting," *IEEE Trans. Educ.*, vol. 45, no. 2, pp. 145–151, May 2002.

[9] F. Senese and C. Bender, "The Internet chemistry set: Web-based remote laboratories for distance education in chemistry," in *Proc. World Conf. Educ. Multimedia, Hypermedia Telecommun.*, Montreal, PQ, Canada, 2000, pp. 1731–1733.

[10] S. T. Park *et al.*, "Web-based nuclear physics laboratory," in *Proc. 3rd Int. Conf. m-ICTE*, 2005, pp. 1165–1169.

[11] R. Berntzen, J. O. Strandman, T. A. Fjeldly, and M. S. Shur, "Advanced solutions for performing real experiments over the Internet," in *Proc. Int. Conf. Eng. Educ.*, Oslo, Norway, 2001, pp. 21–26.

[12] Á. Salaverría, J. G. Dacosta, L. F. Ferreira, and E. Mandado, "Analog electronics basic simulator and virtual laboratory," in *Proc. Int. Workshop VIRTUAL-LAB*, Setubal, Portugal, 2004, pp. 20–27.

[13] L. Gomes and A. Costa, "Remote laboratory support for an introductory microprocessor course," in *Proc. IEEE Int. Conf. MSE*, Anaheim, CA, 2005, pp. 21–22.

[14] J. Murphy *et al.*, "Local and remote laboratory user experimentation access using digital programmable logic," *Int. J. Online Eng.*, vol. 1, no. 1, 2005.

[15] K. Arnold *et al.*, *The Jini Specification*. Reading, MA: Addison-Wesley, 1999.

[16] L. S. Indrusiak, M. Glesner, and R. A. L. Reis, "Lookup-based remote laboratory for FPGA digital design prototyping," in *Proc. Int. Workshop e-Learning Virtual Remote Lab*., Setubal, Portugal, 2004, pp. 3–11.

[17] L. S. Indrusiak, F. Lubitz, M. Glesner, and R. A. L. Reis, "Ubiquitous access to reconfigurable hardware: Application scenarios and implementation issues," in *Proc. IEEE/ACM Des. Autom. Test Eur*., Munich, Germany, 2003, pp. 940–945.

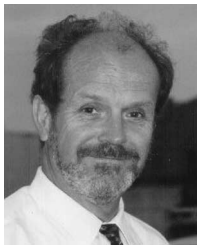
[18] C. Brooks, E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, H. Zheng, Eds., "Heterogeneous concurrent modeling and design in Java (Volume 1: Introduction to Ptolemy II)," EECS Dept. Univ. California, Berkeley, UCB/EECS-2007-7, 2007.

[19] D. Jimenez Orostegui, L. S. Indrusiak, and M. Glesner, *Enhanced Sound Spectrum Simulation*, Mar. 2007. [Online]. Available: http://www.mes.tu-darmstadt.de/staff/lsi/stud_dipl/diegojimenez

Leandro Soares Indrusiak (S'00–M'05) was born in Santa Maria, Brazil. He received the B.S. degree in electrical engineering from the Federal University of Santa Maria, Santa Maria, in 1995, the M.S. degree in computer science from the Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 1998, and the Ph.D. degree in computer science from the Darmstadt University of Technology, Darmstadt, Germany, and UFRGS, in 2003.

From 1998 to 2000, he was a Lecturer with the Informatics Department, Pontifical Catholic University of Rio Grande do Sul, Uruguaiana, Brazil. Since 2000, he has been with the Darmstadt University of Technology, where he is currently a Research Fellow with the Microelectronic Systems Institute and the Coordinator of the International Master Program in Information and Communication Engineering. His research interests include hardware/software design automation, distributed and concurrent computing, remote/virtual laboratories, and collaborative design.

Dr. Indrusiak received the Internationality Award of the Carlo und Karin Giersch Stiftung in 2005 for his contributions to the establishment of international degree programs at Darmstadt University of Technology.



Manfred Glesner (M'93–SM'99–F'00) was born in Saarlouis, Germany. He received the Diploma degree in applied physics and electrical engineering and the Ph.D. degree from Saarland University, Saarbrücken, Germany, in 1969 and 1975, respectively.

From 1975 to 1981, he was a Lecturer with Saarland University in the areas of electronics and CAD. Since 1981, he has been a Professor with the Microelectronic Systems Institute, Darmstadt University of Technology, Darmstadt, Germany. He was a Project Consultant for the European Commission and the United Nation Development Organization. He has contributed original research to areas of electronic circuits design, rapid system prototyping, intelligent signal processing, and reconfigurable hardware.

Prof. Glesner is the recipient of three *honoris causa* doctoral degrees, as well as an *honoris causa* professorship. In 2007, he was nominated Chevalier de l'ordre des palmes académiques (Knight of the Order of the Academic Palms) by the French Minister of Education.

Ricardo Reis (M'81–SM'06) was born in Cruz Alta, Brazil. He received the B.S. degree in electrical engineering from the Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil, in 1978 and the Ph.D. degree in microelectronics from the National Polytechnic Institute, Grenoble, France, in 1983.

Since 1981, he has been a Professor with the Informatics Institute, UFRGS. His research interests include physical design automation, microelectronics education, and fault-tolerant microelectronic systems.