On-line Diagnosis and Reconfiguration of FPGA Systems

Anna Antola*, Vincenzo Piuri+, Mariagiovanna Sami*
*Department of Electronics & Information, Politecnico di Milano, Italy, {antola,sami}@elet.polimi.it
+Department of Information Technologies, University of Milan, Italy, piuri@dti.unimi.it

Abstract

Fault tolerance is becoming an important issue for the effective use of FPGA-based architectures in mission-critical applications.

This paper introduces an innovative approach to design FPGA systems with on-line diagnosis and reconfiguration, at a limited cost in terms of FPGA redundant resources and interconnections. The technique is based on high-level synthesis of the self-checking datapath to be mapped on the FPGA. The analysis of the computation flow allows for locating the necessary checking points. Scheduling is performed so as to minimize the circuit complexity, while satisfying the maximum latency allowed by the application. Allocation is performed as a suited trade-off between the circuit complexity and the reconfiguration efficiency.

Problems and constraints due to re-use of units in different points of the computation are taken into account. The faulty block replacement policy will be discussed, together with its implication in terms of re-use and of interconnection re-routing.

1. Introduction

FPGA devices are widely used to implement complex systems also in applications that increasingly require capacity of autonomous self-checking and possibly of survival to faults, often within cost limitations that exclude massive redundancy.

To introduce fault tolerance in such systems, three basic approaches have been considered in the literature. The first one considers the FPGA at a low abstraction level, as a matrix of basic processing elements (the configurable logic blocks – CLBs – of the device) linked by switched interconnecting busses. The specific characteristics of the application system mapped onto the FPGA are not taken into account. Detection is performed by means of conventional techniques [1-4]. When an error is detected, the faulty FPGA block is identified, removed from the active computation and replaced by spare elements in order to preserve the nominal detection ability as long as possible. To this end, the interconnection structure is properly reconfigured by using one of the many reconfiguration techniques presented in the literature for regular array architectures. The interconnections and the configuration memory are usually considered fault free. The main drawbacks of this approach are due to the hardware redundancy needed to support detection (in particular, if checking is performed at low granularity) and to the fact that reconfiguration may be hampered by lack of sufficient redundant paths

A second approach adopts a high-level view of the system. Fault detection techniques are introduced at application system level. Checking is executed by analyzing the intermediate or final application outputs, according to the adopted detection technique. There is no relationship between the location of the checking operation and the array structure of the underlying FPGA device. Whenever an error is detected, a diagnostic phase must be introduced in order to locate the faulty CLB (e.g., [5-15]). Reconfiguration is then accomplished by excluding such element from the active computation through interconnection reconfiguration and spare activation. In this approach, circuit complexity and performance loss are reduced with respect to the previous case; however, the separate diagnostic phase leads to longer repair time, since appropriate testing need to be executed (obviously, reconfiguration is not "transparent" as far as time is concerned). Here also, reconfiguration may incur in lack of sufficient routing paths, although this case is less likely than in the previous approach since rerouting must be completely recomputed.

The third approach augments the physical structure of the FPGA device to directly support fault detection and confinement [16-22]. The basic CLBs and interconnections are modified so that detection is performed by internal check in every CLB. This solution is highly effective and efficient, but requires large circuit complexity. Besides, it is not applicable when COTS architectures are envisioned.

The present paper introduces an innovative approach that aims at limiting the circuit complexity required by concurrent detection and by subsequent reconfiguration. The solution relates to high-level synthesis of such systems that could be efficiently represented by the "Controller-Datapath" model (also called FSM-Datapath

Proceedings of the First IEEE International Workshop on Electronic Design, Test and Applications (DELTA'02)
0-7695-1453-7/02 $17.00 © 2002 IEEE

model); in particular, the paper focuses on datapath design (design of self-checking sequential machines has been afforded, e.g., in [23]) and the use of coding as detection technique. The proposed approach starts from a computation – described in any high-level language (e.g., C, behavioral VHDL, etc.) – that has to be implemented by an application-specific circuit on FPGA; from such computation, Data Flow Graph and Control Flow Graph are extracted. For simplicity – without loss of generality – we refer to Data Flow Graphs involving arithmetic operations only (this allows us to adopt for our case study one form of error-detection coding, namely arithmetic ECC).

The main goal is achieving concurrent self-checking with relatively low redundancy, and then effecting reconfiguration based on results of such self-checking mechanism – again, keeping redundancy low with respect to the number of faults that can be overcome.

2. The high-level synthesis approach

Our basic idea to achieve fault tolerance in FPGAs by means of reconfiguration consists in partitioning the FPGA architecture into groups of FPGA resources (called *tiles*) on which portions of the computation are mapped. Most of the tiles are used to implement functional blocks of the desired computation. More functional units can be mapped in the same tile. Some tiles (the *spare tiles*) are left unused so that they can be exploited to replace the faulty tiles in reconfiguration to allow the system survival. The overall scheme is shown in Fig. 1.

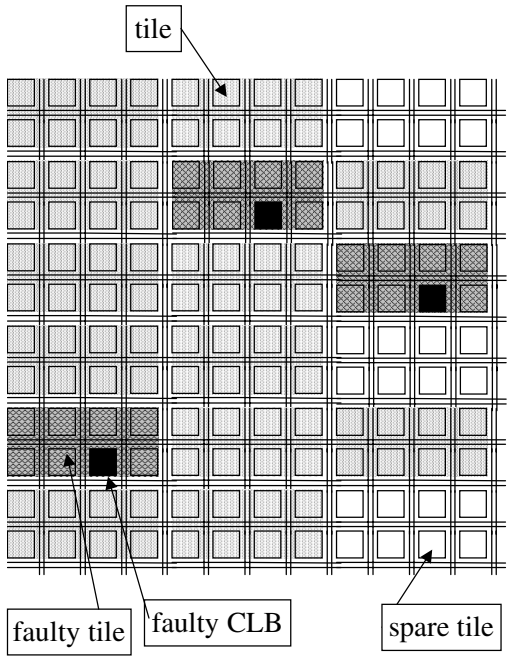

Figure 1. Tiles in the FPGA architecture

To minimize the circuit complexity of the system, the high-level synthesis algorithm should maximize the reuse of components to perform the nominal computation. On the other hand, to minimize the circuit complexity required by checking and reconfiguration, we aim at partitioning the computation in functional blocks that have mainly the following properties:

- the number of checking points that are needed to verify the presence of errors due to faults should be limited,
- the number of input/output connection of the functional blocks should be limited so that the number of interconnection lines needed for reconfiguration is limited,
- the dimensions (i.e., the number of FPGA resources) of the circuits implementing the functional blocks should be sufficiently homogeneous to minimize the unused resources in the spare tiles when faulty tiles are reconfigured.

The objective of our approach is therefore to balance two conflicting goals: to have a good reuse of components and to support efficiently the reconfiguration. This task is very complex if the global optimum solution is desired since scheduling and allocation are strongly dependent on the requirements imposed by the mapping on FPGA tiles and by the reconfiguration approach. These interdependencies make the problem NP-complete and, consequently, induce to adopt a heuristic approach.

To identify the functional blocks that can be verified independently with the minimum number of checkers while guaranteeing the detection ability, we adopt the high-level synthesis approach proposed in [24,25], that will be briefly recalled in section 3. From this methodology we use the technique for computation partitioning which leads to the scheduled self-checking structure. The self-checking approach allows for locating concurrently, with limited redundancy and without requiring a specific diagnostic phase, the FPGA portions that contain the fault. Operations are scheduled according to a time-constrained approach, e.g. based on force-directed algorithms. The scheduling solution is frozen for the subsequent allocation algorithm that completes the system synthesis.

In this paper, we introduce an innovative allocation algorithm, specifically designed to deal with the conflicting circuit complexity issues mentioned above for reconfiguration. This approach reuses the physical units as much as possible to limit the overall circuit complexity, while not increasing too much the complexity in interconnections for reconfiguration. An unused unit will not be reused if this will lead to a high increase of the interconnections. Finally, we will discuss how the functional blocks can be mapped on the FPGA tiles in the perspective of an efficient reconfiguration: constraints and guidelines are introduced on the definition of the size of tiles and, consequently, on the allocation algorithm.

3. The scheduled self-checking structure

Concurrent self-checking is achieved by segmenting the whole Data Flow Graph into subgraphs (called *detectable subgraphs*, DS) such that checking may be performed only on the output of each subgraph without risk of aliasing (this operation is performed prior to scheduling and allocating the DFG). This allows to reduce the number of checking points within the Data Flow Graph (rather than checking each individual operation), as well as reducing the total latency; the number of checking points is an upper bound for the number of self-checking checkers that will be required in the final implementation (in general, a checker may be re-used for different checking points scheduled in different control steps). The basic philosophy has been detailed in [24,25]; a simple example is shown in Fig. 2.

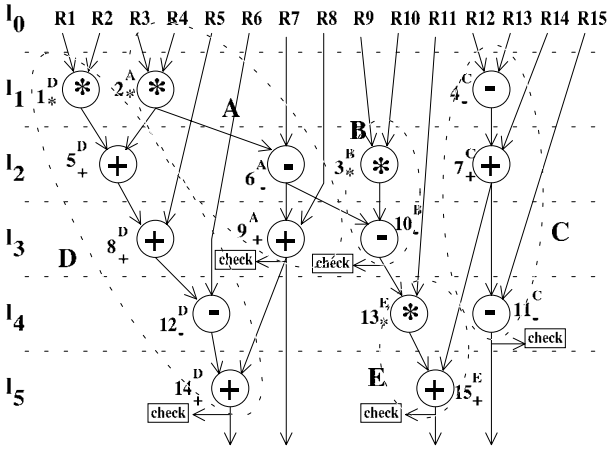

Figure 2. Detectable Subgraphs in a simple DFG

Having extracted all maximum *DS*s that grant absence of aliasing, a scheduling-and-allocation approach has been proposed to achieve the coverage of the given DFG by means of disjoint *DS*s that will limit the number of resources required within the constraints imposed by the self-checking approach. Errors are located at the level of the DS, which corresponds – in general – to a number of functional units and registers. It should be immediately noted here that – in general – the DFG is not directly mapped onto a physical structure; in other words, units (functional units and registers, as well as registers) will be re-used during the allocation phase of different subgraphs of the given DFG. Thus, even in the single-fault assumption, if the whole computation were allowed to proceed from beginning to end, the checkers would in general signal multiple errors whenever the faulty unit is re-used. Actually, the system grants identification of one fault at a time; if reconfiguration is allowed, subsequent faults can in turn be detected and overcome.

The self-checking approach allows to isolate concurrently, with limited redundancy and without requiring a specific diagnostic phase, subsections of the FPGA in which the fault is confined (corresponding to the resources upon which the DS has been mapped). For the subsequent reconfiguration phase, possible reuse by different *DS*s of units within such subsection has to be taken into account; in fact, all units in the "declared faulty" subgraph are excluded from active computation and remapped onto a fault-free section of the FPGA. All connections leading to such units are reconfigured following one of the known approaches (see, e.g., [18,19,26-33]). In fact, lower reconfiguration complexity can be achieved whenever FPGA sections mapping individual *DS*s can be re-used "in toto" within the mapping of larger *DS*s. Then, the re-usable subgraph becomes an atomic tile on the FPGA, and spare identical tiles are reserved on the spare area. Whenever a working tile is declared faulty, reconfiguration of the *DS*s using it implies only rerouting of inputs and outputs to the tile rather than to individual units.

With respect to the low-level approach proposed in the literature, our solution limits the required redundancy since checking points and checkers are introduced to protect groups (possibly large) of FPGA resources on which subgraphs are mapped. With respect to the high-level approach, our technique avoids the diagnosis time since it is able to perform it on line during checking; only extraction and analysis of the few error signals produced by the partitions are needed.

4. The allocation algorithm

The allocation algorithm uses the schedule produced as discussed in the previous section starting from the disjoint *DS*s to map operations onto functional units, by taking into account the reconfiguration needs.

Thus, the allocation algorithm must pursue two goals:
- to allocate the operations onto functional units,
- to allocate the functional units into the tiles,

so that the overall circuit complexity is limited and the reconfiguration feasible with reduced interconnection overhead to re-map the faulty tiles.

From the point of view of reconfiguration simplicity, mapping each individual *DS* (including its checker) on a different tile would be optimum since a one-to-one correspondence is created: fault detection is thus followed by a trivial reconfiguration phase (only units involved in the fault-affected *DS* would be reconfigured). Redundancy could still be kept lower than straightforward system duplication by allowing several *DS*s to be reconfigured onto one spare tile. Still, good resource efficiency for the nominal circuit requires in general that initial allocation of the scheduled DFG involve *reuse* of components, so that

the faulty unit cluster will be associated with more than one *DS* and reconfiguration will affect all such *DS*s. In fact, allocation of some DFGs may involve such an intensive reuse of functional units that detection of a fault associated with one *DS* would lead to discard *all* units allocated to the DFG and thus, once more, to massive replication to achieve fault tolerance. Even outside such extreme instance, reuse of units in a faulty tile by a multiplicity of *DS*s will create problems in reconfiguration (e.g., some *DS* will use other tiles outside the faulty one) and require very complex rerouting.

A viable solution consists in a tradeoff between these two competing goals: we allow more than one *DS* in each tile so as to balance the area increase due to the missing reuse of functional units and the difficulties in interconnections for reconfiguration. The group of *DS*s mapped on the same tile is called a cluster of *DS*s. To simplify the reconfiguration we assume that one *DS* belongs to only one tile so that only one tile needs to be reconfigured when the fault appears.

Since *DS*s have in general different size, the size of the spare tile to be used to reconfigure a tile of a group must be large enough to host the largest tile of that group. To minimize the unused spare area, the size of the tiles of the group should be quite similar.

Thus, the allocation problem consists of identifying:
- the cluster of *DS*s to be mapped on each tile and
- the allocation of the operations belonging to a cluster onto functional units realized in the tile for the cluster.

The first critical aspect is the partitioning of *DS*s into disjoint clusters that will be mapped on tiles. To this purpose we need to define a criterion for compatibility among *DS*s that allows for creating clusters with the desired characteristics discussed above.

We define the *resource compatibility* (*r-compatibility*) between *DS*s as the possibility of reusing functional units (including checkers) among them. Formally: two *DS*s are *r-compatible* if there is at least one functional unit (including checkers) that can be reused to perform at least one operation in each of the two *DS*s while satisfying the constraints imposed by the adopted error detection technique. If the allocation algorithm is not able to reuse any functional unit between the two *DS*s, they are not *r-compatible*.

In general, it will be advantageous to put two *r-compatible DS*s into the same cluster and, thus, to map them onto the same tile. In fact, let's suppose that they are mapped onto two different tiles: each functional unit that they share will be mapped on only one of the two tiles and appropriate interconnections will be introduced to support data transfer. When one of the tiles becomes faulty, it will be reconfigured onto a spare tile: this implies that all functional units mapped on that tile (in particular, those of the *DS* of the pair considered above) are moved and, consequently, all interconnections with the tile containing the *r-compatible DS* need to be rerouted to guarantee the data transfer. This often requires a high number of interconnections. If the two *DS*s are clustered and mapped in the same tile, both are moved by reconfiguration and thus no complex rerouting is required between them by reconfiguration.

The disjoint clusters to be mapped on the tiles can be created by analyzing the *r-compatibility* among all *DS*s and by identifying which groups are disjoint or can be made disjoint at a reasonable cost in terms of redundancy by introducing additional functional units to avoid the increased interconnection redundancy required by reconfiguration. To perform this analysis, we introduce the *r-compatibility graph*, whose nodes are the *DS*s and whose arcs represent the *r-compatibility* relationships (an arc between two nodes exists only if the *DS*s represented by the nodes are *r-compatible*).

If a clique is disjoint from the other nodes of the graph, the *DS*s represented by the clique's nodes are clustered and mapped in the same tile since they reuse functional units only among themselves. Moving these *DS*s does not require complex rerouting for reconfiguration: only inter-tile connections are to be rerouted.

A large clique (i.e., a clique that contains *DS*s having many functional units) can be an unmanageable cluster for reconfiguration. Consequently, we need to partition the large clique into smaller ones (each requiring less circuit complexity) that can be easier treated by reconfiguration.

Since more than one functional unit could satisfy the definition of *r-compatibility*, possibly for different types of operations, different strength of the *r-compatibility* can be envisioned to evaluate the actual advantage of clustering two *DS*s for subsequent mapping on the same tile. To this purpose we introduce the *cluster cost* as the function C_i that measures the cost of clustering a group of *DS*s into the same cluster c_i, defined by:

$$C_i = \alpha\, U_i + \beta\, S_i - \gamma\, R_i =$$
$$= \alpha \sum_{f_k \in U(c_i)} u_k + \beta \sum_{f_k \in S(c_i)} u_k - \gamma \sum_{f_k \in R(c_i)} u_k$$

where:
- U_i is the cost of the functional units of the cluster in terms of circuit complexity;
- S_i is the penalty cost to discourage the non-reuse of functional units; it is proportional to the circuit complexity of the functional units that are used by operations in only one *DS* of the cluster;
- R_i is the premium term to encourage the reuse of functional units: it measures the advantage of reusing functional units available in a tile when some *DS* are mapped on that tile: it is the cost (in terms of circuit complexity) of the functional units that were saved in mapping the *DS*s of the cluster on the same tile with respect to mapping on different tiles;

- α, β, and γ are parameters weighting the relevance of each component in the cluster cost; α is related to the minimization of the overall circuit complexity of the cluster, β to the minimization of the single use of functional units, and γ to the maximization of the reuse;
- u_k is the cost of the functional unit f_k in terms of circuit complexity in the cluster c_i;
- $U(c_i)$ is the set of functional units f_k required to implement the *DS*s grouped in the cluster c_i;
- $S(c_i)$ is the set of functional units f_k that are used only by one *DS* in the cluster c_i;
- $R(c_i)$ is the set of functional units f_k that were not needed to realize the computation of the *DS*s grouped in the cluster c_i with respect to the case in which all *DS*s of the cluster are mapped on different tiles.

The *system cost* *C* induced by the partitioning into disjoint cliques adopted for the *r-compatibility graph* is therefore:

$$C = \sum_i C_i$$

Splitting a large clique of the *r-compatibility graph* to achieve manageable cliques for reconfiguration at limited redundancy cost can be obtained by removing some *r-compatibility* arcs so that the system cost *C* is minimum. This is obviously an NP-complete task and a heuristics need to be adopted.

A viable heuristic algorithm to solve the problem of grouping disjoint *DS*s into suited disjoint clusters by analyzing the *r-compatibility graph* is the following:

1. Create the complete *r-compatibility graph* and consider it as the current graph to be analyzed.
2. Extract the maximum clique (i.e., the clique with the highest number of nodes) from the current graph. This allows to try to maximize the reuse of functional units and, thus, to reduce the overall circuit complexity. If equivalent choices are available, the algorithm must explore the current graph generated by each alternative.
3. Verify that the clique extracted in step 2 is not too large to prevent an effective reconfiguration. In this case, it can be shown that the clique can be suitably split in two parts if the global circuit complexity of the functional units shared between the two parts is smaller than the maximum of the two unshared parts. If no effective splitting is possible, go back to step 2 and extract another clique having a number of nodes immediately not greater than the clique considered in the current iteration. If a large clique still remains despite of this search for smaller cliques, such clique is split anyway to allow reconfiguration.
4. If the current graph is not empty, go back to step 2.
5. For each partitioning generated by the above steps, the cost of each cluster and the system cost are evaluated. The clustering adopted as final solution is the one with the minimum system cost.

Heuristic rules can be envisioned to speed up the exploration of the alternative cliques by considering the cost of the clusters generated by these cliques. For example, the cluster having the lower cost should be preferred since it likely leads to a lower circuit complexity for the whole system. Heuristics may be also considered in partitioning the large cliques: for example, the partition that should be explored first is the one for which the cost of the related clusters is minimum.

Now that we have identified the clusters of *DS*s to be mapped onto the FPGA tiles, we can proceed to complete the solution of the overall allocation problem by creating the allocation of the DFG operations of a cluster onto the functional units available in its tile. This can be obtained by applying the allocation algorithm proposed in [24], which takes into account the constraints imposed by the error detection technique to avoid error aliasing within each *DS*. The algorithm is applied individually to each cluster of *DS*s.

In some cases, the tiles resulting from the previous partitioning algorithm may have too heterogeneous sizes. This may induce a waste of resources in the spare tiles and, thus, reduce the opportunities for successful reconfiguration. To tackle this problem, merging disjoint clusters of *DS*s into a single cluster can be considered to homogenize the resulting tile sizes.

5. Conclusions

An innovative high-level synthesis approach to design fault tolerant systems by using on-line detection and reconfiguration in FPGA architectures has been presented. The proposed idea partitions the dataflow path of the application circuit into disjoint subgraphs in which self-checking is achieved by using one of the error detection techniques available in the literature for the specific type of computation performed. These subgraphs are then suitably grouped into disjoint clusters. Each cluster is mapped onto a tile (portion of the FPGA). When a fault is detected in a subgraph of a tile, the whole tile is reconfigured onto a spare tile of the FPGA. Clustering is performed so that both the circuit complexity of the overall system and the interconnection complexity among tiles for reconfiguration are limited.

With respect to the literature, this approach reduces the overall circuit complexity and finds a better balancing between the circuit complexity and the interconnection complexity for reconfiguration.

6. References

[1] Peterson W.W., and Weldon E.J.: *Error Correcting Codes*, MIT Press, Cambridge, 1972

[2] Rao T.R.N.: *Error Coding for Arithmetic Processors*, Academic Press, NY, 1974

[3] D'Angelo, S.; Metra, C.; Pastore, S.; Pogutz, A.; Sechi, G.R., "Fault-Tolerant Voting Mechanism and Recovery Scheme for TMR FPGA-based Systems", *Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 1998

[4] Abramovici, M.; Strond, C.; Hamilton, C.; Wijesuriya, S.; Verma, V., "Using roving STARs for on-line testing and diagnosis of FPGAs in fault-tolerant applications", *Proc. International Test Conference*, 1999

[5] Park, N.; Ruiwale, S.J.; Lombardi, F., "Testing the configurability of dynamic FPGAs", *Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 2000

[6] Abramovici, M.; Stroud, C., "DIST-based detection and diagnosis of multiple faults in FPGAs", *Proc. International Test Conference*, 2000

[7] Doumar, A.; Ito, H., "Testing approach within FPGA-based fault tolerant systems", *Proc. Asian Test Symposium*, 2000

[8] Feng, W.; Meyer, F.J.; Lombardi, F., "Novel control pattern generators for interconnect testing with boundary scan", *Proc. Symp. Defect and Fault Tolerance in VLSI Systems*, 1999

[9] Feng, W.; Huang, W.K.; Meyer, F.J.; Lombardi, F., "On the Complexity of Sequential Testing in Configurable FPGAs", *Proc. Symp. Defect and Fault Tolerance in VLSI Systems*, 1998

[10] Wei Liang Huang; Meyer, F.J.; Lombardi, F., "Multiple fault detection in logic resources of FPGAs", *Proc. Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 1997

[11] Ferrandi, F.; Fummi, F.; Pozzi, L.; Sami, M.G., "Configuration-specific test pattern extraction for field programmable gate arrays", *Proc. Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 1997

[12] Stroud, C.; Lee, E.; Abramovici, M., "BIST-based diagnostics of FPGA logic blocks", *Proc. International Test Conference*, 1997

[13] Sying-Jyan Wang; Tsi-Ming Tsai, "Test and diagnosis of faulty logic blocks in FPGAs", *Proc. Int. Conf. Computer-Aided Design*, 1997

[14] Chen, X.T.; Huang, W.K.; Lombardi, F.; Sun, X., "A row-based FPGA for single and multiple stuck-at fault detection", *Proc. Workshop Defect and Fault Tolerance in VLSI Systems*, 1995

[15] El-Ayat, K.; Cahn, R.; Chung Lau Chan; Speers, T., "Array architecture for ATG with 100% fault coverage", *Proc. Workshop Defect and Fault Tolerance in VLSI Systems*, 1991

[16] Doumar, A.; Ito, H., "Design of switching blocks tolerating defects/faults in FPGA interconnection resources", *Proc. Symp. Defect and Fault Tolerance in VLSI Systems*, 2000

[17] Lala, P.K.; Walker, A., "An on-line reconfigurable FPGA architecture", *Proc. Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 2000

[18] Fukushi, M.; Horiguchi, S., "Self-reconfigurable mesh array system on FPGA", *Proc. Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 2000

[19] Mahapatra, N.R.; Dutt, S., "Efficient network-flow based techniques for dynamic fault reconfiguration in FPGAs", *Proc. Symp. Fault-Tolerant Computing*, 1999

[20] Lala, P.K.; Singh, A.; Walker, A., "A CMOS-based logic cell for the implementation of self-checking FPGAs", *Proc. Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 1999

[21] Mojoli, G.A.; Salvi, D.; Sami, M.G.; Sechi, G.R.; Stefanelli, R., "KITE: a behavioural approach to fault-tolerance in FPGA-based systems", *Proc. Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 1996

[22] Chapman, G.H.; Dufort, B., "Making defect avoidance nearly invisible to the user in wafer scale field programmable gate arrays", *Proc. Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 1996

[23] Sheu, M.I., Lee, C.L. "Simplifying Sequential Circuit Test Generation", IEEE Design & Test of Computers, 1994,pp.28-38

[24] A. Antola, V. Piuri, M. Sami, "Optimising High-Level Synthesis for Self-Checking Arithmetic Circuits", *Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems DFT'96*, Boston, MA, USA, Nov. 1996

[25] A. Antola, V. Piuri, M. Sami, "A High-Level Synthesis Approach to Optimum Design of Self-Checking Circuits", *Proc. European Design Automation Conference EURO-DAC'96*, Geneve, Switzerland, 1996

[26] Lach, J.; Mangione-Smith, W.H.; Potkonjak, M., "Enhanced FPGA reliability through efficient run-time fault reconfiguration", *IEEE Transactions on Reliability*, 2000

[27] Shi, W.; Kumar, K.; Lombardi, F., "On the complexity of switch programming in fault-tolerant configurable chips", *Proc. Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 2000

[28] Emmert, J.; Stroud, C.; Skaggs, B.; Abramovici, M., "Dynamic fault tolerance in FPGAs via partial reconfiguration", *Proc. IEEE Symp. Field-Programmable Custom Computing Machines*, 2000

[29] Dutt, S.; Shanmugavel, V.; Trimberger, S., "Efficient incremental rerouting for fault reconfiguration in field programmable gate arrays", *Proc. Int. Conf. on Computer-Aided Design*, 1999

[30] Doumar, A.; Kaneko, S.; Ito, H., "Defect and fault tolerance FPGAs by shifting the configuration data", *Proc. Int. Symp. Defect and Fault Tolerance in VLSI Systems*, 1999

[31] Lach, J.; Mangione-Smith, W.H.; Potkonjak, M., "Low overhead fault-tolerant FPGA systems", *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 1998

[32] Hanchek, F.; Dutt, S., "Methodologies for tolerating cell and interconnect faults in FPGAs", *IEEE Transactions on Computers*, 1998

[33] Mathur, A.; Liu, C.L., "Timing driven placement reconfiguration for fault tolerance and yield enhancement in FPGAs", *Proc. European Design and Test Conference*, 1996