Multi-Directional Error Correction Schemes for SRAM-Based FPGAs

Shyamsundar Venkataraman, Rui Santos, Sidharth Maheshwari, Akash Kumar
Department of Electrical & Computer Engineering
National University of Singapore
Email: {shyam, elergvds}@nus.edu.sg, sidharth.iitg@gmail.com, akash@nus.edu.sg

***Abstract*—Readback scrubbing is considered as an effective mechanism to correct errors in Static-RAM (SRAM)-based Field Programmable Gate Arrays (FPGAs). However, current solutions have a low error correction percentage per unit area overhead. This paper proposes two new error detection/correction mechanisms that combine frame readback scrubbing with error correction codes (ECCs) that are applied in multiple directions, to achieve a high error correction percentage per unit area overhead. Experiments conducted show that the proposed schemes have an excellent error correction percentage (over 99%), especially for multi-bit upsets, while using up to 59.37% lesser area overhead compared with other state-of-the-art.**

I. Introduction

Modern Static-RAM (SRAM)-based Field Programmable Gate Arrays (FPGAs) are gaining significant importance in space applications due to their operational capacity and performance. Moreover, these devices can be reconfigured after launch depending on various functional requirements and changes in the device environment [1]. However, due to the technological developments leading to denser chips, these devices become vulnerable to radiation effects called Single Event Upsets (SEUs) that are common in space environments. SEUs can inadvertently change the configuration of the SRAM bits, thereby changing the functionality of the circuit implemented [2]. 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 a Multiple-Bit Upset (MBU) or burst errors (Figure 1).

Several mechanisms have been proposed to mitigate SEUs in SRAM-based FPGAs. The most common solution explores spatial/hardware redundancy [3]. Triple Modular Redundancy (TMR) [4] [5] replicates three times the hardware module to be protected and votes on their outputs, identifying the right results and a possible faulty module. However, this approach imposes a great overhead in terms of area and power consumption. Moreover, this method does not avoid error propagation if more than one component produces erroneous output. Duplication With Compare (DWC) [6] is an alternative approach to reduce the TMR overhead. It compares the output results of duplicated modules in order to identify the errors. However, it cannot correct them, but can trigger the suitable operations to do that, such as a full re-execution.

Blind scrubbing is another traditional method of fault mitigation. This mechanism does not detect the existence of faults, but instead, periodically rewrites the configuration frames on to the FPGA, overwriting possible faulty bits caused by SEUs [7] [8]. An external memory with continuous access is required to store the configuration frames, frequently called as *golden copy*. In order to minimize the faults' impact, the scrubbing frequency must be greater than the expected SEU frequency. However, determining this frequency requires in advance a deep knowledge about fault susceptibility of the device technology, as well as the environmental conditions to which it would be exposed.

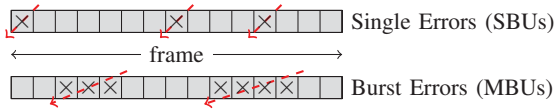

Fig. 1. Error model of single and multi-bit upset.

The balance between error correction and the overhead required has been a challenging problem faced by researchers. This paper proposes two new error correction schemes based on frame readback and ECCs. Depending on the balance of error correction required and area overhead acceptable, one of the proposed solutions can be used. The proposed schemes detect and correct errors in each individual frame of the FPGA by mapping each frame to a $2D$ matrix and then computing hamming or parity bits for the matrix in different directions (rows, columns and diagonals). The mechanism adopted for computing the ECCs ensures a very good error correction performance, especially for burst errors and also uses lesser area overhead as compared with other state-of-the-art.

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

- 2 schemes based on frame readback and ECCs (hamming [9] and parity codes) to repair SEUs in the configuration memory;
- Improving the error correction percentage per unit area overhead;
- A hardware architecture to evaluate the efficiency of the proposed schemes.

The rest of the paper is organised as follows. Section II presents related works in the field of readback scrubbing. Section III presents in detail the proposed error correction schemes. Section V presents the experiments and the corresponding results obtained. Section VI concludes the paper.

II. Related Work

Readback scrubbing has been seen by the researchers as an effective mechanism to provide error correction in SRAM-based FPGAs [10]–[14]. Three categories of readback scrubbers can be found in the literature. The first category enables fault detection with a direct comparison between read frames and the golden copy [13]. The fault is corrected

by overwriting the faulty frame with the respective golden copy frame. The second category does not use an exhaustive comparison with the golden copy [14]. It detects the faults matching the online computed error detection codes (EDCs) with the original ones, previously computed and externally stored for each frame. Similar to the previous category, the fault recovery is performed using the frame's golden copy. The third category enables the fault detection, computing ECCs for each frame [10]–[12]. The ECCs allow the faults' detection, similar to the previous category. However, upon fault detection in a frame, the faults can be easily recovered using the ECCs. Once the fault is recovered, the frame is written back into the FPGA. Different error detection schemes, combined with different ECCs have been proposed in the literature. However, they are not really efficient to handle burst errors.

Lanuzza *et al.* [12] proposes a scheme to correct burst errors in SRAM-based FPGAs by applying hamming codes to a data word obtained by frame bit interleaving. The bit interleaving technique reduces the probability to have several bit-faults in the same data word, thereby increasing the correction efficiency. However, the error correction is limited by the amount of bit interleaving, and hence might not be suitable if a very high error correction efficiency is required.

Argyrides *et al.* [10] introduce a scheme called Matrix Code (MC), that uses hamming codes combined with parity codes to enable the detection and correction of multiple errors in an FPGA configuration frame. A frame word is mapped into a matrix of subwords. Errors are corrected by computing hamming codes for each row, providing Single Error Correction Double Error Detection (SECDED) and computing parity codes for each column. As a result, this scheme is not efficient in handling MBUs, since if more than two errors occur in the same row, they are not detected by the ECC code.

Park *et al.* [11] propose a built-in 2-D Hamming Product Code (2-D HPC) scheme. This technique is able to perform SECDED by using hamming codes built from arranging the FPGA configuration frame in a 2-D array. Therefore, hamming codes are computed for each row and for each column of the 2-D array. Like the previous work, this scheme is not able to detect more than two errors that occur in the same row or column, and hence not efficient in handling burst errors.

III. Proposed Schemes

This paper proposes two novel schemes for error detection and correction in SRAM-based FPGAs, using the readback scrubbers. Both these schemes are based on existing ECCs, i.e. parity codes and hamming codes, which are applied to the FPGA frames. A frame is the lowest reconfigurable granularity that can be found in an FPGA. In particular, the FPGA configuration data frames FRc contain information about the circuit design to be implemented on the FPGA. These schemes map sequentially the content of each frame $fr_i \in FRc$ in a 2D data matrix mfr_i with nr rows and nc columns. ECC codes are then computed for this 2D data matrix and stored in an internal or external storage. During runtime, the frames are periodically read and mapped to the 2D data matrix to see if they contain any errors. Errors are identified by comparing the ECCs to the ones stored in the memory. In the case of an error, the proposed algorithm attempts to rectify the affected bits. Once the errors are rectified, the corrected frame is then written back into the FPGA board.

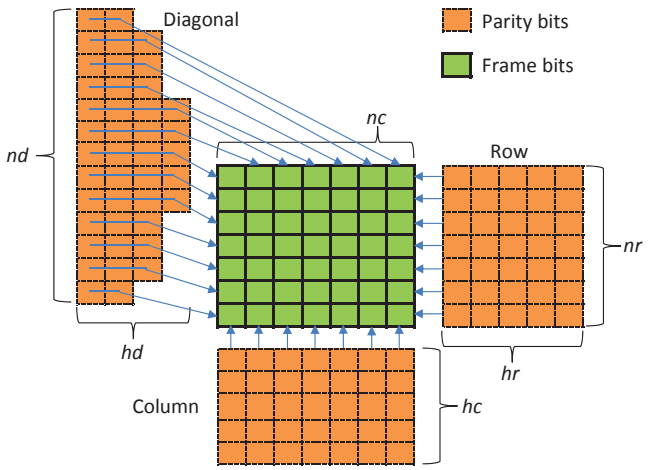

Fig. 2. ECCs used in H^3 scheme.

A. Error Model

There are two different error models considered for this work. Both these models focus on SEUs, i.e., transient soft errors due to a single particle strike and which affect the configuration (both the logic and routing) bits of the FPGAs. The first model considers SBUs (single errors) and the second considers MBUs (burst errors). Both error models are illustrated in Figure 1. According to the literature, the latter model is more realistic since it is very common that a single particle strike might affect one or more neighbouring bits [12]. We try to detect and correct both single errors as well as burst errors in this work. The following sub-sections discuss these schemes in detail with examples to illustrate the working of each.

B. H^3 Scheme

H^3 scheme applies hamming codes to the frame matrix at the design time in three directions, rows, columns and in one of the diagonals as shown in the Figure 2. With this scheme, single error correction (SEC) is available for each row $r_j \in mfr_i$, for each column $c_j \in mfr_i$ and for each line of one of the diagonals $d_j \in mfr_i$. The pseudocode presented in Algorithm 1 illustrates the H^3 scheme behaviour. The matrix frame is received as input and SEC is sequentially applied to the rows, columns and diagonal. While errors are detected in the matrix frame, SEC is continuously applied. When no errors are detected the mfr_i is returned.

Figure 3 illustrates the working of the H^3 scheme. In the first iteration, the algorithm computes the hamming codes for the rows, columns and diagonals respectively and compares it to the ones already stored in memory. As can be seen from the figure, errors are found in rows 2 and 7 (matrix A1), columns 1, 2, 4, and 6 (matrix A2) and diagonal 2 and -3 (matrix A3). These error bits are highlighted in yellow color. All the errors are corrected at the end of the first iteration.

Algorithm 1 H^3 scheme.

Require: mfr_i
Ensure: mfr_i
1: $errorExists = true$
2: **while** $errorExists = true$ **do**
3: **for all** $r_j \in mfr_i$ **do** $sec(r_j)$;
4: **for all** $c_j \in mfr_i$ **do** $sec(c_j)$;
5: **for all** $d_j \in mfr_i$ **do** $sec(d_j)$;
6: **update** $errorExists$;
7: **end while**
8: **return** mfr_i;

sec – single error correction

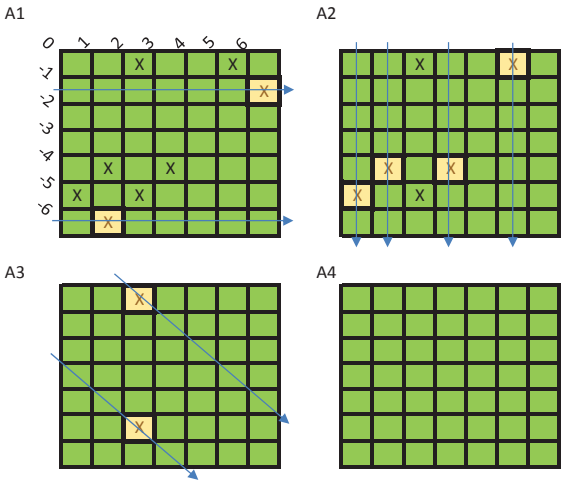

Fig. 3. Example of H^3 scheme.

ECC Overhead: By definition, for SEC, the number of hamming bits h required for a n-bit word is given by the equation, $n + 1 + h \leq 2^h$. This way, the ECC overhead can be given by Equation 1,

$$H^3oh = \sum_{j=1}^{nd} hd_j + (nr \times hr) + (nc \times hc) \quad (1)$$

where, hd_j, the number of hamming bits for each diagonal j, is given by $|d_j| + 1 + hd_j \leq 2^{hd_j}$; hr, the number of hamming bits for the rows, is given by $nc + 1 + hr \leq 2^{hr}$; hc, the number of hamming bits for the columns, is given by $nr + 1 + hc \leq 2^{hc}$; nd, the number of diagonals of the matrix, is given by $nd = nr + nc - 1$; d_j is the set of elements of the diagonal j.

C. P^2H Scheme

P^2H scheme provides error detection and correction through the use of both parity and hamming codes. Parity codes are applied for each row $r_j \in mfr_i$ and for each column $c_j \in mfr_i$, while hamming codes are applied for each line of one of the diagonals $d_j \in mfr_i$, as shown in Figure 4. Since parity code can only detect the presence of a single error, this scheme also employs an efficient algorithm to not only detect multiple errors, but to also correct them. The pseudocode of this scheme is presented in Algorithm 2.

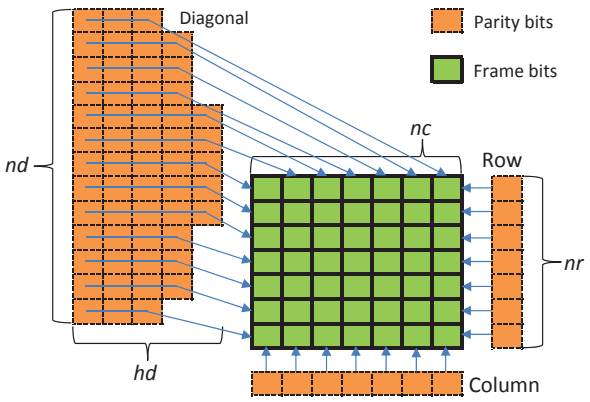

Fig. 4. ECCs used in P^2H scheme.

Algorithm 2 P^2H scheme.

Require: mfr_i
Ensure: mfr_i
1: $errorExists = true$
2: $diagErrLoc = []$; $rowErrLoc = []$;
3: $colErrLoc = []$; $errLocs = []$;
4: **while** $errorExists = true$ **do**
5: **for all** $d_j \in mfr_i$ **do** $sec(d_j)$;
6: **for all** $d_j \in mfr_i$ **do** $diagErrLoc+ = ded(d_j)$;
7: **for all** $r_j \in mfr_i$ **do** $rowErrLoc+ = ped(r_j)$;
8: **for all** $c_j \in mfr_i$ **do** $colErrLoc+ = ped(c_j)$;
9: $eci(diagErrLoc, rowErrLoc, colErrLoc)$;
10: **update** $errorExists$;
11: **end while**
12: **return** mfr_i;

ded - double error detection
ped - parity error detection
eci - error correction by intersection

The first operation detects and corrects any single-bit error for each diagonal $d_j \in mfr_i$ through the function $sec(d_j)$. This first operation can be observed in Figure 6 ($A1 - A2$). Note the frame burst errors can be easily corrected on this first operation. Once all the single bit errors in the diagonals are removed, it is not possible to correct any more errors using only a single direction. Therefore, the next operation set identifies the diagonals, rows and columns with errors and matches them in order to identify their exact location. $ded(d_j)$ identifies the diagonals with double errors, through the hamming codes. $ped(r_j)$ and $ped(c_j)$ identify the rows and the columns with errors, through the parity codes. These locations are submitted to the function eci, which match them and produce a set of intersections in the matrix frame. These intersections are potential bit errors. This way, several operations are then performed to identify and correct the errors. Figure 5 presents a flowchart, which describes the operation flow performed by P^2H scheme. In particular, the shaded area describes the operation flow performed in the scope of eci function.

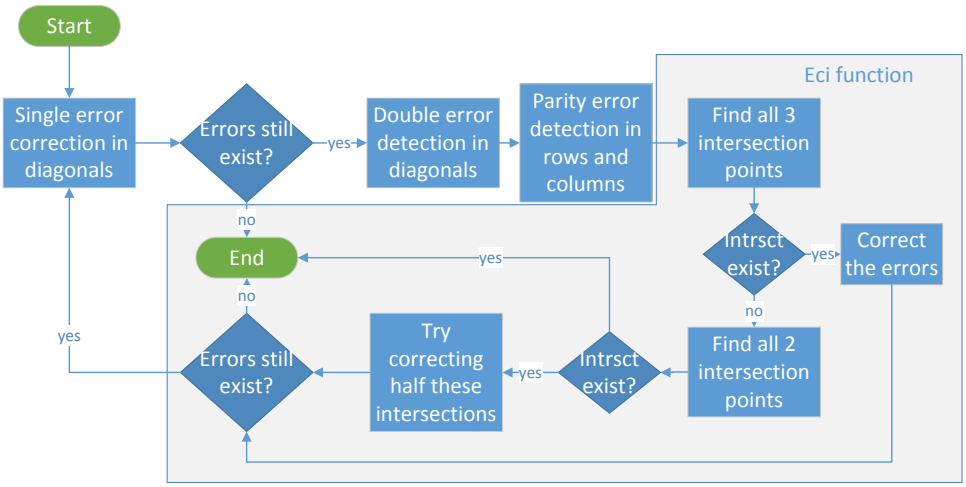

Fig. 5. Flowchart of P^2H scheme.

Find 3 intersection points operation looks for the intersections between the row, columns and diagonals which have errors. The set of points identified by three intersections are considered bit-errors and will be corrected by inverting their bits. However, there are cases where three intersections can identify false errors, as the example described in Figure 6 ($B1 - B3$). In such cases, which we might end up inverting correct bits, the subsequent iterations of the algorithm will then try to correct these errors. Matrix $B2$ shows four bit errors and one false bit error highlighted in yellow. After the bit inversion, the four erroneous bits are corrected, but the false bit error becomes an error. The algorithm returns to the beginning and this last bit error is corrected through the diagonal, as described in the matrix $B3$.

When there are no three intersections possible or errors still exist after the three intersection step, *find 2 intersection points* operation tries to find out all possible erroneous bits by checking the diagonal-row, diagonal-column and row-column intersections. These intersections that are identified might have both correct bits and erroneous bits. In order to reduce the chances of inverting the correct bits, only half of these points are inverted. This operation can be observed in example ($C1 - C4$) of Figure 6. Matrix $C2$ shows the bits identified as faulty in yellow and blue color. Since not all identified bits are erroneous, the algorithm tries to randomly choose half of these bits. It is to be noted here that it is not possible to identify the erroneous bits deterministically. The algorithm then flips these bits that were randomly chosen, highlighted in the Matrix $C2$ in blue. After this operation, the algorithm then starts from the beginning again and tries to find the single errors in the diagonals as shown in matrix $C3$. After performing three intersections again as shown in matrix $C4$, we can see that the algorithm can correct all the errors in the frame.

ECC Overhead: By definition, SECDED requires one extra hamming bit when compared with SEC for the same n-bit data word. Therefore, the ECC overhead for this scheme can be given by Equation 2,

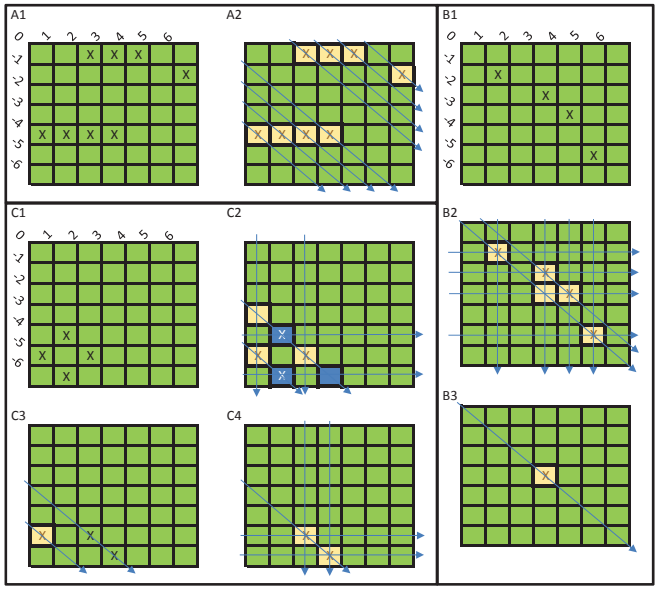

Fig. 6. Example of P^2H scheme.

$$P^2Hoh = \sum_{j=1}^{nd}(hd_j + 1) + nr + nc \tag{2}$$

where, hd_j is given by $|d_j| + 1 + hd_j \leq 2^{hd_j}$ and nd is given by $nd = nr + nc - 1$.

D. Optimal Diagonal

A straightforward way to implement hamming code for the diagonals is to have separate ECCs for every line of the diagonal. However, this might not be the most optimal implementation as the ECCs may support more bits than there are in a particular diagonal. For this reason, we use an optimal diagonal construction as illustrated by the Figure 7. Such a method has already been implemented for tolerating

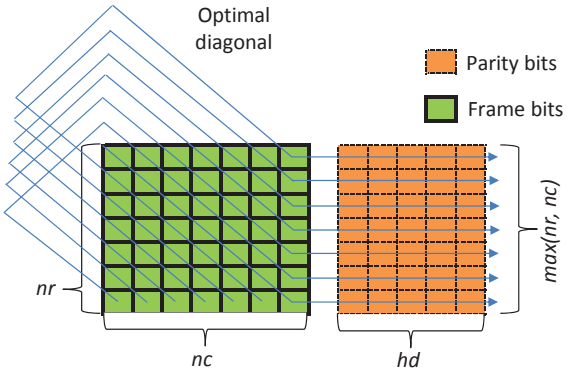

Fig. 7. Illustration of optimal diagonal

disk failures in Redundant Array of Independent Disk (RAID) architectures [15] [16]. Using this optimization reduces the number of ECC bits required. The H^3 and P^2H schemes however, remain the same for the optimized hamming code of the diagonal and require no modification. Even though optimal diagonal is more efficient that a non-optimal diagonal in terms of number of ECC bits used, there are cases where non-optimal diagonal would be preferred to the optimal one, as will be discussed in Section V-A.

ECC Overhead: Considering a frame matrix with nr rows and nc columns, the ECC overhead for H^3 scheme with optimal diagonals is given by Equation 3.

$$H^3oh = (hd \times max(nr, nc)) + (nr \times hr) + (nc \times hc) \quad (3)$$

where, hd is given by $min(nr, nc) + 1 + hd = 2^{hd}$; hr, the number of hamming bits for the rows, is given by $nc+1+hr \leq 2^{hr}$; hc, the number of hamming bits for the columns, is given by $nr + 1 + hc \leq 2^{hc}$.

Similarly, the ECC overhead for the P^2H optimal scheme is given by Equation 4.

$$P^2Hoh = ((hd + 1) \times max(nr, nc)) + nr + nc \quad (4)$$

where, hd is given by $min(nr, nc) + 1 + hd = 2^{hd}$.

IV. Implementation Architecture

The implementation of the proposed schemes on a Commercial Of-The-Shelf (COTS) FPGA presents a few constraints due to the current FPGA architecture. For example, the COTS FPGA architecture does not provide a direct way to access the contents of the frame as columns and diagonals. This necessitates the need for a separate hardware architecture to implement the proposed schemes. This section describes the details of the implementation of the schemes in COTS FPGAs without modifying their architecture. Figure 8 shows the overall architecture of the FPGA along with the necessary modules for implementing the algorithm proposed. As can be seen from the figure, the Internal Configuration Access Port (ICAP) is used to read the individual frames of the design. Each of these frames is then sent to the *Frame Storage Module* which converts the data into the necessary rows, columns and

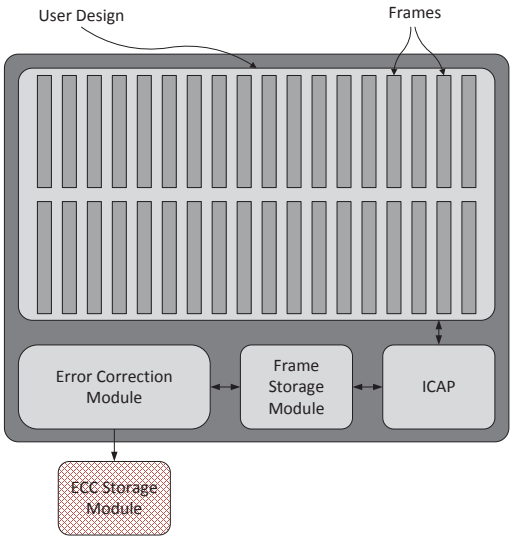

Fig. 8. Proposed implementation architecture.

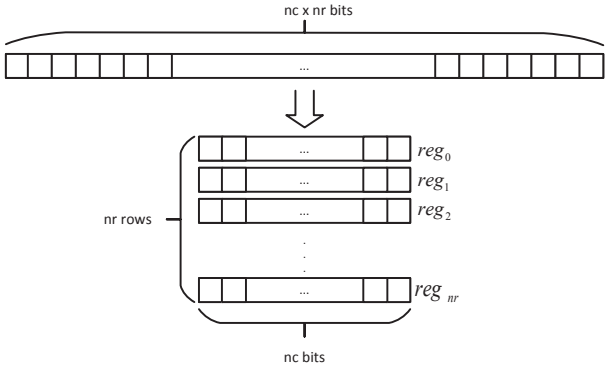

Fig. 9. Storing frame read by ICAP into the registers

diagonals. This data is then provided to the *Error Correction Module* that implements the proposed schemes in hardware or even in software. During of the schemes execution, if errors are detected in the frame, they are corrected and then the frame is written back to the design through the ICAP module. The *ECC Storage Module* is a memory that stores the original ECC codes for all the frames computed at design time.

A. Frame Parsing

As explained previously, the ICAP module reads the data from the design frame by frame. Each frame is then converted to an $nr \times nc$ 2-D grid for which ECC codes must be computed for each of the rows, columns and diagonals according to the schemes. If the frame does not evenly divide into the 2-D grid, it will be padded with extra 0s.

Computing the ECC codes for the rows is the simplest of all. However, computing the ECC codes for the column requires the entire frame data to be read before it can be encoded since a single column spans multiple rows. Due to this, the entire frame is stored inside registers within the Frame Storage Module as can be seen from Figure 9. The frame containing $nr \times nc$ bits is stored in nr registers each having

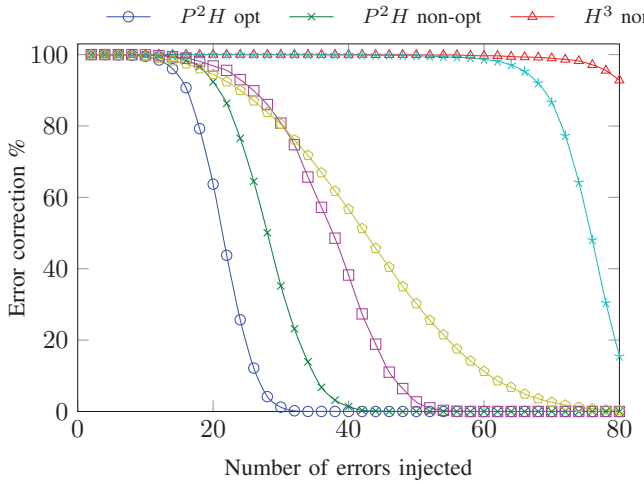

Fig. 10. Error performance of different algorithms for single errors

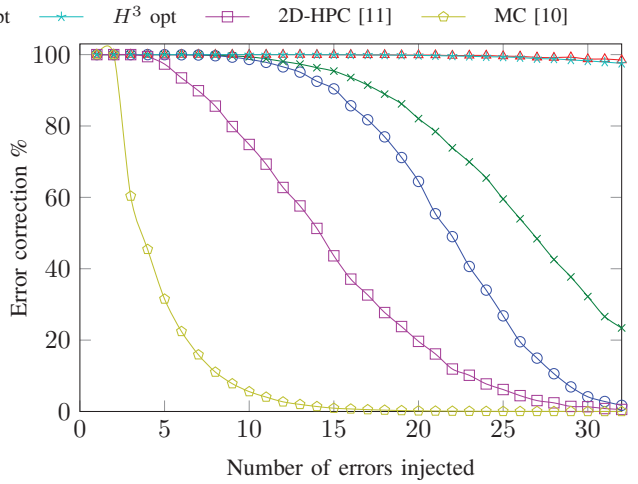

Fig. 11. Error performance of difference algorithms for burst error model

nc bits. Storing the frame in registers helps to quickly retrieve the bits corresponding to a certain column. To retrieve the i^{th} column, the registers would be accessed by

$$col(i) = reg_0(i)\ \&\ reg_1(i)\ \&\ ...\ \&\ reg_{nr}(i).$$

Note that the *&* operator here is used to concatenate the bits from the registers and not logically *and* them.

The ECC codes for the diagonals need to be computed similar to the columns. Since each diagonal spans multiple rows, the diagonals can be encoded only after the entire frame is read. For both optimal and non-optimal diagonals, the Error Correction Module needs to access the frame bits in such a way that it corresponds to each of the diagonals. To access the i^{th} diagonal of the frame, the registers would be accessed by:

$$\begin{cases} diag_{nonopt}(i) = reg_0(i)\ \&\ reg_1(i+1)\ \&\ \ ...\ \&\ reg_{nr-i}(nr), \\ \text{if } i \geq 0 \\ diag_{nonopt}(i) = reg_{-i}(0)\ \&\ reg_{-i}(1)\ \&\ \ ...\ \&\ reg_{nr}(nr+i), \\ \text{if } i < 0 \end{cases}$$

$$diag_{opt}(i) = reg_0(i \bmod n)\ \&\ reg_1((i+1) \bmod n)\ \& \\ ...\ \&\ reg_{nr}((i+nr) \bmod n)$$

V. Experiments and Results

The schemes proposed in this work were simulated in Matlab for $100,000$ simulations. For the purpose of error correction performance, a window size of 32 was used (i.e. a 32×32 matrix of frame). Errors were injected according to the error model defined in Section III-A. The schemes are compared with the closest related works [10] [11] in terms of error correction performance, ECC overhead and ratio between these two. Furthermore, the hardware implementations of the proposed schemes have been evaluated for their area overhead, utilization and frequency.

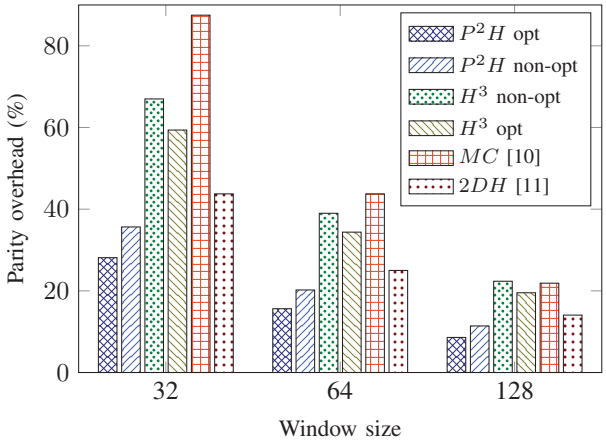

Fig. 12. Area overhead - Percentage

A. Error Correction Performance

For the single error model affecting only a single bit, the graph was plotted for the percentage of errors corrected versus the number of errors injected. As can be seen from Figure 10, the proposed H^3 scheme has an excellent error correction even up to 70 errors per window. Moreover, the P^2H scheme for both optimal and non-optimal diagonals performs equivalent to the 2D-HPC [11] up to 10 errors beyond which it becomes worse. The MC [10] scheme performs slightly better than the $2D$ Hamming scheme but worse than the H^3 scheme.

B. Area Overhead

Similarly, Figure 11 plots the error performance of the different schemes for the burst error model [17]. In the worst case an SBU can affect four consecutive bits. Therefore, every error injected in the simulation affected one to four bits in the frame. Since the burst errors only affect neighbouring bits, burst errors were injected in only one direction. Since burst errors occur only in one direction, calculating hamming bits diagonally has a much better probability of correcting the faults. From the graph, we can see that both the proposed

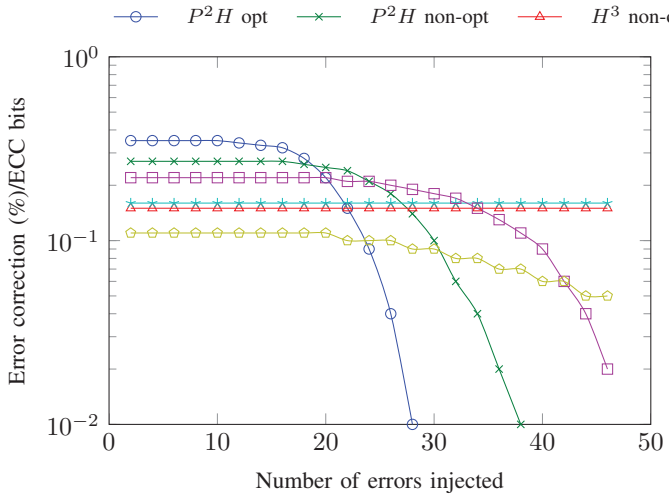

Fig. 13. Ratio between ECCs overhead and error correction percen. for SBU.

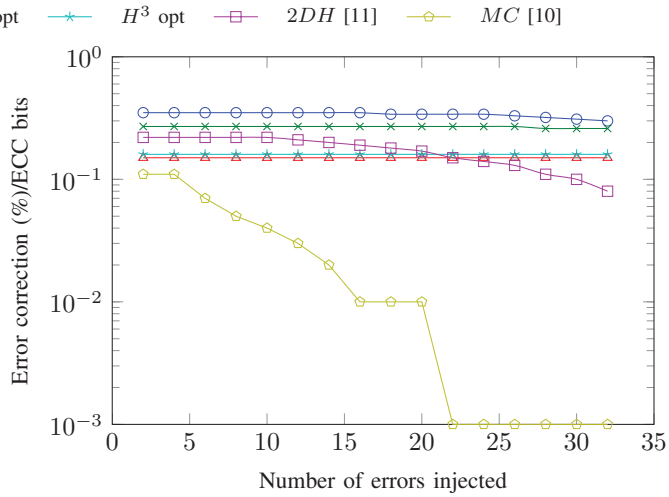

Fig. 14. Ratio between EECs overhead and error correction percen. for MBU.

schemes perform much better under the burst error model as compared with the 2D-HPC and MC schemes.

The non-optimal diagonal for both schemes performs better than the optimal one since there are more ECC bits in its hamming code. Hence, even though the optimal diagonal is more efficient in terms of number of ECC bits used, the non-optimal diagonal would be preferred if the design is likely to encounter more errors than the optimal diagonal can handle.

Figure 12 compares the ECC overhead percentage for the proposed and related schemes with different windows sizes. As expected, the MC scheme has the highest ECC overhead percentage. This is because it uses 28 ECC bits for every word in the frame. However, as the window size increases, the number of ECC bits needed goes down considerably and the H^3 scheme starts to have a higher overhead than the MC scheme. The P^2H scheme with non-optimal diagonal has less ECC overhead than 2D-HPC for smaller window sizes. This is because the proposed scheme uses parity codes for rows and columns, and hamming codes for the diagonals while 2D-HPC uses hamming codes for both rows and columns. For larger window sizes however, the number of ECC bits increases for the non-optimal diagonal hamming code. Summing up, the P^2H optimal and non-optimal techniques have 35.7% and 18.75% lesser ECC overhead as compared to 2D-HPC. Similarly, the H^3 optimal and non-optimal techniques are 32.1% and 23.4%, respectively smaller in overhead than MC for a window size of 32.

C. Ratio between the Error Correction Percentage and the ECC Overhead

This work as mentioned in the introduction, aims to improve the error correction percentage per unit area overhead. Figures 13 and 14 plots the ratio for different numbers of errors injected. A higher ratio implies that the error correction is more efficient by using lesser ECC overhead than a lower ratio one. As the number of errors injected is increased, the ratio decreases since the error correction capability of the schemes reduce drastically. Further, as can been seen from the figures, the proposed schemes in this paper, especially the P^2H scheme does much better than 2D-HPC for both SBUs and MBUs for lesser number of errors (up to ≈ 17 errors in the frame). Moreover, the H^3 scheme has a much lower ratio than all the other schemes for more than 34 errors in a single frame. From this result, we propose using the P^2H scheme for environments with lower SEU conditions, while the H^3 technique would be more effective for high SEU conditions.

TABLE I. HARDWARE RESULTS

	Slices	Utilisation	LUT-FF pairs	Max. Freq.
H^3 module	2324	6%	5694	182 MHz
P^2H module	3703	9%	10393	160 MHz
Frame storage	1792	4%	9694	246 MHz

D. Hardware Implementation

A hardware implementation of the proposed schemes was implemented according to the architecture discussed in Section IV. The error correction and frame storage modules were completely designed in hardware so that computations can be done in parallel. Moreover, the frame and ECC storage was done in registers to make the access of data faster. The implementation was targeted towards the Virtex-6 architecture and the feasibility of the proposed schemes was verified. Note that hardware implementations of the related works were either not available or not possible to implement on the current FPGA architecture used. Table I presents the results of the hardware implementation. Both schemes take up less than 13% of the entire FPGA and run at a frequency of at least 160 MHz. Moreover, the time taken to correct errors was measured for both the proposed schemes. H^3 scheme (optimal) took $20,796$ cycles[1] to correct the errors while the P^2H scheme (optimal) managed to correct the same in $6,445$ cycles[2]. The P^2H scheme performs faster than the H^3 scheme since it only uses parity correction for the rows and columns and hamming for the diagonals.

[1]corresponds to $207.96\mu s$ for ICAP running at 100 MHz
[2]corresponds to $6.445\mu s$ for ICAP running at 100 MHz

VI. Conclusions

Two new error detection/correction schemes are presented. These schemes combined with frame readback scrubbing allow mitigating SBUs and MBUs in SRAM-based FPGAs. A suitable architecture has also been presented for the implementation of the proposed schemes which uses COTS FPGAs. These two schemes are evaluated and compared with the existing related works in terms of ECC overhead and error correction percentage, as well as the ratio between them and their execution time. From the results, we propose using the H^3 scheme for environments with higher SEU conditions and where the error correction percentage is really important. On the other hand, for environments with moderate SEU, P^2H scheme will be more effective as it provides a moderate error correction capability with a lesser ECC overhead.

References

[1] D. Ratter, *FPGAs on Mars*, ser. Journal n50. Xilin Technical Report, xCell Jornal, 2004.

[2] D. White and Xilinx Corporation, *Considerations Surrounding Single Event Effects in FPGAs, ASICs, and Processors*, ser. White Paper WP402, 20012.

[3] I. Koren and C. Krishna, *Fault-tolerant systems*. Morgan Kaufmann, 2007.

[4] 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.

[5] 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 (DFT'07)*, 2007.

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

[7] C. Carmichael, M. Caffrey, and A. Salazar, "Correcting single-event upsets through virtex partial configuration," Xilinx, Tech. Rep., 2000.

[8] J. Heiner, B. Sellers, M. Wirthlin, and J. Kalb, "FPGA partial reconfiguration via configuration scrubbing," in *Field Programmable Logic and Applications (FPL'09)*, 2009.

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

[10] 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.

[11] 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.

[12] 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 (ISIE'10)*, 2010.

[13] M. Berg, "The nasa goddard space flight center radiation effects and analysis group virtex 4 scrubber," *Xilinx Radiation Test Consortium (XRTC) Meeting*, 2007.

[14] Xilinx Corporation, *Virtex FPGA Series Configuration and Readback*, ser. Application Note XAPP138, 2005.

[15] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: an efficient scheme for tolerating double disk failures in RAID architectures," *IEEE Transactions on Computers*, 1995.

[16] C. Huang, L. Xu, and S. Member, "An efficient coding scheme for correcting triple storage node failures," in *4th Usenix Conference on File and Storage Technologies (FAST)*, 2005.

[17] 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.