In *Proc. 3rd IEEE International On-Line Testing Workshop*, pp.191-195.
IEEE Computer Society Press, 1997.

A Self-Repairing FPGA Inspired By Biology

Gianluca TEMPESTI, Daniel MANGE, André STAUFFER
Logic Systems Laboratory
Department of Computer Science
Swiss Federal Institute of Technology (EPFL)
CH-1015 Lausanne, Switzerland
E-Mail: tempesti@di.epfl.ch

1. Introduction

Biological organisms are among the most robust systems known to man. Their robustness is based on a set of processes which cannot be adapted directly to the world of silicon, but can provide an inspiration for the design of robust circuits. The Embryonics project [2,3,5] is an attempt to draw such an inspiration from the embryological processes of living organisms.

The core of the project is the development of a two-dimensional array of *cells*, called MICROTREE [2], computational units based on a small processor, a binary decision machine. These cells will be reconfigurable, and thus application-specific. The distinguishing feature of these processing elements is that each cell contains and runs the same program, which thus completely defines the operation of the array (and can thus be considered the electronic equivalent of the biological *genome*), but executes only a subset of all the instructions, depending on its position within the array, i.e., its *coordinates*. Such an array, which can be seen as a SPMD (Single-Program Multiple-Data) machine, would then be able to self-repair through a simple recomputation of a cell's coordinates, an operation which can be executed transparently as part of the genome.

One major difficulty in the development of such an array lies in the fact that the dimensions of a cell cannot be fixed *a priori* without imposing unacceptable limitations to the versatility of the system. The solution we adopted was to implement the array of cells on an FPGA (Field-Programmable Gate Array) of our own conception, called MUXTREE [2], specifically designed to meet the requirements and exploit the features of our project.

The most important feature of our FPGA is that it is itself capable of on-line self-test and self-repair, thus creating a two-level hierarchical system [4, 6, 7, 8] presenting an interesting degree of robustness. The self-test and self-repair systems are briefly presented in sections 3 and 4, while section 2 describes a feature particular to our system which allows a MUXTREE array to be easily configured as an array of identical cells. Finally, section 5 will describe in some more detail the relationship between the MICROTREE and the MUXTREE layers, with particular emphasis on self-test and self-repair considerations.

2. Configuration

As mentioned above, we designed our FPGA for a particular purpose, i.e. to implement an array of MICROTREE cells. The immediate consequence of this approach is that the FPGA will usually be configured as an array of identical cells (to avoid confusion , we will call *cells* the MICROTREE processors, using the term *element* when referring to the FPGA cells). To exploit this regularity, we developed an approach which will allow us to partition the FPGA into *blocks* of elements of any given size. Once the blocks have been defined, it will then be possible to configure the entire FPGA by entering a single instance of the configuration of a MICROTREE cell.

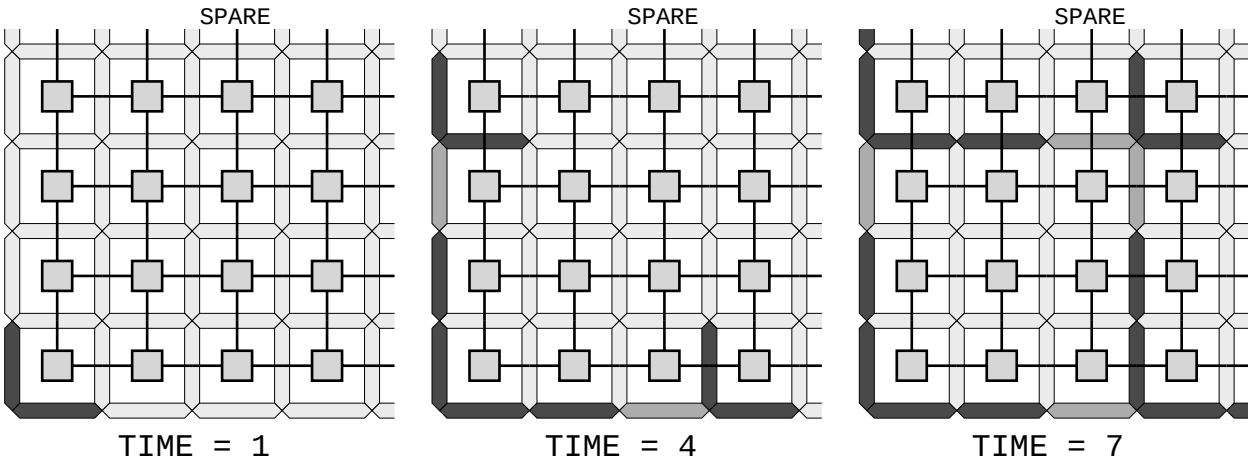

Fig. 1: The cellular automaton colonizing an array of MUXTREE elements

The problem is, of course, that since the dimensions of a cell depend on the application, the size of the blocks cannot be hardwired into the FPGA, and we need a mechanism capable of "colonizing" the MUXTREE array, subdividing the entire FPGA into user-defined blocks. The mechanism we have adopted is a very simple cellular automaton (Fig. 1), capable of creating a set of *boundaries* which define the size of the cell. An added advantage of this approach is that no *a priori* knowledge of the size of the FPGA (i.e., of the number of MUXTREE elements) is required: the automaton will automatically create as many blocks as will fit in the circuit.

Once the boundaries have been defined, they will be used to guide the configuration of the FPGA: the configuration of a single MICROTREE cell will automatically enter every block in parallel, and the boundaries will be used to define a propagation path for the bitstream.

3. Self-Test

The on-line self-test system we developed for the MUXTREE layer had to respect a number of very restrictive constraints. The main constraint was *size*: a MUXTREE element being very fine-grained (a single two-input multiplexer), the logic dedicated to self-test necessarily had to be itself small. This constraint forced us to adopt a set of compromises with regard to the fault-detection capabilities of the system. A second, very restrictive constraint was our desire for the self-test to occur on-line to the greatest possible extent and that the self-test system be completely distributed (preventing the use of a centralized controller). The combination of these and other constraints unfortunately prevented the use of the majority of conventional approaches to self-test [1]. Without attempting an exhaustive list, we will mention the unfeasibility of methods such as pattern testing (which requires external or centralized circuitry), parity checking (which cannot be applied to a MUXTREE element, which deals with single bits), or standard cell duplication (which would impose an unacceptable amount of additional logic).

To implement our system, we thus developed an hybrid system, which borrows concepts from a number of conventional approaches to self test. The key to the system lies in the fact that, for testing purposes, a MUXTREE element can be divided into three parts (Fig.2):

- The functional part of the cell (the multiplexer and the internal flip-flop) are tested by space redundancy (which, in our particular case, actually requires less additional logic than a time-redundant approach): the logic is duplicated (M1 and M2) and the outputs of the two cells compared to detect a fault. A third copy of the flip-flop was added to allow self-repair.
- The configuration register (CREG) is tested as the configuration is being entered (and thus not entirely on-line). Being implemented as a shift register, it can be tested using a special *test sequence* introduced in all the elements in parallel *before* the actual configuration for the system.
- The connections are the weakest point in the system at this level. Faults on the connections (and in the switch block SB) can be detected, but cannot be repaired, both because they cannot be localized to a particular connection, and because our self-repair system exploits the connections to reconfigure the array. In the current system, therefore, we decided not to test the connections directly at the MUXTREE level: as we will see in section 5, we feel that such a test could be performed more efficiently at the MICROTREE level.

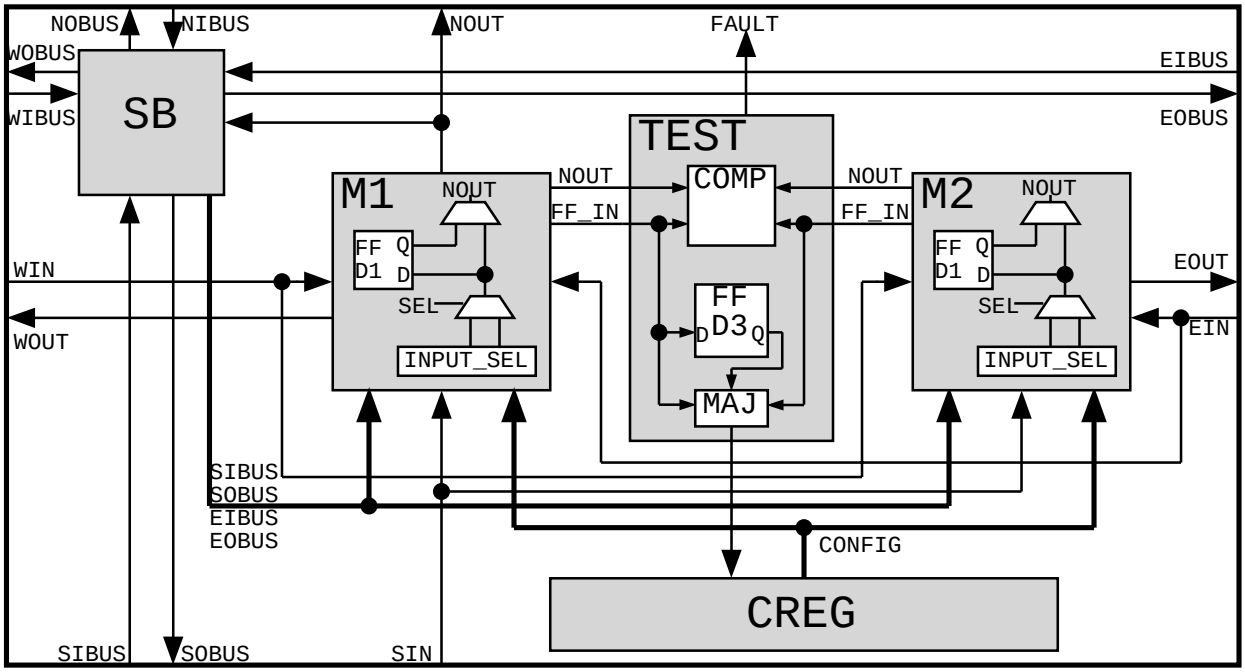

Fig. 2: A self-testing MUXTREE element using space redundancy

4. Self-Repair

The self-repair system had to meet the same constraints of the self-test system, and notably the requirements that the additional logic be small, that the system be completely distributed, and that the system be capable of resuming operation after the self-repair without losing information (i.e., that the *state* of the system be conserved).

To meet these constraints, we used a relatively conventional approach to reconfiguration to design a system based on a set of *spare columns* of elements distributed throughout the array (Fig. 3). Limiting the reconfiguration of the array to a single element per line between two spare columns allows us to minimize the amount of logic required (notably, where the reconfiguration of the connections is concerned). Moreover, the self-test system described above allows us to meet the other major requirements: the fact that the fault detection occurs within each element allows the reconfiguration to occur locally, without the need for a centralized controller, while the presence of a third copy of the flip-flop (the only memory within the functional part of the element), in combination with a simple majority circuit, allows the array to reconfigure without losing its state. The reconfiguration itself is relatively straightforward, consisting of a simple shift to the right of the configuration register (which is implemented as a simple shift register). Attaching the result of the majority circuit (and thus the contents of the flip-flop) to the register, and exploiting the connections already in place for the propagation of the configuration, we are able to perform the shift without requiring additional connections.

One particularly interesting feature of our system is that, exploiting the mechanism used to configure the array (the cellular automaton described in section 2), we are able to define the frequency of spare-to-active columns in array as part of the configuration of the FPGA: the spare columns are defined simply by a particular state of the automaton, and not hard-wired in the circuit. This feature, which allows us to define not only the functionality of the system, but also its robustness to suit a particular application, provides a great amount of flexibility. This possibility of trading functionality for robustness (since increasing the number of spare columns obviously implies decreasing the number of active elements in the circuit, and thus decreasing the overall functionality of the array) could prove very interesting for a number of applications (for example, in circuits designed to operate in high-radiation environments, such as space).

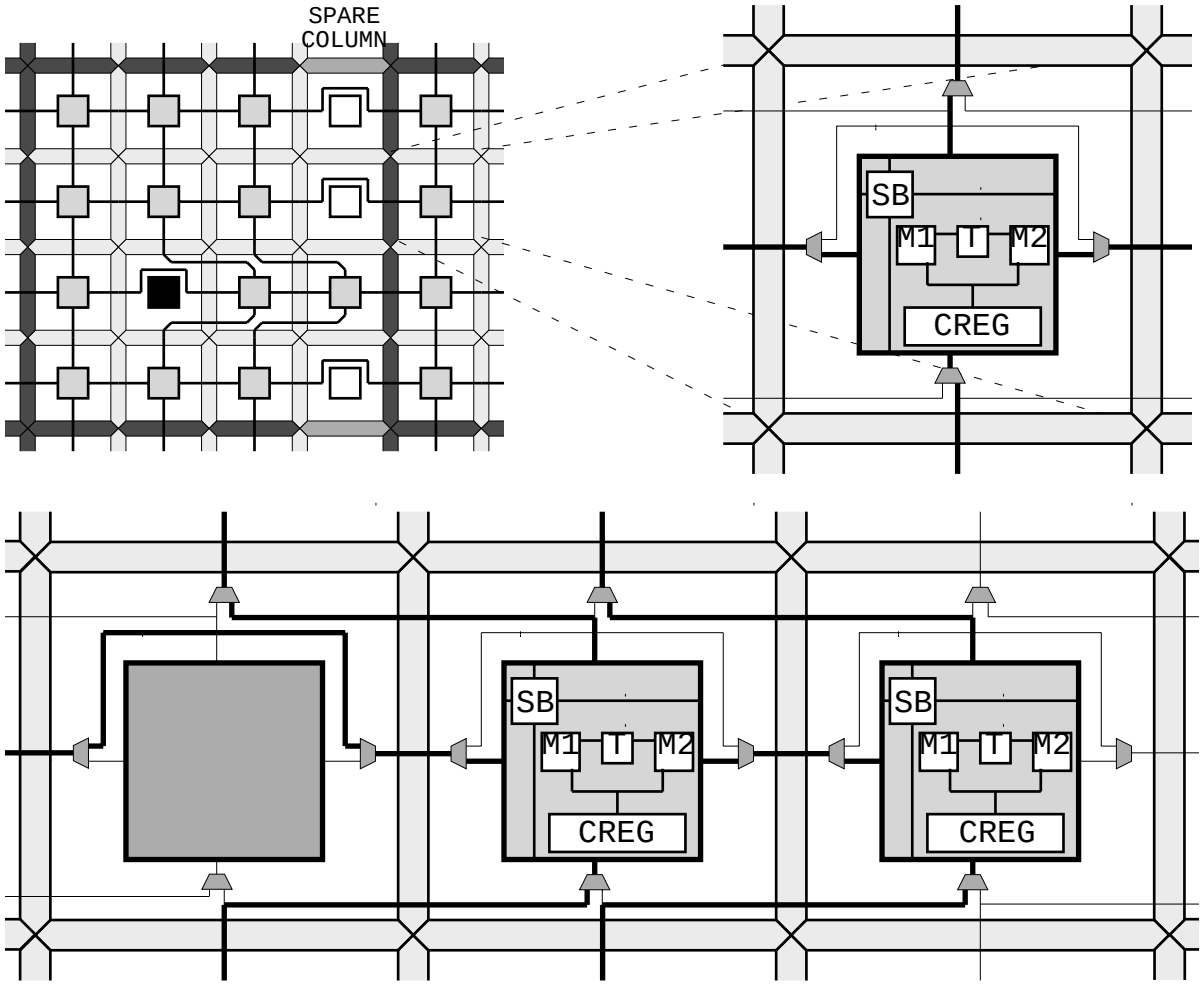

Fig. 3: The self-repair mechanism for an array of MUXTREE elements

5. The MICROTREE cell

As was mentioned, MUXTREE was designed with a specific application in mind, that is, implementing an array of MICROTREE cells. This section gives a brief overview of the basic structure of the cells, keeping in mind that, since the cells are themselves programmable, such a structure cannot be completely defined. However, while the cell as a whole is application specific, it is built around a fixed core, which will be briefly outlined below.

The basic core of the MICROTREE cell (Fig. 4) consists of a *genome interpreter* and of a *genome memory*. The interpreter, a simple binary decision machine, reads the genome (i.e., the program), written in an extremely simple structured language, from the memory (whose size can be configured to fit the size of the genome program), and executes it. The genome itself, executed continuously by the interpreter, consists of two parts: the first deals with the computation of the coordinates (which are thus recomputed every time the program is run), while the second defines the functionality of the cell. The genome is the same for all the cells, and the interpreter chooses which instructions to execute depending on the cell's coordinates. The coordinates are thus a vital part of the system, since they determine the functionality of each cell.

Outside of this basic core, the rest of the cell is entirely programmable. In particular, the number and the size of the registers, the size of the memory (which could be used to contain data as well as the genome program), the connections between cells, and indeed the functionality of the processing elements are all parametrizable, the only limitation being that they be identical for all the cells of the array. In fact, the fixed part of the cell can be seen as a (variable-size) microcontroller for a processing element which is completely programmable.

As we saw above, there are limitations to the fault coverage we can provide at the MUXTREE level, and we are developing a self-test approach for the MICROTREE level. Many of the conventional methods which could not be applied to the MUXTREE elements because of their size can be applied at this higher level, where the complexity of the cells allows for more test logic. As far as self-repair is concerned, it can be implemented very easily, as we mentioned, because of the coordinate-based system: since the coordinates can be recomputed every time the genome is executed, the reconfiguration of the array becomes trivial. Since the loss of an entire MICROTREE cell (or of a column of such cells, depending on the reconfiguration algorithm) is costly, in terms of logic, self-repair at this level will be activated only when the self-repair capabilities of the MUXTREE level are exhausted.

6. Conclusion

Biology and electronics are very different domains, and the mechanisms of one cannot be easily applied to the other. However, with a careful analysis, it is sometimes possible to bridge the gap between the two domains, as we have tried to do with our system.

We developed an FPGA capable of self-test and self-repair, based on a two-level approach where a set of small processors, loosely comparable to biological cells, is implemented on an FPGA of our design. This system is capable of self-repair at both levels, thus providing a kind of robustness which is not quite equivalent to that of biological systems, but does exploit some of its mechanisms.

In developing our system, we borrowed, directly or indirectly, a number of biological concepts. Apart from the immediately obvious analogy between self-repair and healing, a case can be made for comparing the mechanism whereby our FPGA is "colonized" with a number of identical copies of the MICROTREE cell with the biological mechanism of growth, whereby an organism increases in size through cellular division. Additionally, the concept of genome as the description of the function of the entire array, stored and executed in every cell, is clearly inspired by the biological genome, which contains the description of the function of the entire organism, and is also stored and executed in each cell.

Acknowledgments

This work was supported in part by grant 20-42270.94 from the Swiss National Science Foundation.

References

[1] M. Abramovici, M. A. Breuer, A. D. Friedman, Digital Systems Testing and Testable Design (Computer Science Press, New York, 1990).

[2] D. Mange, M. Goeke, D. Madon, A. Stauffer, G. Tempesti, S. Durand, "Embryonics: A New Family of Coarse-Grained Field-Programmable Gate Array with Self-Repair and Self-Reproducing Properties", in: Towards Evolvable Hardware, Lecture Notes in Computer Science (Springer, Berlin, 1996) 197-220.

[3] P. Marchal, P. Nussbaum, C. Piguet, S. Durand, D. Mange, E. Sanchez, A. Stauffer, G. Tempesti, "Embryonics: The Birth of Synthetic Life", in: Towards Evolvable Hardware, Lecture Notes in Computer Science (Springer, Berlin, 1996) 166-197

[4] R. Negrini, M. G. Sami, R, Stefanelli, Fault Tolerance Through Reconfiguration in VLSI and WSI Arrays (The MIT Press, Cambridge, Massachusetts, 1989).

[5] A. Stauffer, D. Mange, E. Sanchez, G. Tempesti, S. Durand, P. Marchal, C. Piguet, "Embryonics: Towards New Design Methodologies for Circuits with Biological-like Properties", in: *Proc. International Workshop on Logic and Architecture Synthesis*, Grenoble, December 1995, pp. 299-306.

[6] C. Stroud, S. Konala, M. Abramovici, "Using ILA Testing for BIST in FPGAs", in: *Proc. 2nd IEEE International On-Line Testing Workshop*, Biarritz, July 1996.

[7] N. Tsuda, T. Satoh, "Hierarchical Redundancy for a Linear-Array Switching Chip", in: *Proc. 2nd IFIP Workshop on WSI*, Brunel, Sept. 1987.

[8] M. Wang, M. Cutler, S.Y.H. Su, "On-Line Error Detection and Reconfiguration with Two-Level Redundancy", in*: Proc. COMPEURO 87*, Hamburg, 1987, 703-706.