Fault-Free: A Framework for Supporting Fault Tolerance in FPGAs

Kostas Siozios¹, Dimitrios Soudris¹ and Dionisios Pnevmatikatos²

¹ School of Electrical & Computer Engineering, National Technical University of Athens, Greece
{kysiop, dsoudris}@microlab.ntua.gr
² Electronic and Computer Engineering Department, Technical University of Crete, Greece
pnevmati@mhl.tuc.gr

Abstract. In this paper we propose a novel methodology for supporting application mapping onto FPGAs with fault tolerance even if this feature is not supported by the target platform. For the purposes of this paper we incorporate three techniques for error correction. The introduced fault tolerance can be implemented either as a hardware modification, or through annotating the application’s HDL. Also, we show that the existing approaches for fault tolerance result to hardware wastage, since there is no demand for applied them uniformly over the whole FPGA. Experimental results show the efficiency of the proposed framework in terms of error correction, with acceptable penalties in device area and Energy×Delay Product (EDP) due to the redundant hardware resources.

Keywords: FPGA; fault tolerant; architecture; exploration; CAD tool.

1 Introduction

SRAM-based Field-Programmable Gate Arrays (FPGAs) are two-dimensional arrays of Configurable Logic Blocks (CLBs) and programmable interconnect resources, surrounded by programmable input/output pads on the periphery. Even though programmability feature of these devices makes them suitable for widely application implementation, there are a number of design issues that have to be taken into consideration during application mapping. Among others, reliability issues have become worse as devices have evolved. For instance, as the transistor geometry and core voltages decreased, while the numbers of transistors per chip and the switching frequency increase, the target architectures become more susceptible to incurring faults (i.e., flipped bit or a transient within a combinatorial logic path). Nowadays, the reliability problem becomes even more important due to the industry trend for increasing the logic density [1]. Consequently, fault tolerance need to be though as a critical design issue, even for real-life applications.

The last ten years many discussions were done about the design of reliable architectures able to overcome from faults occurred either during the fabrication process or the execution time. More specifically, the term fault tolerant corresponds to a design which is able to continue operation, possibly at a reduced level, rather than failing completely, when some part of the system fails [1, 2, 3, 4, 5, 6, 8, 9, 12, 13, 14, 15]. These solutions include fabrication process-based techniques (i.e. epitaxial CMOS processes) [11], design-based techniques (i.e. hardware replicas, time redundancy, error detection coding, self-checker techniques) [4], mitigation techniques (i.e. multiple redundancy with voting, error detection and correction coding) [5], and recovery techniques (reconfiguration scrubbing, partial configuration, rerouting design) [14].

Even though fault tolerance is a well known technique, up to now it was mostly studied for ASIC designs. However FPGAs poses new constraints (i.e. higher power density, more logic and...
interconnection resources, etc), while the existing fault models are not necessarily applicable. To make matters worse, faults in FPGAs can alter the design, not just user data. In addition to that, the designs mapped onto FPGAs utilize only a subset of the fabricated resources, and hence only a subset of the occurred faults may result to faulty operation. Consequently, FPGA-specific mitigation techniques are required, that can provide a reasonable balance among the desired fault prevention, the performance degradation, the power/energy consumption and the area overhead due to the additional hardware resources.

Up to now there are two approaches for preventing faults occurring on FPGAs. The first of them deals with the design of new hardware elements which are fault tolerant enabled [2, 4, 12, 15]. These resources can either replace existing hardware blocks in FPGAs, or new architectures can be designed to improve robustness. On the other hand, it is possible to use an existing FPGA device and provide the fault tolerance at higher level with CAD tools [2, 3, 6, 8, 13, 14].

Both of these approaches have advantages and disadvantages, which need to be carefully concerned. More specifically, the first approach results to a more complex architecture design, while the derived FPGA provides a static (i.e. defined at design time) fault tolerant mechanism. On the other hand, the second approach potentially is able to combine the required dependability level, offered by fault tolerant architectures, with the low cost of commodity devices. However, this scenario imposes that the designer is responsible for protecting his/her own design.

In [12] a fault tolerant interconnection structure is discussed. The faults in interconnection network are corrected by spare routing channels which are not used during place and route (P&R). A similar work is discussed in [13], where a defect map is taken as input to P&R tool and then application’s functionalities are not placed in the faulty blocks. In another approach [14], EDA tools take as input a generic defect map (which may be different from the real defect map of the chip) and generate a P&R according to this. A work that deals with a yield enhancement scheme based on the usage of spare interconnect resources in each routing channel in order to tolerate functional faults, is discussed in [15]. The only known commercial approach for supporting fault tolerance in FPGAs can be found in [8]. This implementation replicates each of the logic blocks that form application. The initial, as well as the two replica blocks work in parallel, while the output is derived by comparing their outputs with a majority voting.

In this work we focus on providing a framework for exploring the efficiency of alternative fault tolerance techniques in terms of numerous design parameters. The introduced fault tolerance can be implemented either on hardware (i.e. block redesign) or software (i.e. HDL annotation) level. In addition to that, the proposed methodology enables conventional FPGA devices to have fault tolerant features, making it suitable for existing commercial architectures. Also, our methodology can provide the desired tradeoff between the required reliability level and the area overhead. Based on experimental results, we achieve significant error correction (similar to existing fault tolerance approaches), but we employ significant fewer hardware resources, leading among others to area, delay and energy savings.

More specifically, the main contributions of this work are summarized, as follows:

- We introduce a novel methodology for supporting fault detection and correction on FPGA devices.
- We identify sensitive sub-circuits (where faults are most possible to occur) and we apply the proposed fault tolerant technique only at these points rather than inserting redundancy in the whole device.
- We develop a new tool that automates the introduction of redundancy into certain portions of an HDL design.
- We validate the results with a new platform simulator based on MEANDER framework [10].

The rest paper is formed as follows: In section 2 we discuss the paper motivation, while section 3 describes the fault tolerant FPGA architectures. The proposed fault tolerance methodology is explained in section 4. Experimental results that show the efficiency of the derived fault tolerance architectures are shown in section 5, while conclusions are summarized in section 6.
2 Motivation

The first step in order to build a reliable system is to identify possible regions with increased failure probability. These regions mostly include hardware resources that implement application’s functionalities with increased switching activity [17]. Since switching activity is an application property which does not depend either to the target platform or the employed mapping tools, it is possible to identify critical functionalities during the profiling task (step 1 of the proposed methodology – we will describe it in section 4). However, the employed toolset introduce some constraints regarding the spatial distribution of regions with excessive high (or low) values of switching activity, and consequently with increased (or decreased) failure probability [18].

In order to show a case study about the variation of switching activity, Figure 1 plots the sensitive FPGA regions, by calculating the failure probability of hardware resources for an array composed by 64×64 slices, which implements the \textit{frisc} application [7]. More specifically, Figure 1(a) depicts the variation of switching activity for the application mapping without fault tolerance. In this figure, different colors denote different failure probabilities, while as closer to red color a slice is the higher probability to occur a fault.

![Figure 1. Spatial distribution of failure probability for \textit{frisc} benchmark: (a) without fault tolerance, and (b) with TMR redundancy](image)

Based on this map (Figure 1(a)), it is evident that the switching activity (and hence the failure probability) is not constant across the FPGA, since it varies between two arbitrary points \((x_1,y_1)\) and \((x_2,y_2)\) of device. From this distribution it is feasible to determine regions on the device with excessive high values of switching activity (regions of importance), where we have to pay effort in order to increase the fault tolerance. Consequently, the challenge, with which a designer is faced up, is to choose only the actually needed redundancy level, considering the associated spatial information from the distribution graph. Figure 1(b) depicts a candidate classification of FPGA’s hardware resources into two regions, with and without increased demands for redundancy.

A second important conclusion is drawn from Figure 1: although the majority of existing fault tolerant techniques exhibits a homogeneous and regular structure, the actually critical for failure resources provide a non-homogeneous and irregular picture. Consequently, careful analysis of the points of failure must be performed, while the target system implementation needs to combine regions with different density of fault tolerance.
Based on exhaustive exploration [18], we found that the region with increased failure probability and hence increased demand for redundancy is placed almost in the middle of the device, while it occupies about 40% of the FPGA’s area. On the other hand, the rest FPGA (i.e. 60%) exhibits limited failure probability, and hence it is not upmost important to apply redundancy.

3 Proposed Fault Tolerant FPGA Architectures

Our target is a generic recent FPGA device (shown in Figure 2) similar to the Xilinx Virtex architecture, consisting of an array of CLBs, memories and DSP cores. Communication among hardware blocks is provided by a hierarchical interconnection network of fast and versatile routing resources. We assume that CLBs are formed by a number of Basic Logic Elements (BLEs), each of which is composed of a set of programmable Look-Up Tables (LUT), multiplexers, and flip-flops. The output of BLE can connect either to an input pin of any other BLE, or to the logic block output. Apart from this architecture, our proposed methodology can also support almost any other existing FPGA device.

![Figure 2. The target FPGA architecture](image)

In order to support the fault tolerance we use a replication technique based on voting, which can be implemented either in hardware (as a dedicated circuit) or in glue logic (through HDL annotation). In contrast to the hardware implementation, the software-based fault repair technique is particular important in the field of FPGAs, as it provides a trade-off between the desired accuracy of detecting and correcting faults with the extra hardware overhead.

The reconfigurable platform is encoded as an $M$-of-$N$ system, consisting of $N$ hardware blocks where at least $M (M \leq N)$ of them are required for proper operation (the platform fails if less than $M$ of the blocks are functional). In our study we assume that different blocks fail with statistically independent order and if a block fails then it remains non-functional (the faults are not temporal). If $R(t)$ is the probability of an individual block to be still operational at time $t$, then the reliability of a $M$-of-$N$ architecture corresponds to the probability that at least $M$ blocks are functional at time $t$. By supposing that $f_p$ denotes the probability that entire architecture suffers by a common failure, the system reliability can be calculated as follows:

$$R_{M,of,N}(t) = \left(1 - f_p\right) \sum_{k=M}^{N} \binom{N}{k} R^k(t) \left[1 - R(t)\right]^{N-k}$$

(1)
where \( \binom{N}{k} = \frac{N!}{k!(N-k)!} \). In case we assume that a fault affects the whole architecture (i.e., \( f_c = 0 \)), then the FPGA’s reliability is calculated based on Equation 2. Additionally, whenever \( R(t) < 0.5 \) the hardware redundancy actually become a disadvantage, as compared to a platform without redundancy.

\[
R_{M,of,N}(t) = \sum_{k=0}^{N} \binom{N}{k} R^k(0) [1 - R(t)]^{N-k}
\]  

(2)

The functionality of voter is to receive a number of inputs from an M-of-N architecture and to generate a representative output. For instance, a typical voter does a bit-by-bit comparison, and then outputs the majority of the N inputs. Figure 3 shows a majority voter instantiation with three inputs. Such a voter can block an upset effect through the logic at the final output. Consequently, they can be placed in the end of the combinational and sequential logic blocks, creating barriers for the upset effects.

During the redundancy approach, a critical design issue is to determine the optimal partition size for the replicated logic that have to be voted, in order to reduce the probability upset faults in the routing to affect two distinct redundant parts that are voted by the same voter. A small size block partition requires a large number of majority voters that may be too costly in terms of area and performance. On the other hand, placing only voters at the last output increases the probability of an upset in the routing infrastructure to affect two (or more) distinct redundant logic blocks overcoming the fault correction mechanism.

Next paragraph describes some of the supported voting schemes from our proposed framework. We have to mention that apart from these, any other voting scheme can also be employed, if it is appropriately described in Boolean form. Note that, our approach can diagnose with accuracy errors in glue logic. In order to correct faults in wire segments, it is possible to add spare routing resources similar to [12].

### 3.1 R-fold modular redundancy

The R-fold modular redundancy (RMR), shown in Figure 4, entails \( R = \{3, 5, 7, ..., C \} \) replica logic blocks working in parallel, while the output results through a majority vote. Such a voter discards errors coming from the faulty replica(s) and selects as block’s output one of the partial outputs from the rest (faulty-free) replicas. This technique can effectively mask faults if: (i) only less than \( (R + 1)/2 \) replicas are faulty (either on combinational and sequential logic), but the fault presents in different register locations, and (ii) the voter if fault free.

A typical instantiation of this approach is Triple Modular Redundancy (TMR), which guarantees effective functionality as far as up to one replica blocks exhibits a single fault. This instantiation is also used at the only known commercial tool for supporting fault tolerance in FPGAs [8].
3.2 Cascaded R-fold modular redundancy

Despite the simplicity of the RMR method, it is ineffective when an upset occurs in the voter or there is an accumulation of upsets. In these cases, an additional mechanism is necessary to correct the faults, before the next upset happens. Cascaded $R$-fold modular redundancy (CMR) is similar to the previous studied architecture, but along with the $R$ replicas of the logic blocks, the majority voter is also replicated. Multiple instantiations of this technique are feasible, which mainly differ in the number of voters, as well as the connectivity among them. Figure 5 depicts the $R$-folded modular instantiation employed in this paper.

3.1 Time redundancy (TR)

Figure 6 depicts another fault tolerant implementation supported by the proposed framework. More specifically, by sampling the signal inserted to the voter more than once with shifted clocks, it is possible to eliminate errors with a pulse width less than the clock cycle. As compared to the rest implementations discussed previously, this one exhibits lower area overheads, but it imposes performance degradation, as well as the requirement for different clock phases.
4 The Proposed Fault-Free Framework

In this section we describe the proposed framework for exploring the efficiency of different fault tolerance techniques. The goal of this methodology is to ensure fault masking at the same time with acceptable area, delay and energy overheads. The methodology consists of three steps (S1, S2 and S3), as they are depicted in Figure 7.

Starting from an HDL system description, during the first step (S1) we perform application profiling and analysis in order to determine sensitive functionalities with increased failure probability. Typical parameter that results at higher failure probability is the increased switching activity, and hence the on-chip temperature, due to the electromigration effect [17]. In section 2 we have already studied the spatial variation of switching activity. Consequently, application’s functionalities with increased switching activity are most candidates for applying redundancy, since they have higher failure probabilities. This information, in conjunction to the user constraints (i.e. desirable reliability, area overhead, etc), defines the employed fault tolerance model. Such a model describes among others the selected fault tolerant technique (i.e. RMR, CMR, TR, etc), as well as the amount of redundancy applied on the target platform.

The second step (S2) of the methodology provides a precautionary mechanism for minimizing (or eliminating) reliability issues from defects occurred during execution time due to permanent faults. In order to apply it, a number of hardware resources need to be reserved. More specifically, this step can provide solutions to the following reliability problems: (i) to discourage functionalities to be placed or routed to already known faulty resources, and (ii) to reserve resources on critical device regions and to invoke them whenever a fault occurs, in order to find a new P&R which occupies only functional components. More specifically, the hardware resources of target device are categorized in three groups, namely: (i) unutilized (they are not employed for application mapping), (ii) utilized (used for application mapping and are fault free), and (iii) faulty (found not to operate properly). Due to the increased complexity of the second task, a co-processor attached to the FPGA is required, in order to find the P&R and re-program faulty resources.

The third step (S3) of the proposed methodology deals with application mapping. The application’s HDL description is synthesized and then technology mapped. The retrieved netlist is appropriately annotated based on the desired fault tolerance, as it was retrieved from S1. This task is involved only whenever the redundancy is applied in software (i.e. HDL) level. Otherwise, i.e. when the FPGA incorporate a hardwired fault tolerant mechanism, we bypass this tasks and proceed to application P&R. Constraints posed during the P&R comes from the reservation of hardware resources (step S2), as well as the faulty map (similar to the Figure 1(a)).
In order to provide the necessary software support for our proposed methodology, we employ an open source tool flow [10], while the fault tolerance is applied as a new CAD tool, named Fault-Free. However, the methodology is completely transparent to the employed CAD tool chain and can be applied as an intermediate step between synthesis and place and route (P&R) tools at existing design flows.

5 Experimental Results

The first step in calculating reliability is the selection of fault models. There are two major sources of logic faults in FPGAs: cosmic radiation and manufacturing/operating imperfections. Since the size of radiation particles is small as compared to the bitstream size of logic blocks, we employ a fault model that follows uniform distribution of non-correlated failures. In addition to that, the device scaling makes them more sensitive to soft faults [9]. The second class of faults is related to manufacturing and operating imperfections. Even though these defects are not important during the testing after fabrication, they become exposed after a long period of operation. In order to model this type of faults, we use the Stapper fault model [19], which is based on gamma distribution.

The efficiency for the alternative fault tolerant implementations is shown in Figure 8. More specifically, here we study the percentage of repaired over the injected faults for different redundant techniques. In order to make this experiment, for each of the curves we calculate the bitstream file for frisc [7] application. This size ranges from \(B=1107\) Kbits (TR approach) up to \(B=1560\) Kbits (CMR approach), while each of these files remains constant along the corresponding curve. Then, we inject randomly a number of faulty bits in the configuration data. The amount of faulty injected bits, mention with \(F\), ranges from 5% up to 25% of the configuration’s file size. We assume that the faults are randomly distributed between logic and interconnection resources.

The horizontal axis of this figure corresponds to the percentage of injected faulty bits over the size of configuration data, while the vertical one shows the percentage of faulty bits over the total injected bits that remain faulty after applying the recovery mechanism. In other words, the vertical axis plots the efficiency to repair faults for the alternative redundant implementations.

![Figure 8. Efficiency of alternative fault tolerant techniques: (a) when they are applied uniformly over FPGA, (b) when applied based on the proposed region-based approach](image-url)
Based on Figure 8 we can conclude that the proposed methodology results to significant error prevention for all the incorporated mechanisms. More specifically, regarding the case where \( F = 5\% \), our region-based redundancy results up to 7% smaller efficiency in correcting faults, as compared to an implementation with full redundancy. However, we have to mention that our approach results to significant area savings, since 60% of the device area does not incorporate redundant blocks. In addition to that, in case the fault tolerance is applied uniformly over the FPGA, there is still a failure probability, since the employed redundancy scheme cannot prevent faults in routing infrastructure.

One of the disadvantages of applying fault tolerant techniques is the performance degradation due to additional hardware blocks. Table 1 shows the result in terms of Energy\( \times \)Delay Product (EDP) for the original (i.e., non fault tolerant) and the three alternative techniques discussed in this paper. For each of these implementations we evaluate two instantiations: (i) all the functionalities are replicated (existing design approach), and (ii) only functionalities that are placed on critical device regions are replicated (region-based design approach). The derived modified fault tolerance schemes are denoted as \( m_{\text{RMR}} \), \( m_{\text{CMR}} \) and \( m_{\text{TR}} \).

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Existing design approaches</th>
<th>Proposed (region-based) approaches</th>
</tr>
</thead>
<tbody>
<tr>
<td>alu4</td>
<td>4.77</td>
<td>7.85</td>
</tr>
<tr>
<td>apex2</td>
<td>8.30</td>
<td>9.42</td>
</tr>
<tr>
<td>apex4</td>
<td>4.13</td>
<td>4.99</td>
</tr>
<tr>
<td>bigkey</td>
<td>1.41</td>
<td>1.66</td>
</tr>
<tr>
<td>clm1</td>
<td>133</td>
<td>146</td>
</tr>
<tr>
<td>des</td>
<td>16.5</td>
<td>17.7</td>
</tr>
<tr>
<td>diffreq</td>
<td>2.21</td>
<td>3.14</td>
</tr>
<tr>
<td>dsip</td>
<td>6.29</td>
<td>7.81</td>
</tr>
<tr>
<td>elliptic</td>
<td>18.2</td>
<td>23.6</td>
</tr>
<tr>
<td>ex1010</td>
<td>37.1</td>
<td>41.2</td>
</tr>
<tr>
<td>ex5p</td>
<td>3.86</td>
<td>4.85</td>
</tr>
<tr>
<td>Fricc</td>
<td>22.0</td>
<td>26.5</td>
</tr>
<tr>
<td>misex3</td>
<td>4.35</td>
<td>5.72</td>
</tr>
<tr>
<td>Pdc</td>
<td>58.0</td>
<td>66.0</td>
</tr>
<tr>
<td>s298</td>
<td>12.5</td>
<td>21.1</td>
</tr>
<tr>
<td>s38417</td>
<td>24.2</td>
<td>32.7</td>
</tr>
<tr>
<td>s38514</td>
<td>25.1</td>
<td>27.0</td>
</tr>
<tr>
<td>Seq</td>
<td>5.47</td>
<td>6.40</td>
</tr>
<tr>
<td>Spla</td>
<td>25.8</td>
<td>39.2</td>
</tr>
<tr>
<td>Tseng</td>
<td>0.69</td>
<td>1.00</td>
</tr>
<tr>
<td>Average:</td>
<td>16.29</td>
<td>19.9</td>
</tr>
<tr>
<td>Ratio:</td>
<td>1.00</td>
<td>1.22</td>
</tr>
</tbody>
</table>

From the results provided in Table 1, the existing way of applying the three fault tolerant techniques exhibit increased EDP values, as compared to initial mapping, ranging from 1.22 up to 1.36 times. On the other hand, when we apply redundancy only on the application’s functionalities mapped onto the 40% of device hardware resources placed on the middle of FPGA array, then the EDP overhead range from 0.99 up to 1.1 times. These values are almost similar to those retrieved when the device is designed without redundancy.

5 Conclusions

A novel framework for exploring and supporting fault tolerance at FPGAs was introduced. This framework can support alternative fault tolerance mechanisms based on voting, but rather than annotating the whole application’s description with redundancy (as existing approaches do), we replicate only application’s functionalities implemented onto regions with increased failure.
Since our framework does not require dedicated hardware blocks for supporting fault tolerance, it leads to increased flexibility, while it can be also applied to commercial devices as an additional step between synthesis and P&R. Moreover, our approach can provide a tradeoff between the desired reliability level and the extra overhead due to extra hardware resources. Based on experimental results, we shown that the proposed fault tolerant technique is not so much expensive in term of area, delay and energy dissipation, as compared to existing redundant solutions, while it leads to almost similar error correction.

References