Using Run-Time Reconfiguration for Fault Injection Applications

Lörinc Antoni, Régis Leveugle, and Béla Fehér

Abstract—**The probability of faults occurring in the field increases with the evolution of the CMOS technologies. It becomes, therefore, increasingly important to analyze the potential consequences of such faults on the applications. Fault injection techniques have been used for years to validate the dependability level of circuits and systems, and approaches have been proposed to analyze very early in the design process the functional consequences of faults. These approaches are based on the high-level description of the circuit or system and classically use simulation. Recently, hardware emulation on FPGA-based systems has been proposed to accelerate the experiments; in that case, an important characteristic is the time to reconfigure the hardware, including re-synthesis, place and route, and bitstream downloading. In this paper, an alternative approach is proposed, based on hardware emulation and run-time reconfiguration. Fault injection is carried out by direct modifications in the bitstream, so that re-synthesizing the description can be avoided. Moreover, with some FPGA families (e.g., Virtex or AT6000), it is possible to reconfigure the hardware partially at run-time. Important time-savings can be achieved when taking advantage of these features, since the injection of a fault necessitates the reconfiguration of only a few resources of the device. The injection process is detailed for several types of faults and experimental results are discussed.**

Index Terms—**Fault injection, FPGA, Hardware Prototyping, partial run-time reconfiguration (RTR).**

I. Introduction

THE fault injection techniques have been recognized for a long time as necessary to validate the dependability of a system by analysing its behavior when a fault occurs. Also, reconfigurable devices such as FPGAs are appropriate to implement and test prototypes by synthesising descriptions in high-level languages such as VHDL. Another advantage of prototyping is the possibility to perform "in-system" emulation before any manufacturing. Approaches have been recently proposed to take advantage of prototyping to improve the efficiency of fault injection campaigns. These approaches are based on the implementation in a FPGA device of a specific version of the circuit under analysis, instrumented to perform the injection of internal faults by changing the value of dedicated circuit inputs.

Manuscript received May 26, 2001; revised May 29, 2003.

L. Antoni is with the Techniques of Informatics and Microelectronics for Computer Architecture (TIMA) Laboratory, Grenoble, France, and also with the Department of Measurement and Information Systems, Budapest University of Technology and Economics, Budapest, Hungary (e-mail: antoni@mit.bme.hu).

R. Leveugle is with the Techniques of Informatics and Microelectronics for Computer Architecture (TIMA) Laboratory, Grenoble, France (e-mail: Regis.Leveugle@imag.fr).

B. Fehér is with the Department of Measurement and Information Systems, Budapest University of Technology and Economics, Budapest, Hungary (e-mail: feher@mit.bme.hu).

Digital Object Identifier 10.1109/TIM.2003.817144

Run-time reconfiguration (RTR) is a well-known technique that reconfigures the hardware during the execution of an application. Such a reconfiguration may be applied to the whole device or only to a given subset of the internal elements. Also, in the later case, the other elements may either remain in operation or become idle during the reconfiguration.

In this paper, a new approach is proposed for fault injection in circuit prototypes. This approach is based on RTR to inject the faults, avoiding any instrumentation of the initial circuit description. Experiments reported in this paper include the injection of permanent or transient stuck-at faults in the combinatorial parts of the circuit, as well as the injection of asynchronous transient faults such as single event upsets (SEUs) in both combinatorial parts and flip-flops.

The first goal of the study was to demonstrate the feasibility of the proposed approach based on RTR. The second goal was to analyze the potential benefits of this approach in terms of time required to perform the fault injection campaign, and to identify the potential bottlenecks. The aim of this paper is, therefore, not to report detailed fault injection results for a given circuit example, but to set the bases of a new approach for fault injection.

The paper is organized as follows. Section II gives an overview of fault-injection techniques and a brief introduction to RTR. Alternative fault injection approaches are presented in Section III and the proposed approach is detailed in Section IV. Section V discusses the results achieved by the experiments and points out bases for further work.

II. Preliminaries

A. Fault-Injection and Hardware Prototyping

As previously mentioned, fault injection techniques have been proposed for a long time to evaluate the dependability of a given circuit or system implementation. Most of the approaches proposed up to now apply once the system or circuit is available. Such approaches include pin-level fault injection, memory corruption, heavy-ion injection, power supply disturbances, laser fault injection, or software fault injection.

More recently, several authors proposed to apply fault injection early in the design process. The main approach consists in injecting the faults in high level models (most often, VHDL models) of the circuit or system. [1] describes, for example, the injection of faults in behavioral VHDL descriptions of microprocessor-based system. [2] and then [3] or [4] consider the injection of different types of faults in the VHDL model of a circuit at several abstraction levels and using various techniques, including modifications of the initial VHDL description. As in the case of [1], simulations are used to evaluate the impact of the faults on the circuit behavior. As mentioned in [5], the main drawback related to the use of simulations is the huge amount of time required to run the experiments when many faults have to be injected in a complex circuit.

To cope with the time limitations imposed by simulation, it has been proposed to take advantage of hardware prototyping, using a FPGA-based hardware emulator [6]. Another advantage of emulation is to allow the designer to study the actual behavior of the circuit in the application environment, taking into account real-time interactions [7]. When an emulator is used, the initial VHDL description must of course be synthesisable. In some limited cases, the approaches developed for fault grading using emulators (e.g., [8], [9]) may be used to inject faults. However, such approaches are classically limited to stuck-at fault injection. In most cases, specific modifications are therefore required in the initial description to perform the injections. After these modifications, the description must remain synthesisable and must satisfy a set of constraints related to the emulator hardware [5].

The main goal of the work presented hereafter is to evaluate the feasibility of a new fault injection approach, avoiding any modification of the initial circuit description before it is implemented onto the emulation hardware.

B. Run-Time Reconfiguration

In most cases, the functions implemented onto a FPGA-based system are completely defined during an initialization phase and cannot change anymore during the execution of the application. The same system may be reconfigured to implement other functions, but not during the execution of a single application. This is called compile-time reconfiguration (CTR) [10].

On the opposite, RTR is a technique reconfiguring hardware resources during the execution of a given application, for example to adapt the functions performed by the hardware to the specific requirements of different application phases. This approach has been proposed for example in [11] and [10]. Of course, RTR can only be used in the case of FPGAs that can be re-programmed in the field, e.g., using SRAM or Flash memories to store their configuration.

RTR can be applied in different ways. A first approach consists in temporarily halting the execution of the application in order to reload a new configuration in the whole FPGA memory. This will be called global RTR, as the reconfiguration applies globally to the device. In this case, the application is divided into distinct temporal phases and each phase corresponds to a single system-wide configuration that occupies all FPGA resources [10].

Another approach consists in reconfiguring only some resources in the device. Such a partial reconfiguration corresponds to a local RTR and the functions used in several phases of the application are not redefined during the reconfiguration (e.g., function A in Fig. 1). In this case, noticeable time-savings can be achieved with respect to a complete reconfiguration of the component. Of course, such an approach requires specific capabilities of the reconfigurable device and is therefore limited to a subset of the commercial devices (e.g., Virtex or AT6000).

When a partial reconfiguration is used, the functions remaining unchanged may in some cases remain active during

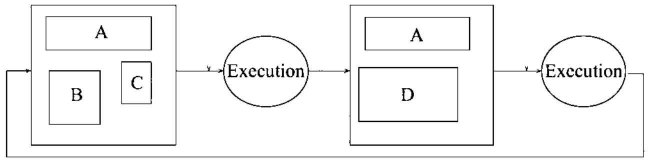

Fig. 1. Illustration of a local RTR.

the reconfiguration. In this particular case, the application may not be interrupted between two subsequent phases.

III. Fault Injection Alternatives

A. CTR-Based Versus RTR-Based Flows

As previously mentioned, the most widely used approach for fault injection is based on the instrumentation of the circuit by modification of the high level description (often assumed to be in VHDL). In a second step, the VHDL code is synthesised and a netlist is generated. After placement and routing of this netlist, a bitstream is obtained that can be downloaded onto the FPGA. Finally, a set of experiments is performed and the analysis of the circuit responses is made. For each experiment, one or several faults are injected during the application run. The analysis may aim at categorising the faults (identifying those having some types of effects) or may end up with a functional model of the system behavior, that shows how the system evolves when the faults occur (identification of the error propagation paths).

With the classical approach, each fault to inject requires additional hardware implemented in the prototype, and additional control signals to define the fault activated at a given time during the experiments. Due to the hardware limitations of the emulator, it is not always possible to include in a single prototype all the modifications required to inject all the faults. Several instrumented descriptions may therefore be necessary, each of them allowing the designer to inject a subset of the faults and thus to run a subset of the experiments. The synthesis, placement, and routing of each description has to be done separately and the emulator is reconfigured globally (using CTR) at the end of each subset of experiments [Fig. 2(a)].

The basic idea proposed in this paper is to replace the modifications performed in the VHDL description by modifications performed directly in the bitstream obtained after synthesis, placement, and routing of the initial circuit description. The bitstream modification must be done for each fault to inject, but only one synthesis, placement and routing has to be done, no matter the number of faults to inject. Since these design steps can be very time consuming, avoiding to repeat them may lead to reduce the time globally spent when running a fault injection campaign on a hardware emulator.

For each experiment, the bitstream modified for a new fault configuration has to be downloaded onto the emulator at the injection time, using RTR. This may of course be done by a global reconfiguration of the device [Fig. 2(b)]. However, in practice, only a small number of bits has to be changed in a bitstream to inject a fault and this implicitly corresponds to a local reconfiguration, that can take advantage of partial reconfiguration capabilities of the device [Fig. 2(c)]. The emulator has to be reconfigured at least one time per experiment to inject permanent faults, and at least two times per experiment if transient faults are targeted (one time to inject the fault, and one time to

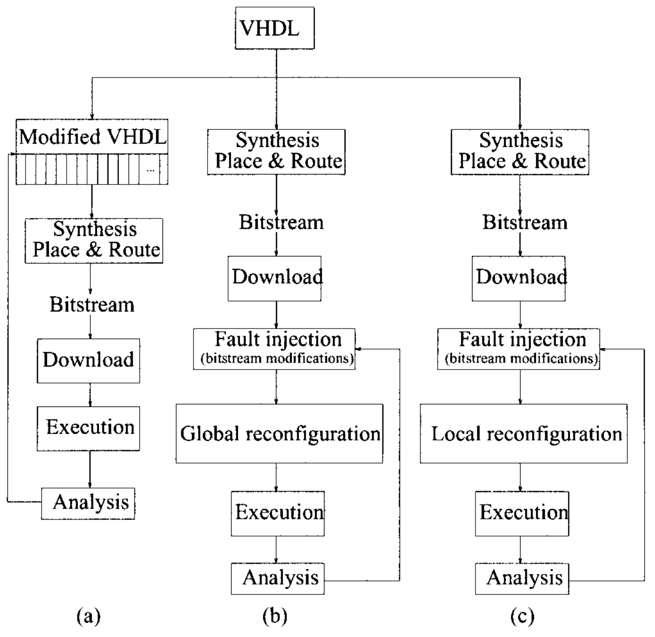

Fig. 2. Alternative fault injection flows. (a) Use of VHDL modifications with CTR or direct injection into the prototype with (b) global or (c) local reconfiguration.

remove it). The total number of configurations of the emulator is therefore much higher than in the classical case, but the total configuration time is optimized in the proposed approach by the possible use of partial reconfigurations.

B. Handling the System State During Reconfiguration

In the case of global RTR, the system state must be handled to keep the information computed before each configuration step: a means must be provided to support inter-configuration communication [10]. Because each phase occupies all reconfigurable resources, interfaces between configurations are fixed and all circuit modules can be designed under the same general context. This approach was used to develop a hardware accelerator for image processing [12]. Another application implementing global RTR, the RTR artificial neural network (RRANN), is described in [13], where a host PC stores all configuration information for the FPGAs, monitors the progress of each stage of execution and supplies the appropriate configuration data to the FPGA board. In the case of fault injection, the internal state of the circuit under analysis remains unchanged during the partial reconfiguration and thus does not require external management.

When partial reconfiguration is applied, the parts of the circuit that are not reconfigured may stay in operation while the other parts are reconfigured. In the case of fault injection, the application must in fact be temporarily halted in order to inject the fault at the desired cycle in the execution.

IV. Developed Fault Injection Technique

This section details the approaches illustrated in Fig. 2(b) and (c). Faults are injected at "low-level," directly in the bitstream, so that the circuit must not be modified or re-synthesised.

The study has been carried out using FPGAs from Xilinx, and especially Virtex FPGAs with partial reconfiguration capabilities. The bitstream modifications were automated using the JBits toolset [14], that is described in Section IV-A.

Different fault models have been considered, targeting both combinatorial parts and memory elements. For the combinatorial parts, the classical stuck-at model has been chosen and faults are injected on the inputs or outputs of the configurable logic blocks (CLBs) by modifying the contents of the look-up tables (LUTs). The approach used for stuck-ats can also be extended to signal inversions, modeling some types of SEUs in the combinatorial logic. RTR-based reconfiguration to inject these types of faults is detailed in Section IV-B. Another application of RTR is then detailed in Section IV-C. Here, asynchronous bit-flips are injected directly in the functional flip-flops used in the circuit under analysis.

A. JBits API

The JBits application programming interface (API) is a Java-based tool set that allows designers to write information directly to a Xilinx FPGA to carry out whatever customer logic operations were designed for it [14] [15]. Using JBits, the FPGA bitstream can be modified quickly and easily, allowing for fast reconfiguration of the FPGA.

In Virtex FPGAs, JBits can partially or fully reconfigure the internal logic of the hardware device. The Virtex architecture allows this reconfiguration to be as extensive as necessary and still maintain timing information [14] [15]. JBits also makes possible to integrate the operations of the FPGA with other system components such as an embedded processor, a graphics coprocessor, or any digital peripheral device.

B. Fault Injection in Combinatorial Parts

To implement the approach in Fig. 2(b), the initial bitstream can be read from a file, and the modified bitstream can be written to a file that is then downloaded onto the board. The approach in Fig. 2(c) is more interactive and directly sends new configuration data to the FPGA. Similarly, the bitstream can be read directly back from the device.

As far as combinatorial parts are concerned, two types of faults have been considered: stuck-at faults and signal inversions. The faults can only be injected on signals connecting CLBs, i.e., on LUT inputs or outputs. This is not a limitation in practice since injecting faults on other types of signals would be meaningless, due to the very different implementation in the final circuit and in the prototype. From a functional point of view, meaningful injection targets classically correspond to primary inputs of the circuit or to flip-flop outputs, connected to the primary inputs of combinatorial blocks. In some cases, the injection can be useful also on a signal connecting two different blocks in the circuit description hierarchy. In all cases, these signals are mapped to LUT inputs if the synthesis maintains the description hierarchy. Of course, the injection of a fault on an intermediate signal that disappears during the synthesis optimizations is meaningless, since this signal will not exist in the actual circuit.

When modifying the FPGA configuration, each injection target is determined by a set of low-level parameters: the CLB row and column numbers, and the input of the CLB (F1, G4, etc.). In order to simplify the specification of the injection

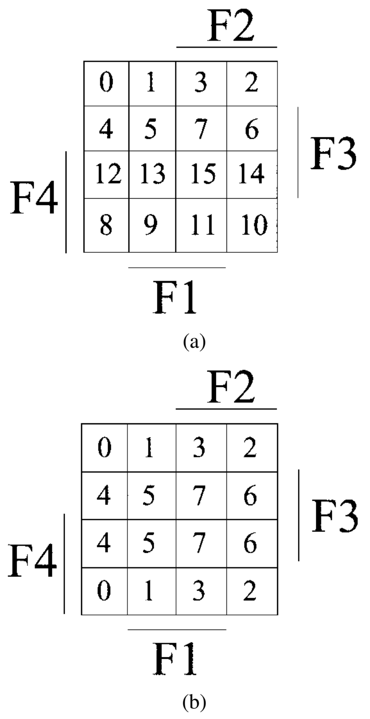

Fig. 3. Injection of a stuck-at-0 fault on the F4 input of a CLB: (a) original LUT state and (b) modified LUT.

campaign, the program developed for our experiments starts from a list of targets identified by the name of the signals in the initial high-level description. If the targets are meaningful, these names appear in the netlist after synthesis. Then, using the reports generated during the placement and routing phases, it is possible to identify the position of the targets in the FPGA array. Finding the correspondences between the high-level signal names and the low-level parameters defining the assigned LUT inputs is therefore conceptually very easy. The process can however be tricky, depending on the exact information given by the tools used for synthesis, placement and routing. Such a process was automated for the version of the tools used during the experiments; it would have to be revisited for new versions or other tools.

Once the targets are localized in the FPGA array, the faults are injected by modifying the contents of the LUT(s). Let us consider the example shown in Fig. 3. In this example, a stuck-at-0 fault is injected on the F4 input of the CLB. It can easily be seen that in this case, the output value for F4F3F2F1 $= 0000$ must be copied to the output value for F4F3F2F1 $= 1000$, the value for 0001 to 1001, etc. The bitstream used for the experiment is modified accordingly.

Similar modifications of the contents of a LUT can be defined for each kind of fault (stuck-at-0, stuck-at-1 or signal inversion) and for each input of the CLB. Similar modifications also allow to inject the fault on the output of a LUT, that is more efficient in the case of a signal with high fanout.

C. SEU Fault Injection in CLB Flip-flops

The SEU fault model is currently receiving an increasing attention and is often associated with asynchronous bit-flips of the memory cells in a circuit. Three main differences can be pointed out compared with the previous case.

- The modification of the functional data must be performed directly in a flip-flop (FF) element.
- The modification must occur asynchronously with respect to the system clock (that could be a gated clock in some cases).
- The value to inject in the FF depends on the correct functional value at injection time since the flip-flop must commute.

For these reasons, it is not possible to inject SEUs in memory elements by modifying the function (or LUT configuration) of the circuit.

In fact, the only way of changing asynchronously the state of a FF in a Virtex FPGA is to apply a set or reset to it, depending on whether it should be changed to 1 or 0. However, in the context of run-time reconfiguration, it is only possible to pulse the global set/reset (GSR) line of the device, that sets or resets simultaneously all the FFs in the device. The actual effect of the GSR line is determined for each flip-flop by the position of a switch.

The reconfiguration process to inject a SEU is therefore much more complex than the process previously presented. In order to change the state of only one FF, the states of all FFs and set/reset switches must be read back first at the injection time. If there are FFs where the switch is not in a position coherent with the FF contents (e.g., the switch is in "set" position but the FF state is "0"), the switch must be changed before pulsing the GSR line in order to leave the FF in its initial state. Conversely, in the case of the FF where the SEU should be injected, the switch must be set in the opposite position. The basic steps of the algorithm used for asynchronous SEU injection in FFs are thus as follows:

- Initialization: reading of the bitstream and full configuration of the device;
- read the states of set/reset switches.

Then, for each injection experiment:

- start execution of the application;
- stop the execution at injection cycle;
- read the states of FFs;
- change the set/reset switches where needed (depending on FF states);
- pulse the GSR line.
- re-change the switches modified before the GSR pulse;
- continue the execution of the application until either the end of the experiment or the next injection.

Partial reconfiguration as well as partial read-back of the configuration and functional data are controlled through the JBits API.

V. Results

Preliminary experiments were realized on a XS40 board [16] using JBits 1.1. This board contains an XC4010XL FPGA device [17] that is not able to implement partial reconfiguration. Global reconfiguration was therefore used in this case. Other experiments have been carried out on a XSV development board based on a Virtex XCV50 device and supporting partial reconfiguration [18]. In the two cases, the application that has been developed requires only a fraction of a second to inject a fault at low level, directly in the bitstream. This may be compared with the minutes or hours that could be required for a single synthesis, placement, and routing process. The feasibility of RTR-based fault injection has been demonstrated for the various fault models discussed in Section IV, using simple design examples, both combinatorial and sequential.

Evaluations made on the Virtex devices have also shown that configuration time savings of several orders of magnitude can be expected when using partial reconfiguration, with respect to global reconfiguration [19]. As an example, injecting a stuck-at implies to modify eight frames that require 0.424 ms on a XCV400 device, instead of 169.738 ms for a complete reconfiguration. The same injection in a XCV2000E device would require 0.816 ms to be compared with 677.309 ms for a complete reconfiguration.

In order to make efficient fault injection campaigns using the proposed approach, it is, however, not sufficient to use partial reconfiguration. The speed of the reconfiguration and readback has also to be carefully optimized. Ideally, this would require a device architecture better suited to this application than the Virtex architecture. As an example, one could imagine to group all the configuration bits of a LUT in a single frame. In that case, only one frame should be reconfigured to inject a stuck-at fault, instead of eight in a Virtex device. Also, it has been shown that the Virtex architecture imposes strong constraints on the injection of asynchronous bit-flips. Here, again, a different architecture may noticeably shorten the injection.

Apart from the device architecture, other characteristics must be optimized at the system level, including:

- the place and route algorithms, that may group the CLBs used by a given design so that a minimum number of frames has to be used during the configuration of the device;
- the configuration bandwidth on the development board (high frequency configuration clock and/or configuration data sent in parallel mode onto the FPGA);
- the bandwidth of the interface between the development board and the host computer.

Developing such an efficient environment for RTR-based fault injection is a subject for further work.

VI. Conclusion

The experiments reported in this paper demonstrate the feasibility of a new fault injection approach, based on RTR in a FPGA-based circuit prototype. The modifications required in the bitstream to inject the faults can be automated for various fault models, including permanent and transient stuck-at and SEU faults.

Using partial reconfiguration capabilities available in some FPGAs, the proposed approach could lead to noticeable time-savings compared with other fault injection approaches. However, in order to achieve efficient fault injection campaigns, the speed of the reconfiguration (and readback) must be maximized since one or two partial reconfigurations are necessary for each injection. This requires the development of a prototyping environment taking into account this constraint that is quite different from the constraints classically considered for other applications needing only a few reconfigurations.

References

[1] T. A. Delong, B. W. Johnson, and J. A. Profeta, "A fault injection technique for VHDL behavioral-level models," *IEEE Design Test Comput.*, vol. 13, pp. 24–33, Winter 1996.

[2] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, "Fault injection into VHDL models: The MEFISTO tool," in *24th Int. Symp. Fault-Tolerant Comput.*, June 1994, pp. 66–75.

[3] S. Svensson and I. Karlsson, "Dependability Evaluation of the THOR Microprocessor Using Simulation-Based Fault Injection," Chalmers Universiity of Technology, Department of Computer Engineering, 295, 1997.

[4] J. Boué, P. Pétilton, and Y. Crouzet, "MEFISTO-L: A VHDL-based fault injection tool for the experimental assessment of fault tolerance," in *28th FTCS*, June 1998, pp. 168–173.

[5] R. Leveugle, "Toward modeling for dependability of complex integrated circuits," in *IEEE Int. On-Line Testing Workshop*, July 1999, pp. 194–198.

[6] ——, "Behavior modeling of faulty complex VLSIs: Why and how?," in *Baltic Electronics Conf.*, Oct. 1998, pp. 191–194.

[7] E. Böhl, W. Harter, and M. Trunzer, "Real time effect testing of processor faults," in *5th IEEE Int. On-Line Testing Workshop*, July 1999, pp. 39–43.

[8] K.-T. Cheng, S.-Y. Huang, and W.-J. Dai, "Fault emulation: A new methodology for fault grading," *IEEE Trans. Comput.-Aided Design Integrated Circuits Syst.*, vol. 18, no. 10, pp. 1487–1495, Oct. 1999.

[9] R. W. Wider, Z. Zhang, and R. D. McLeod, "Emulating static faults using a xilinx based emulator," in *IEEE Symp. FPGAs for Custom Computing Machines*, Apr. 1995, pp. 110–115.

[10] B. L. Hutchings and M. J. Wirthlin, "Implementation approaches for reconfigurable logic applications," in *5th Int. Workshop on Field Programmable Logic and Applications*, Aug. 1995, pp. 419–428.

[11] P. Lysaght and J. Dunlop, "Dynamic reconfiguration of field programmable gate arrays," in *Proc. Int. Workshop on Field Programmable Logic and Applications*, Sept. 1993, pp. 82–94.

[12] D. Ross, 0. Vellacott, and M. Turner, "An fpga-based hardware accelerator for image processing," in *More FPGAs: International Workshop on Field-Programmable Logic and Applications*, Sept. 1993, pp. 299–306.

[13] J. G. Eldredge and B. L. Hutchings, "Run-time reconfiguration: A method for enhancing the functional density of sram-based fpgas," *J. VLSI Signal Processing*, vol. 12, pp. 67–86, 1996.

[14] S. A. Guccione, D. Levi, and P. Sundararajan, "JBits: Java-based interface for reconfigurable computing," in *Second Annual MAPLD*, 1999.

[15] *JBits 2.8*, Xilinx, San Jose, CA, 2001.

[16] *XS40, XSP Board V1.4*, XESS Corporation, Apex, NC, 1999.

[17] *XC4000E and XC4000X Series Field Programmable Gate Arrays*, Xilinx, San Jose, CA, 1999.

[18] *Virtex Field Programmable Gate Arrays*, Xilinx, San Jose, CA, 2000.

[19] L. Antoni, R. Leveugle, and B. Fehér, "Using run-time reconfiguration for fault injection in hardware prototypes," *IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems*, pp. 405–413, Oct. 2000.

Lörinc Antoni received the M.Sc. degree in electrical engineering from the Budapest University of Technology and Economics (BUTE), Budapest, Hungary, in 1998. The defense of his Ph.D. degree was held in September 2003 at the Techniques of Informatics and Microelectronics for Computer Architecture (TIMA) Laboratory, National Polytechnical Institute of Grenoble (INPG), Grenoble, France, in collaboration with the Department Of Measurement and Information Systems, BUTE.

His research interests include reconfigurable computing, hardware synthesis, object-oriented conception in hardware design, and fault injection techniques.

Régis Leveugle received the Ph.D. degree in microelectronics from the National Polytechnical Institute of Grenoble (INPG), Grenoble, France, in 1990.

He is currently a Professor at the INPG and a member of the Techniques of Informatics and Microelectronics for Computer Architecture (TIMA) Laboratory. His main interests are computer architecture, VLSI design methods and CAD tools, dependability evaluation, fault-tolerant architectures, and concurrent checking. He has authored or coauthored more than 90 scientific papers and served as a Reviewer for many journals and conferences. He has also served on several program and organization committees.

Dr. Leveugle was Program co-Chair for the 2001 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'01), vice-General Chair for the 2002 IEEE International On-Line Testing Workshop (IOLT'02), and General co-Chair for DFT'02. He is vice-Program Chair for the 2003 IEEE International On-Line Testing Symposium (IOLTS'03).

Béla Fehér received the Ph.D. degree in electrical engineering from the Budapest University of Technology and Economics (BUTE), Budapest, Hungary, 1995.

He is currently an Associate Professor with the Department of Measurement and Information Systems, BUTE. His research fields include digital signal processing, parallel architectures, embedded computers, SoPCs, reconfigurable computing, and fault tolerant architectures. He holds lectures in digital systems design and high performance VLSI DSP architectures. He has published more than 30 scientific papers and served as a member of program committees of different conferences.