Open access • Proceedings Article • DOI:10.1109/VTS.2005.75

Soft error mitigation for SRAM-based FPGAs — Source link

Ghazanfar Asadi, Mehdi B. Tahoori

Institutions: Northeastern University

Published on: 01 May 2005 - VLSI Test Symposium

Topics: FPGA prototype, Soft error, Reconfigurable computing and Integrated circuit design

Soft Error Mitigation for SRAM-Based FPGAs

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

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

Abstract

FPGA-based designs are more susceptible to single-event upsets (SEUs) compared to ASIC designs, since SEUs in configuration bits of FPGAs result in permanent errors in the mapped design. Moreover, the number of sensitive configuration bits is two orders of magnitude more than user bits in typical FPGA-based circuits. In this paper, we present a high-reliable low-cost mitigation technique which can significantly improve the availability of designs mapped into FPGAs. Experimental results show that, using this technique, the availability of an FPGA mapped design can be increases to more than 99%.

1 Introduction

Soft errors, also called transient errors, are intermittent malfunctions of the hardware that are not reproducible [17]. These errors, which can occur more often than hard (permanent) errors [13], arise from *Single Event Upsets* (SEU). These SEUs are caused by energetic neutrons and alpha particles hitting the surface of silicon devices.

Device scaling significantly affects the susceptibility of integrated circuits to soft errors [19]. As the feature size shrinks, the amount of charge per device decreases enabling a particle strike to be much more likely to cause an error. As a result, particles of lower energy, which are far more plentiful, can generate sufficient charge to cause a soft error. Hence, in the absence of error correction schemes, the error rate of vulnerable parts will grow in direct proportion to the number of bits on the chip. Thus, while Moore's Law gives an exponential increase in the transistor count, this growth comes at the cost of exponential increases in error rates for unprotected chips [16].

Field Programmable Gate Arrays (FPGAs) are widely utilized in many application domains such as industrial, spacecraft, storage systems, cryptography, and embedded applications due to their high performance, no Non-Refundable-Engineering cost and fast Time-To-Market. However, the importance of dependability issues limit their widespread use in mission- critical applications [14]. FPGAs are vulnerable to *Single Event Upsets* (SEUs) [8]. 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 [18]. Moreover, radiation-hardened devices are few generations behind state-of-the-art COTS devices.

Memory elements in an FPGA device can be categorized into *configuration* and *user* bits. Configuration bits are used for specifying the particular circuit mapped into the FPGA, whereas the user bits, such as flip-flops (FFs) or on-chip memory bits, hold the current state of the circuit. After loading a design into an FPGA, the contents of configuration bits are supposed to remain unchanged, while the contents of user bits can be changed at any clock cycle. Majority (more than 99%) of memory bits in an FPGA are configuration bits. Therefore, the probability of soft errors in configuration bits is much more than that in user bits. Moreover, a particle hit on a configuration bit cause a permanent error in the mapped design. Conventional fault-tolerant schemes [12] can protect only user bits but not configuration bits. The only applicable fault-tolerant mechanism to protect configuration bits is *Triple Modular Redundancy* (TMR) scheme in all used logic and routing resources [4] [14]. TMR-based SEU mitigation techniques impose 100%-200% overhead in terms of area and power [4, 6, 14, 15]. This extra overhead also affects the performance of the design mapped into the FPGA.

In this paper, we present a low-cost mitigation technique which can significantly reduce the failure rates of FPGA-based designs. We reserve some rows of the FPGA to store checksum and use a small *auxiliary* FPGA for error checking and recovery. We also present an analytical reliability analysis of this technique. Experimental results show that the availability of a system protected with this scheme can be improved to more than 99.6%.

The rest of the paper organized as follows. Section 2 presents the previous SEU mitigation techniques as well as error models of SRAM-based FPGAs. Section 3 presents the proposed mitigation technique. Experimental results are given in Section 4. Finally, Section 5 concludes the paper.

2 Previous Work and Background

2.1 Mitigation Techniques

A conventional SEU mitigation technique is to use TMR scheme. This approach imposes 2x area and power overhead [4].

While TMR schemes can mask single errors, 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.

In Xilinx Virtex devices, the configuration memory is segmented into *frames* [20]. These FPGAs 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. The FPGA mitigation technique presented in [5] exploits this reconfiguration feature and is based on using a *cyclic redundancy code* (CRC) checker for each FPGA frame. In this method, CRC is periodically generated for each frame (during the readback) and 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 never requires any updates or changes (i.e. fixed bitstream), CRC constants are pre-generated and stored in system ROM for a specific FPGA design. For the applications that can accept updates for the FPGAs bit-stream, CRC constants are 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.

2.2 FPGA Error Models

The effects of SEUs in digital circuits can be classified into *transient* and *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 bits and flip-flops. These errors are called transient because they can be overwritten or corrected using error-detection-and-correction techniques. So, transient errors impact the user-defined logic and flip-flops of the FPGA.

Moreover, SEUs can make permanent errors in a FPGA if they alter the contents of configuration bits. Note that these errors differ from those errors caused by permanent failures in the hardware. In this case, the configuration bits remain erroneous until the new configuration is downloaded into the FPGA. So, these permanent errors are recoverable[1].

Analysis of transient errors has been well described in [1], [7], and [11]. These methods investigate the circuit behavior by injecting faults into the simulated or emulated model of the design. The fault injection in these techniques implies the alteration of memory elements, such as data-path and control-unit registers, as well as input, output or internal signals [18]. Consequently, the effect of SEUs in the presence of the errors can be straightforwardly studied using conventional simulation or emulation tools.

[1]In the rest of this paper, when we refer to permanent errors, we mean recoverable permanent errors.

The study of permanent errors due to configuration alteration requires a 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 or affect the routing signals between different CLBs. Moreover, an SEU may change the functionality of the logic component by affecting the look- up tables (LUT) [14] [18]. Permanent errors are classified as routing error, LUT bit-flip, and control/clocking bit-flip.

3 Soft Error Mitigation

Availability is the probability that the system is operating properly when it is requested for use. Availability is a function of both reliability and maintainability. Unprotected FPGA-based designs have very poor availability since once an SEU occurs and then manifests to system outputs, no recovery (repair) is performed. Hence, the steady state availability becomes zero.

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).

We use an *auxiliary* FPGA in order to 1) store the correct state of the main system for checkpointing, and 2) re-calculate and compare the CRC checksums for configuration bits. As will be shown in our experiment, this auxiliary circuitry can be mapped to 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 FPGAs (such as Actel antifuse FPGAs) since they provide much lower susceptibility to soft errors compared to SRAM-based FPGAs. To further reliability assurance of the CRC checker circuit, it is implemented in the auxiliary FPGA using the TMR technique. Since the CRC checker circuit is very small, the TMR implementation of the CRC checker can easily fit in the auxiliary FPGA.

In our approach, we do not keep the original copy of configuration bits in the auxiliary FPGA since otherwise it requires an excessively large memory core. For example, at least 1MB RAM is required for storing the configuration bits of Xilinx XCV1000 FPGA. Therefore, only the contents of the flip-flops (and user memory bits) are stored in the auxiliary FPGA which are protected by *error correction codes* (ECC). As shown in [3], the number of user bits is less than 0.5% of the number of configuration bits for ISCAS89 benchmark circuits mapped into Xilinx Virtex FPGAs. For example, user bits of a large FPGA (e.g., XCV1000) can fit in a 8KB RAM available in the auxiliary FPGA.

One of the challenges in mitigation techniques is when some configuration bits are used as user-bits. For example, an LUT can be also configured as a 16-bit RAM. This difficulty, known as *coherence* problem, is common in all mitigation techniques whether the mitigation technique uses checksum or the original copy of bitstream for error detection [10]. In checksum-based mitigation techniques, these bits should be excluded during checksum computation. In our solution, we put some *tag-bits* in the last bytes of each frame indicating which LUT tables in that frame are used as user-RAMs. During checksum recomputation by the auxiliary FPGA, those configurations bits which are used as user bits, indicated by tag-bits, are excluded.

In the proposed approach, the frames are repeatedly read back from the main FPGA by the auxiliary FPGA and CRC checksums of the 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 the last checkpoint values. This mechanism is known as *rollback recovery* [9]. The architecture of the proposed mitigation technique is shown in Figure 1.

Since CRC values are kept inside the main FPGA, the main FPGA can also be connected to the host system as well. In other words, 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 design, debug, and test of the system. Moreover, the way CRC values stored inside the FPGAs is scalable for larger FPGAs. In system-on chip (SoC) application in which processor cores are integrated with FPGA cores (e.g. Xilinx Virtex II FPGAs or eFPGAs), the role of the auxiliary FPGA can be performed by the embedded processor.

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 row of the FPGA device unused.

Note that if an error occurs in the configuration bits, the error can be propagated to the system states (FFs or system RAMs) in a few clock cycles. Therefore, in case of configuration bit-flip, the system state is no longer valid and should be restored to the correct state.

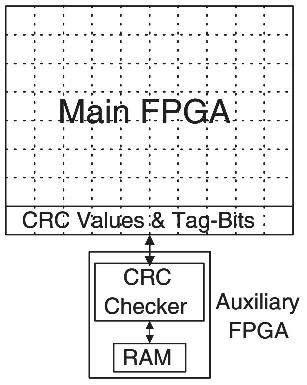

Figure 1. The architecture of the proposed mitigation technique.

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

1. The state of the system is read back, ECC is computed for the system state, and system state along with ECC is stored in the auxiliary memory. This step is called *checkpointing*. The checkpointing time (t_c) depends on the FPGA device and mapped circuit sizes. The time interval between two checkpoints, *checkpointing period*, (t_{cp}) 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 application. Note that checkpointing does not intervene the operation of the main system.

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

3. Occurrence of error in any configuration frame implies that the state of the circuit can also be erroneous. In this situation, both configuration data and user data (system state) must be restored to the 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 the auxiliary FPGA RAM) are written into the FFs and user memory. After these tasks, the operation of the main system can be resumed.

The configuration bits for a particular mapped design can be categorized into *sensitive* and *non-sensitive* bits. Any bit-flips in sensitive bits will eventually affect the user-bits (system state), whereas bit-flips in non-sensitive bits does not affect the functionality of the mapped design. Frames containing sensitive configuration bits are defined as sensitive frames. The simple approach for error-checking is to uniformly check all configuration frames in order. Assume 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 $t_{ep} \geq t_f$. Also, assume N_f is the number of all sensitive frames of the main FPGA. The term *Mean Time To Detect* (MTTD) an error is defined as the interval between the time a particle hit cause a bit-flip in a sensitive bit and the time that an erroneous configuration bit is detected (by the auxiliary FPGA). MTTD increases linearly with N_f and t_{ep}. Using uniform error-checking approach, $MTTD = 0.5 \times t_{ep} \times N_f$.

Mean Time To Manifest errors (MTTM) is defined as 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 in the same proportion 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 *Error Manifestation Rate* (EMR) according to the equation 1.

$$EMR = \begin{cases} \left(\frac{MTTD-MTTM}{MTTD}\right).S, \; MTTD > MTTM \\ 0, \; MTTD \leq MTTM \end{cases} \tag{1}$$

If MTTD is less than MTTM, the error is detected and corrected before it is propagated to the outputs. In this case, EMR equals to zero. If MTTD is bigger than MTTM, the probability that an error is manifested to the outputs equals to $(MTTD - MTTM)/MTTD$. This is shown in Figure 2.

Note that S is the failure rate of an unprotected design, which is computed using SER estimation techniques [2]. By decreasing MTTD or increasing MTTM, EMR can be reduced. Therefore, EMR is the failure rate of the protected design. EMR equals to S for an unprotected design.

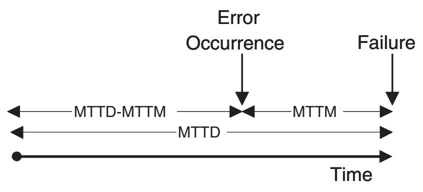

Figure 2. Error manifestation rate

EMR is the failure rate due to SEUs occurring in the configuration bits. In this computation, we have not included the failure rate of user-bits (S_{user}). The total failure rate of a system is computed according to the equation 2.

$$Total\ failure\ rate\ S_{total} = EMR + S_{user} \tag{2}$$

Note that EMR and S_{user} are computed by summing EMR-per-bit and S_{user}-per-bit quantities over all sensitive configuration and user bits, respectively. As shown in [3], the ratio of user bits to configuration bits is less than 0.5% for ISCAS89 benchmark circuits. Therefore, in SRAM-based FPGAs, the failure rate of user-bits can be easily ignored compared to the failure rate of configuration bits, i.e., $S_{total} \approx EMR$.

Finally, it is notable to mention that keeping CRC values in the last row of each frame (random configuration values) may cause an illegal configuration of the FPGA resources. This may cause conflict between routing signals and also lead to an abnormal power consumption. Some of these conflicts can be resolved by using appropriate input values driving the checksum rows (We can use the IOs in the last rows for this purpose). For example, if two signal lines are illegally bridged together, we can set the inputs driving those signals such that the bridged signals always have the same logic value. Another possible solution is to put some status bits to relocate CRC bits, and leave the conflicting configuration bits unused. In Virtex family, the last row of each frame contains at least 20 bits. In this case, some of these bits can be used to determine where the CRC bits are located in the last row of each frame and which bits are unused.

3.1 Dependability Improvement Analysis

In this section, we compute the dependability parameters of a design protected by this mitigation technique and compare it to the unprotected version. These parameters include MTTF, MTTR, reliability, and availability.

The *Mean Time To Failure* (MTTF) of a system is conversely proportional to the failure rate of the system. So, $MTTF = \frac{1}{EMR}$. The *Mean Time To Repair* (MTTR) is the time required to detect an error (MTTD) plus the time required to recover the system to the last correct state before the error occurrence. The recovery time equals to the reconfiguration time of the FPGA plus the time needed to return the system back to the latest state before error occurrence, i.e. the time between last checkpoint and error occurrence. This time, on average, equals to half of the checkpointing period. These are shown in Equation 3. In this equation, t_{ep} is the error checking period, t_{cp} is the checkpointing period, and N_f is the number of sensitive frames.

$$\begin{aligned} MTTR &= MTTD + t_{recovery} \\ MTTR &= MTTD + t_{reconfiguration} + \frac{t_{cp}}{2} \\ MTTD\ (uniform\ error\ checking) &= \frac{t_{ep} \times N_f}{2} \end{aligned} \tag{3}$$

The reliability of an unprotected FPGA-based design depends only on the system failure rate, S (Equation 4).

$$Reliability(t) = e^{-S \times t} \tag{4}$$

The reliability of a design protected by the presented mitigation technique can be computed according to Equation 5. We recall that EMR is the failure rate of a protected design.

$$Reliability = e^{-EMR \times t} \tag{5}$$

Availability, as a function of reliability and repairability, equals to $MTTF/(MTTF + MTTR)$. In a sensitive frame, the sensitive bits and non-sensitive bits can not be easily distinguished, from the error checking perspective. Therefore, even if a non-sensitive bit of a sensitive frame is corrupted, the recovery process is performed. Note that in this case, no failure occurs. However, the system is stalled for a few milliseconds. Therefore, the MTTF in the equation for availability should be computed with respect to all bits inside the sensitive frames (including both sensitive and non-sensitive bits). In this application, MTTF means the mean time to stop the functionality of the system, as shown in Equation 6. Note that N_f is the number of sensitive frames, and r_f is the error rate of a sensitive frame (per hour).

$$MTTF = \frac{1}{N_f \times r_f} \tag{6}$$

Equation 7 is used for the availability of a protected system. t_{cp} is the checkpointing period (hour) and $t_{reconf.}$ is the reconfiguration time (hour).

$$\begin{aligned} Availability &= \frac{MTTF}{MTTF + MTTR} \\ &= \frac{1}{1 + N_f \times r_f \times (MTTD + t_{reconf.} + \frac{t_{cp}}{2})} \end{aligned} \tag{7}$$

An unprotected system is only available before the first failure occurrence since there is no recovery mechanism ($MTTR = \infty$ for an unprotected system). As will be shown in the experiments, our mitigation technique gives very high availability with very low area overhead.

3.2 How to reduce EMR?

A smaller EMR results in higher reliability and better availability. We propose two methods to reduce EMR as follow.

Column-based Placement: The first approach is to modify the placement algorithm based on *column filling*. In this approach, it is tried to fill each column as much as possible before using the next column. The reason is if there is only one used CLB in a column i, all frames of column i should be read back during the error-checking step since all frames associate with this column are considered as sensitive frames. Therefore, to reduce the number of all sensitive frames (N_f), it is better to perform the placement column by column rather than row by row (or any other placement strategy). As an example, consider a 32×48 FPGA device (32 rows and 48 columns). A conventional placement algorithm may place 200 used CLBs in 20 columns. However, using a column-based placement algorithm, these 200 used CLBs can be placed in only 7 columns. In this case, the MTTD is reduced to 1/3.

Selective Readback Frequency: By more frequently reading back the frames with smaller MTTMs, EMR can be reduced. For example, assume that MTTMs of the frames $f1$ and $f2$ are 100 and 10 cycles, respectively. If an error occurs in $f2$, less time is available to detect (and recover) the error compared to an error occurring in frame $f1$. Hence, the error-checking should be done more frequently on $f2$ than $f1$. We can compute the MTTM of different parts of the FPGA using the technique presented in [2] and assign higher priorities to the frames with smaller MTTMs.

Another approach to reduce EMR is to increase MTTM by reducing the clock frequency, or decrease MTTD by reducing error checking intervals. This technique is applicable only to the applications in which it is possible to reduce the clock frequency. For example, consider an application with an average MTTM of 20 cycles. If the clock frequency is reduced from $40Mhz$ to $20Mhz$, the MTTM is doubled and hence the EMR is reduced to one-half. Note that the MTTD is not affected by the reduction of the clock frequency of the main FPGA, since MTTD is determined by read back frequency which is performed by the auxiliary FPGA.

4 Experimental Results

Table 1 shows the number of sensitive bits of ISCAS89 benchmark circuits mapped into Virtex XCV300, as well as the mean time to manifest errors from the error site to the system outputs. The sensitive bits are classified according to the error models described in Sec. 2.2. As shown in this table, the configuration routing bits constitute almost 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. MTTM values are reported for different types of configuration bits (routing, LUT, and control/clocking) and user bits (FFs). As shown in this table, the average MTTM of routing, LUT, and control/clocking resources for the Virtex device are 3.6, 25.6, and 1.6 cycles, respectively. This shows that control/clocking bits are the most sensitive ones. As the results show, the average manifestation time of an erroneous flip-flop 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 (implementation of parity schemes in LUTs is very straightforward). If we consider *normalized manifestation rate* for each category (routing, LUT, control/clock, and FF) which is normalized to the number of sensitive bits as $\#sensitive\ bits/MTTM$, routing bits are the most vulnerable ones. The error rate is mainly a function of the number of sensitive bits and to some extend the structure of the design. Placement and routing algorithms have a major impact on the number of sensitive bits. As shown in Sec. 3, placement policy

Cir.	# of Sensitive Bits				MTTM (cycles)			
	R	L	C	F	R	L	C	F
s298	982	410	1525	14	2.7	20.2	1.2	4.9
s344	1035	392	1595	15	2.5	17.5	1.4	5.6
s349	1450	520	2157	15	2.9	20.4	1.4	5.6
s382	1849	712	2768	21	3.3	22.1	1.4	7.0
s386	1689	660	2509	6	3.9	30.6	1.8	15.8
s400	1949	700	2867	21	3.1	20.8	1.4	7.2
s444	1690	692	2590	21	3.0	21.9	1.4	7.3
s510	3122	1244	4665	6	4.9	34.8	2.1	3.8
s526	2401	856	3484	21	4.0	27.7	1.5	8.1
s641	3038	1056	4469	19	2.4	16.9	1.4	12.1
s713	2793	988	4136	19	2.4	16.8	1.5	13.1
s953	7906	2644	11147	29	3.2	21.2	1.5	24.2
s1196	7980	2976	11569	18	5.2	36.6	2.2	23.8
s1238	8608	3224	12484	18	5.6	41.2	2.3	24.1
s1488	9660	3688	14050	6	4.5	29.9	1.9	3.8
s1494	9670	3628	13993	6	4.6	31.4	2.0	3.8
ave.	4114	1524	6001	16	3.6	25.6	1.6	10.6

Table 1. Number of sensitive bits and MTTMs for ISCAS'89 circuits mapped into Xilinx XCV300
R: Routing, L: LUT, C: Control/Clocking, F: FFs

can also have a considerable impact on the reliability improvement of mitigation techniques.

In our experiments, we use an Actel AX125 (anti-fuse) FPGA for the auxiliary FPGA and Xilinx XCV300 (Virtex) as the main FPGA. We use a 16-bit CRC polynomial ($CRC-16 = X^{16} + X^{15} + X^2 + 1$). This can be implemented by 16-bit shift register and 3 XOR gate, which is mapped into 16 logic cells. To improve the reliability of the auxiliary FPGA, we implement a TMR version of the CRC checker. The TMR implementation imposes about 200% overhead which can be easily mapped into an AX125 device. The AX125 FPGA contains 18,432 RAM bits. This RAM can keep a copy of FFs of XCV300 device. For larger Virtex devices (e.g. XCV1000), we can use AX500 which contains 73,728 RAM bits. Figure 3 shows the reliability of protected and unprotected designs for two circuits (s526, s1494) in ISCAS'89 benchmark. These circuits, s526 and s1494, occupy 34 and 145 slices, respectively. As an example, the reliability values of the unprotected and protected implementations of s526 in 10^6 hours are 0.987 and 0.994, respectively.

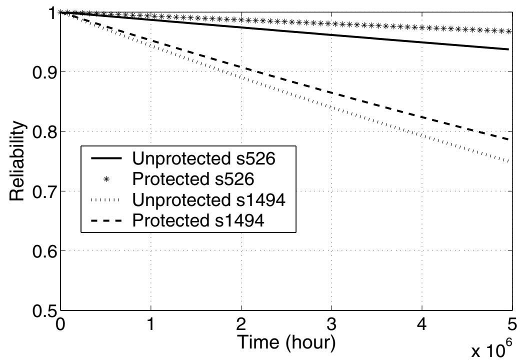

Figure 3. Reliability of unprotected designs vs protected designs

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

Table 2. Availability of a protected design implemented in Xilinx XCV300

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 2 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 ranges between 1.0e-12 and 1.0e-11 (bit/hour). However, this amount at the elevation of 10Km, is 100x higher [21].

5 Conclusions

Designs mapped into FPGAs are more susceptible to soft errors than ASIC implementations. More than 99% of memory bits on an FPGA are used for storing the configuration state of the FPGA. Therefore, FPGAs are much more vulnerable to soft errors in the configuration bits than in user bits.

We have presented a low-cost and high reliable soft error mitigation technique based on checkpointing. A very small auxiliary FPGA is utilized to store checkpoints, compare the checksums, and reconfigure the main FPGA. Experimental as well as analytical results show that the availability of a protected design based on this technique increases to more than 99.6%.

Since no host system or pre-store configuration is required, this solution can be used for embedded applications. In SoC designs in which processor cores are integrated with FPGA cores, the embedded processor can play the role of the auxiliary FPGA.

References

[1] 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 Trans. on Software Engineering, Vol. 16, No. 2, Feb. 1990.

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

[3] G. Asadi and M. B. Tahoori, "Soft Error Rate Estimation and Mitigation for SRAM-Based FPGAs," Proc. of the 13^{th} ACM Intl. Symp. on Field-Programmable Gate Arrays (FPGA), Monterey, CA, Feb. 2005.

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

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

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

[7] A.R. Ejlali, S.G. Miremadi, H.R. Zarandi, G. Asadi, and S. Sarmadi, "A Hybrid Fault Injection Approach Based on Simulation and Emulation Co-operation," Proc. of the Intl. Conf. on Dependable Systems and Networks (DSN'03), pp. 479-488, San Francisco, USA, June 2003.

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

[9] W. J. Huang and E.J. McCluskey, "Transient Errors and Rollback Recovery in LZ Compression," Proc. of the Pacific Rim Intl. Symp. on Dependable Computing, IEEE Press, pp. 128-135, 2000.

[10] W. J. Huang and E.J. McCluskey, "A Memory Coherence Technique for Online Transient Error Recovery of FPGA Configurations," Proc. of the ACM Intl. Symp. on Field Programmable Gate Arrays (FPGA), pp. 183-192, Monterey, CA, Feb. 2001.

[11] 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.

[12] B. W. Johnson, "Design & analysis of fault tolerant digital systems," Addison-Wesley Longman Publishing, ISBN:0201075709, 1988.

[13] J. Karlsson, P. Ledan, P. Dahlgren, and R. Johansson, "Using Heavy-Ion Radiation to Validate Fault Handling Mechanisms," IEEE Micro, 14(1), pp. 8-23, Feb. 1994.

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

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

[16] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessor," Proc. of the 36^{th} Intl. Symp. on Micro-architecture (MICRO-36), pp. 29-40, 2003.

[17] H. T. Nguyen and Y. Yagil, "A Systematic Approach to SER Estimation and Solutions," Proc. of the 41^{st} Intl. Reliability Physical Symp., pp. 60-70, Dallas, Texas, 2003.

[18] M. Rebaudengo, M. Sonza Reorda, and M. Violante, "Simulation-Based Analysis of SEU Effects on SRAM-Based FPGAs," Proc. of the 12^{th} Intl. Conf. on Field-Programmable Logic and Applications (FPL2002), Sep. 2002.

[19] 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 Intl. Conf. on Dependable Systems and Networks (DSN'02), Washington D.C., June 2002.

[20] "The Programmable Logic Data Book", Xilinx Inc, 2004.

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