Run-Time Management of Logic Resources on Reconfigurable Systems

Manuel G. Gericota, Gustavo R. Alves
Department of Electrical Engineering – ISEP
Rua Dr. António Bernardino de Almeida – 4200-072 Porto – PORTUGAL

Miguel L. Silva, José M. Ferreira
Department of Electrical and Computer Engineering – FEUP
Rua Dr. Roberto Frias – 4200-465 Porto – PORTUGAL

Abstract

Dynamically reconfigurable systems based on partial and dynamically reconfigurable FPGAs may have their functionality partially modified at run-time without stopping the operation of the whole system.

The efficient management of the logic space available is one of the biggest problems faced by these systems. When the sequence of reconfigurations to be performed is not predictable, resource allocation decisions have to be made on-line. A rearrangement may be necessary to get enough contiguous space to implement incoming functions, avoiding the spreading of their components and the resulting degradation of system performance.

A new software tool that helps to handle the problems posed by the consecutive reconfiguration of the same logic space is presented in this paper. This tool uses a novel on--line rearrangement procedure to solve fragmentation problems and to rearrange the logic space in a way completely transparent to the applications currently running.

1. Introduction

Reconfigurable computing experienced a considerable expansion in the last few years, due in part to the fast run--time partial reconfiguration features offered by recent Field Programmable Gate Arrays (FPGAs). The Virtex and Spartan families from Xilinx, used to validate this work, are the most recent examples. This kind of devices enabled the implementation of the concept of virtual hardware defined in [1] ten years ago: to use temporal partitioning to implement those applications whose area requirements exceed the reconfigurable logic space available (i.e. to assume the availability of unlimited hardware resources). The static implementation of a circuit is separated in two or more independent hardware contexts, which may be swapped during runtime [2]. Extensive work was done to improve the multi-context handling capability of these devices, by storing several configurations and enabling quick context switching [3, 4]. The main goal was to improve the execution time by minimising external memory transfers, assuming that some amount of on-chip data storage was available in the reconfigurable architecture. However, this solution was only feasible if the functions implemented on hardware were mutually exclusive on the temporal domain, e. g. context-switching between coding/decoding schemes in communication, video or audio systems; otherwise, the length of the reconfiguration intervals would lead to unacceptable delays in most applications.

These restrictions have been overtaken by higher levels of integration, due to the employment of sub-micron scales, and by the use of higher frequencies of operation. The increasing amount of logic available in FPGAs and the reduction on the reconfiguration time, partly owing to the possibility of partial reconfiguration, extended the concept of virtual hardware to the implementation of multiple applications sharing the same logic resources in the spatial and temporal domains.

An application comprises a set of functions that are predominantly executed sequentially, or with a low degree of parallelism, in which case their simultaneous availability is not required. On the other hand, the reconfiguration intervals offered by new FPGAs are sufficiently small to enable functions to be swapped in real time. If a proper floorplanning schedule is devised, it becomes feasible to use a single device to run a set of applications, which in total require far more than 100% of

♦ This work is supported by an FCT program under contract POCTI/33842/ESE/2000

the FPGA available resources, by swapping functions in and out of the FPGA as needed.

Partial reconfiguration times are in the order of a few milliseconds, depending on the configuration interface and on the complexity (and thus on the size) of the function being implemented. However, the reconfiguration time overhead may be virtually zero, if new functions are swapped in advance with those already out of use, as illustrated in figure 1. A number of applications share the same available reconfigurable logic space in both the temporal and spatial domains [5]. After the execution of a given function, a new function may be set up in its place during the interval r_t, in order to be available when required by the application flow (r_t should therefore not be mistaken by the reconfiguration time). Notice that an increase in the degree of parallelism may retard the reconfiguration of incoming functions, due to lack of space in the FPGA. Consequently, delays will be introduced in the application execution, systematically or not, depending on the application flow.

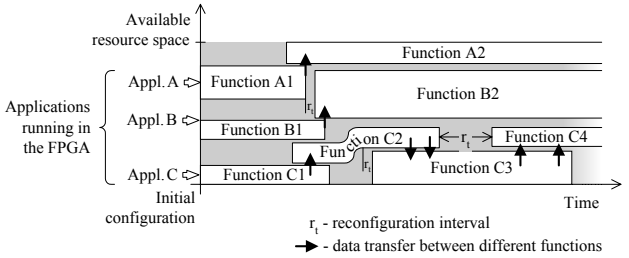

Fig. 1. Temporal scheduling of applications in the temporal and spatial domains

The main goal behind the temporal and spatial partitioning of the reconfigurable logic resources is to achieve the maximum efficiency of the reconfigurable systems, pushing up resource usage and taking the maximum advantage of its flexibility. However, this approach comprises several problems. An incoming function may require the relocation of other functions already implemented and running, in order to release enough contiguous space for its configuration (see function C2 in figure 1). Since each of the multiple independent functions sharing the logic space occupies a different amount of resources, many small pools of resources are created as they are released. These unallocated areas tend to become so small that they fail to satisfy any request and for that reason remain unused, leading to a fragmentation of the FPGA logic space [6].

Suitable arrangements can be designed if the requirements of each function and their sequence are known in advance, but an increase in the available resources will in most cases be necessary to cope with the allocation problem [7]. However, when placement decisions have to be made on-line, or the need for extra space is only temporary, an increase on the available resources is a poor solution, since it decreases the efficiency of the system.

The problem described may be solved through on-line management of the available resources, whereby the system tries to avoid that a lack of contiguous free resources prevents the configuration of new functions (provided that the total number of resources available is sufficient). Note that spreading the components of an incoming function, due to fragmentation of available resources, would degrade its performance, delaying tasks and reducing the utilisation of the FPGA. If a new function cannot be allocated immediately due to lack of contiguous free resources, a suitable rearrangement of a subset of the functions currently running may solve the problem. Three methods are proposed in [5] to find such (partial) rearrangements, in order to increase the rate at which waiting functions are allocated, while minimising disruptions to running functions that are to be relocated. However, no physical execution of these rearrangements is proposed other than halting those functions, stopping the normal system operation.

A mechanism to implement such rearrangements without disturbing the system operation is presented in this paper. To address this problem, a new concept is introduced – dynamic relocation –, which enables the relocation of each FPGA CLB (Configurable Logic Block) and of its associated interconnections, even if the CLB is part of a function that is actually being used by an application [8, 9]. This concept enables the rearrangement and defragmentation of the FPGA logic space on-line (i. e. concurrently with all applications currently running), without any time overheads.

We will start by describing the dynamic relocation of each CLB, highlighting the constraints imposed by the FPGA architecture, and then we will introduce the relocation mechanism proposed. The relocation of routing resources will then be considered, including the software tool that was developed to automate the generation of the required partial configuration files.

2. Dynamic CLB relocation

Conceptually, an FPGA could be described as an array of uncommitted CLBs, surrounded by a periphery of IOBs, which are interconnectable by configurable routing resources, controlled by an underlying set of memory cells.

Any on-line management strategy implies a dynamic relocation mechanism, whereby a CLB currently being used by a given function has its functionality transferred into another CLB, without disturbing system operation. This relocation mechanism does more than just copying the functional specification of the CLB to be replicated:

the corresponding interconnections with the rest of the circuit have to be re-established; additionally, according to its current functionality, internal state information may also have to be copied.

The transparent relocation of a CLB is not a trivial task due to two major issues: i) configuration memory organisation and ii) internal state information.

The configuration memory can be visualised as a rectangular array of bits, which are grouped into one-bit wide vertical frames extending from the top to the bottom of the array. A frame is the smallest unit of configuration that can be written to or read from the configuration memory. Frames are grouped together into larger units called columns. Each CLB column corresponds to a configuration column with multiple frames, mixing internal CLB configuration and state information, and column routing and interconnect information. The partitioning of the entire FPGA configuration memory into frames enables on-line concurrent partial reconfiguration, facilitating the implementation of on-line rearrangement procedures. The configuration procedure is a sequential mechanism that spans through some (or eventually all) CLB configuration columns. When the functionality of a given CLB is dynamically relocated, even into a CLB in the same column, more than one column may be affected, since its input and output signals (as well as those in its replica) may cross several columns before reaching its source or destination.

Any reconfiguration action must therefore ensure that the signals from the original CLB are not broken before being totally re-established from its replica, otherwise its operation will be disturbed or even halted. It is also important to ensure that the functionality of the CLB replica is perfectly stable before its outputs are connected to the system, to prevent output glitches. A set of experiments performed with an XCV200 from Xilinx demonstrated that the only possible solution is to divide the relocation procedure in two phases, as illustrated in figure 2 (for reasons of intelligibility, only the relevant interconnections are represented).

In the first phase, the internal configuration of the CLB is copied into the new location and the inputs of both CLBs are placed in parallel. Due to the slowness of the reconfiguration procedure when compared with the speed of operation of current applications, the outputs of the CLB replica are already perfectly stable when they are connected to the circuit, in the second phase. To avoid output glitches, both CLBs (the original and its replica) must remain in parallel for at least one clock cycle. Notice that rewriting the same configuration data does not generate any transient signals, so this procedure does not affect the remaining resources covered by the rewriting of the configuration frames that are needed to carry out the relocation procedure.

Successful completion of the procedure described above cannot be achieved without correct transfer of state information. If the current CLB function is purely combinational, this two-phase relocation procedure will suffice to accomplish successful relocation. However, in the case of a sequential function, the internal state information must be preserved and no update operations could be lost during the copying phase. The solution to this problem depends on the type of implementation. In this paper we shall consider three implementation cases: synchronous free-running clock circuits, synchronous gated-clock circuits, and asynchronous circuits.

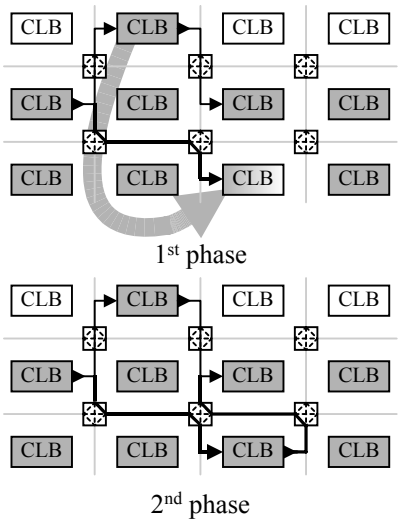

Fig. 2. Two-phase CLB relocation procedure

When dealing with synchronous free-running clock circuits, the two-phase relocation procedure described previously is a good solution. Between the first and the second phase the CLB replica has the same inputs as the original CLB, and all its flip-flops (FFs) acquire the same state information. The experimental replication of CLBs with FF driven by a free-running clock has confirmed the effectiveness of this method. No loss of state information or the presence of output glitches was observed.

When using synchronous gated-clock circuits, where input acquisition by the FFs is controlled by the state of the clock enable signal (CE), the previous method does not ensure that the CLB replica captures the correct state information, because CE may not be active during the relocation procedure. Besides, it is not feasible to simply set this signal as part of the relocation procedure, because the value present at the input of the replica FFs may change in the meantime, and a coherency problem would then occur.

An auxiliary relocation circuit needs to be implemented to solve this problem. This circuit manages the transfer of the state information from the original FFs to the replica

FFs, while enabling their update by the circuit at any instant, without delaying the relocation procedure. The whole relocation scheme is represented in figure 3, where only one logic cell is shown, for reasons of simplicity. In the Virtex and Spartan families, each CLB comprises four of these cells; however, and for the purpose of implementing this procedure, each CLB cell can be considered individually. The temporary transfer paths established between the original cells and their replicas do not affect their functionality, since only free routing resources are used. No changes in the cell structure are required to implement this procedure. The relocation control and the clock enable control signals are driven through the reconfiguration memory, so no extra external pins are required. The OR gate and the 2:1 multiplexer that are part of the auxiliary relocation circuit must be implemented during the relocation process in a nearby (free) CLB.

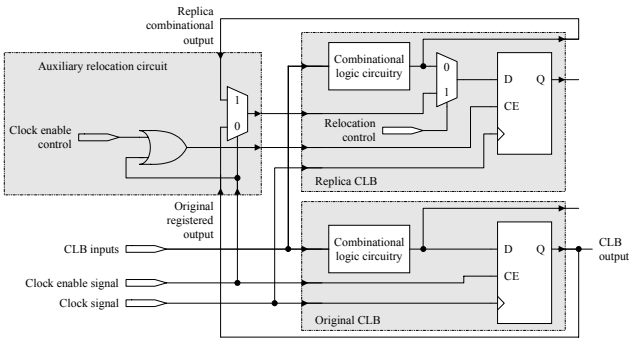

Fig. 3. CLB relocation for synchronous gated--clock circuits

The inputs of the 2:1 multiplexer present in the auxiliary relocation circuit receive one temporary transfer path from the output of the original CLB FF and another one from the output of the combinational logic circuitry of the replica CLB, which, in normal operation, is applied to the FF input. This multiplexer is controlled by the clock enable signal of the original CLB FF. If this signal is not active, the output of the original CLB FF is applied to the input of the replica CLB FF. The clock enable control signal is then activated, which forces the replica CLB FF to capture the value coming from the original CLB FF.

If the clock enable signal is active, or is activated during this process, the multiplexer selects the output of the combinational block in the replica CLB and applies it to its FF input. In this case, both the original and the replica FFs are updated at the same time and with the same values, guaranteeing state coherency.

After the state has been transferred, the input signals involved in the execution of the relocation procedure are placed in parallel, all the signals to and from the auxiliary relocation circuit are disconnected, and the outputs of both CLBs are also placed in parallel. After at least one function clock cycle, the original CLB is disconnected from the rest of the circuit (first the outputs and then the inputs, in order to prevent any transient instability in the output signals), and becomes part of the pool of free resources.

Figure 4 represents the flow diagram of the proposed relocation procedure. Several relocation experiments were carried out in a group of circuits from the ITC'99 Benchmark Circuits from the Politécnico di Torino [10] implemented in a Virtex XCV200, proving the effectiveness of our approach. These circuits are purely synchronous with only one single-phase clock signal present. However, this approach is also applicable to multiple clock/multiple phase applications, since only one clock signal is involved in the relocation of each CLB (CLBs relocation is performed individually, even if many of these blocks were replicated simultaneously). No loss of information or functional disturbance was observed during the execution of these experiments.

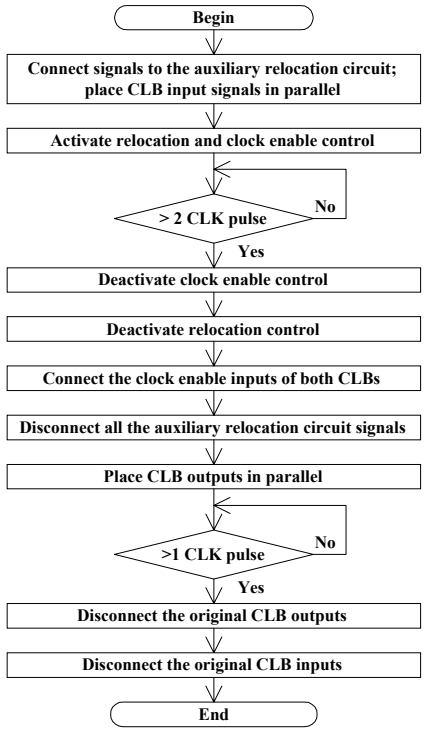

Fig. 4. Relocation procedure flow diagram

The average relocation time of each CLB implementing synchronous gated-clock circuits is about 22,6 ms, when the Boundary Scan [11] infrastructure is used to perform the reconfiguration, at a test clock frequency of 20 MHz.

This method is also effective when dealing with asynchronous circuits, where transparent data latches are used instead of FFs. In this case, the control enable signal is replaced in the latch by the input control signal, with the

value present in the input of the FF being stored when the control signal changes from '1' to '0'. The same auxiliary relocation circuit is used and the same relocation sequence is followed.

In the Virtex family of FPGAs, LUTs (Look-Up Tables) can be configured as Distributed RAMs. However, it is not feasible to extend this on-line relocation concept to the relocation of those LUT/RAMs. The content of the LUT/RAMs could be read and written through the configuration memory, but the system would have to be stopped to ensure data coherency, in the case of a writing attempt during the relocation interval, as stated in [12]. Furthermore, since frames span an entire column of CLB slices, the same LUT bit in all of them is updated with a single write command. It must be ensured that all the remaining data in the slice is either constant, or it is also modified externally through partial reconfiguration. Even not being relocated, LUT/RAMs should not lie in any column that could be affected by the relocation procedure.

3. Rearranging routing resources

Due to the scarcity of routing resources, it might also be necessary to perform a rearrangement of the existent interconnections, to optimise the occupancy of such resources, after the relocation of one or more CLBs, and to increase the availability of routing paths to incoming functions. The relocation of routing resources does not pose any special problems, since the same two-phase relocation procedure is effective on the relocation of local and global interconnections. The interconnections involved are first duplicated in order to establish an alternative path, and then disconnected, becoming available to be reused, as illustrated in figure 5.

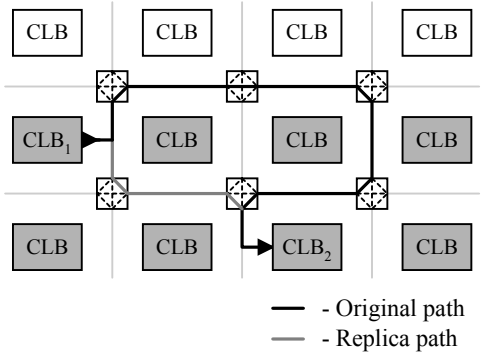

Fig. 5. Relocation of routing resources

A last remark must be made about the relocation of routing resources. Since different paths are used while paralleling the original and replica interconnections, each of them will have a different propagation delay. This means that if the signal level at the output of the CLB source changes, the signal at the input of the CLB destination will show an interval of fuzziness, as shown in figure 6. However, the impedance of the routing switches will limit the current flow in the interconnection, and hence this behaviour does not damage the FPGA. Nevertheless, and for transient analysis, the propagation delay associated to the parallel interconnections, shall be the longer of the two paths.

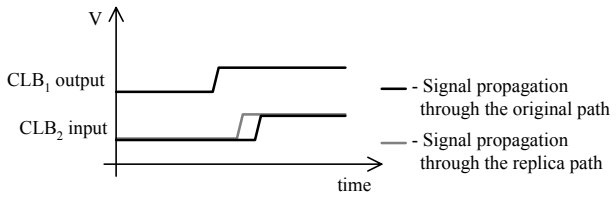

Fig. 6. Propagation delay during the relocation of routing resources

The dynamic relocation of CLBs and interconnections should have a minimum influence (preferably none) in the system operation, as well as reduced overhead in terms of reconfiguration cost. This cost depends on the number of reconfiguration frames needed to relocate each CLB, since a great number of frames would imply a longer rearrangement time. The impact of the relocation procedure in those functions currently running is mainly related to the delays imposed by rerouted paths, since the relocation procedure might imply a longer path, therefore decreasing the maximum frequency of operation.

The placement algorithms (in an attempt to reduce path delays) gather in the same area the logic that is needed to implement the components of a given function. It is unwise to disperse it, since it would generate longer paths (and hence, an increase in path delays). On the other hand, it would also put too much stress upon the limited routing resources. Therefore, the relocation of the CLBs should be performed to nearby CLBs. If necessary, the relocation of a complete function may take place in several stages, to avoid an excessive increase in path delays during the relocation interval.

4. The FPGA rearrangement and programming tool

To support the implementation of this management process, a software tool was developed, based on the JBits software – a set of Java classes that provide an Application Programming Interface (API) to access the Xilinx FPGA bitstream [13]. This tool is responsible by the creation of the partial configuration files and carries out the partial and dynamic reconfiguration of the FPGA through the Boundary Scan interface. Since the partial configuration files that implement the rearrangements

defined by the relocation procedure are generated automatically (without designer intervention), the usage of this tool becomes very straightforward. The input information may be provided in the form of a complete configuration file (generated by the traditional development tool with a new placement for the functions currently running or those that are about to be implemented) or by providing the co-ordinates - source and destination - of the CLB to be relocated. The tool implements a series of algorithms based on artificial intelligence techniques that manage the routing of the signals coming in and out of the CLBs that are relocated, optimising the usage of the routing resources. The program always keeps a complete copy of the current configuration, enabling system recovery in case of failure. The user interface is shown in figure 7.

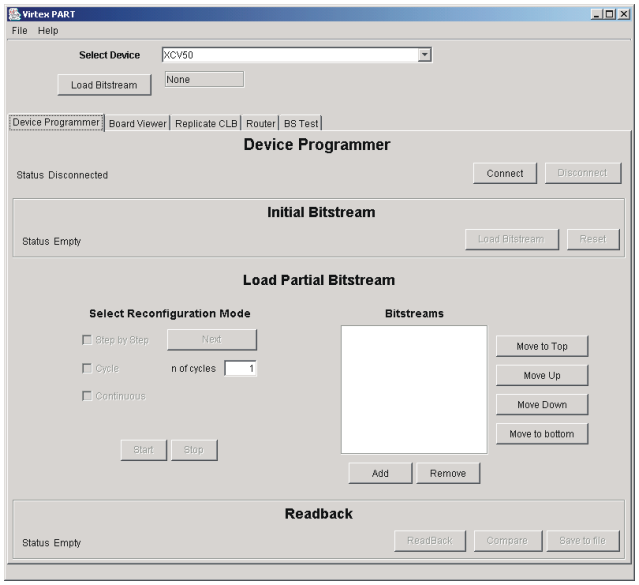

Fig. 7. Interface of the FPGA Rearrangement and Programming tool

5. Conclusion

A novel relocation procedure to perform the dynamic relocation of CLBs, without halting their operation, was presented in this paper. The proposed procedure enables the implementation of a truly on-line management of the FPGA logic space, supporting the rearrangement of running functions, releasing enough contiguous space for the configuration of new incoming functions, and performing defragmentation. Therefore, on-line dynamic scheduling of tasks in the spatial and temporal domains becomes possible, enabling the implementation of the virtual hardware concept [1]. Several applications may share the same hardware platform, with their respective functions running and being swapped in and out of the FPGA, without generating any time overhead to the running applications, or disturbing their operation. The software application that was developed to support the implementation of the relocation procedure enables the complete automation of the whole process and an optimised management of the available resources. Further work is under way to increase the functionality and flexibility of this tool.

References

[1] X. P. Long, H. Amano, "WASMII: a Data Driven Computer on a Virtual Hardware", *Proc. 1st IEEE Workshop on FPGAs for Custom Computing Machines*, 1993, pp. 33-42.

[2] J. M. P. Cardoso, H. C. Neto, "An Enhanced Static-List Scheduling Algorithm for Temporal Partitioning onto RPUs", *Proc. 10th Intl. Conf. on VLSI*, 1999, pp. 485-496.

[3] R. Maestre, F. J. Kurdahi, R. Hermida, N. Bagherzadeh, H. Singh, "A Formal Approach to Context Scheduling for Multicontext Reconfigurable Architectures", *IEEE Trans. on VLSI Systems*, Vol. 9, No. 1, Feb. 2001, pp. 173-185.

[4] M. Sanchez-Elez, M. Fernandez, R. Maestre, R. Hermida, N. Bagherzadeh, F. J. Kurdahi, "A Complete Data Scheduler for Multi-Context Reconfigurable Architectures", *Proc. Design, Automation and Test in Europe*, 2002, pp. 547-552.

[5] O. Diessel, H. El Gindy, M. Middendorf, H. Schmeck, B. Schmidt, "Dynamic scheduling of tasks on partially reconfigurable FPGAs", *IEE Proc.-Computer Digital Technology*, Vol. 147, No. 3, May 2000, pp. 181-188.

[6] M. Vasilko, DYNASTY: A Temporal Floorplanning Based CAD Framework for Dynamically Reconfigurable Logic Systems, *Proc. 9th Intl. Workshop on Field-Programmable Logic and Applications*, 1999, pp.124-133.

[7] M. Teich, S. Fekete, J. Schepers, "Compile-time optimization of dynamic hardware reconfigurations", *Proc. Intl. Conf. on Parallel and Distributed Processing Techniques and Applications*, 1999, pp. 1097-1103.

[8] M. G. Gericota, G. R. Alves, M. L. Silva, J. M. Ferreira, "Active Replication: Towards a Truly SRAM-based FPGA On-Line Concurrent Testing", *Proc. 8th IEEE Intl. On-Line Testing Workshop*, 2002, pp. 165-169.

[9] M. G. Gericota, G. R. Alves, M. L. Silva, J. M. Ferreira, "On-line Defragmentation for Run-Time Partially Reconfigurable FPGAs", *Proc. 12th Intl. Conf. on Field Programmable Logic and Applications*, 2002, pp. 302-311.

[10] Politécnico di Torino ITC'99 benchmarks. http://www.cad.polito.it/tools/itc99.html

[11] IEEE Std. Test Access Port and Boundary Scan Architecture (IEEE Std 1149.1), *IEEE Std. Board*, May 1990.

[12] W. Huang, E. J. McCluskey, "A Memory Coherence Technique for Online Transient Error Recovery of FPGA Configurations", *Proc. 9th ACM Intl. Symposium on Field--Programmable Gate Arrays*, 2001, pp. 183-192.

[13] S. A. Guccione, D. Levi, P. Sundararajan, "JBits Java based interface for reconfigurable computing", *Proc. 2nd Military and Aerospace Appl. of Prog. Devices and Technologies Conf., 1999.*