A Bit-Interleaved Embedded Hamming Scheme to Correct Single-Bit and Multi-Bit Upsets for SRAM-Based FPGAs

Shyamsundar Venkataraman, Rui Santos, Anup Das and Akash Kumar
Department of Electrical and Computer Engineering
National University of Singapore
Email: {shyam, elergvds, akdas, akash}@nus.edu.sg

***Abstract*—Single Event Upsets (SEUs) inadvertently change the configuration bits of Static-RAM (SRAM)-based Field Programmable Gate Arrays (FPGAs), leading to erroneous output until the error has been corrected. Scrubbing using an Error Correction Code (ECC) such as hamming is a popular method to correct such faults. However, current works either require a large external memory to store the ECCs or can at most correct only one error in a frame. This paper proposes a novel bit-interleaved embedded hamming scheme along with scrubbing, to correct single (SBUs) and multi-bit upsets (MBUs) in SRAM-based FPGAs. This scheme does not require an external memory to store the ECCs, as they are embedded within the configuration memory itself. Experiments conducted on various benchmarks show that the proposed scheme can handle multiple errors per frame very well, with an embedding efficiency of over 99.3%.**

I. Introduction

Static-RAM (SRAM)-based Field Programmable Gate Arrays (FPGAs) have been gaining significant popularity due to their operational capability and reconfigurability. However, they are prone to radiation from high energy particles which can cause a Single Event Upset (SEU), thereby inadvertently changing the values of the SRAM bits. If SEUs affect only one bit, this effect is known as a Single-Bit Upset (SBU) or single error. On the other hand, if several bits are consecutively affected, this effect is known as Multi-Bit Upset (MBU) or burst errorase (Figure 1). Several mechanisms have been proposed to mitigate SEUs in SRAM-based FPGAs. The most common techniques exploit spatial/hardware redundancy [1], like Triple Modular Redundancy (TMR) [2], [3] and Duplication With Compare (DWC) [4]. However, these techniques require a very large area and power overhead.

From our analysis of different designs, we notice that more than 90% of the user design contains more than 50% unused bits (discussed further in Section III). In this work, we make use of these unused bits to store ECCs, which help repair errors at runtime, without the need for an external memory. Moreover, the ECCs are interleaved within the configuration memory, enabling us to correct MBUs in addition to SBUs. To the best of the authors' knowledge, no other works have proposed a similar approach to correct faults in FPGAs.

Contributions: The following are the key contributions of this paper.

- A novel algorithm to embed hamming code into the bitstream of the FPGA
- Providing multi-bit error correction through bit interleaving and decomposition of frames

Experiments with various benchmark circuits have shown an embedding efficiency of over 99.3% without any extra memory overhead. In cases where 100% embedding efficiency is desired, internal block RAM or a small external memory (typically of size less than 2 KB) can be used to achieve this. Overall, the technique only uses less than 1% memory overhead as compared with other works.

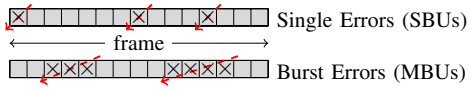

Fig. 1. Error model of single and multi-bit upset.

The remainder of this paper is organised as follows. Section II presents the related works and Section III then motivates the need for the proposed scheme. Section IV presents the scheme in detail, while Section V then discusses the experiments performed and results obtained. Finally, Section VI presents the conclusions.

II. Related Work

Scrubbing is a common method of fault mitigation which requires an external memory to store the configuration frames, frequently called *golden copy*. In order to minimise the impact of faults, the scrubbing frequency must be greater than the expected SEU rate. An alternate solution proposed to avoid using the golden copy is to use ECCs [5]–[7]. The ECC of the frames are computed at runtime and compared against the ECCs stored externally. If an error is identified, the ECCs can be used to correct the errors and the corrected frame is then written back to the FPGA. This research focuses on the existing readback scrubbers that use error correction schemes combined with different ECCs.

Commercial FPGA vendors like Xilinx provide a soft error mitigation strategy using a dedicated set of ECCs for every frame of the FPGA [8]. Few bits[1] from every frame are set aside to store the ECCs which are then readback during runtime. This method is able to detect and correct a single error or detect up to two errors per frame. Though this method is very effective to correct SBUs, it is unable to correct MBUs. Lanuzza *et al.* [5] propose a scheme to mitigate the effects of MBUs in SRAM-based FPGAs. Frame bit interleaving is used to compose a data word and then hamming codes are applied to it. The bit interleaving technique reduces the probability to have several bit-faults in the same data word, increasing the correction efficiency of the hamming codes. However, this scheme imposes a large memory overhead to store the hamming bits. Argyrides *et al.* [7] and Park *et al.* [6] propose solutions that make use of hamming codes combined with parity codes. The ECCs are applied to a frame which is arranged as a 2D matrix. Though these schemes can handle SBUs well, they are not efficient in correcting MBUs.

[1] Virtex-6 FPGA has 13 bits while Virtex-5 has 12

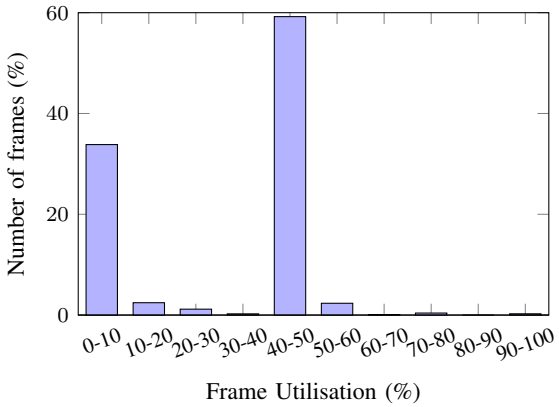

Fig. 2. Average utilisation of the essential frames for 20 MCNC benchmarks

III. Motivation

The lowest reconfigurable granularity of an FPGA is called a *configuration frame* (f). Mathematically, the set of configuration frames of an FPGA can be represented as $C_{bit} = \{f_i | 1 \leq i \leq \eta\}$, where C_{bit} is the configuration bitstream and η is the total number of frames in the FPGA. For example, the Xilinx Virtex-6 (XC6VLX240T) FPGA board contains $28,464$ frames in total. Every frame contains *bits* that represent the configuration data for Look-Up Tables (LUTs), Flip Flops (FFs), switch matrices, etc. Each frame f_i can be represented as $f_i = \{b_{ij} | 1 \leq j \leq \kappa\}$, where κ represents the total number of bits for each frame and b_{ij} represents the j^{th} bit of f_i. For example, the Virtex-6 FPGA contains $2,592$ bits (81 words) per frame. Before we discuss the utilisation of the frames in the FPGA, the following terms are defined.

Definition 1. (Essential and non-essential bits) *Essential bits (B_e) are a set of bits that are associated with the circuitry of the user design [9]. Similarly, non-essential bits (B_{ne}) are defined as those bits that are completely not associated with the circuitry of the user design.*

The *mask bitstream* (M_{bit}) provides information about the essential and non-essential bits in the FPGA and is represented by $M_{bit} = \{f_i^M | 1 \leq i \leq \eta\}$, where f_i^M are the frames corresponding to the mask bitstream.

Definition 2. (Essential frames) *Those frames that contain at least one essential bit in them.*

Mathematically, the set of essential frames can be represented as $F_e = \{f_i | (1 \leq i \leq \eta) \wedge (\exists b_{ij} \in B_e^i) \wedge (1 \leq j \leq \kappa)\}$ where B_e^i represents the set of essential bits for frame f_i. Similarly, a *non-essential frame* is a frame that does not have any essentials bits and is represented by the difference of the set of all frames in the FPGA and the set of essential frames, $F_{ne} = C_{bit} \setminus F_e$.

Definition 3. (Frame Utilisation) *The ratio of the number of essential bits to the total number of bits in a frame is defined as the utilisation of the frame.*

Formally, it can be represented as $f_i^{util} = |B_e^i|/\kappa$, where $|B_e^i|$ is the number of essential bits in frame f_i. Figure 2 shows the average utilisation of the essential frames for all the MCNC benchmark circuits implemented on a Virtex-6 ML605 FPGA board. The utilisation of the frames were computed from the Xilinx Essential Bits tool [9] which provides the necessary C_{bit} and M_{bit} files of the user design. From the utilisation ratios of the different MCNC benchmarks, more than 90% essential frames are only half or less than half utilised. These frames contain more than 50% of non-essential bits that can be used to store ECCs. Since less than 0.25% frames have 100% utilisation, it would not hinder the effectiveness of embedding ECCs in the non-essential frames. These observations motivate the use of non-essential bits to store ECC bits instead of an external memory.

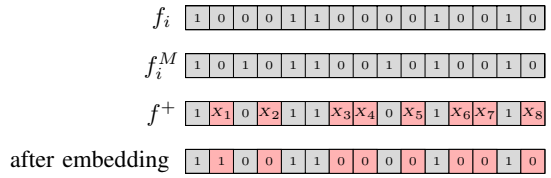

Fig. 3. Example of error detection using embedded parity

IV. Proposed Scheme

This paper proposes a novel bit-interleaved embedded hamming scheme to correct SBUs and MBUs for SRAM-based FPGAs. The scheme uniformly interleaves the bits of each frame into a number of *sub frames* and uses the non-essential bits of the frames to embed hamming parity bits, making the entire sub frame hamming code compliant [10]. At runtime, each frame of the user design is readback and the sub frames are decoded to check for errors. If an error is detected, the embedded hamming code is used to find the location of error and the corrected frame is written back to the design. The following sections discuss the scheme in detail.

A. Embedded Hamming Code

In order to illustrate the technique of embedded hamming code, a simpler technique of *embedded parity code* is first explained. A *parity bit* is a check bit that represents whether the number of 'ones' in the frame is even (p_e) or odd (p_o). In a traditional parity code, the parity bit is appended after the frame. However, in the technique of embedded parity code, the parity of the essential bits is stored in one of the non-essential bits of the same frame. This converts the entire frame into a parity compliant code since any single bit error in the frame can be detected by computing its parity.

For the rest of the examples illustrated in this section, f_i and f_i^M shown in Figure 3 are used. Both these frames are combined into one frame (f^+) for easier comprehension, where the non-essential bits are represented by don't care bits "x". In order to apply embedded parity code to f^+, p_e for the essential bits is calculated. $p_e = 1$ in this case since there are an odd number of ones (5) in the essential bits. This value is then stored in one of the non essential bits (x_1 in this example) while the other non-essential bits are set to 0 as shown in Figure 3. The entire frame f^+ is now parity compliant. When a bit-flip occurs, the error can be detected as the p_e of the frame changes to 1.

Extending the same concept used in embedded parity code, an *embedded hamming code* manipulates the non-essential bits of a frame such that the entire frame becomes hamming code compliant. A typical hamming code consists of a number of data bits (d_i) with parity bits (p_i) embedded in indices (ix) that are powers of two as shown in Figure 4. These parity bits are computed by choosing a different set of data bits for each parity bit as specified by a *hamming check matrix (H)* [11]. Each row of H corresponds to a parity bit while the columns specify whether a data bit is included in the parity bit of that

$$ix \quad \begin{array}{|cccccccccccccc|} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \end{array}$$
$$f_i \quad \begin{array}{|cccccccccccccc|} p_1 & p_2 & d_1 & p_3 & d_2 & d_3 & d_4 & p_4 & d_5 & d_6 & d_7 & d_8 & d_9 & d_{10} & d_{11} \end{array}$$

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Fig. 4. Hamming check matrix

f^+ | 1 | X_1 | 0 | X_2 | 1 | 1 | X_3 | X_4 | 0 | X_5 | 1 | X_6 | X_7 | 1 | X_8 |

after embedding | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |

Fig. 5. Example of error detection and correction using embedded hamming

row. For the H shown in Figure 4, the parity bit p_1 of f_i is given by $p_1 = d_1 \oplus d_2 \oplus d_4 \oplus d_5 \oplus d_7 \oplus d_9 \oplus d_{11}$. Other parity bits are computed in a similar manner according to H.

For the example frame f^+, computing the values of the parity bits using H, results in a set of XOR equations with the different non-essential bits as shown below.

$$X_3 \oplus X_7 \oplus X_8 = 1$$
$$X_1 \oplus X_3 \oplus X_5 \oplus X_8 = 1$$
$$X_2 \oplus X_3 \oplus X_6 \oplus X_7 \oplus X_8 = 1$$
$$X_4 \oplus X_5 \oplus X_6 \oplus X_7 \oplus X_8 = 0$$

Solving these equations gives the values of the non-essential bits that make the frame hamming code compliant. For the example considered, we obtain $X_1 = X_2 = X_4 = X_5 = X_6 = X_7 = X_8 = 0$ and $X_3 = 1$. Figure 5 shows the frame f^+ before and after embedding.

Algorithm 1 illustrates the steps needed to embed the hamming code into a configuration frame. The algorithm essentially tries to find a suitable value for the non-essential bits, such that the entire frame becomes hamming compliant.

Algorithm 1 Embedded hamming code technique

Input: f_i, f_i^M
Output: f_i
1: $\kappa = |f_i|$, $\delta = 0$
2: **repeat**
3: $\quad \delta \leftarrow (\delta + 1)$
4: **until** $(\delta + \kappa + 1) \leq 2^\delta$
5: $H \leftarrow gen\_ham\_mat(\delta, \kappa)$
6: $b \leftarrow ((f_i \times H^T) \bmod 2)^T$
7: $tmp \leftarrow H \oplus repmat(f_i^M, \delta)$
8: $A \leftarrow tmp(:, find(f_i^M == 0))$
9: $f_i(find(f_i^M == 0)) \leftarrow solve\_lin\_eq(A, b)$
10: **return** f_i

Lines 1-4 compute the δ required for f_i. The *hamming check matrix* H (dimensions $\delta \times \kappa$) is then generated (line 5) in order to compute the parity bits of the hamming code [11]. The column vector b, which is then computed using H and f_i (line 6), represents the constants of the linear equality constraints of the hamming code. The function *repmat* in line 7 replicates the mask frame f_i^M for δ rows so that it can be XORed with H in order to obtain the coefficients of the linear equality constraints (tmp) of the hamming code. Line 8 selects only those columns of matrix tmp which correspond to non-essential bits of f_i. Finally the values of the non-essential bits are computed by solving the binary linear equation function $solve\_lin\_eq(A, b)$.

B. Bit Interleaving

If the embedded hamming code was applied to the entire frame, only SBUs can be corrected, as hamming code is capable of correcting only one error. Bit interleaving has been considered in combination with embedded hamming to increase the number of errors than can be detected and corrected.

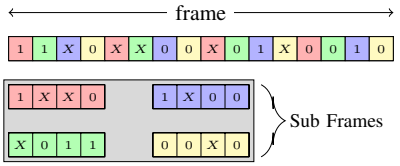

Fig. 6. Example of decomposition of a frame into sub frames through bit interleaving

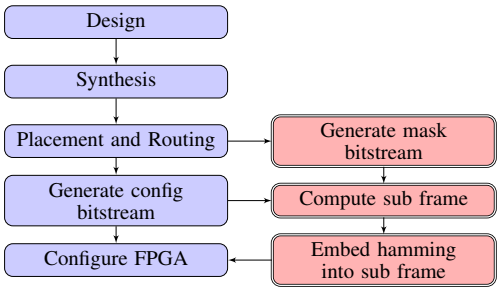

Fig. 7. Overall flow of the proposed scheme

Definition 4. (SUB FRAME) *A frame $f_i \in FPGA$ is subdivided into φ number of equal sub frames in which the bits of each sub frame are distributed uniformly along the length of frame f_i as shown in Figure 6.*

A frame can now said to be composed of a set of sub frames $f_i = \{(s_{ij} \subset f_i) | (1 \leq j \leq \varphi)\}$. As shown in the figure, a bigger frame is decomposed into smaller sub frames, each of them containing some essential and non-essential bits. Moreover, the bit interleaving also spreads out the non-essential bits of the frame to all sub frames. This is very important as the embedded hamming technique, can only work if there are enough non-essential bits to create a hamming compliant code.

Definition 5. (EMBEDDING EFFICIENCY) *The ratio of the number of sub frames that can be embedded with hamming code to the total number of sub frames in the user design.*

Since the hamming bits are directly embedded into the frames of the bitstream, there is no external memory required to store the ECC bits. Moreover, since the proposed scheme can both detect and correct errors autonomously, it obviates the need for a golden copy of the design as well. However, there are cases where the sub frames are not embeddable. In these cases, it becomes necessary to track the sub frames that were not embedded. If the user requires 100% embedding efficiency, the hamming codes for these sub frames can be stored in the internal BRAM or in an external memory.

The overall flow of the scheme is shown in Figure 7 with the steps in blue showing the normal flow of the FPGA configuration. The steps in red highlight the necessary changes required to the normal flow.

V. EXPERIMENTS AND RESULTS

All experiments were conducted on a Virtex-6 FPGA board[2]. The benchmarks were synthesised for the same board

[2]XC6VLX240T-1f1156

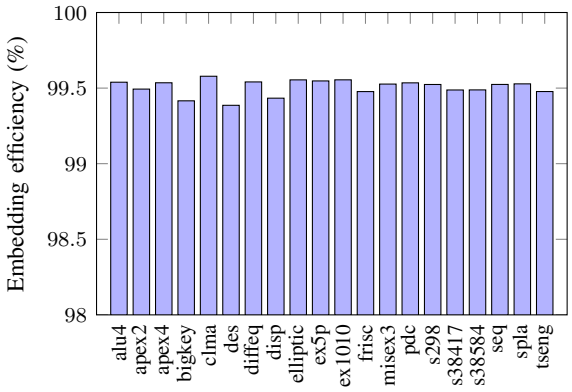

Fig. 8. Embedding efficiency for different benchmark circuits

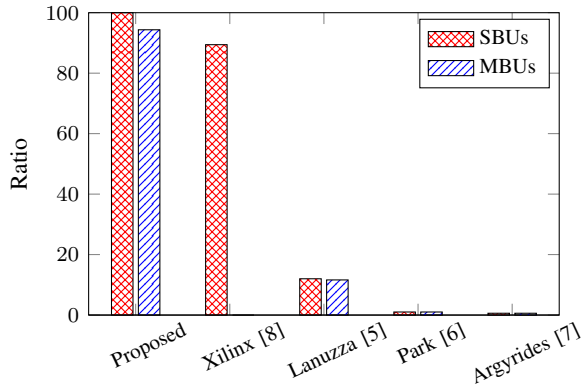

Fig. 9. Ratio of error correction performance to memory overhead

with the synthesis option set to speed. Moreover, the mask file was generated through the Xilinx Essential Bits BitGen parameter *-g essentialbits:yes*. Since the reserved bits of the Xilinx ECC (13 bits) are not used by the proposed scheme, these bits are used to track the sub frames that were not embedded with hamming code. Hence, the number of sub frames (φ) has been set to 13 for all experiments.

A. Embedding Efficiency

Figure 8 shows the embeddeding efficiency for various MCNC benchmark circuits considered. The proposed scheme has an embedding efficiency of over 99% for all designs. Moreover, if the other $< 1\%$ of the sub frames require error correction ability, the ECCs of these frames can be stored in a memory external to the frame (like a BRAM). This memory would be very small (typically < 2 KB).

B. Error Correction Performance and Memory Overhead

Overall, the proposed scheme is able to correct more than 90% of the errors when up to $5,000$ SBUs or $2,000$ MBUs[3] are injected. Moreover, its performance is better or on par with the other state-of-the-art techniques. Table I compares the memory overhead required by the different schemes considered. The proposed scheme and [8], excel over other works since they require no memory overhead. The other works require at least 1 MB of extra memory to store the ECCs. Moreover, this extra memory needs to be further protected from SEUs. The proposed method does not require any extra protection for the ECCs since they are already embedded within the FPGA configuration frames.

TABLE I. MEMORY OVERHEAD REQUIRED

Proposed	Xilinx [8]	Lanuzza [5]	Park [6]	Argyrides [7]
0 MB	0 MB	1.05 MB	4.19 MB	7.07 MB

The ratio between the error correction performance and the memory overhead represents how good the error correction is for a unit memory overhead. A larger ratio implies a better error correction performance using a smaller memory overhead. Figure 9 shows the ratios obtained for the various state-of-the-art considered for an error injection of 500 SBUs and 500 MBUs. The values have been normalised with reference to the proposed scheme. The proposed scheme has a ratio over $8\times$ better than Lanuzza *et al.* and over $90\times$ better than Park *et al.* and Argyrides *et al.*

[3]MBU burst error length was fixed to 4 bits per burst error [12]

C. Timing Analysis for Bitstream Generation

The time taken to embed hamming code is directly proportional to the size of the user design. On average, the time taken to embed hamming code into the designs was less than a minute. It is to be noted that this time is just a one time overhead which is required after the bitstream of the design has been generated.

VI. CONCLUSION

This paper proposes a novel bit-interleaved embedded hamming scheme to mitigate radiation induced SBUs and MBUs. The scheme uniformly interleaves the bits of each frame into a number of sub-frames and uses the non-essential bits of the frames to embed hamming parity bits, making the entire sub frame hamming code compliant. Experiments conducted with various benchmarks have shown that the ratio of error correction performance to the memory overhead is highest for the proposed scheme as compared with other state-of-the-art.

REFERENCES

[1] I. Koren and C. Krishna, *Fault-tolerant systems*, 2007.

[2] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin, "Improving FPGA Design Robustness with Partial TMR," in *44th Annual IEEE International Reliability Physics Symposium Proceedings*, 2006.

[3] C. Bolchini, A. Miele, and M. D. Santambrogio, "TMR and Partial Dynamic Reconfiguration to mitigate SEU faults in FPGAs," in *Defect and Fault-Tolerance in VLSI Systems*, 2007.

[4] A. Sarkar, F. Mueller, H. Ramaprasad, and S. Mohan, "Push-assisted migration of real-time tasks in multi-core procs," *SIGPLAN Not.*, 2009.

[5] M. Lanuzza, P. Zicari, F. Frustaci, S. Perri, and P. Corsonello, "A self-hosting configuration management system to mitigate the impact of Radiation-Induced Multi-Bit Upsets in SRAM-based FPGAs," in *IEEE International Symposium on Industrial Electronics*, 2010.

[6] S. P. Park, D. Lee, and K. Roy, "Soft-Error-Resilient FPGAs Using Built-In 2-D Hamming Product Code," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2012.

[7] C. Argyrides, D. Pradhan, and T. Kocak, "Matrix codes for reliable and cost efficient memory chips," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2011.

[8] K. Chapman, "SEU Strategies for Virtex-5 Devices," Xilinx Application Note, 2010.

[9] R. Le, "Soft Error Mitigation Using Prioritized Essential Bits," Xilinx Application Note, 2012.

[10] A. D. Houghton, *The engineer's error coding handbook*. Chapman & Hall, 1997.

[11] S. Wicker, *Error control systems for digital communication and storage*. Prentice Hall, 1995.

[12] H. Quinn, P. Graham, J. Krone, M. Caffrey, and S. Rezgui, "Radiation-induced multi-bit upsets in SRAM-based FPGAs," *IEEE Transactions on Nuclear Science*, 2005.