社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－ARC－177 (1)
2008－HPC－114 (1)
2008／3／5

Fault-tolerant FPGA Architecture with Distributed Internal Configuration-memory Access

Pierre Devautour, Shuichi Sakai, Masahiro Goshima
University of Tokyo

Abstract

As FPGAs are vulnerable to aging and radiation induced effects that can result in hard errors, they need proper autonomous recovery schemes to become really fault-tolerant. But current FPGA architectures have not be designed specifically to support such schemes. This paper presents a fault-tolerant FPGA architecture featuring TMR-based error detection and localization and a distributed internal access to configuration-memory supporting autonomous recovery.

1. Introduction

Reconfigurable hardware devices such as Field Programmable Gate Arrays (FPGA) are being increasingly used, not only for ASIC prototyping, but also in application domains such as aerospace and defense, cryptography, medical imagery or computer vision, because they allow relatively cheap and fast production of prototypes and final designs in low volume. However, major dependability issues such as vulnerability to radiation induced Single Event Effects (SEE) limit their deployment in harsh environments like space or nuclear plants. A careful and targeted use of Triple Modular Redundancy (TMR), configuration-memory scrubbing and other basic fault-tolerance techniques may help achieve some level of dependability, but important flaws still remain. Autonomous recovery schemes based on partial dynamic reconfiguration seem to be the most adequate technique to improve SRAM-based FPGA's dependability, but current FPGA architectures do not fully support it.

The aim of this paper is to describe an FPGA architecture with inherent support of autonomous recovery. It first presents a short overview of FPGAs and dynamic partial reconfiguration, as well as the dependability issues they face. Then, it makes a review of currently investigated techniques. In a third section, it describes the novel architecture we propose, and then it makes a rough evaluation of it before presenting our conclusions and the work left to be done.

2. Background and Motivation

2.1. FPGA Architecture

The FPGA, or Field Programmable Gate Array, is a semiconductor device containing programmable logic elements called Configurable Logic Blocks (CLB) and a network of programmable interconnects. Each CLB can be programmed to perform simple combinational functions using a set of lookup tables, flips-flops and multiplexers. Connected together through the interconnection network, they can perform the same computations as an ASIC. The data describing the function of each CLB and interconnection switch is stored in a configuration memory.

There are different possible architectures for FPGAs but conceptually they are all based on the same principles: the configuration logic blocks are arranged as a two-dimensional array and are connected with each other through a hierarchical interconnection network that spans all the device horizontally and vertically. Spread on all the periphery of the CLB array, Input/Output blocks act as an interface between the FPGA and the other digital devices on the board. In most FPGAs, some additional resources are included in the device, such as RAM memory blocks and specialized digital signal processors (DSP), for enhanced computing power.

2.2. Dynamic Partial Reconfiguration

For reconfigurable computing applications, the configuration of the FPGA has to be partially modified at run-time so that the unchanged logic runs continuously. This is achieved by using a SRAM configuration memory and Dynamic Partial Reconfiguration (DPR).

Xilinx Virtex devices are such SRAM-based FPGAs that support DPR. Their configuration architecture is described in [1]: the configuration is stored in a SRAM memory that can be read from or written to without halting the device. The smallest unit of configuration memory that can be read or written is a frame. In Virtex 4 devices, the frame is a 1 bit wide column which spans 16 rows of CLBs, as shown on Figure 1, which makes it a more flexible reconfigurable computing platform. It is also important to note that the reconfiguration process is glitch-less: if a configuration bit

holds the same value before and after reconfiguration, the resource controlled by that bit will not experience any discontinuity in operation.

FPGAs usually offer several external configuration interfaces. As an example, Virtex devices have a Serial interface, a SelectMAP interface which provides an 8-bit bidirectional data bus interface to the configuration logic that can be used for both configuration and read-back, and a JTAG chain. In addition to these, Virtex devices also offer an interface that allow an FPGA to access its configuration memory from within its user logic: the Internal Configuration Access Port (ICAP). This ICAP allows to implement a reconfiguration controller, usually in the form of a soft microprocessor, on the user logic of the FPGA, which is the base of autonomous systems.

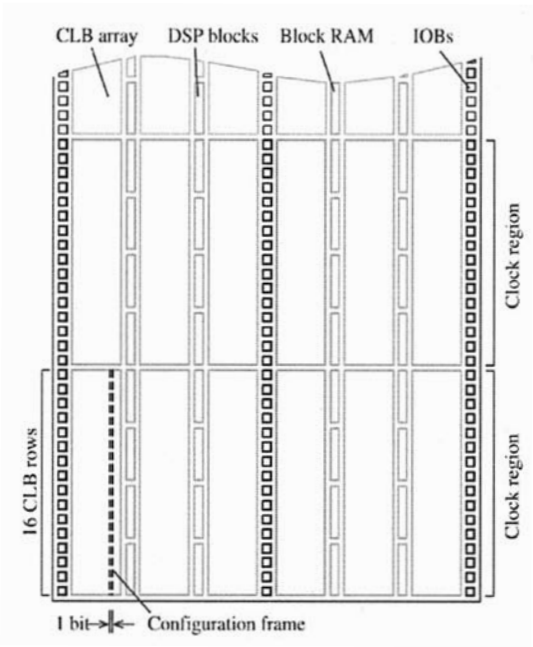

Figure 1: *Xilinx Virtex 4 configuration frames*

2.3. Dependability Issues

Shrinking feature size allowed improved performances and functionalities but also made FPGAs more vulnerable to aging and radiation effects. Time dependent dielectric breakdown, electromigration, hot carrier effects, stress migration and thermal cycling are examples of accelerated aging mechanisms that cause dysfunction in semiconductor devices by altering the gates'characteristics, eventually leading to hard errors. Total ionizing dose (TID) is the accumulation of ionizing radiations over time and causes a degradation of the performance parameters of the device, eventually resulting in permanent failures.

Single Event Effects (SEE), in the other hand, result from the strike of a single high energy particle which can generate both soft and hard errors. Single Event Upsets (SEU) result in firm errors. A high energy particle strikes the semiconductor device causing the ejection of heavy ions that can create a momentary pulse in the signal. A SEU occurs when the parasite pulse is latched in a memory cell. Errors caused by Single Event Upsets are called firm errors because in FPGAs, as the functionality of the device is described in the configuration memory, errors can alter the functionality of the device so these can no longer be called soft. Figure 2 shows how an upset of a bit within the configuration memory modifies its behavior.

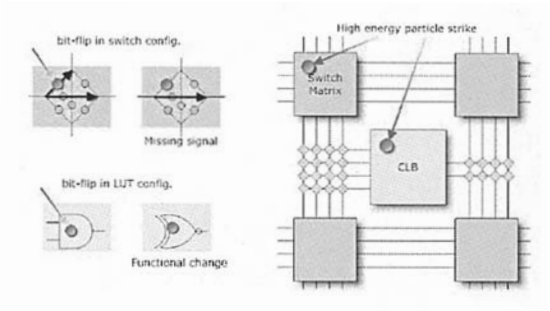

Figure 2: *SEU's effects on FPGAs*

3. Related Work

Manufacturers have proposed several solutions to mitigate the upsets in the configuration layer. In [2] is used a Cyclic Redundancy Check (CRC) feature to detect corrupting SRAM bits. This feature enables a reconfiguration whether a SEU is detected, thus correcting the configuration bits.

In [3] is proposed a detection/correction method based on the read-back feature. The principle is similar to [2]. What is different is the need of an external controller to perform the checking and to enable the reconfiguration of the device. The triple modular redundancy (TMR) with voting uses three identical modules that perform the same task with corresponding outputs compared through a majority voter circuit. This scheme can be applied to different level of modules: device, design or gate.

In the test community, several techniques have been proposed to test and diagnose FPGAs [4], [5], [6]. In addition,

some authors have proposed fault recovery mechanisms and self-repair capabilities for bypassing one or several faults occurring in a fault-tolerant design [4], [7]. The main idea is to use the re-configurability nature of FPGAs for bypassing the faults.
As an example, in [4] is proposed the use of Roving STARs in fault-tolerant designs. Roving STAR method has been proposed to perform on-line testing and fault diagnosis in FPGAs. The FPGA can be divided into two areas: working area and self-testing area (STAR). The STAR is composed of non configured resources, i.e. those not involved in the design. At each time, the STAR is moved. At the same time, the functionality of the working-part under test is moved into the discharged STAR. This process is done by dynamically reconfiguring a part of the device and this is done without disturbing the operation in the rest of the application. In [4], the self testing area is composed of two full rows and two full columns.

4. Proposed Architecture

The problems with the techniques mentioned above is that most of them allow a fault latency that can not be tolerated in some critical applications. Some of them also rely on a recovery managing unit that becomes a single point of failure.
Our approach is to eliminate fault latency using TMR and to reduce the number of single points of failures by also triplicating the unit in charge of the recovery and providing a distributed internal access to the configuration-memory.

4.1. TMR for error localization

The voting logic in basic TMR systems takes three inputs and sends as an output the value given by the majority of the three inputs. To use TMR for error localization in order to support autonomous recovery, we want a TMR system to be able to send information about the localization of the error to the recovery manager so that appropriate measures can be taken to repair or relocate the faulty unit. Figure 3 shows a TMR system that is able to tell which of its inputs is faulty, if any. The error signal then has to be forwarded to the reconfiguration manager so that appropriate measures can be taken. How this is done will be described later.

4.2. Tile-based architecture

The TMR system described above is the base of our fault-tolerant architecture. As the voting logic is a single point of failure of the device, it should be implemented as static hard-wired logic, with thick gates if possible. However, TMR is not enough to provide acceptable dependability, and it should be accompanied by an autonomous recovery mechanism. To support such a mechanism, we define a novel FPGA architecture that will support fine grain logic

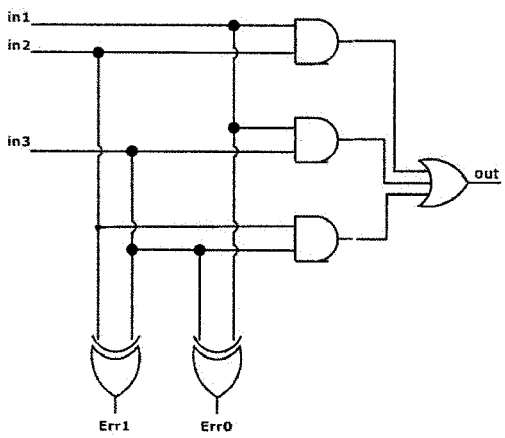

Figure 3: *TMR for error localization*

local relocation, which seems to be the most efficient recovery scheme. Our architecture is based on a building-block called a tile. Figure 3 describes such a tile.

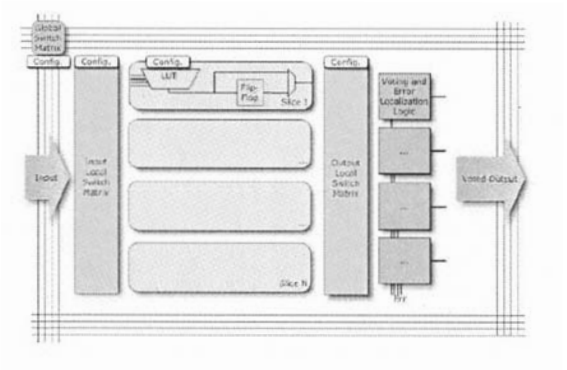

Figure 4: *A tile: building-block of the FPGA*

Each tile contains n slices, which are the equivalent of simple CLBs, that are in charge of the actual computation. An input local switch matrix feeds these slices with the proper inputs coming from the global interconnection network, while a local output switch matrix sends the outputs of the slices to k hard-wired voting and error localization units. The voted outputs are then sent to other tiles through the global interconnection network while the error signals are sent to the recovery manager.
The configuration memory of this tile is called a frame and is the smallest portion of the configuration memory that can

be read from or written to. It contains the configuration data of the global switch matrix, the two local switch matrices, and the slices.
Such a tile-based architecture is suitable for autonomous recovery mechanisms: if a certain number of the slices are left unused as spare units, simple manipulations on the content of the tile's configuration frame allow to copy configuration data from one slice to another and to re-route the output of the spare slice to the right voting unit.

4.3. Autonomous recovery mechanism

In the system we propose, a unit responsible for the management of the recovery process (recovery manager) is implemented on the user logic of the FPGA and is triplicated like the rest of the design. A general purpose soft-core processor such as Microblaze is suitable for this task. The recovery process that we propose is as follows:

- The recovery process starts upon error detection by a voting logic. The associated error localization logic sends to the recovery manager a signal specifying which one of its three outputs is faulty
- The recovery manager orders a read-back of the configuration frame of the tile where the error has occurred
- The recovery manager extracts appropriate information from the configuration data of the programmable switch matrix to determine the slice where the error may have occurred and the healthy copies of this slice
- The recovery manager reconfigures the supposedly faulty slice with the configuration data of one of its healthy copies
- If there is no more error signal from the voting logic, then the error was just a firm error in the slice's configuration memory and no other measure needs to be taken. Else, if the error signal persists, then we can either suppose that an hard error occurred within the slice, or that its inputs have been misrouted
- Then the recovery manager relocates the supposedly faulty logic in a spare slice of the tile, extracting appropriate information from the configuration data of the input switch matrix to re-rout the inputs of the slice correctly
- If there is no more error signal from the voting logic, then the error was probably a hard error within the logic of the slice, like a bit stuck at 0, or a short circuit for example, and no other measure needs to be taken. However, if the the error signal persists, either the input of the slice was misrouted and there is no way to determine which signal should be taken as an input, or the spare slice used to replace the faulty slice is faulty itself
- If another spare slice is available, the recovery manager makes a second attempt to relocate the supposedly faulty slice
- If there is no more error signal, the recovery is successful: the first spare slice used for relocation was probably faulty itself and is marked as such by the recovery manager. If the error signal persists, a test can be run on the slice to check for hard error (such a process is possible but not described here). If the slice is diagnosed as healthy, then we suppose that its inputs have been misrouted

4.4. Distributed internal configuration-memory access

The feasibility of the recovery process described above depends on whether we are able to send error localization information to the recovery manager and to send reconfiguration data from the recovery manager to the faulty tile.
To solve these problems, in our system, we propose a communication network connecting all the slices to each other, in the shape of a token ring as shown on Figure 5.

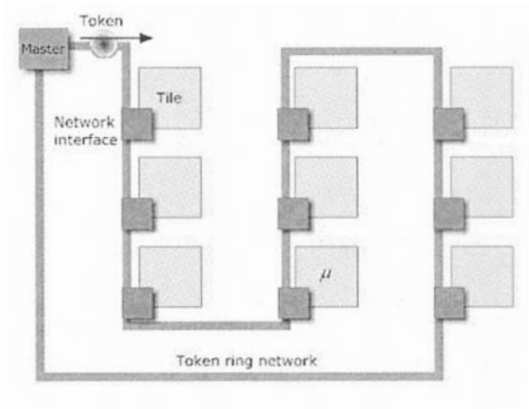

Figure 5: *A token ring for distributed internal configuration memory access*

Each tile contains a local interface to this token ring network, including the tile on which the recovery manager is implemented. Error signals and reconfiguration data are sent from tiles to tiles on this network. here is a description of the communication protocol:

- an error is signaled in a tile T

- the network interface of tile T assembles a packet containing an error report and waits for the token
- when the token arrives, tile T catches it, sends its packet, and switches to a state where it is ready to receive a packet containing a request for local memory read-back
- the tile Mu where the recovery manager is implemented intercepts the packet,forwards the error report to its slices, and sends on the network a packet containing a read-back request
- tile T still has the token so it intercepts the packet containing the read-back request which is sent to the local memory access controller
- tile T assembles a packet containing its configuration data, sends it on the network, and switches to a state where it is ready to receive a packet containing reconfiguration data from the the tile Mu
- tile Mu intercepts the packet, forwards the configuration data to its slices, and wait until the slices have computed new reconfiguration data. It then assembles a packet containing this reconfiguration data and a write operation order and sends it on the network
- tile T still has the token so it intercepts the packet and forwards the write operation order and the reconfiguration data to its local configuration memory access controller
- if no further error is observed, tile T can release the token

Provided with such a communication structure and protocol, our architecture can really support the autonomous recovery mechanism described in the previous paragraph.

5. Evaluation

The architecture described above is designed specifically for fault-tolerance and to support autonomous recovery. A study of its behavior in errors occur allows a rough estimation of its survivability. Figure 6 shows a representation of the behavior of a tile. After an error occurs, the tile goes to a transitional state while the recovery manager attempts an on-site reconfiguration of the supposedly faulty module. If another error occurs in the same TMR system during this process, it leads to a fatal failure. If the reconfiguration succeeds in recovering from the error, then the tile goes back to its initial state. If not, it the supposedly faulty module is relocated and the tile goes to a state where all the TMR systems are healthy again but one spare sliced has been used. This process can be repeated N times where N is the initial

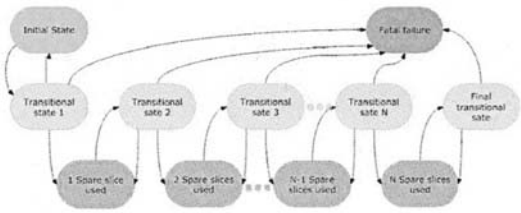

Figure 6: *Behavioral model*

number of spare slices.
In fact, the time used by the recovery manager to either reconfigure or relocate a faulty slice is about seconds, while the time between two failures is about days. As a consequence, the length of a transitional state is negligible compared to the mean time between failures. So we can simplify the model described above as follows. A tile is in state k, with k from 0 to N, when k of the N initial spare slices have been used. And the mean time of transition between two consecutive states is T, where T is the mean time to failure for persistent errors. With such a model, it is easy to estimate that the mean time to failure of the device is (N + 1) T . As T depends of the environment and of the device, we can not give precise figures to illustrate the dependability of our architecture, but we can estimate that it will survive N +1 times longer than a non fault-tolerant design, with N the number of spare slices in a tile.

6. Design of an experimental platform

In the previous chapter, we have made a very rough estimation of the survivability of our architecture. But the only way to perform a better evaluation of our proposal and to demonstrate its capabilities is to make a prototype of it and measure its performances. In this chapter, we describe such an experimental platform. It consists in a multi-FPGA board. We want to use this board to demonstrate the efficiency of the fault-tolerant schemes supported by our architecture. An ideal demonstration would be to observe a design performing correctly continuously while we trigger failures on the device.
Our experimental board will be consisting of two arrays of FPGAs: one to implement the tiles, and one to implement the token ring network to simulate a distributed internal configuration memory access scheme.
To implement a tile on an FPGA, we used the modular flow for dynamic partial reconfiguration: independent modules are implemented for slices, switch matrices and voting logic.

To each tile-FPGA corresponds a network-FPGA which will be its interface to the token ring network and its access port to the reconfiguration memory.

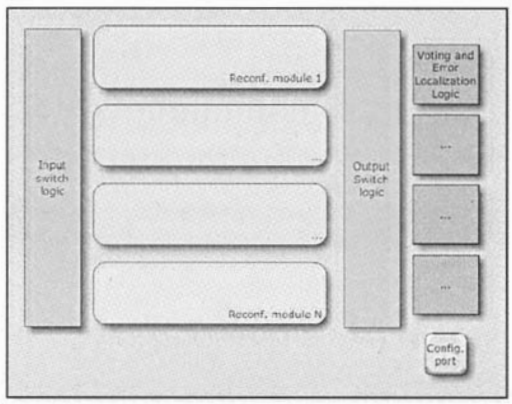

Figure 7: *Tile FPGA*

7. Conclusion and Future Work

In this paper, we proposed a novel FPGA architecture with inherent support for autonomous recovery schemes. The main contribution of this paper is the description of an error localization and reporting method relying on a distributed internal configuration memory access and a token ring network. We roughly evaluated the survivability of our proposal and described a prototyping board. We hope that this paper will be used as a basis for further investigation eventually leading to the implementation of a prototype. We left a few questions unanswered, such as the feasibility of a recovery scheme able to repair errors occurring in the global routing resources, and also concerning the performance overhead generated by our architecture.

Acknowledgments :
This research is supported by JST, CREST.

References

[1] S. Kelem. Virtex series configuration architecture user guide. *Xilinx Application Note*, (151), 2003.

[2] Altera. Error detection using crc in altera FPGA devices. *Application Note*, (357), 2004.

[3] Xilinx. Correcting single-event upsets through virtex partial configuration. *Xilinx Application Note*, (216), 2000.

[4] M. Abramovici, C. Stroud, C. Hamilton, S. Wijesuriya, and V. Verma. Using roving Stars for on-line testing and diagnosis of FPGAs in fault-tolerant applications. *IEEE Int. Test Conf.*, pages 973-982, 1999.

[5] F. Lomardi, D. Ashen, X. Chenn, and W.K. Huang. Diagnosing programmable interconnect systems for FPGAs. *ACM Int. Symp. On Field-Programmable Gate Arrays*, pages 100-106, 1996.

[6] C. Stroud, E. Lee, and M. Abramovici. Using ILA testing for BIST in FPGAs. *IEEE Int. Test Conf.*, pages 68-75, 1996.

[7] J. Lach, W.H. Mangione-Smith, and M. Potkonjak. Low overhead fault-tolerant FPGA systems. *IEEE Trans. On VLSI Systems*, vol. 6, pages 212-221, 1998.