Manfred Glesner
Peter Zipf
Michel Renovell (Eds.)

LNCS 2438

Field-Programmable Logic and Applications

Reconfigurable Computing Is Going Mainstream

12th International Conference, FPL 2002
Montpellier, France, September 2002
Proceedings

Springer

On-line Defragmentation for Run-Time Partially Reconfigurable FPGAs

Manuel G. Gericota[1], Gustavo R. Alves[1], Miguel L. Silva[2], José M. Ferreira[2]

[1] Department of Electrical Engineering – DEE/ISEP
Rua Dr. António Bernardino de Almeida, 4200-072 Porto – PORTUGAL
{mgg, galves}@dee.isep.ipp.pt
http://www.dee.isep.ipp.pt/~mgg/indexe.htm
[2] Dep. of Electrical and Computers Engineering – DEEC/FEUP
Rua Dr. Roberto Frias, 4200-465 Porto – PORTUGAL
{mlms, jmf}@fe.up.pt

Abstract. Dynamically reconfigurable systems have benefited from a new class of FPGAs recently introduced into the market, which allow partial and dynamic reconfiguration at run-time, enabling multiple independent functions from different applications to share the same device, swapping resources as needed. When the sequence of tasks to be performed is not predictable, resource allocation decisions have to be made on-line, fragmenting the FPGA logic space. A rearrangement may be necessary to get enough contiguous space to efficiently implement incoming functions, to avoid spreading their components and, as a result, degrading their performance. This paper presents a novel active replication mechanism for configurable logic blocks (CLBs), able to implement on-line rearrangements, defragmenting the available FPGA resources without disturbing those functions that are currently running.

1 Introduction

Reconfigurable logic devices, namely Field Programmable Gate Arrays (FPGAs), experienced a considerable expansion in the last few years, due in part to an increase in its size and complexity, with gains in board space and flexibility. With the advent of a new kind of SRAM-based FPGAs, capable of implementing fast run-time partial reconfiguration (e. g. the Virtex family from Xilinx used to validate this work), the advantages of these devices were considerably reinforced, wide-spreading their usage as a base for reconfigurable computing platforms.

The new features offered by these devices enabled the concept of "virtual hardware", where resources are supposed to be unlimited and those implementations that exceed the reconfigurable area are resolved by temporal partitioning. The static implementation of a circuit is separated in two or more independent hardware contexts, which may be swapped during runtime [1]. Extensive work is under way to improve the capability of these devices to handle multi-context, by storing several

♦ This work is supported by the Portuguese Foundation for Science and Technology (FCT), under contract POCTI/33842/ESE/2000.

M. Glesner, P. Zipf, and M. Renovell (Eds.): FPL 2002, LNCS 2438, pp. 302-311, 2002.

configurations and enabling quick context switching [2, 3]. The main objective is to improve the execution time by minimising external memory transfers, assuming that some amount of on-chip data storage is available in the reconfigurable architecture. However, this solution is only viable if the functions implemented on hardware are mutually exclusive on the temporal domain; otherwise, the length of reconfiguration intervals would imply delays unacceptable to most applications.

An application comprises a set of functions that are frequently 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 more than 100% of the FPGA resources, by swapping functions in and out of the FPGA as needed.

Partial reconfiguration times are in the order of microseconds, 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 reduced to zero, if new functions are swapped in advance with those already out of use, as illustrated in figure 1. The reconfiguration interval (r_t) refers to the period where the configuration of a new function may be performed in order to be available when required by the application flow (and 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. Delays will therefore be introduced in the application execution, systematically or not, depending on the application flow.

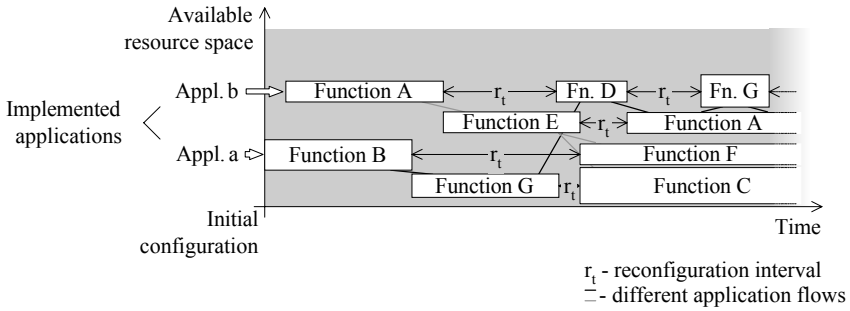

Fig. 1. Temporal scheduling of functions.

The FPGA logic space comprises three types of elements:

- Configurable Logic Blocks (CLBs);
- Input/Output Blocks (IOBs);
- Routing resources.

All these resources may be shared in the spatial and/or temporal domains. The goal is to allocate to each function as much resources as it needs to execute independently of all the others, as if it were the sole application running on a chip just large enough to support it. The partitioning of the FPGA logic space in a three-dimensional basis,

two spatial dimensions and one temporal, is addressed in [4]. The resources are shared among multiple independent functions, each having its own spatial and temporal requirements. As the resources are allocated to functions and later released, many small areas of free resources are created. These portions of unallocated resources tend to become so small that they fail to satisfy any request and so remain unused – the FPGA logic space gets fragmented. This problem is illustrated in figure 2 in the form of a 3-D floorplan and function schedule [5], where each shadow area corresponds to the optimal space occupied by the implementation of a single function.

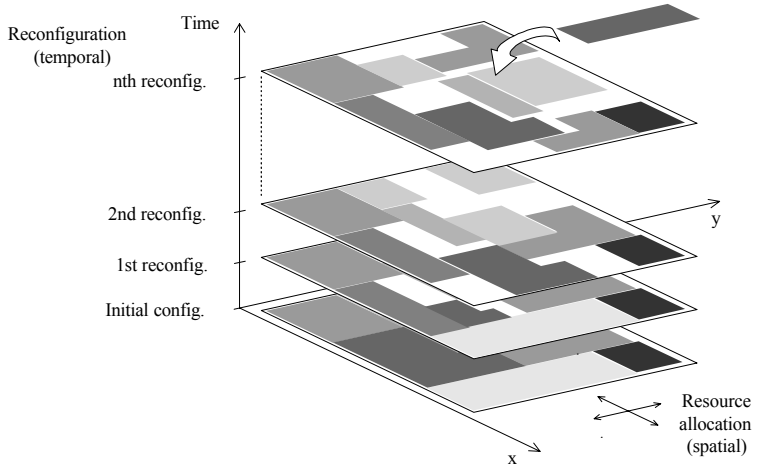

Fig. 2. 3-D representation of the fragmentation problem.

If the requirements of functions and their sequence are known in advance, suitable arrangements can be designed and sufficient resources can be provided to handle them in parallel [6]. However, when placement decisions need to be made on-line, it is possible that a lack of contiguous free resources will prevent functions from executing, even if the total number of resources available is sufficient. Notice that spreading the components of incoming functions by the available resources would lead to a degradation of its performance, delaying tasks from completion and reducing the utilisation of the FPGA.

When a new function cannot be allocated immediately due to lack of contiguous resources, a suitable rearrangement of a subset of the executing functions might solve the problem. Three methods are proposed in [4] to find such (partial) rearrangements, with the goal to increase the rate at which waiting functions are allocated, while minimising disruptions to executing functions that are to be moved. However, the physical execution of these rearrangements implies halting currently running functions and consequently halting their respective applications. A mechanism to implement the rearrangements without disturbing the running functions is presented in this paper. To address this problem, a new concept is introduced – the active replication –, which enables the relocation of each FPGA CLB (and therefore of its associated routing) even if it is active, i.e. the CLB is part of an implemented function that is actually being used by an application [7]. This concept enables the

defragmentation of the FPGA logic space on-line, without introducing time overheads.

We start by describing the active CLB replication and releasing mechanism, followed by a summary of its limitations, be it due to the FPGA architecture or to the particular configuration mechanism used.

2 Resource replication and release

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, whose configuration is controlled by a set of memory cells that lies beneath.

The implementation of an on-line defragmentation strategy implies the use of a dynamic replication mechanism, where CLBs currently being used by a given function have their functionality relocated to other CLBs, without disturbing function execution. The replicated CLBs are then free to be allocated to new functions. One of the major problems facing the implementation of such replication mechanism is the dynamic replication of active elements, be it CLBs or routing resources. Dynamically replicating an active CLB is not just a matter of relocating its functional specification: the corresponding interconnections with the rest of the circuit have to be established; additionally, internal state information also has to be copied, depending on the functionality it is implementing. All these actions must be carried out without interfering with its normal operation. The same happens when dealing with routing resources, although their replication is a simpler job.

The transparent release of active CLBs it is not trivial, due to two major issues: i) configuration memory organization and ii) internal state information.

The configuration memory can be visualized 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. One frame is the smallest unit of configuration that can be written to or read from the configuration memory. These 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 replicating an active CLB 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 replicated 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 must be perfectly stable before its outputs are connected to the system, so as to avoid output glitches.

A set of experiments performed with a XCV200 from Xilinx demonstrated that the only possible solution is to divide the replication procedure in two phases, as illustrated in figure 3.

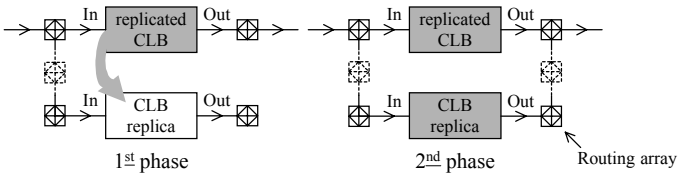

Fig. 3. Two-phase CLB replication procedure.

In the first phase, the internal configuration of the CLB is replicated and the inputs of both CLBs are placed in parallel. Due to the slowness of the reconfiguration procedure, when compared with the application speed of operation, 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 (replicated and 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 replication procedure.

Another major requirement for the success of the replication procedure is the correct transfer of state information. If the current CLB function is purely combinational, a simple read-modify-write configuration procedure will suffice to accomplish the replication. However, in the case of a sequential function, the internal state information must be preserved and no writes could be lost during the copying procedure. 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.

When dealing with synchronous free-running clock circuits, the two-phase replication procedure described earlier is a good solution. Between the first and the second phase, the CLB replica has the same inputs as the replicated CLB, and all its four flip-flops acquire the state information. Several experiments made using this class of circuits have shown the effectiveness of this method in the replication of active CLBs. No loss of state information or the presence of output glitches was observed.

Despite the effectiveness of this solution, its restriction to synchronous free--running clock circuits is a serious limitation. A broad range of circuits uses gated--clocks, where input acquisition by the flip-flop is controlled by the state of the clock enable signal. As we cannot ensure that this signal will be active between the first and the second phase of the replication procedure, it is uncertain that the CLB replica will capture the state information. On the other hand, it is not feasible to set this signal as part of the replication procedure, because the value present at the input of the replica flip-flops may change in the meantime, and a coherency problem will then occur.

A replication aid block was used to solve this problem, which manages the transfer of the state information from the replicated flip-flops to the replica flip-flops, while

enabling their update by the circuit at any instant, without delaying the replication procedure. The whole replication scheme is represented in figure 4, where only one logic cell is shown, for reasons of simplicity. Each CLB comprises four of these cells, which can be considered individually for the purpose of implementing this procedure. The temporary transfer paths established between the replicated cells and their replica do not affect their functionality, since they use only free routing resources and do not modify their structure.

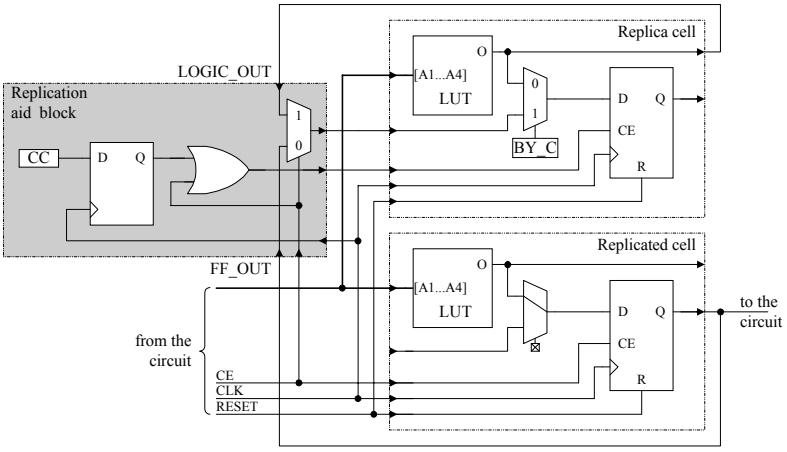

Fig. 4. Implementation of the gated-clock flip-flop replication scheme.

The inputs of the 2:1 multiplexer present in the replication aid block receive one temporary transfer path from the output of the replicated flip-flop (FF_OUT), and another one, in this example, from the output of the Look-Up Table (LUT) in the replica logic cell (LOGIC_OUT), which is normally applied to the flip-flop. If the LUT in the logic cell being replicated is not used by the current implementation, the input of the flip-flop cell will be directly connected to one of the cell inputs. In this case, LOGIC_OUT will be connected to that input. The multiplexer is controlled by the clock enable signal (CE) of the replicated flip-flop. If this signal is not active, the output of the replicated flip-flop (FF_OUT) is applied to the input of the replica flip--flop. A clock enable signal, generated by the replication aid block (capture control signal - CC), forces the replica flip-flop to hold the transferred value. The replica flip--flop acquires the state information present in the replicated flip-flop. If the CE signal is active or is activated during this procedure, the multiplexer selects the LOGIC_OUT signal and applies it to the input of the replica flip-flop, which is updated at the same time and with the same value as the replicated flip-flop, guaranteeing state coherency. Figure 5 represents the flow diagram describing the replication procedure, while figure 6 shows the waveform simulation of state transfer and update operations during the replication procedure. No loss of information or functional disturbance was observed during the execution of the procedure.

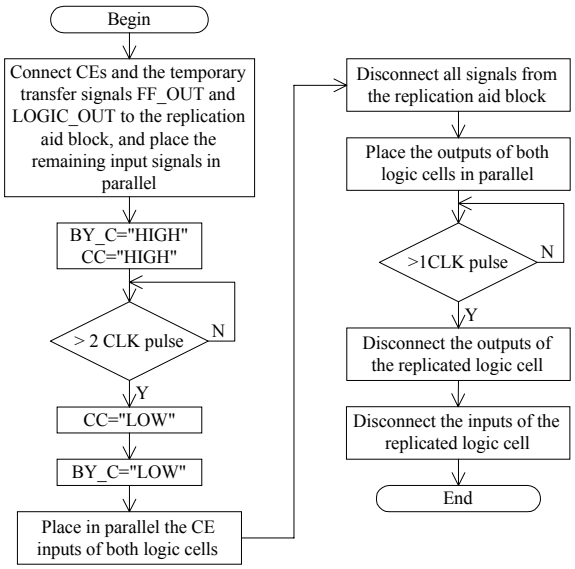

Fig. 5. Replication procedure flow.

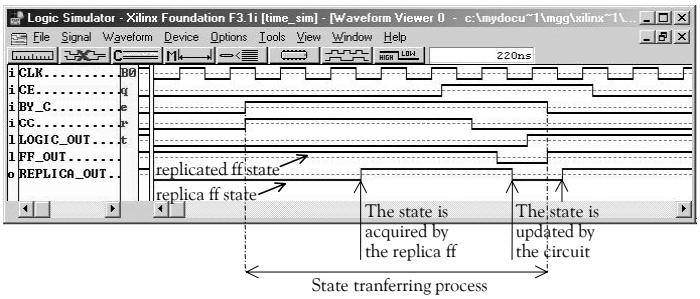

Fig. 6. Simulation of a state transfer and update during the replication procedure.

The control signals CC and BY_C are configuration memory bits whose values are driven through reconfiguration of the configuration memory. BY_C directs the state signal to the input of the replica flip-flop, while CC enables its acquisition. It is therefore possible to control the whole replication procedure through the configuration interface used, and as such no extra FPGA pins are required.

Each replication aid block makes use of one CLB slice. Since four of these blocks are required to replicate the four logic cells of a CLB, two extra CLBs will be required to implement this procedure. However, this overhead is only apparent, since the occupancy of those CLBs takes place only during the replication procedure. After the replication they become free to be used by new functions.

Since we decided to control all signals through the configuration memory, so as not to use extra FPGA pins, the CC net includes the flip-flop shown in figure 4. However,

it is there simply as a consequence of the structure of the CLB slice, and does not play any role in the implementation of the procedure that was described.

After the state has been transferred, the input signals involved in the execution of the replication procedure are placed in parallel, all the signals to and from the replication aid block are disconnected (and consequently the occupied CLBs are free), and the outputs are also placed in parallel. After at least one function clock cycle the replicated block is disconnected and becomes free.

Practical experiments performed over the ITC'99 Benchmark Circuits from the Politécnico di Torino [8] implemented in a Virtex XCV200 proved 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 CLB replication is performed individually. Only one clock signal is involved in the replication of each CLB, even if many of these blocks were processed simultaneously.

Using the Boundary Scan [9] interface to perform the reconfiguration, the average replication time of each CLB implementing synchronous gated-clock circuits is about 22,6 ms, for a 20 MHz frequency of the test clock.

This method is also effective when dealing with asynchronous circuits, if transparent data latches are used instead of flip-flops. In this case, the CE signal is replaced in the latch by the input control signal G. Data present in the D input is stored when the control input G changes from '1' to '0'. The same replication aid block is used and the same replication sequence is followed. The register present in the replication aid block may be configured as a latch, instead as a flip-flop, if this is preferred or if no adequate clock signal is available.

In the Virtex family of FPGAs, LUTs can be configured as Distributed RAMs. However, the extension of this on-line replication concept to the replication of those LUT/RAMs is not viable. The content of the LUT/RAMs could be read and written through the configuration memory, but there is no feasible way, other than to stop the function, capable of ensuring data coherency, if there is a write attempt during the replication interval, as stated in [10]. 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. We must ensure that either all the remaining data in the slice is constant, or it is also modified externally through partial reconfiguration. Even if not being replicated, LUT/RAMs should not lie in any column that could be affected by the replication procedure.

3 Reconfiguring routing resources

Each CLB comprises, in addition to its logic resources, three routing arrays: two local (input and output) and one global. The routing resources in these arrays may be unidirectional or bi-directional, as indicated in figure 7. No routing resources are available in the local arrays to establish direct interconnections with other CLBs, so those required by the replication procedure can only be established through the global routing array. Between local and global routing arrays, only unidirectional routing resources are available, as seen in figure 7.

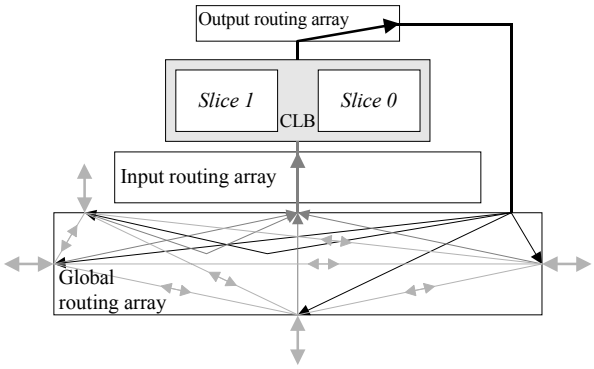

Fig. 7. CLB routing arrays resources.

To place the inputs in parallel, interconnection segments between global arrays may be unidirectional (from the replicated CLB inputs towards the CLB replica inputs), or bi-directional. Concerning the outputs, interconnection segments between global arrays may also be unidirectional (from the CLB replica outputs towards the replicated CLB output), or bi-directional. Otherwise, since signals do not propagate backwards, no signals will exist at the inputs of the CLB replica, and the outputs of both CLBs will not be in parallel. As a result, output glitches will occur when the outputs of the replicated CLB are disconnected from the system, and no signals will be propagated to the rest of the circuit.

As mentioned before, the relocation of functional active CLBs should have a minimum influence (preferably none) in the system operation, as well as a reduced overhead in terms of reconfiguration cost. This cost depends on the number of reconfiguration frames needed to replicate and free each CLB, since a great number of frames would imply a longer rearrangement time. The impact of the replication procedure in the running functions is mainly related to the delays imposed by re--routed paths, since the relocation procedure might imply a longer path, thus diminishing their maximum frequency of operation (the longest path delay determines 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 in 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, in order to avoid an excessive increase in path delays.

The replication of routing resources – in order to rearrange their positioning and to free some segments to be used in incoming functions – posed no problems. The same two-phase replication procedure is effective on the replication and release of local and global active interconnections. The interconnections involved are first duplicated in order to establish an alternative path, and then disconnected, becoming available to be reused. Notice that due to the lack of routing resources following the replication of the CLB, it might be necessary to perform a rearrangement of the routing

interconnections, in order to optimize them or to increase the availability of routing resources to incoming functions.

4 Conclusion

This paper presented a novel replication procedure able to replicate active CLBs without halting their operation. The proposed procedure enables the implementation of a truly on-line non-intrusive FPGA logic space defragmentation, which allows the dynamic scheduling of tasks in the spatial and temporal domains. It is therefore possible that several applications 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.

References

1. Cardoso, J. M. P, Neto, H. C.: An Enhanced Static-List Scheduling Algorithm for Temporal Partitioning onto RPUs. *Proc. of the 10th Intl. Conf. on Very Large Scale Integration*, Lisbon, Portugal, 1999, pp. 485-496.
2. Maestre, R., Kurdahi, F. J., Hermida, R., Bagherzadeh, N., Singh, H., A Formal Approach to Context Scheduling for Multicontext Reconfigurable Architectures. *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 9, No. 1, February 2001, pp. 173-185.
3. Sanchez-Elez, M., Fernandez, M., Maestre, R., Hermida, R., Bagherzadeh, N., Kurdahi, F. J.: A Complete Data Scheduler for Multi-Context Reconfigurable Architectures. *Proc. of the Design, Automation and Test in Europe*, Paris, France, 2002, pp. 547-552.
4. Diessel, O., ElGindy, H., Middendorf, M., Schmeck, H., Schmidt, B.: Dynamic scheduling of tasks on partially reconfigurable FPGAs. *IEE Proc.-Computer Digital Technology*, Vol. 147, No. 3, May 2000, pp. 181-188.
5. Vasilko, M.: DYNASTY: A Temporal Floorplanning Based CAD Framework for Dynamically Reconfigurable Logic Systems. *Proc. of the 9th International Workshop on Field-Programmable Logic and Applications*, Glasgow, Scotland, 1999, pp.124-133.
6. Teich, M., Fekete, S., Schepers, J.: Compile-time optimization of dynamic hardware reconfigurations. *Proc. of the Intl. Conf. on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, USA, 1999.
7. Gericota, M. G., Alves, G. R., Silva, M. L., Ferreira, J. M.: Dynamic Replication: The Core of a Truly Non-Intrusive SRAM-based FPGA Structural Concurrent Test Methodology. *3nd IEEE Latin-American Test Workshop Digest of Papers*, Montevideo, Uruguay, 2002, pp. 70-75.
8. Politécnico di Torino ITC'99 benchmarks. *http://www.cad.polito.it/tools/itc99.html*
9. *IEEE Standard Test Access Port and Boundary Scan Architecture (IEEE Std 1149.1)*, IEEE Std. Board, May 1990.
10. Huang, W., McCluskey, E. J.: A Memory Coherence Technique for Online Transient Error Recovery of FPGA Configurations. *Proc. of the 9th ACM Int. Symposium on Field-Programmable Gate Arrays*, Monterey, California, 2001, pp. 183-192.