IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 6, June 2016

Protecting Scram Based FPGA Configuration Frames against Low Cost Multiple Bit Upsets

Ramya R.S[1] , Sakthivel.B[2]

PG Scholar, Department of ECE Coimbatore institute of engineering and technology, Coimbatore[1]

Assistant Professor, Department of ECE Coimbatore institute of engineering and technology, Coimbatore[2]

Abstract: In nanoscale technology nodes Radiation induced multiple bit upsets are the major reliability. Functionality of the mapped design is permanently affected by the occurrence of such errors in the configuration frames of a field programmable gate arrays. Permanent effect of these errors can be avoided by periodic configuration scrubbing combined with a low cost error correction scheme is an efficient approach. In this paper, we present a low-cost error-detection code to detect MBUs in configuration frames as well as a generic scrubbing scheme to reconstruct the erroneous configuration frame based on erasure codes. The proposed scheme does not require any modification to the FPGA architecture. Implementation of the this scheme on a Xilinx Virtex-6 FPGA device shows that it can detect 100% of MBUs in the configuration frames with the recovery time which is comparable to the previous schemes and with only 3.3% resource occupation.

Keywords: MBU, configuration frames, erasure codes, scrubbing

I. INTRODUCTION

FPGA are silicon devices that can be electrically programmed in the field to become almost any kind of digital circuit or system. For high volume productions, FPGAs provide cheaper solution and faster time to market as compared to Application Specific Integrated Circuits ASIC. Also for varying requirements, a portion of FPGA can be partially reconfigured while the rest of an FPGA is still working. Any future updates in the final product can be easily upgraded by simply downloading a new application bit stream. However, flexibility is the main advantage of FPGAs is also the major cause of its drawback. Flexible nature of FPGAs makes them significantly slower, larger and more power consuming than their ASIC counterparts. Because of the routing interconnect of FPGAs these disadvantages arise largely, which comprises of almost 90% of total area of FPGAs. Instead, FPGAs present a compelling alternative for digital system implementation due to their less time to market and low volume cost. The reconfigurability of an FPGA is based on an underlying programming technology, which can cause a change in behavior of a pre-fabricated chip after its fabrication. Normally FPGAs comprise of: Programmable logic blocks, which is used to implement the logic functions. Programmable routing that connects these logic functions. I/O blocks are connected to the logic blocks through routing interconnect and that make off-chip connections.

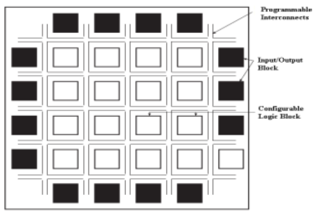

Fig 1 Basic FPGA Architecture

A generalized example of an FPGA is shown in figure 1, where configurable logic blocks are arranged in a two dimensional grid and are interconnected by programmable routing resources. I/O blocks are arranged at the periphery of the grid and they are also connected to the programmable routing interconnect. The “reconfigurable” term in FPGAs indicates their ability to implement a new function on the chip after its fabrication is complete. Different types of programming technologies are used in reconfigurable architectures.

Each of these technologies has different characteristics which in turn have significant effect on the programmable architecture. Some of the well known technologies include static memory, flash and anti-fuse. SRAM-based FPGAs uses static memory cells as the basic cells. Most commercial vendors use static memory (SRAM) based on the programming technology of various devices. These devices use static memory cells which are divided throughout the FPGA to yield configurability. An example of such memory cell is shown in figure 2.

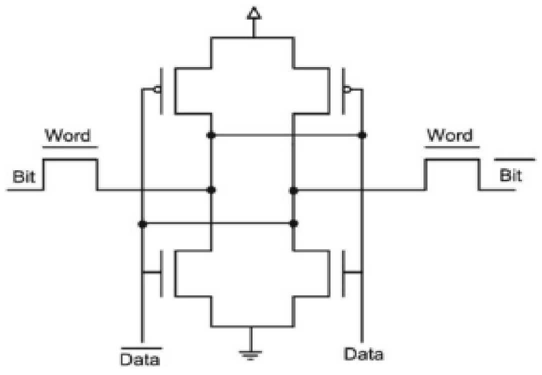

Fig. 2 Static memory cell

In an SRAM-based FPGA, SRAM cells are used for to program the routing interconnect of FPGAs which are generally steered by small multiplexors. To program Configurable Logic Blocks which is also used to implement logic functions SRAM-based programming technology is the dominant approach used in FPGAs because of its re-programmability and the use of standard CMOS process technology and therefore leading to increased integration, higher speed and dynamic power consumption of new process with smaller geometry. There are number of drawbacks with SRAM-based programming technology. For example an SRAM cell requires 6 transistors which make the use of this technology costly in terms of area compared to other programming technologies. Further, SRAM cells need external devices to permanently store the configuration data. These external devices add cost and area overhead of SRAM-based FPGAs.

MBU PATTERNS

In order to fairly quantify the MBU correction capability of the proposed scheme, need to have detailed information about the possible MBU patterns and its occurrence probabilities. In this regard, a 3-D-TCADbased neutron particle strike simulation is conducted by employing a commercial soft error assessment tool [1]. The SPICE net list and the memory layout as well as the radiation environment information are provided as inputs to the tool to compute the distribution of generated current pulses for each cell according to a nuclear database. Afterward, the SEU and MBU rates are extracted by injecting the obtained current pulses in the SPICE netlist. Using this commercial tool, we have acquired the occurrence probabilities of neutron-induced MBU patterns in the terrestrial environment on an SRAM memory designed for a 45-nm technology. In this experiment, the neutron energy distribution is described according to the JEDEC89a standard [2]. Furthermore, the secondary particles reaction that occur when neutrons interact with the atoms in the CMOS structure is modeled according to a nuclear database.

FPGA CONFIGURATION FRAMES

The configuration memory of FPGAs is organized into configuration frames that are the smallest addressable units and constitute the majority of SRAM cells in FPGAs. The number and the size of the configuration frames vary from one device to another. For example, in Xilinx Virtex-6 XLV240T device, which is employed as a case study in this paper, there are 28 464 configuration frames, each comprising of 81 32-bit words (total of 72 049 k bit), whereas there are only 461 of 36 k bit BRAMs.

Therefore, for this particular device, 81.28% of the total SRAM cells belong to the configuration frames. For each mapped design, the FPGA device utilizes only a subset of all the available configuration frames, which are known as sensitive frames. Any errors occurring in these sensitive frames of the device might lead to a system malfunction.

LOW-COST MBU DETECTION

The main idea of this paper is to exploit erasure codes to recover the contents of the erroneous configuration frame. Nevertheless, since erasure codes cannot detect errors, an effective detection technique is required as well. Therefore, each configuration frame has to be equipped with a low-cost error detection code. A scrubber unit periodically investigates the configuration frames for possible errors. Once an error is detected, by assuming that the erroneous frame is erased, its contents are recovered using an erasure code. Considering the fact that the entire configuration frame could be recovered using an erasure code, the identification of the exact location of erroneous bits is not of our interest, rather a low-cost error detection technique with a very high detection coverage is required. In this regard, we present a very efficient error detection coding technique called IND parity for MBU detection in the configuration frames. The error detection capability of this technique in the configuration frame for two particular cases where N = 2 and N = 3 is investigated in detail. In the proposed error detection technique, we exploit the fact that the sizes of large MBU patterns are typically much smaller than the size of a configuration frame. Since an MBU incident affects several bits in a localized manner, the bits that are located far enough cannot be simultaneously affected with one MBU incident. Therefore, having separate parities for such bits in configuration frames neither increases the error detection capability nor improves the performance, rather only imposes unnecessary area overhead.

IND PARITY

In order to increase the cost-efficiency of error detection for the configuration frames, we introduce the idea of IND parity. The main idea is to use the same parity bit for the bits that are separated by a constant distance to minimize the area overhead (i.e., interleaved parity). In addition, the parity bits are distributed at several dimensions to increase the detection coverage with respect to probable MBU patterns as there are some MBU patterns that cannot be detected using one or two dimensions, no matter how many parity bits are employed.

The number of parity bits in each dimension theoretically has to be at most equal to the largest MBU spread on that dimension. However, in practice, large MBU patterns are typically detected using the parity bits on the other dimensions. Consequently, the number of parity bits required by this technique is always smaller than this theoretical limit.

I2D PARITY

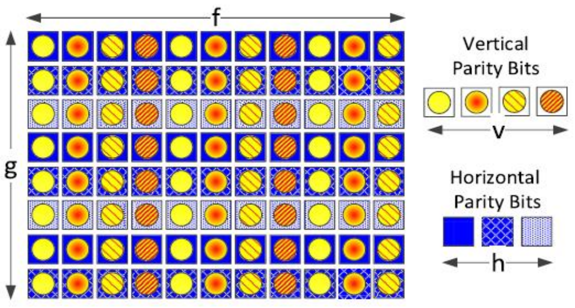

Fig. 3 Example of I2D with vertical and horizontal distance of 4 and 3, respectively.

Figure 3 shows an example of I2D coding with the horizontal and the vertical interleaving distance of 3 and 4, respectively. In the complete (traditional) 2-D parity, a parity bit is associated for each row (column) which is constructed by XOR ing all the bits in that particular row (column). In the I2D parity technique, each horizontal (vertical) parity bit is the XOR of the bits in multiple rows (columns).

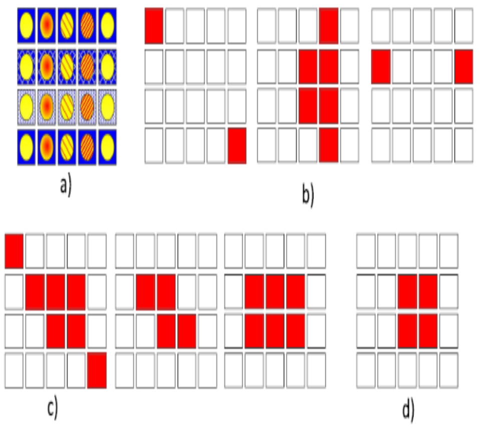

Fig. 4 Examples of MBU patterns for comparison of detection capability of 2D and I2D (vertical and horizontal distance of 4 and 3. (a) I2D: vertical 4 and horizontal 3.(b)Not detected by I2D;detectable by 2D 3.(c) Detected by both(d) Not detected by both.

I3D PARITY

In general, there might be some MBU patterns, which affect an even number of cells in each row and column. This kind of patterns cannot be detected by the I2D parity technique since parity bit can only detect an odd number of erroneous bits. Examples of MBU patterns, in this technology, with significant occurrence probabilities that cannot be detected by either I2D or traditional 2-D parity technique. In order to detect such MBUs, a more powerful MBU detection technique is required. In this regard, we propose I3D parity technique that has an additional set of parity bits for diagonals.

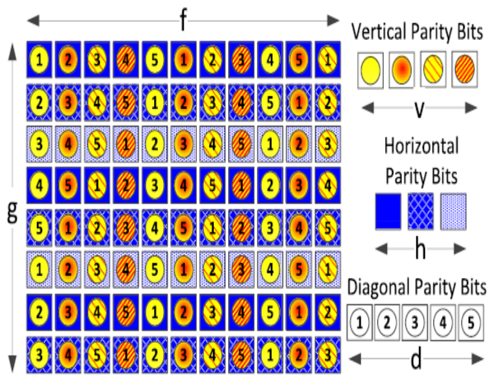

Fig.5 Example of I3D with vertical, horizontal, and diagonal distance of 4, 3, and 5, respectively.

Example of I3D with vertical, horizontal, and diagonal distance of 4, 3, and 5, respectively. Computation of the horizontal and the vertical parity bits in I3D parity follows the same rules explained for I2D parity. Similar to the interleaving technique employed in I2D parity to reduce the number of horizontal and vertical parity bits, in I3D parity, several interleaving groups are formed for the Diagonals as well.

COMPARISON OF I2D AND I3D

The number of parity bits required by I2D to reach its maximum detection coverage is equal to that of I3D. However, the error detection coverage of I2D is less than that of I3D. The maximum error detection coverage's for various number of parity bits employed in the I2D and I3D parity techniques are shown in figure 5. This is obtained by comparing the detection coverage of all possible cases that result in the desired number of parity bits. The I3D parity always provides higher detection coverage than I2D parity when the number of parity bits is more than two. Although the I3D parity technique provides more detection coverage with the same number of bits, it occupies more logical resources on the FPGA. This is because more parity bits are required to be stored for the third dimension and the complexity of the controller unit increases for the generation of such additional parity bits.

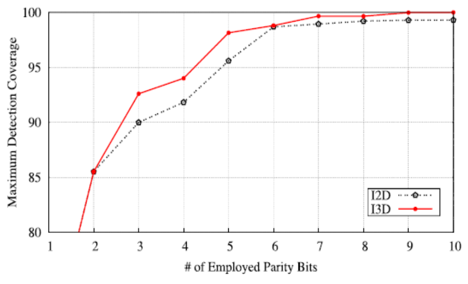

Fig.5 Maximum detection coverage obtained by I2D and I3D parity for different number of parity bits.

ERASURE CODE

Our proposed error correction scheme is based on the concept of erasure codes [3] [4]. An optimal erasure code is a data-recovery technique, were it transforms m blocks into m + n blocks such that the original m blocks can be reconstructed from any arbitrary set of m blocks among m+n coded blocks are shown in figure 6.

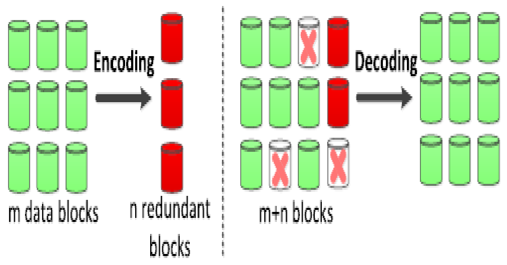

Fig. 6 Encoding and Decoding Of Erasure Codes

Data-recovery technique is widely implemented in storage devices such as hard disks and CDs,[5] error correction in cache array [6] [7] multimedia multicasting, and signal transfer protocols[8]. A variety of erasure codes with recovery coverage, area overhead, and complexity in encoding/decoding are proposed. The area overhead of an erasure code is expressed as the ratio of redundant blocks to the data blocks (i.e., n/m). The recovery coverage of an erasure code is defined as the maximum number of erasures that could be tolerated. For the optimal erasure codes, the recovery coverage is equal to the number of redundant blocks (i.e., n). Erasure codes are not proposed to detect or correct errors rather to retrieve the original blocks when a subset of blocks is not available (i.e., erased). However, once an error is detected in some blocks, by assuming that those blocks are not available, the original blocks could be recovered by means of an erasure code.

RECOVERY BASED ON ERASURE CODES

For error detection, we propose to implement a scrubber unit that periodically checks the parity bits of the configuration frames for possible errors. Upon a detection of an error, by assuming that the erroneous frame is erased, its contents are recovered using an erasure code. There are plenty of erasure codes with different characteristics in the literature. An effective erasure code should be selected in the context of the error recovery for the configuration frames. Since soft error occurrence rate is relatively small and scrubbing is continuously performed to detect and recover possible errors, it is unlikely to have multiple erroneous configuration frames in each scrubbing iteration.

In addition, the encoding time of the erasure code is not a major issue in this context as it is done only once in advance during the design mapping to the FPGA device. In contrast, a high decoding time prolongs the error recovery process. Therefore, an erasure code with one redundant block and short decoding time satisfies our requirements. In order to reduce the error recovery time, FPGA frames could be divided into several clusters, each of which has its own redundant block.

STORING REDUNDANT DATA

The proposed scheme generates error detection codes (InD parity bits) for each configuration frame and also a redundant erasure block for each cluster. These additional data have to be stored in BRAMs of the FPGA device or in the spare bits of the configuration frames. In case the parity bits are stored in BRAMs, one important concern is not to store parity bits of different frames from one cluster very close together because an MBU might affect parity bits of several frames in that cluster.

In such a scenario, affected bits cannot be recovered as the number of erroneous frames is more than the available erasure blocks (only one erasure block for each cluster).

A similar issue also exists for storing error-detection data and the erasure block of the same cluster, i.e., if an MBU affects both the erasure block of a cluster as well as the parity bits of one frame in the same cluster, the recovery is not possible. Considering these important issues, it is essential to interleave the redundant data in a way that for each cluster at most either error detection parity bits of one frame or the redundant erasure block could be affected. Hence, the size of the clusters has to be larger than one, to facilitate an interleaving distance of at least one among data from each cluster.

II. CONCLUSION

In this paper, a cost efficient scheme based on erasure codes for MBU detection and correction in the configuration and correction in the configuration frames of SRAM based FPGAs. It is implemented as a generic soft core along side with the user design and does not require any changes to the existing FPGA architecture. Comparing to other solutions, this scheme provides a highest level of MBU protection at very low costs with a negligible recovery time.

III. REFERENCES

[1] E. Costenaro, D. Alexandrescu, K. Belhaddad, and M. Nicolaidis, “A practical approach to single event transient analysis for highly complex design,” J. Electron. Test., vol. 29, no. 3, pp. 301–315, 2013.

[2] JEDEC89C Standard, document JEDEC89C. [Online]. Available:http://www.jedec.org/standards-documents, accessed Apr. 2015.

[3] J. S. Plank, “Erasure codes for storage applications,” in Proc. 4th Usenix Conf. File Storage Technol., 2005, pp. 1–74.

[4] L. Rizzo, “Effective erasure codes for reliable computer communication protocols,” ACM SIGCOMM Comput. Commun. Rev., vol. 27, no. 2, pp. 24–36, 1997.

[5] J. S. Plank and M. G. Thomason, “A practical analysis of low-density parity-check erasure codes for wide-area storage applications,” in Proc. IEEE Int. Conf. Dependable Syst. Netw., Jun./Jul. 2004, pp. 115–124.

[6] A. BanaiyanMofrad, M. Ebrahimi, F. Oboril, M. B. Tahoori, and N. Dutt, “Protecting caches against multiple bit upsets using embedded erasure coding,” in Proc. Eur. Test Symp. (ETS), 2014.

[7] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, and J. C. Hoe, “Multi-bit error tolerant caches using two-dimensional error coding,” in Proc. 40th Annu. IEEE/ACM Int. Symp. Microarchitecture, Dec. 2007, pp. 197–209.

[8] J. M. Park, E. K. P. Chong, and H. J. Siegel, “Efficient multicast stream authentication using erasure codes,” ACM Trans. Inf. Syst. Secur., vol. 6, no. 2, pp. 258–285, 2003.