SEU Recovery Mechanism for SRAM-Based FPGAs

Uros Legat, Student Member, IEEE, Anton Biasizzo, and Franc Novak, Member, IEEE

Abstract—The application of SRAM-based field-programmable gate arrays (FPGAs) in mission-critical systems requires error-mitigation and recovery techniques to protect them from the errors caused by high-energy radiation, also known as single event upsets (SEUs). For this, modular redundancy and runtime partial reconfiguration are commonly employed techniques. However, the reported solutions feature different tradeoffs in the area overhead and the fault latency. In this paper, we propose a low area-overhead SEU recovery mechanism and describe its application in different self-recoverable architectures, which are experimentally evaluated using a specially designed fault-emulation environment. The environment enables the user to inject faults at selected locations of the configuration memory and experimentally evaluate the reliability of the developed solutions.

Index Terms—Fault emulation, partial runtime reconfiguration, self-recovery, single event upset.

I. INTRODUCTION

SRAM-BASED field-programmable gate array (FPGA) devices are increasingly being used for implementing mission-critical and reliable systems. The advantage of FPGAs is their high reconfigurability, which enables fast prototyping, on-site hardware upgrades, flexible functionality through partial reconfiguration, and on-site configuration recovery. Due to the increasing integration density FPGA devices are getting prone to faulty behavior, caused by cosmic or artificial radiation [1]–[3]. This radiation exposure can cause a bit flip in the user memory or configuration memory of the FPGA. These soft errors are modeled as single event upsets (SEUs).

Radiation is a major concern in space [4], while the systems in avionics and at ground level are less exposed to it because of the planet’s atmospheric and magnetic radiation shield. However, experiments [1], [2] have shown that with an increased density of integrated circuits, the neutron particles present in the atmosphere are also capable of producing SEUs. It is therefore imperative that FPGA-based applications, where high reliability is required, include mechanisms that can mitigate or easily and quickly recover the system from SEUs.

Many techniques have been developed recently to protect critical systems based on SRAM FPGAs against an SEU [5]. At the design level of the FPGA, these techniques are classified as SEU-mitigation techniques, which prevent the SEU from affecting the target design, and SEU-recovery techniques, which repair erroneous bits of the FPGA configuration memory.

The most common SEU-mitigation techniques employ modular redundancy like duplication and comparison [6], or triple modular redundancy (TMR). The TMR technique was proposed by [7] and was extensively investigated for mitigating the SEUs in FPGAs. Since the voter in the TMR circuit is vulnerable to faults, Carmichael [8] proposed a method for hardening the TMR in Xilinx FPGAs by triplicating the voter and implementing it using dedicated FPGA resources. TMR designs on the FPGA are susceptible to faults in the FPGA’s internal routing that can cause errors on multiple modules [9]. Various algorithms and design methods have been proposed to reduce the number of such errors [10]–[13]. Error-correcting codes (ECCs) are also used to mitigate SEUs in integrated circuits. Many such codes are used to protect systems against single and multiple SEUs. The most common ECCs are Hamming codes and Reed–Solomon codes. ECCs are mostly used to protect system memories.

SEU-recovery techniques in SRAM FPGAs are also known as configuration scrubbing. The basic principle of these techniques is to use the FPGA configuration interface to recover the original state of the FPGA configuration memory. Simple scrubbing mechanisms periodically reconfigure the whole device. However, the scrubbing period has to be shorter than the estimated mean time between two SEUs. More advanced recovery mechanisms use ECCs [14] or CRCs [15] to check the integrity of the FPGA configuration memory and use partial reconfiguration to recover the device.

Depending on which FPGA configuration interface is used to reconfigure the device, scrubbing techniques are classified as either external or internal. External scrubbing techniques use external configuration ports (i.e., JTAG, SelectMap) and require an external radiation-hardened scrubbing controller (processor [16], FPGA [17], or ASIC [18]), and an external radiation-hardened memory to store the so-called “Golden copy” of the FPGA configuration bits. Internal SEU-recovery techniques use an internal configuration interface [for example, the internal configuration access port (ICAP)] to access the configuration memory of the FPGA. The scrubbing is also controlled internally, and the controller usually consists of an embedded microprocessor [14], [20], [21].

The SEU detection-and-recovery mechanism should be fast in order to reduce the system-error latency. Besides, since it is implemented with similar FPGA resources as the target application, the mechanism itself is subject to SEUs. It is therefore imperative that its hardware overhead is as small as possible. In this paper, we describe the implementation of an SEU detection-and-recovery mechanism that outperforms the existing solutions in terms of speed and/or a small hardware overhead.
Fig. 1. The structure of the configuration memory in the Xilinx FPGA.

Based on the mechanism, we propose different recovery architectures and evaluate them using a specially designed fault-emulation environment that allows the injection of faults into specified FPGA resources. The obtained failure-in-time (FIT) estimation can then be used to select the appropriate level of reliability.

II. SINGLE EVENT UPSETS IN SRAM FPGA

The functionality of a SRAM FPGA is determined by the state of its configuration memory (i.e., the SRAM cells). In the Xilinx FPGA, the configuration memory is organized in a network of configuration frames that are laid out on a device according to their frame address. A configuration frame is the smallest reconfigurable part of an FPGA. The size of a frame in the Virtex 4 and Virtex 5 FPGA is 41 words of 32-bits. (It is to these two Xilinx families that the solutions presented in this paper are directly applicable.) The frame address is composed of the block type, the top/bottom bit, the row, the major address, and the minor address. The structure of the FPGA configuration is depicted in Fig. 1.

The SRAM FPGA configuration memory is susceptible to radiation. A charged particle can cause a configuration bit to change its state and consequently alter the FPGA’s functionality. A configuration bit is associated with a particular part of the FPGA. It can be a part of an internal memory of the device, like block random-access memory (BRAM) or a flip-flop, or it can represent a functional part of the design, like a configurable logic block (CLB) or internal routing.

Note that a typical user design employs only a part of the whole FPGA configuration image. Therefore, an SEU in the redundant parts of the FPGA can be ignored. The faults in the configuration bits that are used to define the target design can affect the operation of the design. These configuration bits are considered to be potentially critical.

An SEU can also occur in a vital control part of the FPGA configuration memory, which causes a regional or device-wide failure. These faults are referred to as single event functional interrupts (SEFIs) [14]. The observed SEFIs in the Virtex 4 and 5 devices are power-on-reset (POR), select map or ICAP registers, digital clock managers (DCMs), and global signals (Global Write Enable, Global 3-state Control, GHIGH, etc.).

III. INTERNAL ERROR-RECOVERY MECHANISM

The errors in the FPGA configuration memory are recovered by a small internal error-recovery mechanism through the ICAP. The mechanism occupies a small portion of the FPGA, while the rest of the configuration memory can be used for the target application. The efficient implementation of the error-recovery mechanism is very important, since a smaller design has a lower probability of the SEU affecting the correct operation of the mechanism.

A. Configuration Check and Recovery Technique

The configuration frames can be read (readback) or written (reconfigured) in runtime using the ICAP. Each configuration frame in the Virtex 4 and Virtex 5 FPGA contains 12 parity bits. These are the parity bits of the Hamming error correction code (ECC). The error within a frame is determined by a syndrome value, which is calculated from the 12 parity bits and the other 1300 bits of the read frame data. Table I provides a decomposition of the syndrome value and its corresponding error status.
The first 11 bits of the syndrome value $S[10 : 0]$ identify the location of a single erroneous bit within the frame (including the errors in the parity bits), while the last bit of the syndrome value $S[11]$ indicates a double error in the frame. Virtex 4 and Virtex 5 FPGA have an embedded ECC circuit that calculates the syndrome value during each frame readback.

**B. Implementation of Error-Recovery Mechanism**

The hardware architecture of our error-recovery mechanism is depicted in Fig. 2. It consists of an ICAP device, frame ECC device, dual-port block RAM, and control logic.

The FPGA device is configured by writing the configuration commands into the configuration registers. The Xilinx Virtex 4, 5 Configuration User Guides give a detailed description of the register types and commands to perform the configuration operations.

The ICAP device has direct access to the configuration registers. The error-recovery mechanism uses the ICAP to read and write a configuration frame. The readback and reconfiguration operations are performed using an appropriate sequence of 32-bit configuration commands sent to the ICAP input. These commands are predefined and stored in the internal memory of the FPGA BRAM.

The frame ECC device is used to detect and locate errors inside the FPGA configuration frame. It works in parallel with the ICAP device. While the ICAP device reads the particular frame, the frame ECC uses the frame data to compute the syndrome value.

The internal block RAM is used to store the configuration commands and to buffer the frame data during the error correction. The BRAM is also susceptible to SEU. To protect the integrity of the BRAM content in the Virtex 4 and Virtex 5 FPGAs, the BRAM ECC option is enabled. BRAM can be configured as a single 512 × 64-bit RAM with Hamming error correction using extra eight bits in the 72-bit-wide RAM. The eight protection bits are generated during each read operation to correct any single error or to detect any double error. The status output indicates the error status. The ECC in the Virtex 4 FPGA uses two vertically adjacent 18-kb BRAMs, while the Virtex 5 FPGA uses one 36-kb BRAM.

The control logic manages the error detection-and-correction process. The controller is composed of a finite state machine (FSM), a frame address counter, and an error detection logic. The frame address counter consists of separate counters for the major address, minor address, row, and top/bottom bit that form a 32-bit frame address. When the next frame address is required, the counters are incremented. When the frame address reaches the last frame, the counters are reset back to their initial values pointing to the first frame. The error-detection logic determines the location of a single error within the frame. It examines the syndrome value output from the ECC device and

---

**TABLE I**  
SYNDROME VALUE AND CORRESPONDING ERROR STATUS

<table>
<thead>
<tr>
<th>Syndrome</th>
<th>Error status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 11</td>
<td>Bit 10 to 0</td>
</tr>
<tr>
<td>$S[11]=0$</td>
<td>$S[10:0]=0$</td>
</tr>
</tbody>
</table>

---

Fig. 2. Hardware architecture of the error-recovery mechanism.
calculates the BRAM address and the bitmask used for the error recovery.

C. Operation of the Internal Recovery Mechanism

The state machine of the internal recovery mechanism is shown in Fig. 3. The scrubbing process begins with a start state where the signals are initialized. Then, the device configuration readback is started in the initiate readback state. In the check frame state, the syndrome value of the current frame is checked through the frame ECC device by the error-detection logic. The correct frame address is maintained by incrementing the frame address counter for every pulse of the SYNDROMEV ALID signal. Depending on the number of faults inside a single frame, the following scenarios are possible:

- In the case of a single fault, the single error is detected, and the device readback is stopped. The erroneous frame is read and stored in the BRAM (read frame state). The faulty bit determined by the error-detection logic is corrected, and the frame is reconfigured (correct frame state). After the reconfiguration, the frame is rechecked in the check frame state, and if the error has been recovered, the mechanism restarts the device readback (initiate readback state). If the error has not been recovered, the mechanism switches to the stop state, and it reports an uncorrectable single error, which may be a result of multiple faults or a hard-fault.

- In the case of a double fault, the error is detected but not corrected. The mechanism switches to the stop state. The scrubbing process is stopped, and the double error is re-reported.

- If more than two faults occur in one configuration frame, the operation of the mechanism becomes unreliable. It may happen that the faults are undetected (fault masking) or wrongly identified and treated as a single fault. In the latter case, the error recovery mechanism injects another fault in the configuration frame. By rechecking the frame, the mechanism switches to the stop state and reports the uncorrectable single error.

Empirical evidence [2], however, suggests that the probability of a multiple fault in a frame is extremely low and can be neglected.

When the last frame is read, the mechanism goes back to the initiate readback state and restarts the device readback.

D. Hardware-Implementation Comparison

The internal error-recovery mechanism was implemented in the Virtex 4 and Virtex 5 FPGAs. In Virtex 4, it occupies 156 slices, 118 flip-flop registers, and 1 BRAM. If the BRAMs are protected by an ECC, two BRAM instances are occupied. In Virtex 5, the scrubbing mechanism occupies just 72 slices, 115 flip-flop registers, and 1 BRAM with ECC. The implementation results can differ slightly when different options are selected in the synthesis tool.

Table II shows a comparison of the used hardware resources with two other reported scrubber implementations. Both of them use an embedded microprocessor to control the scrubbing process, resulting in significantly higher hardware overheads. The microprocessor indeed offers some additional options, like control or debug through an RS232 interface and fault injection. However, these options are not required for the actual recovery process and may even reduce the reliability of the system.

E. Error-Recovery-Time Comparison

The error-recovery time is an important characteristic of the error-recovery mechanism. Shorter recovery time means lower probability that a SEU-induced fault will affect the operation of the target application. The error-recovery time is the sum of the error-detection time and the error-correction time. The worst-case error-detection time is the period in which the mechanism checks all the configuration frames and it depends on the size of the FPGA device. Our error-recovery mechanism first initiates the configuration readback and then checks each frame in 41 clock cycles. It checks one 32-bit word per clock cycle. When a single error occurs, our mechanism determines the erroneous frame, corrects the fault, and reconfigures it in 210 clock cycles.

Table III shows a comparison of the error-recovery times with other reported implementations.

The scrubbing mechanism for Virtex 4 was compared with the report of Heiner et al. [20]. The error detection is performed by a PicoBlaze processor through a device readback. Their mechanism achieves a comparable error-detection time, but when the error is detected, they do not stop the readback, and they do not have the information about the frame address of the erroneous frame. Hence, they have to perform another
check on a frame-by-frame basis, which results in a very long error-correction time.

Virtex 5 devices have an autonomous dedicated circuit (readback CRC) that performs a continuous readback of the device. The scrubbing mechanism for Virtex 5 [21] employs a readback CRC circuit to detect the errors. The error-detection time of the mechanism is the same as in our case; however, the correction time is longer (~12,500 clock cycles). The scrubbing is controlled by an 8-bit PicoBlaze processor. The frame read and reconfiguration is four times slower due to the 8-bit processor bus.

F. Implementation of the Internal Error-Recovery Mechanism in Triple Modular Redundancy

An internal error-recovery mechanism is also susceptible to SEUs. A critical fault in the mechanism could cause a system-wide corruption of the configuration data. Therefore, in the case of highly reliable systems, the logic of the mechanism should be protected by a SEU-mitigation technique.

In our case, the original implementation of the error-recovery mechanism is small, and the BRAM is protected by an ECC. To further increase the reliability of the mechanism, we implemented the mechanism in TMR. The hardware architecture of the TMR is depicted in Fig. 4. The TMR is applied to the control logic and the BRAMs. The majority voter is placed at the inputs of the ICAP device. The outputs of the ICAP device and the frame ECC device are triplicated and fed back to the inputs of the design modules. These triplicated design modules are clocked by separate synchronous clock signals. The ICAP and ECC devices cannot be triplicated because there are only one frame ECC and only two ICAP devices in a Virtex 4 or Virtex 5 FPGA. Furthermore, ICAP devices cannot be used simultaneously. The fact that ICAP and ECC devices cannot be triplicated could be regarded as a concern for a single point of failure. However, these devices are hardwired components of the FPGA and therefore more robust.

Effective implementation of the recovery mechanism in TMR requires the following considerations.

• The triplicated modules and the majority voter have to be placed separately, isolated from each other.
• The internal signals have to be carefully routed to limit the possibility that a SEU would affect more than one module.
• The connections between the voter and the ICAP device have to be as short as possible.
• The output signals of the ICAP and the ECC device have to branch into three separate signals as close as possible.

These considerations are difficult to achieve using the standard implementation tools such as standard Xilinx ISE and PlanAhead tools that we used. In the ISE tool, the synthesis options and constraints were set to prevent merging of the modules during optimization procedure. In the PlanAhead tool, different placement partitions were made for the three modules and the voter. The modules were placed separately in different clock regions, and the voter was placed close to the ICAP inputs. The triplication of the ICAP and ECC output signals was achieved by using internal buffers (BUFCFs). In order to shorten the length of the nontriplicated part of the signals, the buffers were placed close to the outputs of the ICAP and ECC devices.

Using standard Xilinx tools, we are able to place the components of the TMR modules but are unable to directly affect the routing. Our partial solution to this problem was to place the modules far apart. This way, the probability that the signals of one module pass through the partition of another module is reduced.

The remaining single points of failure due to SEU of TMR design are as follows:

• voter logic and voter output signals;
• outputs of ICAP and ECC device before they branch into the three separate signals;
• failures in routing which affect signals of more than one module;
• an occurrence of SEFI.

The number of failures in routing that affect more than one module could be further reduced if we had at disposal a tool that would allow us to constrain routing from certain partitions and sufficiently physically separate the signals to avoid bridging faults.

Hardware-utilization results of the TMR scrubber implemented in Virtex 5 are shown in Table IV. In comparison with the original version of the recovery mechanism, the TMR increases the number of occupied resources by some more than three times.

We have analyzed the power consumption of our mechanism using Xilinx XPower tool on Virtex 5 FPGA at a 100 MHz clock rate. The TMR version of the mechanism has approximately

---

TABLE III

<table>
<thead>
<tr>
<th>Time is stated in clock cycles</th>
<th>Virtex 4 (XC4VLX15)</th>
<th>Virtex 5 (XC5VLX30)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst-case Error-detection time</td>
<td>147650</td>
<td>226115</td>
</tr>
<tr>
<td>Worst-case Error-correction time</td>
<td>~150000</td>
<td>210</td>
</tr>
<tr>
<td>Worst-case Error-recovery time</td>
<td>~2400000</td>
<td>226325</td>
</tr>
</tbody>
</table>

---

Fig. 4. Hardware-architecture block diagram of error-recovery mechanism implemented in TMR.
TABLE IV

Comparison of Resources, Power, and Timing of the Original and TMR Version of Error-Recovery Mechanism

<table>
<thead>
<tr>
<th>Virtex 5</th>
<th>Original mechanism</th>
<th>TMR mechanism</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resources</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slices</td>
<td>72</td>
<td>321</td>
</tr>
<tr>
<td>Flip-flops</td>
<td>115</td>
<td>345</td>
</tr>
<tr>
<td>BRAM</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Power consumption</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic (mW)</td>
<td>20</td>
<td>56</td>
</tr>
<tr>
<td>Static leakage (mW)</td>
<td>894</td>
<td>895</td>
</tr>
<tr>
<td>Total (mW)</td>
<td>914</td>
<td>951</td>
</tr>
<tr>
<td>Timing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max Clock (MHz)</td>
<td>282</td>
<td>261</td>
</tr>
</tbody>
</table>

TABLE V

Resource Comparison of TMR Scrubber on Virtex 4

<table>
<thead>
<tr>
<th>TMR on Virtex 4</th>
<th>Our mechanism</th>
<th>Heiner et al. [20]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slices</td>
<td>671</td>
<td>1308</td>
</tr>
<tr>
<td>Flip-flops</td>
<td>354</td>
<td>1082</td>
</tr>
<tr>
<td>BRAM</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

three times higher dynamic power consumption than the original mechanism. On the other hand, due to the high static power consumption of the FPGA, the total power consumption of the TMR version is only 4% higher than the power consumption of the original mechanism.

A timing analysis was done using Xilinx ISE tool. The maximum clock frequency of the TMR version of the error recovery mechanism is 8% lower than the clock frequency of the original version. The decrease of the maximum clock frequency is the result of a slightly longer critical path. The frequency of 261 MHz is still more than enough since the ICAP circuit that is used by the mechanism is recommended to run below 100 MHz.

A TMR version of scrubber was also implemented by Heiner et al. [20]. They triplicated the PicoBlaze controller and the BRAM. The utilization results presented in Table V show that their hardware design is considerably larger.

IV. SELF-RECOVERABLE ARCHITECTURES

The proposed error-recovery mechanism can be employed in different configurations of self-recovery systems.

A. Recovery System With Internal Recovery Mechanism

The basic error-recovery architecture is shown in Fig. 5. It contains only the internal error-recovery mechanism, which runs at the same time as the target application. It corrects single faults and detects double faults in the configuration memory. This recovery architecture is suitable for systems with limited resources. The internal scrubber occupies less than 1% of the slices of the smallest Virtex 5 FPGA. This error-recovery architecture does not have any external recovery procedure. The system fails if an upset occurs on a critical bit of the internal error-recovery mechanism and it cannot recover itself from the SEFI.

B. Recovery System With Internal Recovery Mechanism and External Watchdog Timer

Fig. 6 shows the block diagram of error-recovery architecture with internal and external recovery procedures. The error-recovery architecture consists of the following:

- an FPGA, that contains the internal error-recovery mechanism and the user application, which run concurrently;
- an external watchdog timer, which monitors the vital signals of the internal error-recovery mechanism;
- an external nonvolatile memory, which holds the original (golden) configuration data.

When the external watchdog timer detects the wrong operation of the internal error-recovery mechanism, it recovers the system from the external memory. The recovery from the external memory is also triggered by the double error signal from the error-recovery mechanism. This self-recovery architecture is effective against SEFI. In order to provide an efficient operation of the watchdog timer, the monitored signals of the internal error-recovery mechanism must be carefully selected. In some circumstances, the failure of the internal error-recovery mechanism remains undetected, and the system fails to recover. An experimental case study is given in the next section.
C. Recovery System With Internal Recovery Mechanism in the TMR

The recovery architecture with an internal scrubber in the TMR is shown in Fig. 7. This system is vulnerable to the upsets occurring in the small voter circuit or to the upsets that affect the operation of two modules at the same time. The architecture does not have any external recovery procedure, and it is not effective against SEFI. It is suitable for reliable systems that have enough available internal FPGA resources and are not able to include any additional external circuitry.

D. Recovery System With Internal Recovery Mechanism in TMR and External Watchdog Timer

This architecture offers the highest level of reliability. It combines internal scrubber in TMR with the external watchdog timer. Fig. 8 shows a block diagram of the architecture. The watchdog timer monitors the vital signals of the internal scrubber and recovers the system if it detects a failure. The system can be recovered from most failures, including the failures in the majority voter of the TMR scrubber and SEFI failures.

V. FAULT-EMULATION EXPERIMENT

In order to determine the reliability of the proposed recovery architectures, we developed an environment for SEU emulation. The fault injection was performed by changing the logic state of individual FPGA configuration bits using a runtime reconfiguration. Such a fault-emulation approach enabled us to inject faults at a precise location within the FPGA configuration memory.

The function of the particular configuration bit has been determined using the device configuration user guide and by the acquired knowledge of the bitstream structure. We are able to discriminate between the configuration bits corresponding to routing, lookup tables (LUT) and other components of the CLB, and BRAM.

The fault-emulation experiment was performed using an internal fault-injection mechanism controlled and monitored by a computer. The hardware structure of the SEU emulation is shown in Fig. 9. The computer serves as a fault-emulation controller, external memory, and reconfiguration device. It also acts as a watchdog timer for the error-recovery mechanism.

The error-recovery mechanism and the fault-injection mechanism are placed in the FPGA configuration separately from each other. Faults are injected only into the part of the configuration memory containing the error-recovery mechanism since the faults in other parts of the configuration do not affect the error-recovery mechanism and are always corrected.

During the fault-emulation experiment, the fault-injection mechanism injects faults and controls the error-recovery mechanism: It triggers or stops it, and it is able to check if the injected fault has been corrected. The error-recovery mechanism performs only a single configuration scan in contrast with the normal operation, where it continuously performs configuration scans.

The process of fault emulation begins by configuring the FPGA device with an error-free configuration. After the initial configuration, the following actions are performed for every fault from the fault list:

- the fault injection mechanism injects the fault into the configuration memory of the error-recovery mechanism by partial runtime reconfiguration;
- the error-recovery mechanism is triggered. It performs one scan of the configuration memory.
Depending on how the error-recovery mechanism was affected by the injected faults, the following situations may occur:

1) The error-recovery mechanism performs correctly. In this case, the error-recovery mechanism corrects the fault and completes the scan cycle, which resets the watchdog timer. The fault-injection mechanism confirms that the fault has been corrected and goes to the next fault on the fault list. The computer logs the result.

2) The error-recovery fails and the failure is detected by the watchdog timer. In this case, the error-recovery mechanism does not complete the scan cycle and does not reset the watchdog timer. The computer logs the result, reconfigures the error-recovery mechanism from the stored partial configuration image, and proceeds with the injection of the next fault.

3) The error-recovery mechanism fails, and the failure is not detected by the watchdog timer. In this case, the error-recovery mechanism completes the cycle and resets the watchdog timer but does not correct the error. The fault-injection mechanism detects the failure by verifying if the recovery mechanism corrected the fault. The computer logs the result, reconfigures the error-recovery mechanism from the stored partial configuration image, and proceeds with the injection of the next fault.

Faults are injected in all bits of configuration frames occupied by the error recovery mechanism (regular or TMR version). Two separate fault-emulation experiments were performed to evaluate the reliability of the four proposed self-recovery architectures.

In the first experiment, 181 056 faults were injected into the 138 configuration frames occupied by the original implementation of the internal error-recovery mechanism to evaluate the architectures described in Sections IV-A and IV-B. The results of the fault emulation are presented in Table VI. Among all injected faults, 9177 faults affected the operation of the internal error-recovery mechanism. These faults (denoted as critical faults) were further classified by their location. As presented in Table VI, the majority of the faults (78%) correspond to the FPGA routing, 21% correspond to CLBs, and 1% to the BRAM configuration.

Using the external watchdog timer, 8458 faults were detected among 9177 critical faults, while 719 (7.8%) remained undetected. We further examined the operation of the error-recovery mechanism under the influence of these undetected faults. The mechanism failed to correct the injected fault but did not further corrupt the configuration memory.

In the second experiment, 708 480 faults were injected into the 540 configuration frames occupied by the TMR implementation of the internal error-recovery mechanism to evaluate the architectures described in Sections IV-C and IV-D. The results of the fault-emulation are presented in Table VII. The faults of each TMR module and the majority voter are analyzed separately.

The fault analysis of the TMR modules shows the following.

- All faults of the LUT content and BRAM configuration were successfully mitigated.
- Some faults in the routing of TMR modules affect the performance of the TMR. The reason for this is the fact that the synthesis tool routes some signals of one module through the partition of another module. Faults in such a signal can affect both modules and the TMR. This situation could be eliminated by manually rerouting these signals or using dedicated routing tools.
- Some faults injected in CLB configuration bits of the triplicated modules also affect the TMR. These faults alter the basic functionality of the resources in the CLB.

The fault analysis of voter reveals that it is the most vulnerable part of the TMR structure, as follows.

- The majority of routing faults (250) occurred at the voter circuit. There are two types of routing failure: the bridging faults at the input of the voter, and faults in the connections between the voter and the ICAP port.
- The logic of the majority voters is implemented by LUTs. 69 faults injected into LUT content corrupted the logic of the voter.

An erroneous operation of the mechanism was detected in 417 cases. The watchdog timer detected the erroneous behavior of the error-recovery mechanism in 93%, reducing the SEU cross-section to only 30 critical faults.

### Table VI: Fault-Emulation Results for the Architectures Sections IV-A and IV-B on Virtex 5 FPGA

<table>
<thead>
<tr>
<th>Number of critical faults (of 181056 injected faults)</th>
<th>Errorous operation of internal scrubber</th>
<th>Detected by watchdog timer</th>
<th>Undetected by watchdog timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routing</td>
<td>7180</td>
<td>6626</td>
<td>554</td>
</tr>
<tr>
<td>LUT content</td>
<td>1429</td>
<td>1307</td>
<td>122</td>
</tr>
<tr>
<td>CLB configuration</td>
<td>509</td>
<td>472</td>
<td>37</td>
</tr>
<tr>
<td>BRAM configuration</td>
<td>59</td>
<td>53</td>
<td>6</td>
</tr>
<tr>
<td>Sum</td>
<td>9177</td>
<td>8458</td>
<td>719</td>
</tr>
</tbody>
</table>

### Table VII: Fault-Emulation Results for the Architectures Sections IV-C and IV-D on Virtex 5 FPGA

<table>
<thead>
<tr>
<th>Number of critical faults (of 708480 injected faults)</th>
<th>Errorous operation of TMR scrubber</th>
<th>Detected by watchdog timer</th>
<th>Undetected by watchdog timer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Routing</td>
<td>16</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>LUT content</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CLB configuration</td>
<td>26</td>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>BRAM configuration</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Module 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Routing</td>
<td>9</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>LUT content</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CLB configuration</td>
<td>24</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>BRAM configuration</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Module 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Routing</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>LUT content</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CLB configuration</td>
<td>23</td>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>BRAM configuration</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Voter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Routing</td>
<td>250</td>
<td>227</td>
<td>23</td>
</tr>
<tr>
<td>LUT content</td>
<td>69</td>
<td>62</td>
<td>7</td>
</tr>
<tr>
<td>CLB configuration</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>BRAM configuration</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sum</td>
<td>417</td>
<td>387</td>
<td>30</td>
</tr>
</tbody>
</table>
TABLE VIII
SEU RELIABILITY ESTIMATION

<table>
<thead>
<tr>
<th>System on xc5vlx110t</th>
<th>Number of critical bits</th>
<th>FIT (SEU/10^9h)</th>
<th>MTBF (Years)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The whole FPGA device</td>
<td>31.1 Mb</td>
<td>5039.8</td>
<td>22.7</td>
</tr>
<tr>
<td>Average design without error recovery</td>
<td>3.42 Mb</td>
<td>554.4</td>
<td>206</td>
</tr>
<tr>
<td>Self-recovery architecture IV A</td>
<td>9177 b</td>
<td>1.5</td>
<td>7.68 · 10^7</td>
</tr>
<tr>
<td>Self-recovery architecture IV B</td>
<td>719 b</td>
<td>0.12</td>
<td>1.0 · 10^8</td>
</tr>
<tr>
<td>Self-recovery architecture IV C</td>
<td>417 b</td>
<td>0.07</td>
<td>1.7 · 10^8</td>
</tr>
<tr>
<td>Self-recovery architecture IV D</td>
<td>30 b</td>
<td>0.005</td>
<td>23.0 · 10^5</td>
</tr>
</tbody>
</table>

The presented TMR fault emulation results were archived by using standard implementation tools. By following the TMR design considerations from Section III-F, we were able to reduce the number of critical faults for more than 90%. A further hardening could be achieved by applying a fault-tolerant routing algorithm [22] and/or using dedicated tools for FPGA TMR design. For older FPGA families, there was the Xilinx TMR tool; however, it is no longer available nor supported.

VI. RELIABILITY ESTIMATION

The reliability of Xilinx devices is being evaluated in an ongoing Rosetta experiment [2]. The current reliability estimation of the devices can be found in the Xilinx device reliability report [23]. The SEU error rate is stated in terms of the FIT or the mean time between two failures (MTBF). The FIT is the number of failures that can be expected in 10^9 hours of operation. Our experimental results are obtained from a Virtex 5 FPGA; therefore, we concentrated on the reliability of our self-recovery architectures implemented on this particular device. The nominal failure rate in the Earth’s atmosphere for Virtex 5 FPGA according to [23] is 162 FIT/Mb. Experiments in [24, 25] show that the failure rate of the whole device can be multiplied with the percentage of critical bits acquired with fault emulation to predict the failure rate of the application on FPGA. Therefore, we can estimate the reliability of particular target application by multiplying the nominal failure rate of the FPGA device with the number of critical bits of the application.

Table VIII presents the reliability estimation for different design scenarios. The entire FPGA configuration has over 5000 FIT. However, a design uses only a certain number of configuration bits. We made an estimation of the reliability of an average design on the XC5VLX110T. According to [25], an average design with a device utilization of 80% has about 11% of bits that are critical to its operation. The average design without error recovery has approximately 554 FIT.

The self-recovery architecture Section IV-A has an internal error-recovery mechanism included in the target design. Only the bits that compromise the operation of the error-recovery mechanism are critical. The fault-emulation experiment indicated that 9177 bits in the configuration of the error recovery mechanism are critical. The estimated reliability of our internal error-recovery mechanism is 1.49 FIT. For comparison, the Xilinx SEU controller reported in [21] has an estimated 8.6 FIT and is less reliable. The self-recovery architecture Section IV-B with the external watchdog timer also recovers from the majority of cases where the internal recovery fails. The number of critical bits is reduced to 719, and the reliability is increased to 0.12 FIT, which results in 1 million years of MTBF.

The self-reparable architecture IV C has an internal error-recovery mechanism implemented in the TMR. The failure of the internal TMR was detected in 417 injected faults, which results in an estimated 0.07 FIT. The external watchdog timer in the architecture IV D detected 93% of the failures and increased the reliability by ten times to 23 million years of MTBF.

VII. CONCLUSION

A SEU-recovery mechanism for SRAM-based FPGAs controlled by FSM was developed. The SEU-recovery mechanism has the smallest hardware overhead. The mechanism was also hardened by the TMR technique. According to the required levels of reliability, the mechanism can be employed in different self-recoverable architectures.

The efficiency of the proposed approach was evaluated with a specially developed fault-emulation environment. The resulting estimated reliability is superior to the other reported solutions. In contrast to conventional statistical methods based on radiation techniques, the developed fault emulation enables the user to inject faults at selected locations of the configuration memory. Then, individual parts of the recovery infrastructure can be analyzed. This way, modifications and possible improvements to the recovery infrastructure can be easily evaluated.

Our error-recovery mechanism supports the Xilinx Virtex 4 and Virtex 5 FPGA families, and it can be easily extended to include Virtex 6 devices. The same recovery principle can also be applied to FPGA devices of other manufacturers.

REFERENCES


