Soft Error Rate Estimation and Mitigation for SRAM-Based FPGAs

Ghazanfar Asadi
Northeastern University
Dept. of Electrical & Computer Engineering
Boston, MA 02115
gasadi@ece.neu.edu

Mehdi B. Tahoori
Northeastern University
Dept. of Electrical & Computer Engineering
Boston, MA 02115
mtahoori@ece.neu.edu

ABSTRACT

FPGA-based designs are more susceptible to single-event upsets (SEUs) compared to ASIC designs. Soft error rate (SER) estimation is a crucial step in the design of soft error tolerant schemes to balance reliability, performance, and cost of the system. Previous techniques on FPGA SER estimation are based on time-consuming fault injection and simulation methods.

In this paper, we present an analytical approach to estimate the failure rate of designs mapped into FPGAs. Experimental results show that this technique is orders of magnitude faster than fault injection method while is very accurate. We also present a high-reliable low-cost mitigation technique which can significantly improve the availability of FPGA-based designs. This technique is able to tolerate SEUs in both user and configuration bits of mapped designs.

Categories and Subject Descriptors

B.2.3 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance; B.6.2 [**Logic Design**]: Reliability and Testing

General Terms

Reliability, Design, Performance

Keywords

SRAM-Based FPGA, Soft Error Rate Estimation, Error Recovery

1. INTRODUCTION

FPGAs are widely used in many application domains such as industrial, spacecraft, and embedded applications due to their high performance, no Non-Refundable- Engineering cost and fast Time-To-Market.

FPGA'05, February 20–22, 2005, Monterey, California, USA.

Although FPGAs provide the advantages of low-cost design and fast Time-To- Market, the importance of dependability issues limit their widespread use in mission- critical applications [19]. FPGAs are vulnerable to *Single Event Upsets* (SEUs) [7, 23]. SEUs are induced by energized particles hitting the silicon device. An SEU with sufficient energy changes the logic state of the memory element, producing a *soft error*. One possible solution to this problem is to use *radiation-hardened* FPGA devices. These devices, however, are much more expensive than Commercial-Off-The-Shelf (COTS) FPGAs; thus when cost is a major issue, the COTS devices are affordable [25]. Moreover, radiation-hardened devices are few generations behind state-of-the-art COTS devices.

A particle hit on a configuration bit causes a permanent error in the mapped design. Conventional fault-tolerant schemes [14] can only protect user-bits but not configuration bits. The only applicable fault-tolerant mechanism to protect configuration bits is to use *Triple Modular Redundancy* (TMR) scheme in all used logic and routing resources[8] [19]. However, this solution enforces high area and performance penalties. It may not be affordable to put redundancy in each and every module (or component) when power and area are important constraints. To achieve a high level of reliability, efficient approaches combine both hardware (spatial) and software (temporal) redundancies.

The first step in designing such schemes is to study the effect of soft errors at the system level and identify the most vulnerable components in the system. Using this analysis and based on the redundancy budget, these components are protected at higher priorities (using hardware or software redundancy).

Previous work on soft error rate (SER) estimation is mainly simulation-based, radiation-based, or a combination of both [2, 7, 9, 10, 11, 19, 25]. All these methods have been based on *Fault Injection* (FI) strategies. Using this methodology, a limited number of error sites are targeted for fault injection. Several workloads are then run to measure the number of detected failures by comparing the results of each run to the clean run. These steps make FI approaches both very time-consuming and inaccurate (the analysis is based on statistics). Moreover, these approaches cannot be used during design phases since they need physical implementation.

SEU mitigation is the next step after SER estimation. The previous SEU mitigation techniques impose 100%-200% overhead in terms of area and power [7, 8, 19, 20]. This extra overhead also affects the performance of the design

mapped into the FPGA. The extra power overhead limits the widespread use of these devices in reliable embedded applications.

In this paper, an accurate SER estimation method for FPGA designs is presented. The presented method does not require physical implementation, i.e. only a synthesis tool and a software program are used. In this method, we first compute the netlist error rate based on FPGA used resources (placement and routing information) for the given mapped design. Then, we compute error propagation probabilities using the gate-level netlist. Based on the netlist error rate and the error propagation probabilities, we compute the failure rate of a particular mapped design (the probability of an error appearing at system outputs).

We also report on *Mean Time To Manifest* (MTTM) error for different resources of FPGAs (e.g., routing, look-up tables, and control/clocking). Finally, we present a high-reliable and low-cost mitigation technique which can significantly reduce the failure rates of FPGA-based designs. This technique protects the configuration bits for a particular mapped design.

The rest of the paper organized as follows. Section 2 explains the previous SER estimation and SEU mitigation techniques. Section 3 describes the error models of SRAM-based FPGAs. In Section 4, the failure rate estimation of FPGA designs is presented. Section 5 describes the SER estimation of user-bits. Section 6 presents the mitigation technique. Experimental results are given in Section 7. Finally, Section 8 concludes the paper.

2. PREVIOUS WORK

2.1 SER Estimation Techniques

Error propagation probability (EPP) is one of the main factors for SER estimation of the circuit nodes [22]. Previous SER estimation techniques use fault injection and random-vector simulation [2, 7, 9, 10, 11, 19, 25]. To compute the EPP of a node, several random vectors are applied to the circuit inputs. Then the system outputs are observed to calculate the probability that the erroneous value is sensitized by the input values and is propagated to the outputs. Fault injection is done using either radiation-equipment or bit-stream alteration. Radiation-based methods [7, 9] are very expensive and they are not commonly used. These methods are mainly used for device characterization, not SER estimation of a particular mapped design.

The methods presented in [2], [10], [11], [19], and [25] compute the SER of an implemented design based on the alteration of the configuration bitstream. The device is configured for every faulty bitstream, i.e., one configuration bit is flipped for each workload. Then, it is run several clock cycles with different input vectors to compare the results with the golden-run results. These methods can be further classified into two groups, as follows. The first group [2] [25] gathers and compares the results in a host system. So, it takes too much time to do experiments for all possible faults. The second group [10] [11] [19] uses one FPGA for the faulty run and one or two other FPGA(s) for the golden run and the comparison of results. The prototype board consists of two or three FPGAs. To implement and evaluate larger designs, a new prototype board equipped with higher density FPGAs is required. Finally, as there are much more candidate locations for FI in FPGA-based designs, FI techniques are more time-consuming for FPGA designs than ASIC designs.

We previously presented our preliminary error rate estimation approach in [3]. In that work, only open errors were considered. In this work, we extend the previous work by including short errors to accurately compute the SER of a circuit mapped into an FPGA.

2.2 Mitigation Techniques

Previous SEU mitigation techniques are based on TMR. This approach imposes 2x area and power overhead [8]. While TMR schemes can mask single error, they will fail if errors accumulate in the circuit. To prevent accumulated errors, *scrubbing* can be used. Scrubbing includes reading back the configuration bits, comparing those with the original configuration bits, and writing the correct bits once there is an error. The combination of TMR and scrubbing gives a high-reliable framework with the cost of 200% area overhead.

Another mitigation technique presented in [6] is based on using a CRC checker for each frame. On the Virtex devices, the configuration memory is segmented into frames. Virtex devices are partially reconfigurable and a frame is the smallest unit of reconfiguration. The number of frames and the bits per frame is different for different devices in the Virtex family. The number of frames is proportional to the *configurable logic block* (CLB) width of the device. In that method, CRC is generated for each frame during the readback and it is compared to the expected CRC value. This method greatly reduces the amount of system memory required to perform SEU detection. Two different methods have been proposed to implement CRC frame constants. For an application that will never require any updates or changes to the design after the development phase, CRC constants can be pre-generated in software and stored in system ROM for a specific FPGA design. For applications that can accept updates for the FPGAs bit-stream, CRC constants should be generated by the host system and stored in a RAM. If the FPGA bitstream is updated, then CRC values must be refreshed. In the first approach, the reconfigurability of the FPGA is missed. In the second approach, a host system is required to reconfigure the FPGA. So, it cannot be used in embedded applications.

3. BACKGROUND: FPGA ERROR MODELS

The effects of SEUs on digital circuits can be classified into a) *transient* and b) *permanent* errors. SEUs can cause transient errors in the combinational logic components, which can be propagated and captured in flip-flips. Also, SEUs can directly make transient errors on memory elements and change the contents of memory caches, main memories, register files and flip-flops (FFs). These errors are called transient because they may be overwritten or corrected using error-detection-and-correction techniques. So, transient errors impacts the user-defined logic and flip-flops of the FPGA.

Moreover, SEUs can make permanent errors on a FPGA if they alter the contents of configuration bits. Note that these errors differ from those errors which damage the device (hard errors or physical defects). In this case, the configuration bit remains erroneous until the new configuration is downloaded into the FPGA. So, these permanent errors are recoverable. In the rest of this paper, when we refer to permanent errors, we mean recoverable permanent errors.

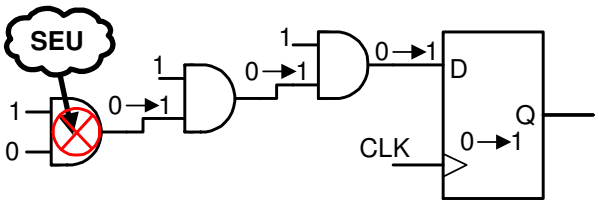

Figure 1: An SEU affects one of inputs of the AND gate and makes a bit-flip error.

The study and analysis of transient errors have been well described in [1], [5], [13], and [17]. They have investigated the circuit behavior by injecting faults into the simulation or emulation models of the design. The fault injection in these techniques implies the alteration of memory elements such as data-path registers and control-unit registers, as well as alteration of inputs, outputs, or internal signals [25]. Consequently, the effect of SEUs in the presence of the errors can be straightforwardly studied using common simulation or emulation tools.

The study of permanent errors due to configuration alteration requires more complex analysis since the simple bit-flip fault model cannot be exploited. An SEU in the device configuration bits can modify the interconnect inside a CLB. It can also affect the routing signals between different CLBs. Moreover, an SEU may change the functionality of the logic part by affecting the content of look-up tables (LUT). This issue has been addressed in [9], [19], and [25].

To summarize, there are two memory resources in FPGAs, a) user bits, and b) configuration bits. An SEU on user bits cause a transient error, and an SEU on configuration bits leads to a permanent error.

3.1 Transient Errors

Transient errors do not alter SRAM configuration bits but they affect user-defined logic and flip-flops as follows:

A bit-flip on the combinational part inside CLBs: An SEU affecting a combination part makes a transient error in logic gates. This can be propagated to the sequential part and make a bit-flip error. Figure 1 illustrates how an SEU makes a bit-flip error in a flip-flop. It has been shown that in ASIC designs, combinational logic is less susceptible to soft errors than memory elements [18] [27]. This is because the combinational logic provides some natural resistance to soft errors, including logical masking, electrical masking, and latch-window masking [27].

A bit-flip on user-defined flip-flops and memory elements: An SEU may directly affect the contents of flip-flops and memory elements. The content of the flip-flop will remain erroneous until it is rewritten with another data or it is corrected by appropriate error detecting and correcting techniques.

3.2 Permanent Errors

An SEU changing a configuration SRAM cell makes a permanent effect until the original configuration bitstream is re-downloaded into the FPGA. This type of error is the major error type in FPGAs because the number of SRAM cells dominates user-defined memory elements. Typically,

Table 1: The number of configuration bits versus the number of flip-flops in Virtex FPGAs.

Device	No. of FFs (Nff)	No. of Config. Bits (Ncb)	Nff/Ncb
XCV50	1,536	559,200	0.27%
XCV100	2,400	781,216	0.31%
XCV200	4,704	1,335,840	0.35%
XCV300	6,144	1,751,808	0.35%
XCV400	9,600	2,546,048	0.38%
XCV800	18,816	4,715,616	0.4%
XCV1000	24,576	6,127,744	0.4%
XC2V2000	21,504	7,492,000	0.29%
XC2V4000	46,080	15,659,936	0.29%
XC2V8000	93,184	29,063,072	0.32%

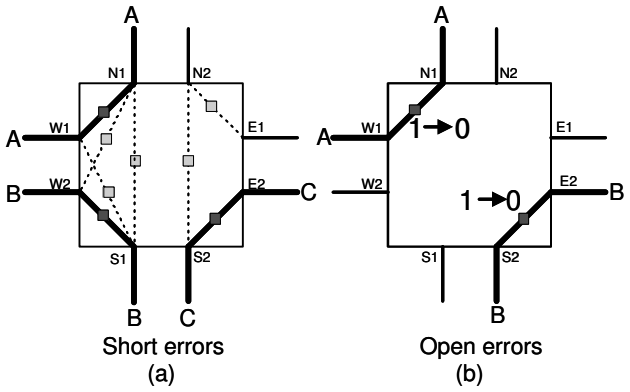

Figure 2: An impact of SEU on routing signals

the number of SRAM configuration cells are more than 98% of all memory elements inside an FPGA [29, 30]. As an example, Table 1 shows the number of configuration bits and the number of flip-flops for some Xilinx Virtex FPGA devices.

The configuration memory bits are categorized into *sensitive* and *non-sensitive* bits, according to their vulnerabilities to SEUs. An SEU in a sensitive configuration bit affects the functionality of the particular circuit mapped into the FPGA. Non-sensitive bits act as "don't care" configuration bits for that particular mapped design. Hence, the sensitivity of particular configuration bit is *application-dependent*.

Permanent errors are classified into routing errors, LUT bit-flips, and control/clocking bit-flips.

Routing errors: Programmable interconnect points (PIPs), multiplexers and buffers constitute the programmable routing network of a segmented-routing FPGA (e.g. Xilinx Virtex FPGAs). More than 80% of transistors in an FPGA are used in the routing network [28].

Routing resources can be *inter-CLBs* or *intra-CLB*. An inter-CLB routing signal connects two or more CLBs. Those that used inside a CLB are called intra-CLB signals. Switch matrices and line segments are used to route inter-CLB while multiplexers and buffers are mostly used for intra-CLB routing. Select-bits of multiplexers comprise more than half of the total susceptible SRAM cells to SEUs, as shown in [11].

An SEU changing a configuration routing bit causes a switch open, switch short, or bridging error (wired-or, wired-

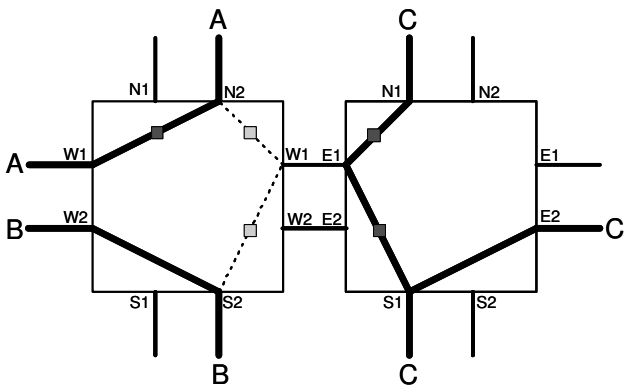

Figure 3: Inter switch matrices shorts

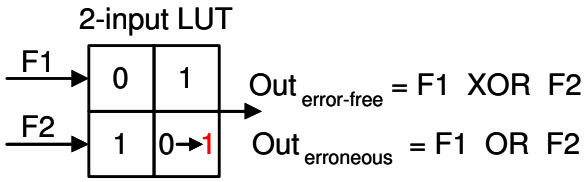

Figure 4: LUT bit-flip

and), as shown in Figure 2. Buffer errors are *buffer on* and *buffer off* errors.

Consider a chain of PIPs connecting two nodes of the circuit. An SEU changing a PIP control from 1 to 0 causes a switch open in this chain resulting in an open error in the gate-level netlist. However, the situation for 0 to 1 SEUs is different. Not all switch shorts due to a $0 \rightarrow 1$ SEU cause bridging errors in the netlist. Consider the switch matrix shown in Figure 2(a). Three different nets are shown which use PIPs (W1,N1), (W2,S1), and (S2,E2). An SEU ($0 \rightarrow 1$) on the unused PIP (W1,S1) or (N1,S1) causes a bridging error between nets A and B. These PIPs are called *sensitive* PIPs and the corresponding controlling configuration bits are *sensitive bits*. However, an SEU on the unused PIP (N2,E1) does not cause any bridging error since no two nets are adjacent to this PIP. Hence, (N2,E1) is a *non-sensitive* PIP.

Similar situation for bridging errors can happen across two adjacent switch matrices. This is due to the fact that if a switch is turned on, the wire segments adjacent to that switch are also connected. Consider the example shown in Figure 3. If there is an SEU ($0 \rightarrow 1$) on the PIP (N2,W1) in the left switch matrix, it will cause a bridging error between nets A and B, since the wire segment (W1,E1) along with this PIP form a bridge between these two nets.

A bit-flip on LUT configuration bits: A look up table implements a logic function by storing all values for the truth table. The example in Figure 4 shows how a bit-flip changes the functionality of an LUT. As shown in the figure, a bit-flip in one of LUT cells changes the original function (XOR) to an OR gate.

A bit-flip on control/clocking bits: There are some control bits inside CLBs and input-output blocks (IOBs) to determine miscellaneous functionalities. As an example, there are some control bits that determine whether the LUT performs as a look up table, a dual-ported RAM, or a programmable shift register. Also, there are some SRAM cells for clock signal routing throughout the circuit. A bit-flip on the control/clocking bits affects the functionality of the mapped design drastically.

4. FAILURE RATE ESTIMATION OF FPGA DESIGNS

There are some differences between failure probability computation in ASIC designs and FPGA-based designs:

- In ASIC designs, only the propagation of an erroneous value from the error site to primary outputs (POs) or flip-flops (FFs) has to be considered. However in FPGA designs, the activation probability as well as the propagation probability are required for failure rate estimation. This is because in FPGAs, a failure occurs if an erroneous node is first activated and then the error is propagated to POs or FFs (permanent errors).

- In FPGAs, the errors occurring in the configuration bits remain unchanged during the next clock cycles after the bit-flip. So, the same failure probability is valid for the next clock cycles. However in ASIC designs, if an erroneous value, due to an SEU, is masked and not propagated to the outputs, the effect of that SEU will be completely disappeared in the system.

- The error sites in ASICs are mainly logic gates rather than routing signals. But in FPGAs, routing signals (controlled by SRAM cells) constitute more than 70% of the total sensitive SRAM bits. [11].

- In FPGAs, if an SEU flips the content of a configuration bit, an erroneous value can be propagated from the error site to the system outputs without any attenuation (electrical masking). However, electrical masking is one of the key factors that causes the combinational logic in ASIC designs to be less susceptible to soft errors than memory elements [27]. Electrical masking occurs when the pulse resulting from a particle strike is attenuated by subsequent logic gates due to the electrical properties of the gates. This attenuation reaches to the point that the SEU does not affect system outputs.

To compute the failure rate of a design mapped into an FPGA, we perform the following steps. First, we compute the netlist failure probability. In this step, the gate-level netlist of the mapped design is used. Second, we compute the error rate of all nodes of the circuit. This step is performed based on the detailed FPGA placement and routing information of the mapped design (i.e. the detailed information regarding used and unused FPGA resources). Finally, the system failure rate is computed based on these two steps. The details of these steps are provided in the subsequent subsections.

4.1 Error Propagation Probabilities (PP_i)

While particle flux uniformly encounters the entire system, the probability of an erroneous value observed at the system outputs highly depends on which node a particle is striked and the values of other nodes of the circuit at that time (i.e. the system state).

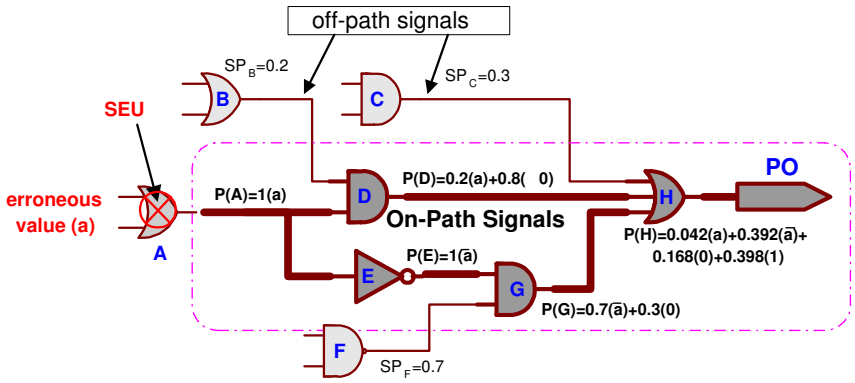

Figure 5: A typical path from an erroneous node to primary outputs/flip- flops

Table 2: Computing probability at the output of a gate in terms of its inputs

GATE	RULE
AND	$P_1(out) = \prod_{i=1}^{n} P_1(X_i)$
	$P_a(out) = \prod_{i=1}^{n} [P_1(X_i) + P_a(X_i)] - P_1(out)$
	$P_{\bar{a}}(out) = \prod_{i=1}^{n} [P_1(X_i) + P_{\bar{a}}(X_i)] - P_1(out)$
	$P_0(out) = 1 - [P_1(out) + P_a(out) + P_{\bar{a}}(out)]$
OR	$P_0(out) = \prod_{i=1}^{n} P_0(X_i)$
	$P_a(out) = \prod_{i=1}^{n} [P_0(X_i) + P_a(X_i)] - P_0(out)$
	$P_{\bar{a}}(out) = \prod_{i=1}^{n} [P_0(X_i) + P_{\bar{a}}(X_i)] - P_0(out)$
	$P_1(out) = 1 - [P_0(out) + P_a(out) + P_{\bar{a}}(out)]$
NOT	$P_1(out) = P_0(input)$
	$P_a(out) = P_{\bar{a}}(input)$
	$P_{\bar{a}}(out) = P_a(input)$
	$P_0(out) = P_1(input)$

In our proposed approach for error propagation probability computation, we use the signal probabilities of all nodes in the combinational part and the topological structure of the circuit [4]. The signal probability (SP) of a signal line is the probability of logic value 1 (versus 0) on that line [24]. SP estimation techniques have been presented in [16, 21, 26]. In our approach, the structural paths from each error site to all reachable outputs and flip-flops are extracted. Then, these paths are traversed to trace and compute the error propagation probabilities to reachable outputs or flip-flops. For a particular error site (hit by a particle), we classify the circuit nets and gates as follows. An *on-path* signal is a net on a path from the error site to a reachable output. An *on-path gate* is defined as the gate with at least one on-path input. Finally, an *off-path* signal is a net which is not on-path and is an input of an on-path gate. These three are shown in Figure 5. In this figure, gates D, E, G, and H are on-path gates.

In the general case in which reconvergent paths might exist, the propagation probability to the output of the reconvergent gate depends not only on the type of the gate and the signal probabilities of the off-path signals, but also on the polarities of the propagated erroneous values on the on-path signals. To address this issue, we consider error propagation rules for reconvergent gates. These rules are presented in Table 2. In summary, as we traverse the paths, we use signal probability for off-path signals and use these error propagation rules for on-path signals. The example of application of these rules to calculate error propagation probabilities is shown in Figure 5. In this example, considering the output of gate A has an erroneous value, first we compute the propagation probabilities for gates D, E, and G. Then, we compute the propagation probability for gate H, as shown in Figure 5. Now, the propagation probability of an erroneous value to the output of gate H is calculated as $0.042 + 0.392 = 0.434$. More details of this approach can be found in [4].

4.2 Netlist Failure Probability (N_i)

Netlist failure probability, N_i, depends on the error model and the circuit topology. In general, N_i is the product of the activation probability of node i (AP_i) and the propagation probability of that node (PP_i). We use signal probability for AP_i. Computation of PP_i has been presented in Section 4.1.

In the case of open and stuck-at errors, N_i can be computed according to Equation 1. The first part of this equation accounts for the erroneous value being 0 and the second part accounts for the erroneous value being 1. Each part expresses that the erroneous value should be first activated (signal probability, SP_i, is used for the activation probability) and then propagated to the outputs (PP_i). Note that $PP_i(0) = PP_i(1)$.

$$N_i = SP_i \times PP_i(0) + (1 - SP_i) \times PP_i(1) = PP_i \qquad (1)$$

PP_i: *Propagation probability*
SP_i: *Signal probability*

For wired-AND bridging errors (between nets i and j), N_i can be computed according to Equation 2. The first term of the equation expresses the probability of node i being 1 and node j being 0. The second term calculates the probability of node i being 0 and node j being 1.

$$N_i = [SP_i \times (1 - SP_j) \times PP_i(0)] + [(1 - SP_i) \times SP_j \times PP_j(0)] \quad (2)$$

We use the same approach to compute N_i for wired-OR bridging errors between nets i and j, as shown in Equation 3.

$$N_i = [SP_i \times (1 - SP_j) \times PP_j(1)] + [(1 - SP_i) \times SP_j \times PP_i(1)] \quad (3)$$

For a bit-flip in one of LUT cells (cell i), N_i is computed according to Equation 4. To compute the failure probability due to LUT SRAMs, we use the propagation probability of the LUT output. N_i is computed as the product of the propagation probability of the LUT output and the activation probability of the SRAM cell from LUT inputs which is computed during SP estimation.

$$N_i = AP_i \times PP(LUT_{out}) \quad (4)$$

PP_i: *Propagation probability*
AP_i: *Activation probability of cell i*

4.3 Node Error Rate (PR_i)

PP_i and N_i depends only on the gate-level netlist of the mapped design. In contrast, the node error rate depends on the detailed FPGA placement and routing information of the design. For each gate-level circuit node i, PR_i is defined as the permanent-error rate of node i. PR_i is calculated based on the raw error rate of the device, the error model, and the number of SRAM configuration bits used for implementing node i in the FPGA. The error rate of a node is directly proportional to the number of SRAM configuration cells controlling that node. So, nodes with more configuration bits have higher error rates. Accordingly, we define the vector $PR(n)$ (n: total error sites).

For example, consider two nodes n_1 and n_2 which consist of 4 and 2 PIPs, respectively (Figure 6). The permanent-error probability of n_2 is less than n_1 because n_1 has more candidate locations for permanent-errors than n_2. PR_i is computed as shown in Equation 5. In this equation, f is the total number of possible errors which can occur on node i. For example, f is one for LUT SRAMs. f_{MUX} is equal to the number of select bits. For a routing node, f is directly proportional to all ON/OFF switches connected to that node. As an example, the permanent-error rate of node AB, shown in Figure 7, is equal to six times the raw error rate of an SRAM-cell. In our method, we compute PR_i for both short and open errors.

$$PR_i = r \times f_i \quad (5)$$

r: *Raw error rate of an SRAM cell*
f: *The number of all possible errors*

The raw error rate of an SRAM cell (r) depends on the device characteristics and the flux encounters the device. The raw error rate can be expressed in terms of either MTBF (Mean Time Between Failures) or FIT (Failure in Time). FIT is inversely proportional to MTBF and is equal to one failure in a billion hours (10^9). Designers usually work with

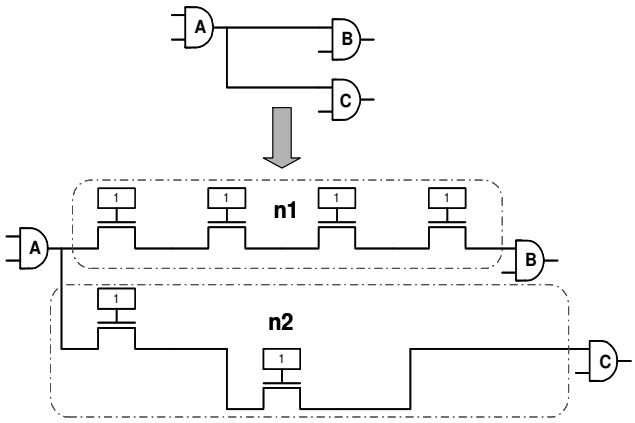

Figure 6: Nodes n_1 and n_2 with different error rates

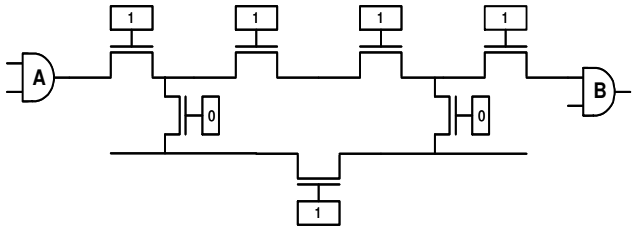

Figure 7: Nets implemented with different numbers of switches

FIT units because they are additive, unlike MTBF. Current predictions show that typical FIT rates for latches and SRAM cells (measured at sea level) vary between 0.001-0.01 FIT/bit [12, 15, 23]. The FIT/bit increases with the elevation. At 10Km, the FIT/bit is approximately 100x higher [31] than the sea-level value.

4.4 Failure Rate Computation

After the computation of PR_i and N_i, the system failure probability due to node i can be compute as follows:

$$System\ failure\ rate\ S_i = PR_i \times N_i \quad (6)$$

Having the system failure rates computed for all nodes (S_i), we compute the system failure rate for the entire circuit in the clock cycle after the particle hit, according to Equation 7.

$$S = 1 - \prod_{i=1}^{n} (1 - S_i) = 1 - \prod_{i=1}^{n} (1 - PR_i \times N_i) \quad (7)$$

Finally, the system failure rate of the entire circuit c clock cycles after the particle hit is calculated as follows:

$$S = 1 - \prod_{i=1}^{n} (1 - PR_i \times (1 - (1 - N_i)^c)) \quad (8)$$

Note that the node error rate factor (PR_i) has the same impact on the system failure rate in Equation 7 and Equation 8. But the way netlist failure probability (N_i) is used to calculate the system failure rate c clock cycles after the particle hit is different from Equation 7.

5. SER ESTIMATION OF USER-BITS

In this section, we present an analytical framework to estimate SER in multiple clock cycles after particle strike. In general, an erroneous value in a flip-flop can be propagated to primary outputs either directly or through other flip-flops. As a result, the system failure probability can increase at each cycle after the error occurrence.

To formalize this analysis, an $n \times n$ *error propagation matrix* M is defined where M_{ij} is the probability of error in flip-flop FF_j given the content of flip-flop FF_i is erroneous (n in the number of flip-flops).

$$Matrix\ M:$$
$$M_{ij} = P(error\ appears\ in\ FF_j | FF_i\ is\ erroneous)$$

$$\mathbf{M} = \begin{pmatrix} P(FF_1|FF_1) & P(FF_2|FF_1) & \dots & P(FF_n|FF_1) \\ P(FF_1|FF_2) & P(FF_2|FF_2) & \dots & P(FF_n|FF_2) \\ \vdots & \vdots & \ddots & \vdots \\ P(FF_1|FF_n) & P(FF_2|FF_n) & \dots & P(FF_n|FF_n) \end{pmatrix}$$

$$System\ failure\ probability\ vector\ S:$$
$$S_i = P(system\ failure | FF_i\ is\ erroneous)$$

Also, the system failure vector S is defined, where S_i is the probability of system failure at the first clock given the content of FF_i is erroneous (i.e., $S_i = P(SF|FF_i)$.

$$\mathbf{S} = \begin{pmatrix} P(SF|FF_1) \\ P(SF|FF_2) \\ \vdots \\ P(SF|FF_n) \end{pmatrix}$$

Using the error propagation matrix, the probability of a system failure at clock c given that FF_i is erroneous is calculated as follows:

$$P(SF\ at\ cycle\ c | FF_i\ erroneous) = i^{th}\ element\ of M^{c-1} S \quad (9)$$

Matrix M and vector S are computed using the propagation probability computation approach presented in Section 4.1. Each column of M (as well as the corresponding entry in S) is computed by a traversal of the circuit from the corresponding flip-flop.

Note that the simulation time of random-simulation method increases exponentially with c, making it intractable for large sequential circuits. However, the presented approach requires only a matrix multiplication to compute the system failure rate in next clock cycles.

6. SOFT ERROR MITIGATION

Unprotected FPGA-based designs have very poor *availability* since once an SEU occurs and then manifests to system output, the steady state availability becomes zero. In this section, a high-reliable framework for soft error tolerance in the configuration bits with very low area overhead is presented. To protect the configuration bits, we reserve some rows of the FPGA to store the checksum and the status for all configuration frames. Specifically, the last two bytes of each frame is used to keep the CRC values of that frame. We also use the *checkpointing* technique for recovery of user bits by keeping the latest correct state of the circuit (user bits including FFs).

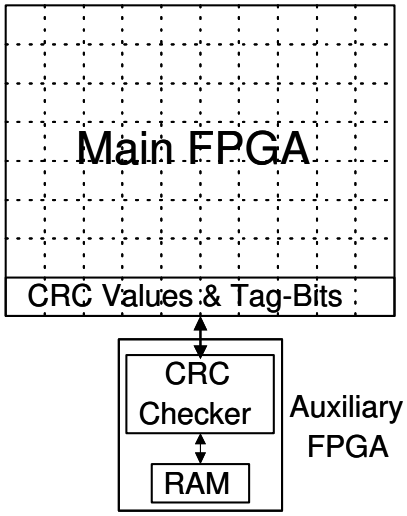

Figure 8: The architecture of the proposed mitigation technique.

We use an *auxiliary* FPGA device to store the correct state of the main system (checkpointing), re-calculate, and compare the CRC checksums for configuration bits. As will be shown in our experiment, this auxiliary circuitry can be mapped into the smallest FPGA device even if the main system is implemented on the largest FPGA device.

To assure the reliability of the auxiliary FPGA, we use non-SRAM based FPGA (such as Actel antifuse family). To further reliability assurance of the CRC checking circuitry, it is implemented using the TMR technique. Since the CRC checking circuit is very small (16 logic cells), the TMR CRC checker can be easily mapped into one of the smallest Actel devices (AX125).

In our presented approach, we do not keep the original copy of the configuration bits since it requires a large storage (e.g. at least 1MB RAM for Xilinx Virtex XCV1000). Only the contents of the FFs are stored in the auxiliary FPGA which are protected by error correction codes (ECC). As shown in Table 1, the number of FFs is much less than the number of configuration bits. The FFs of a large FPGA (e.g., XCV1000) can fit in a 8KB RAM available in the auxiliary FPGA.

The main difficulty in the mitigation techniques is when some configuration bits are used as user-bits. For example, an LUT can be also used as a 16-bit RAM. This problem is common in all mitigation techniques whether the mitigation technique uses the checksum or keeps the original copy of the bitstream to detect errors. In checksum-based mitigation techniques, these bits should be excluded from the checksum computation. In our approach, we put some *tag-bits* in the last bytes of each frame indicating which LUT tables in that frame are used as user-RAMs. Using these tag-bits, those configurations bits which are used as user bits are excluded during checksum recomputation by the auxiliary FPGA.

Since CRC values are kept inside the main FPGA, it can also be connected to the host system as well. Specifically, the main FPGA can be connected to either the host system or the auxiliary FPGA without any changes. This feature is

very useful during the development, debug, and test of the system. Moreover, the way CRC values stored inside the FPGAs is scalable for larger FPGAs.

This mitigation technique adds a constraint on the *placement* algorithm since the last few bytes of all frames should be reserved for the CRC values and tag-bits. The straightforward way is to leave the last rows of the FPGA device unused.

The frames are repeatedly read back from the main FPGA by the auxiliary FPGA and CRC checksums of these frames are re-calculated and compared to the last two bytes of the frame (the original checksum). Using this error correction code (ECC), any detected single bit-flip can be corrected. If any error is detected, the system must be restored to its latest correct state which is obtained from the most recent checkpoint values. This mechanism is known as *rollback recovery*.

Note that if an error occurs in the configuration bits, the error can be propagated to the system states (FFs or system RAMs) in just few clock cycles. Therefore, in the event of configuration bit-flip, the system state is no longer valid and should be restored to the correct state. The architecture of the proposed mitigation technique is shown in Figure 8.

The following steps are performed to recover an error in the configuration bits.

1. The state of the system (user bits) is read back, ECC is computed, and all stored in the auxiliary memory. This step is called *checkpointing*. Using the SelectMap interface for Xilinx FPGAs, checkpointing takes between $1ms$ to $4ms$ to read and save the content of FFs. The checkpointing time (t_c) depends on the device and circuit size. The time interval between two checkpoints (t_{cp}) (*checkpointing period*) can be determined based on the raw error rate of the SRAM cell, the reliability requirements of the system, and the specification of the mapped design. Note that checkpointing does not intervene with the operation of the main system.

2. The configuration frames are read back by the auxiliary FPGA, frame CRC checksums are computed and compared with the original CRC checksums in each frame. This process, which is called *error-checking*, is repeated for all configuration frames and in case of error occurrence, the next step is executed.

3. If any configuration frame is erroneous, it implies that the state of the circuit might be erroneous, as well. In this situation, both configuration data and user data (system state) must be restored to correct values. The functionality of the system needs to be stopped, the corrected configuration frames (including CRC checksums) are re-downloaded into the main FPGA, and the correct system state (stored in the auxiliary FPGA RAM) is written into the FFs and user memory. After these tasks, the operation of the main system can be resumed.

The simple approach for error-checking is to uniformly check all configuration frames in order. Suppose that t_f is the time required to complete the error-checking step for one frame. If t_{ep} is defined as the error-checking period, then t_{ep} is bigger than or equal to t_f. Also, assume N_f is the number of all *sensitive frames* of the main FPGA. Sensitive frames are those that contain sensitive configuration bits. The term

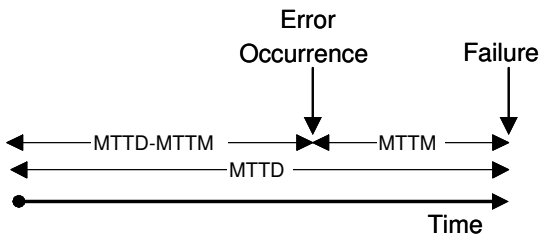

Figure 9: Error manifestation rate

Mean Time To Detect (MTTD) an error is defined as the time interval between the time a particle strike causes a bit-flip in a sensitive bit and the time that the erroneous configuration bit is detected (by the auxiliary FPGA). MTTD increases linearly with the N_f and t_{ep}. Using uniform error-checking approach, $MTTD = 0.5 \times t_{ep} \times N_f$.

MTTM is defined as the mean time to manifest errors from the error site to the circuit outputs, i.e. the time interval between bit-flip and the time the error appears at system outputs. To reduce the overall failure rate of the system, MTTD should be reduced with respect to MTTM. This means that before an erroneous configuration bit manifests to the system outputs, it should be detected and corrected by the auxiliary circuitry. We define the term *Error Manifestation Rate* (EMR) according to Equation 10. If MTTD is less than MTTM, the error is detected and corrected before it is propagated to the outputs. In this case, EMR is zero. If MTTD is bigger than MTTM, the probability that an error is manifested to the outputs is equal to $(MTTD - MTTM)/(MTTD)$. This is shown in Figure 9.

$$EMR = \frac{MTTD - MTTM}{MTTD} \times S, \quad if\ MTTD > MTTM \quad (10)$$
$$EMR = 0, \quad if\ MTTD \leq MTTM$$

Note that S is the failure rate of an unprotected design, which is computed in the previous sections. By decreasing MTTD or increasing MTTM, EMR is reduced. In other words, in a protected design, some failures of unprotected designs are masked by decreasing MTTD or increasing MTTM. Therefore, EMR is the failure rate of the protected design. EMR is equal to S for an unprotected design.

Finally, it is notable to mention that keeping CRC values in the last row of each frame may cause an invalid configuration of the FPGA resources. This may cause conflict between routing signals and also lead to an abnormal power consumption. One possible solution to avoid any contention is to put some status bits to relocate CRC bits. In Virtex family, the last row of each frame contains at least 20 bits. Hence, some of these bits can be used to determine where the CRC bits are located in the last row of each frame.

7. EXPERIMENTAL RESULTS

7.1 SER Estimation

In the experiments, we use Xilinx FPGA devices [29]. To extract detailed placement and routing information of a design mapped into an FPGA, the Xilinx design language (XDL) is used. Figure 10 shows the overall flow of our SER estimation method.

Table 3: Number of sensitive SRAM bits for each part

Circuit	Routing	LUT	Control/ Clocking	Total Conf Bits	No. of FFs
s298	982	410	133	1525	14
s344	1035	392	168	1595	15
s349	1450	520	187	2157	15
s382	1849	712	207	2768	21
s386	1689	660	160	2509	6
s400	1949	700	218	2867	21
s444	1690	692	208	2590	21
s510	3122	1244	299	4665	6
s526	2401	856	227	3484	21
s641	3038	1056	375	4469	19
s713	2793	988	355	4136	19
s953	7906	2644	597	11147	29
s1196	7980	2976	613	11569	18
s1238	8608	3224	652	12484	18
s1488	9660	3688	702	14050	6
s1494	9670	3628	695	13993	6
ave	4114	1524	362	6001	16

The error list considered in the experiments includes mux-open, PIP open, PIP short, buffer-on, buffer-off, LUT bit-flip, and control/clocking bit-flip, and bridging errors. A software tool has been developed to extract the netlist information from the XDL file including the list of used resources, sensitive bits, and the error list. The failure rate of all circuit nodes are computed based on the above information. The experiments have been executed on a Sun Blade 1500 © workstation equipped with 1GB main memory and running Solaris 9 © operating system.

Table 3 shows the number of sensitive bits of the Virtex XCV300 device for ISCAS89 benchmark circuits. The sensitive bits are classified according to the error models described in Section 3. As shown in this table, the configuration routing bits constitute more than half of the total sensitive configuration bits. As shown in this table, the number of FFs, on average, is less than 0.5% of the number of configuration bits.

Table 4 shows the mean time to manifest errors from the error site to the system outputs. MTTM is classified for different types of configuration bits (routing, LUT, and control/clocking) as well as user bits (FFs). The MTTM results have been measured for the XCV300 (Virtex) and XC2s200 (Spartan) devices. As shown in this table, the average MTTM of routing, LUT, and control/clocking resources for the Virtex device are 3.64, 25.63, and 1.63 cycles, respectively. Also, the average MTTM of routing, LUT, and control/clocking resources for the Spartan device are 3.66, 26.22, and 1.65 cycles, respectively. This shows that control/clocking bits are the most sensitive ones. For both families, the MTTM of the routing and control/clocking resources is much less than the MTTM of the LUT resources. In this table, the MTTM of flip-flops (user bits) is also shown.

As the results show, the average manifestation time of an erroneous FF to the primary outputs is about 10 cycles. These results also show that LUTs are the least sensitive bits to SEUs, although they are easiest to be protected against

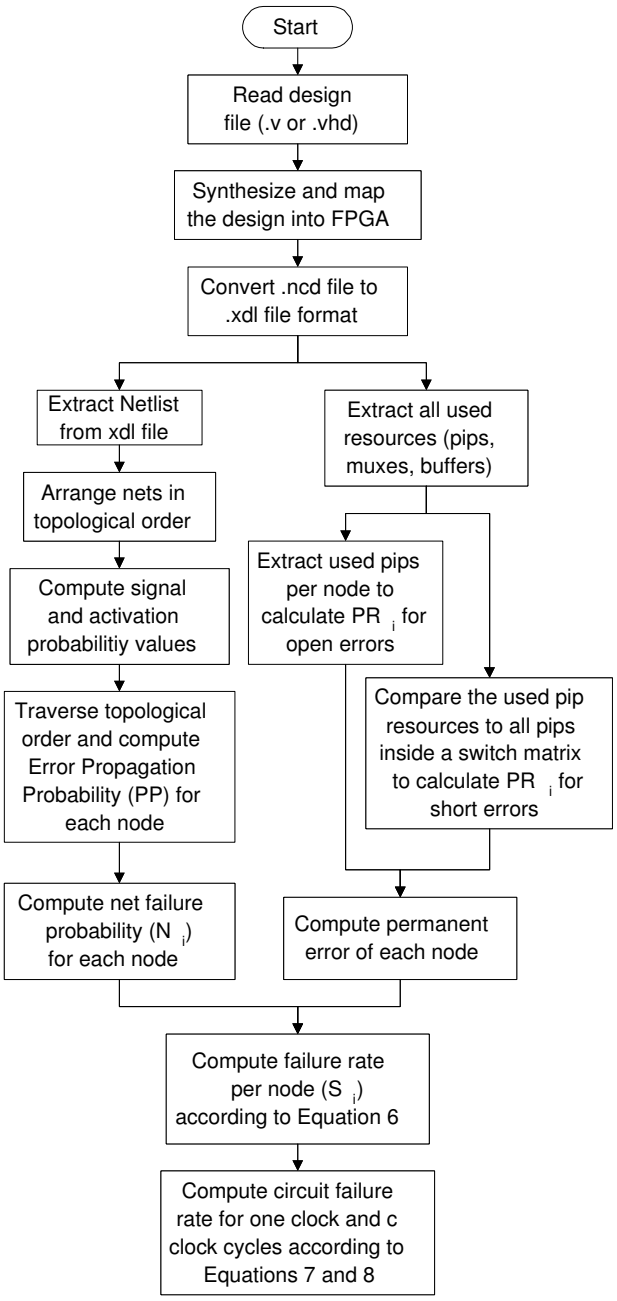

Figure 10: SER Estimation Flowchart

Table 4: Mean Time To Manifest (MTTM) errors to outputs (Results in terms of cycles)

Circuit	Routing		LUT		Cnt/clocking		FFs
	Virtex	Spartan	Virtex	Spartan	Virtex	Spartan	
s298	2.73	2.76	20.22	21.41	1.23	1.31	4.95
s344	2.59	2.61	17.48	21.15	1.37	1.39	5.62
s349	2.91	3.06	20.44	24.66	1.39	1.47	5.62
s382	3.30	3.24	22.11	22.59	1.40	1.38	6.98
s386	3.88	4.11	30.61	28.49	1.79	1.82	15.76
s400	3.13	3.35	20.80	22.54	1.40	1.39	7.23
s444	3.00	3.18	21.92	22.25	1.39	1.39	7.34
s510	4.87	4.27	34.77	35.52	2.14	2.15	3.80
s526	4.03	3.90	27.68	27.70	1.48	1.49	8.14
s641	2.42	2.40	16.96	16.34	1.41	1.40	12.10
s713	2.41	2.47	16.82	16.81	1.37	1.37	13.06
s953	3.23	3.29	21.22	21.27	1.49	1.50	24.16
s1196	5.17	5.16	36.59	36.65	2.16	2.16	23.81
s1238	5.56	5.77	41.23	41.23	2.33	2.33	24.06
s1488	4.47	4.48	29.92	30.22	1.90	1.92	3.81
s1494	4.60	4.58	31.39	30.82	1.98	1.99	3.82
average	3.64	3.66	25.63	26.22	1.63	1.65	10.63

soft errors (implementation of parity schemes in LUTs is very straightforward).

If we consider *normalized manifestation rate* (NMR) for each category (routing, LUT, control/clock, and FF) according to Equation 11, routing bits and LUT bits are the most vulnerable and the least vulnerable ones, respectively.

$$NMR = \frac{Number\ of\ sensitive\ bits}{MTTM} \tag{11}$$

The detailed execution time of our SER estimation method is reported in Table 5. The total time of this SER estimation method includes **a)** the time needed to extract the netlist and also to extract SRAM cells information from the XDL file, **b)** SP computation time, and **c)** error propagation probability computation time including the time needed to compute PR_i and N_i for all nodes. As shown in this table, the SER of an ISCAS'89 circuit is computed, on average, in 108 seconds. As the results show, SP computation is the most time-consuming part of our estimation method. Our simulation time is much less than the results reported in [25]. The reported simulation time in [25] varies from 453 seconds to 11,831 seconds. They have measured the simulation time for only 100 injected faults for some small circuits of ITC'99 benchmarks. The accuracy of our SER method versus the random-simulation method has also been reported in Table 5. The accuracy, on average, is about 95%.

The system failure rates of these benchmark circuits for both Virtex and Spartan devices are reported in Table 6. In these experiments, the raw error rate of an SRAM cell is assumed to be 0.01 (FIT/bit). Failure is observed for 1 and 50 clock cycles after an SEU flips the content of an SRAM cell. Since SEUs cause permanent errors in the configuration bits, the failure rate in 50 cycles after an SEU occurrence is much bigger than the failure rate in the first clock cycle.

These results, Table 4 and Table 6, show that the choice of the FPGA architecture has an indistinguishable effect on the error rate of a particular mapped design. Nevertheless, it has to be mentioned that Virtex and Spartan architectures are similar. The error rate is mainly a function of the number

Table 5: System Failure Estimation Time.
SP Time: Signal Probability computation time (second)
SFR Time: System Failure Rate computation time (second)
Number of Clock cycles: 50

Circuit	Execution Time (sec)				Accuracy (%)
	Extract	SP	SFR	Total	
s298	0.79	1.18	0.09	2.06	95
s344	0.95	34.52	0.12	35.59	96
s349	1.01	41.59	0.15	42.75	96
s382	1.05	47.64	0.14	48.83	95
s386	1.40	0.10	0.19	1.69	97
s400	1.30	46.89	0.19	48.38	95
s444	1.05	45.83	0.14	47.02	95
s510	2.64	84.25	0.38	87.27	96
s526	0.84	54.57	0.13	55.54	96
s641	3.10	88.19	0.55	91.84	91
s713	3.80	77.11	0.58	81.49	90
s953	3.33	179.58	0.73	183.64	95
s1196	4.94	212.69	1.24	218.87	96
s1238	4.83	231.97	1.39	238.19	96
s1488	6.86	267.29	1.20	275.35	95
s1494	7.16	266.29	1.26	274.71	95
ave	2.82	104.98	0.53	108.33	95

Table 6: System Failure Rate for Virtex and Spartan family.
Number of Clock cycles: 50
SFR: System Failure Rate
FIT of an SRAM cell: 0.01

	SFR of Virtex		SFR of Spartan	
Circuit	clk=1	clk=50	clk=1	clk=50
s298	7.46	13.82	7.14	13.86
s344	7.31	14.28	8.09	15.57
s349	9.91	19.45	9.67	19.73
s382	12.10	24.72	12.16	23.70
s386	9.46	18.79	8.72	18.30
s400	13.23	25.31	12.52	24.85
s444	11.60	22.88	12.16	23.93
s510	16.99	36.70	17.19	35.91
s526	15.38	28.23	12.70	25.21
s641	21.11	39.32	21.87	39.77
s713	19.41	36.35	19.32	36.44
s953	49.54	88.41	49.76	88.47
s1196	34.89	80.98	38.16	84.61
s1238	39.27	83.20	38.62	84.11
s1488	53.80	114.36	56.06	117.07
s1494	53.14	113.84	54.34	116.61
average	23.41	47.54	23.66	47.12

of sensitive bits and the structure of the design. Placement and routing algorithms have a major impact on the number of sensitive bits. For example, signals routed with fewer switches have less node error rate.

7.2 Mitigation Technique: A Case Study

In our experiments, we use AX125 (Actel) as the auxiliary FPGA and XCV300(Virtex) as the main FPGA. We use 16-bit CRC polynomial ($CRC-16 = X^{16} + X^{15} + X^2 + 1$). This can be implemented by a 16-bit shift register and 3 XOR gate, which can be mapped into 16 logic cells. The TMR version of this circuit imposes about 200% overhead which can be easily mapped into the AX125 device. The AX125 contains 18,432 RAM bits. This memory can keep a copy of the FFs of XCV300 device. For larger Virtex devices (e.g. XCV1000), we can use AX500 which contains 73,728 RAM bits.

An unprotected system will no longer be available after the first failure in the system. Therefore, the steady-state availability of an unprotected system is zero. Table 7 illustrates the availability of a protected design which occupies all columns of XCV300 device (the worst case scenario for our method). As shown in this table, the availability of the protected design is more than 0.9962 if the raw error rate equals to $1.0e-9$ (bit/hour). Note that the typical raw error rate at sea level is between 1.0e-12 and 1.0e-11 bit/hour [1].

8. CONCLUSIONS

Designs mapped into FPGAs are more susceptible to soft errors than ASIC implementations. Analysis of the effect of soft errors in different resources of an FPGA is a key factor in development of low cost, high performance, and high reliable solutions. In this paper, a very fast and accurate SER estimation technique for FPGA-based designs has been presented.

The experimental results show that the presented method is orders of magnitude faster than conventional fault injection methods while more than 95% accurate. The results also show that the error manifestation time for routing and clock/control resources is 10 times less than that for LUT bits.

We have also presented a low-cost and high reliable soft error mitigation technique based on checkpointing. An auxiliary FPGA (a small device) is utilized to store checkpoints, compare the checksums, and reconfigure the main FPGA. Experimental results show that the availability of a protected design based on this technique is increased to more than 99.6%. Since no host system or pre-store configuration is required, this solution can be used for embedded applications.

[1]These raw error rates almost equal to 0.001-0.01 FIT/bit.

Table 7: Availability of a protected design with different error rates.
(A design occupies all CLBs of XCV300 device)

Error Rate (errors/hour)	Steady-State Availability
1.0e-10	0.9996
2.0e-10	0.9992
3.0e-10	0.9989
4.0e-10	0.9985
5.0e-10	0.9981
6.0e-10	0.9977
7.0e-10	0.9973
8.0e-10	0.9969
9.0e-10	0.9966
1.0e-9	0.9962

9. REFERENCES

[1] L. Antoni, R. Leveugle, and B. Feher, "Using Run-Time Reconfiguration for Fault Injection in Hardware Prototypes," Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT), pp. 405-413, 2000.

[2] G. Asadi, G. Miremadi, H.R. Zarandi, and A. Ejlali, "Fault Injection into SRAM-Based FPGAs for the Analysis of SEU Effects," Proceedings of the IEEE International Conference on Field-Programmable Technology (FPT), pp. 428-430, Tokyo, Japan, Dec 2003.

[3] G. Asadi and M. B. Tahoori, "An Analytical Approach for Soft Error Rate Estimation of SRAM-Based FPGAs," Proceedings of the Military and Aerospace Applications of Programmable Logic Devices (MAPLD), Washington D.C., September 2004.

[4] G. Asadi and M. B. Tahoori, "An Accurate SER Estimation Method Based on Propagation Probability," In the IEEE/ACM Intl. Conference on Design, Automation and Test in Europe (DATE), Munich, Germany, March 2005.

[5] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.C. Fabre, J.-C. Laprie, E. Martins, and D. Powell, "Fault Injection

for Dependability Validation: A Methodology and Some Applications," IEEE Transactions on Software Engineering, Vol. 16, No. 2, February 1990.

[6] C. Carmichael, M. Caffrey, and A. Salazar, "Correcting Single-Event Upsets Through Virtex Partial Configuration," Xilinx Application Notes, XAPP216 (v1.0), June 2000.

[7] C. Carmichael, E. Fuller, J. Fabula, and F. Lima, "Proton Testing of SEU Mitigation Methods for the Virtex FPGA," Proceedings of the Military and Aerospace Applications of Programmable Logic Devices (MAPLD), Washington D.C., September 2001.

[8] C. Carmichael, E. Fuller, P. Blain, and M. Caffrey, "SEU Mitigation Techniques for Virtex FPGAs in Space Applications," Proceedings of the Military and Aerospace Applications of Programmable Logic Devices (MAPLD), Washington D.C., 1999.

[9] E. Fuller, M. Caffrey, A. Salazar, C. Carmichael, and J. Fabula, " Radiation Characterization and SEU Mitigation of the Virtex FPGA for Space-Based Reconfigurable Computing," Proc. of the IEEE Nuclear and Space Radiation Effects Conference (NSREC), Reno, Nevada, July 2000.

[10] M. Gokhale, P. Graham, E. Johnson, N. Rollins, and M. Wirthlin, "Dynamic Reconfiguration for Management of Radiation-Induced Faults in FPGAs," Proceedings of the 18^{th} International Parallel and Distributed Processing Symposium (IPDPS'04), pp. 145-150, Santa Fe, New Mexico, April 2004.

[11] P. Graham, M. Caffrey, J. Zimmerman, D. E. Johnson, P. Sundararajan, and C. Patterson, "Consequences and Categories of SRAM FPGA Configuration SEUs," Proceedings of the Military and Aerospace Applications of Programmable Logic Devices (MAPLD), Washington DC, September 2003.

[12] S. Hareland, J. Maiz, M. Alavi, K. Mistry, S. Walstra, and C. Dai, "Impact of CMOS Scaling and SOI on soft error rates of logic processes," Symposium on VLSI Technology, Digest of Technical Papers, PP. 73-74, June 2001.

[13] R. K. Iyer and D. Tang, "Experimental Analysis of Computer System Dependability," in Chapter 5 of Fault-Tolerant Computer System Design, D. K. Pradhan (ed.), Prentice-Hall, 1996.

[14] B. W. Johnson, "Design & analysis of fault tolerant digital systems," Addison-Wesley Longman Publishing, ISBN:0-201-07570-9, Boston, MA, 1988.

[15] T. Karnik, B. Bloechel, K. Soumyanath, V. De, and S. Borkar, "Scaling Trends of Cosmic Rays Induced Soft Errors in Static Latches Beyond 0.18μ," Symposium on VLSI Circuits, Digest of Technical Papers, pp. 61-62, June 2001.

[16] R. Kodavarti and D. E. Ross, "Signal Probability Calculation Using Partial Functional Manipulation," Proc. of the VLSI Test Sym., [Digest of Papers], April 1993.

[17] R. Leveugle, "Fault Injection in VHDL Descriptions and Emulation," Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 414-419, Yamanashi, JAPAN, October 2000.

[18] P. Liden, P. Dahlgren, R. Johansson, and J. Karlsson, "On Latching Probability of Particle Induced Transients in Combinational Networks," Proceedings of the 24^{th} Symposium on Fault-Tolerant Computing (FTCS-24), pp. 340-349, 1994.

[19] F. Lima, C. Carmichael, J. Fabula, R. Padovani, and R. Reis, "A Fault Injection Analysis of Virtex FPGA TMR Design Methodology," Proceedings of the Radiation Effects on Components and Systems Conference (RADECS2001), Grenoble, FRANCE, 2001.

[20] F. Lima, L. Carro, and R. Reis, "Designing Fault Tolerant Systems into SRAM-Based FPGAs," Proceedings of the IEEE/ACM Design Automation Conference (DAC), pp. 650-655, June 2003.

[21] A. Majumdar and S. B. K. Vrudhula, "Analysis of Signal Probability in Logic Circuits Using Stochastic Models," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, pp. 365-379, Vol. 1, No. 3, Sep. 1993.

[22] K. Mohanram and N. A. Touba, "Cost-Effective Approach for Reducing Soft Error Failure Rate in Logic Circuits," Proceedings of the International Test Conference (ITC), pp. 893-901, Charlotte, NC, October 2003.

[23] E. Normand, "Single Event Upset at Ground Level," IEEE Transactions on Nuclear Science, vol. 43, No. 6, December 1996.

[24] K. P. Parker and E. J. McCluskey, "Probabilistic Treatment of General Combinational Networks," IEEE Trans. on Computers, Vol. c-24, No.6, pp. 668-670, June 1975.

[25] M. Rebaudengo, M. Sonza Reorda, and M. Violante, "Simulation-Based Analysis of SEU Effects on SRAM-Based FPGAs," Proceeding of the 12^{th} International Conference on Field-Programmable Logic and Applications (FPL2002), Montpellier, France, September 2002.

[26] J. Savir, G. S. Ditlow, and P. H. Bardell, "Random Pattern Testability," IEEE Trans. on Computers, Vol. c-33, No. 1, pp. 79-90, January 1984.

[27] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, L. Alvisi, "Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic," Proc. of the International Conference on Dependable Systems and Networks (DSN'02), Washington D.C., June 2002.

[28] M. B. Tahoori, S. Mitra, S. Toutounchi, and E. J. McCluskey, "Fault Grading FPGA Interconnect Test Configurations," Proceedings of the International Test Conference (ITC), pp. 608-617, Baltimore, MD, October 2002.

[29] Xilinx, "Virtex 2.5V Field Programmable Gate Arrays," Data Sheet DS003-1, Xilinx, San Jose, CA, April 2001.

[30] Xilinx, "Virtex-II 1.5V Field-Programmable Gate Arrays," Data Sheet DS031-1 (v1.7), Xilinx, San Jose, October 2001

[31] J. F. Ziegler, "Terrestrial Cosmic Rays," IBM Journal of Research and Development, pp. 19-39, Vol. 40, No. 1, Janaury 1996.