Scatter Scrubbing: A Method to Reduce SEU Repair Time in FPGA Configuration Memory

Mahsa Mousavi, Hamid Reza Pourshaghaghi, Henk Corporaal
Eindhoven University of Technology
m.mousavi, h.r.pourshaghaghi, h.corporaal@tue.nl

Akash Kumar
Dresden University of Technology
akash.kumar@tu-dresden.de

***Abstract*—SRAM-based FPGAs are widely used in many critical systems in which dependability is an essential factor. However, SRAM-based FPGAs are sensitive to Single Event Upsets (SEUs), especially when they are used in space. Scrubbing is an effective technique to protect FPGA Configuration Memory (CM) against SEUs. One major hurdle in read-back scrubbing techniques is that they suffer from long Mean Time To Repair (MTTR). In this paper, we propose *scatter scrubbing*, a new method that reduces MTTR by exploiting the locality of SEUs sensitive bits in CM. It is based on 1) splitting FPGA CM into several partitions based on how critical the CM bits are for proper operation of the FPGA circuit, and 2) deriving a smart schedule for scrubbing the partitions. Finding an optimal partition and scheduling has non-polynomial complexity; therefore we rely on clever heuristics, especially for the first step. However, for small designs, we developed an accelerated brute-force method giving the optimal solution, which we can use as a reference. The experimental results show, for real FPGA designs, up to 64% reduction in MTTR compared to state-of-the-art techniques.**

***Index Terms*—FPGA, fault tolerance, SEU, scrubbing, configuration memory**

I. Introduction

Many of the modern digital embedded systems, such as used in space and automotive domains, make use of custom off the shelf Static-RAM Field Programmable Gate Arrays (SRAM FPGAs) due to all the advanced provided features, like huge capacity, high performance, relatively low power consumption, and reconfigurability. Reconfigurability is one of the most interesting items, as it offers the opportunity to update or correct the system design even after implementing a final version. The reconfigurability feature is provided by using a large SRAM based Configuration Memory (CM), which stores the circuit bitstream. However, SRAM cells are typically susceptible to Single Event Upsets (SEUs) i.e. transient bit flips in the memory cell state induced by high energetic particle strike. SEUs can potentially corrupt correct operation of any design. In a design where there is no room to accept any significant risk of functional failure, fault tolerance techniques can essentially play a key role and ensure the applicability of SRAM-based FPGAs, especially in radiation harsh environments.

Two well-known fault tolerance techniques widely used are: Modular redundancy (MR) [1] and scrubbing [2]. Modular redundancy is based on replicating modules to detect or even correct an error. Error detection is possible by Dual Modular Redundancy (DMR) where outputs of two identical copies of one circuit are compared. Instantaneous error detection and error correction are achievable by Triple Module Redundancy (TMR) using majority vote of three identical copies of one specific circuit. Still, neither DMR nor TMR could actually clear SEUs from configuration memory (CM). Thus, errors can be potentially accumulated and lead to a non-functioning modular redundant unit. To clean CM from SEUs and overcome error accumulation problem, scrubbing methods are combined with modular redundancy. This paper concentrate on scrubbing.

In the simplest scrubbing method i.e. blind scrubbing, errors are removed from CM by periodically overwriting memory contents with a golden copy of bitstream stored in reliable external memory. Several scrubbing methods are introduced to improve blind scrubbing in term of efficiency. However, in all of the existing scrubbing methods, the Mean Time to Repair (MTTR) the error is proportional to the time required to traverse CM, which is a relatively slow process. For instance, rewriting the entire CM of Zynq-7000 FPGA using 32-bit wide 100 MHz Internal Configuration Access Port (ICAP) takes about 8 ms. Reducing this time leads to improvement in system availability which is critical especially for real-time systems.

Several works have been reported on how to reduce the MTTR in FPGA designs inspired by the idea of partitioning whole design into sub-partitions with zero overlaps [1]. MTTR is reduced by applying scrubbing only on the sub-partition in which error is occurred. G. L. Nazar et al. [3] recently introduced a technique that even further reduces MTTR by starting scrubbing of each sub-partition from a frame where likely a lot of *critical bits* are to be found. CM bits are critical if flipping their value, i.e. an *error* occurred, results in observable different circuit behavior, that is, a circuit *failure*.

This paper proposes a new scrubbing technique, the first one based on *non-sequential* scrubbing of CM frames. It switches between CM frame partitions prioritized based on SEU vulnerability analysis. Some consecutive parts of memory are more probable to result in failure when infected by SEUs. They are, therefore, prioritized to be scrubbed earlier than other parts. Although switching among partitions introduces time overhead due to accessing non-continuous part of CM, taking this overhead into account, our proposed technique shows promising results. This paper major contributions are:

- New non-sequential scrubbing technique called Scatter Scrubbing, that reduces the MTTR based upon exploiting the locality of critical bits. It includes:
  - Heuristic methods for partitioning the CM taking into consideration the locality of critical bits (Section IV-C)
  - A method for optimally scheduling obtained set of partitions (Section IV-A)
- An accelerated brute-force scheme to find the partitions and partition order giving the optimal MTTR (Section IV-B)
- Performance analysis of scatter scrubbing (Section V)

Experimental results show that scatter scrubbing reduces

978-1-7281-2260-1/19/$31.00 ©2019 IEEE

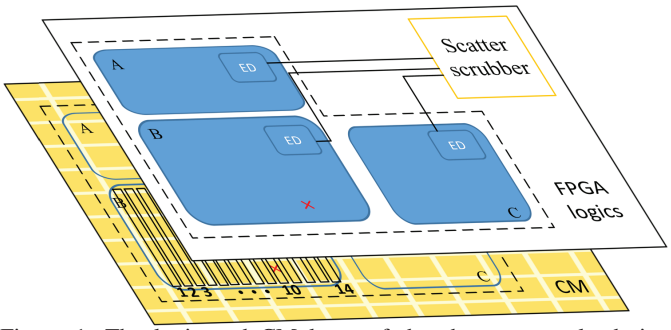

Figure 1: The logic and CM layer of the shown example design consists of 3 partitions A,B and C, implemented on an FPGA. ED logic implements the used error detection mechanism. The red cross in CM frame 10 of partition B indicates an error (SEU / bitflip).

MTTR up to 64% compared to the state-of-the-art techniques.

II. Preliminaries and Related Work

Scatter scrubber targets partitioned circuits [1]. We assume one error detection mechanism, like DMR, for each of those individual partitions. Scrubbing is only applied to CM frames corresponding to the logic in which an error occurs. After detecting an error, scrubbing of frames starts, and stops when the erroneous CM frame has been updated (see Fig.1). Our goal is to reduce the time it takes from detecting an error till repairing it. This time depends on the number of frames required to be traversed until the infected frame is reached.

A. Existing Scrubbing Methods

Scrubbing is a method to remove non-permanent effects of SEUs from CM. In this paper we have investigated most well-known scrubbing methods: 1) Blind scrubbing [4] in which bitstream is periodically rewritten with golden copy stored in radiation hardened memory. There exists some timing overhead in blind scrubbing technique as obviously the entire bitstream is not necessary to be rewritten; 2) read-back scrubbing instead corrects the detected error by rewriting only the infected part [5]- [8]. CM frames are read back from frame with lowest address to highest one in a consecutive manner Scrubbing is halted after repairing the erroneous frame; 3) Shifted scrubbing [3] that is similar to read-back scrubbing except that the starting point of scrubbing is addressed to a frame other than the lowest one in CM [3]. Despite all existing methods, frames are not consecutively scrubbed in scatter scrubbing method. A criticality metric is defined for every bunch of frames and the ones with higher criticality are scrubbed earlier in time. Thus, the probability of repairing an error in a shorter time is increased compared to other methods. Table I compares the aforementioned scrubbing methods in more details.

B. SEU effect in FPGA Configuration Memory

In SRAM-based FPGAs, circuit design is stored in a large memory i.e. CM. The content of CM is accessible for read and write operations frame by frame. Each frame has a linear address in a range from 0 to the total number of FPGA frames. For instance, the XC7Z020 chip contains 7951 frames in which each frame includes 3232 bits. Essential bits are the subset of CM bits which if "they are changed unintentionally by a SEU, it is possible that the function in the FPGA does not behave as intended" [9]. Xilinx tools provide users with essential bits. When a high energetic particle strikes an essential bit, it causes a bit flip in it. However, not all the essential bits leads to failure if they are flipped by SEUs. For example, they may be masked by upcoming logic. The subset of essential bits which leads to a failure when they are flipped, is called critical bits.

Table I: Comparison of various scrubbing methods. CF, CP, EP are Critical Frame, Critical Partition and Erroneous Frame respectively

Scrubbing method	Starting point	Error check	Consecutive	Write back frames
Blind	frame 1	no	yes	all frames
Read-back	frame 1	yes	yes	only EF
Shifted	most CF	yes	yes	only EF
Scattered	most CP	yes	no	only EF

III. Problem Statement

The goal of this paper is to further reduce MTTR. We exploit the fact that the probability of getting a circuit failure due to an error in one of the CM frames is not the same for all frames, i.e. not all frames have the same criticality. Therefore, on average repairing an error takes less time when the frame with more critical bits is scrubbed earlier than other frames. Thus frames are prioritized based on their number of critical bits to be scrubbed. The maximum reduction in MTTR can be achieved when the frames are sorted based on their critical bits in descending order (i.e ordered scrubbing).

Take an example with a simple design including 14 frames with addresses 1 to 14. Assume the number of critical bits of each frame is as shown in Fig.2 (a). Taking these numbers into account, the optimal scrubbing order (S3) of the frames 1 to 14 is 14, 7, 4, 2, 5, 11, 8, 9, 10, 3, 1, 6, 12, 13 respectively (i.e frame 1 is the 14th frame which are scrubbed, etc). The scrubbing order of frames are also shown in Fig 2 in case read-back scrubbing method (S1) and shifted scrubbing (S2) are used. For example, first, frame numbers 3 to 14 are scrubbed consecutively, then frame number 1, and finally frame 2 are scrubbed, in shifted scrubbing. As shown in Fig.2 (b), more reduction in MTTR is achieved when S3 is used.

Above mentioned optimal MTTR reduction is achieved assuming that accessing non-consecutive CM frames costs zero overhead. However, according to Xilinx documentations [10], each non-consecutive access, either read or write, to CM frames introduces some time overhead. For example, for the 7 series FPGAs, each non-consecutive access, read/write, introduces 1.5 times of one frame read/write latency overhead. This time overhead includes the time required for ICAP interface synchronization and initiation (i.e. 60 cycles), in addition to the time required for read/write a dummy frame (i.e. 101 cycles). Applying this time overhead (J) in the MTTR calculation makes the MTTR of ordered scrubbing the worst among all as shown in Fig.2 (b) (i.e S4). Thus, in order to avoid increment in MTTR due to non-consecutive access, the number of jumps over the CM is required to be limited. Taking this into account, we propose *scatter scrubbing* in which the frames are partitioned such that each partition consists of consecutive frames. Thereafter, partitions are ordered to be scrubbed. This frame partitioning and ordering should potentially result in a significant reduction in MTTR. For instance, assume that

Figure 2: a) a sample design with 14 frames, red numbers are the order of traversing frames corresponding to scrubbing method S1-5, b) MTTR of each scrubbing method (units is the time of reading a frame and J is time overhead due to non-continues CM access)

frames in Fig. 2 (a) are clustered into partitions of frames 1-2, 3-9, 10-12, and frames 13-14. In this case, these 4 partitions are scrubbed with the order of 4,2,1,3 respectively (i.e S5 scrubbing method). MTTR is clearly reduced when compared to the cases of applying S1, S3, and S4.

A. Problem Formulation

Consecutive frames of a design, from lowest to highest address, are associated respectively to $f_1, f_2, ..., f_N$, where N is the total number of frames, and the set of all frames is $\mathbb{F} = \{f_i : 1 \leq i \leq N\}$. If the number of critical bits in each frame i.e $h(f_i)$ is given, the *scatter scrubbing problem* is to find partitions and order them such that MTTR is minimized as formulated by:

$$given : \begin{cases} f_i \\ h(f_i) \end{cases} \quad wants : \begin{cases} p_j \\ o(p_j) \end{cases} \quad mimimize\ MTTR \tag{1}$$

where $p_j s,\ 1 \leq j \leq M\ (M \leq N)$, are partitions such that $p_j \subset \mathbb{F}$, $\bigcup_{j=1}^{M} p_j = \mathbb{F}$ and $p_i \cap p_j = \phi\ (i \neq j)$. Note that each partition should include only consecutive frames. In addition, indexing of partitions is defined such that partition with lower index includes frames with lower indexes. Moreover, $o(p_j),\ 1 \leq j \leq M$ stands for the scrubbing order of partition p_j. For example, S5 partitions highlighted by underline in Fig.2 (a) are $p_1 = \{f_1, f_2\}$, $p_2 = \{f_3, f_4, ..., f_9\}$, $p_3 = \{f_{10}, f_{11}, f_{12}\}$, $p_4 = \{f_{13}, f_{14}\}$ and $o(p_1)$, $o(p_2)$, $o(p_3)$, $o(p_4)$ are as 4, 2, 1, 3 respectively. For example, in this case p_1 is the fourth partition which is scrubbed.

The MTTR is defined as the average time required to repair an occurred error, and is calculated as (2):

$$MTTR = \sum_{i=1}^{N} pr(f_i) \times d(f_i) \tag{2}$$

where $pr(f_i) = \frac{h(f_i)}{B}$ (B=total critical bits number) is the probability of failure provided error is occurred in f_i. Further, $d(f_i)$ is required time to traverse all the f_j that have scrubbing order equal and less than f_i, including f_i:

$$d(f_i) = o(f_i) \times T_s + j(f_i) \times J \tag{3}$$

In (3), $o(f_i)$ is scrubbing order of f_i and $j(f_i)$ is number of necessary jumps in CM addresses before reaching f_i. In addition, $T_s = \frac{S}{R}$ is defined as the required time for read/write one frame with S bits and rate of R bits/sec. Last definition in (3), J is time overhead due to non-consecutive access to CM, e.g. J for Xilinx 7-series FPGA is 1.5 T_s.

IV. Scatter Scrubbing

Scatter scrubbing problem is a non-linear discrete optimization problem. According to our knowledge, none of the existing optimization solutions can be used directly to find the optimal solution for scatter scrubbing in finite time. The exhaustive search for finding the optimal solution is not possible in finite time. To overcome complexity, scatter scrubbing problem is split into two separate sub-problems i.e. ordering problem and partitioning problem.

A. Proposed Ordering Method

For the sake of simplicity and without loss of generality, let's assume that all the partitions are known and predefined. The goal with ordering problem is to order these known partitions such that MTTR is minimized:

$$given : \begin{cases} f_i, h(f_i) & wants : o(p_j) \\ p_j,\ 1 \leq j \leq M & mimimize\ MTTR \end{cases} \tag{4}$$

Given (3) and (2) and the knowledge of predefined partitions MTTR can be rewritten as

$$\sum_{i=1}^{N} (pr(f_i) \times ((\sum_{j=1}^{o(p(f_i))-1} N(p^j)) + I(f_i) + o(p(f_i)) \times J)) \tag{5}$$

where p^j is partition corresponding to order j, $p(f_i)$ is the partition that f_i belongs to, $N(p^j)$ is the number of frames in the partition with order j and $I(f_i)$ is the difference between the index of f_i and the first frame in $p(f_i)$. Actually, $d(f_i)$ is rewritten based on the information obtained from pre-known partitions. By expanding (5), MTTR will be:

$$\begin{aligned} MTTR = &\sum_{i=1}^{N} \sum_{j=1}^{o(p(f_i))-1} (N(p^j) \times pr(f_i)) + \sum_{i=1}^{N} (pr(f_i) \times I(f_i)) \\ &+ \sum_{i=1}^{N} (pr(f_i)(o(p(f_i) \times J)) \end{aligned} \tag{6}$$

Note that the second term in (6) is always a constant value, independent from partition order, and it can be eliminated in the process of minimization. For simplicity, we can now define a simplified version of MTTR by combining first and last terms in (6) while sigma indices are changed in the range of partitions:

$$MTTR' = \sum_{l=1}^{M} (pr(p^l) \times (\gamma(p^l) + K \times l)) \tag{7}$$

where $pr(p^l)$ is the summation of $pr(f_i)$ for all frames in p^l i.e. $\sum_{f_i \in p^l} pr(f_i)$, $\gamma(p^l)$ is the number of frames are traversed until p^l is reached i.e. $\sum_{j=1}^{l-1} N(p^j)$ and $K = \frac{J}{T_s}$. It is important to note that minimizing $MTTR'$ is the same as minimizing MTTR. In order to find the minimum value of $MTTR'$, the Theorem 1 is exploited.

Theorem 1. *$MTTR'$ in (7) is minimized when partitions are ordered such that $\frac{pr(p^l)}{N(p^l)+K}$, for $l = 1$ to M form a decreasing sequence.*

Proof. Let $\mathbb{P} = p^1, p^2, ..., p^j, p^{j+1}, ..., p^M$ be an arbitrary order and $\mathbb{P}'$ be the same order except p^j and p^{j+1}, $(1 \leq j \leq M)$ is interchanged. Thus, $pr'(p'^l)$=$pr(p^l)$, $\gamma'(p'^l)$=$\gamma(p^l)$ for $l \neq j, j+1$; $pr'(p'^i)$=$pr(p^{j+1})$, $pr'(p'^{j+1})$=$pr(p^j)$, $\gamma'(p'^j) = \gamma(p^j)$ and $\gamma'(p'^{j+1}) = \gamma(p^j) + N(p^{j+1})$. Referring to Smith

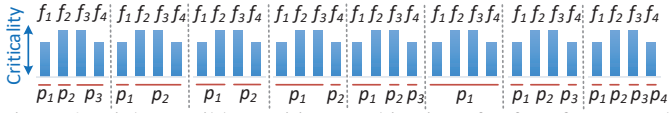

Figure 3: Eight possible partition combinations for four frames, red line indicates frames grouped to a partition

Rule [11], $\mathbb{P}$ minimize the $MTTR'$ value, if and only if for any $\mathbb{P}'$:

$$\sum_{l=1}^{M}(pr(p^l)\times(\gamma(p^l)+K\times l)) \leq \sum_{l=1}^{M}(pr'(p'^l)\times(\gamma'(p'^l)+K\times l)) \tag{8}$$

where by expanding (8), using above equalities and $\gamma(p^{j+1}) = \gamma(p^j) + N(p^j)$, it is concluded that $\mathbb{P}$ minimize the $MTTR'$ value if and only if for any l, $(1 \leq l \leq M)$, (9) is satisfied:

$$\frac{pr(p^{l+1})}{N(p^{l+1})+K} \leq \frac{pr(p^l)}{N(p^l)+K} \tag{9}$$

Thus, MTTR is minimum when partitions with higher $\frac{pr(p)}{N(p)+K}$ are ordered to be scrubbed earlier. For instance, consider the example in Fig.2, the given partitions are $p_1 = \{f_1, f_2\}$, $p_2 = \{f_3, f_4, ..., f_9\}$, $p_3 = \{f_{10}, f_{11}, f_{12}\}$, $p_4 = \{f_{13}, f_{14}\}$ and $\frac{pr(p)}{N(p)+K}$ for each partition is 2, 5.64, 8.8, 1.7 respectively. Thus, based on Theorem 1, the minimum MTTR is achieved when partitions 1 to 4 are scrubbed with the order of 3, 2, 1, 4 respectively. □

B. Accelerated Brute-force

In brute-force all possible partition combinations and, for each combination, all possible orderings are searched to find out a) right partition combination and b) corresponding scrubbing orders, in which MTTR is minimized. Obviously, this is an extreme search algorithm for large designs.To understand how a brute-force search works in this framework, assume that a given design consists of four frames (f_1, f_2, f_3, f_4) with various criticality as shown in Fig.3. There are eight different partition combinations derived from grouping frames, as also shown in Fig.3. For each of the partition combinations, there exists different scrubber ordering. For instance, in the first partition combination from the left side in Fig. 3, the orders of p_1, p_2, p_3 can be respectively 1 2 3, 1 3 2, 2 1 3, 2 3 1, 3 1 2 and 3 2 1. Each of these possible orderings results in one different final MTTR value. The minimum value among all obtained MTTR for a partition combination is selected. This value still needs to be compared with other obtained MTTR values coming from seven other partition combinations. This algorithm can be extended to any other larger design, which could be very compute intensive.

Instead of a normal brute-force algorithm, we propose an accelerated brute-force algorithm based on Theorem 1, to avoid searching for all possible orders of a partition combination. Thus, in the example shown in Fig.3, for each of the eight partition combinations, Theorem 1 determines the only ordering which result in the minimum MTTR. To a larger extent, for each combination with M partitions, computation complexity reduces with a factor of $\frac{1}{M!}$.

C. Proposed Partitioning Method

In Accelerated brute-force, $\sum_{M=1}^{N-1}\binom{N-1}{M}$ various partition combinations are required to be searched, which in terms of computation time is only feasible for a small number of frames. We, therefore, propose a heuristic partitioning method which evaluates only a subset of all possible partition combinations in order to find a solution close to minimum MTTR. Taking into account the features of optimal solutions obtained by accelerated brute-force, the proposed heuristic includes two different phases. First, an initial partition combination is generated. Second, consecutive partitions are merged when merging them leads to a further reduction in MTTR.

1) Generating Initial Partition Combination Using Max-TH Algorithm : The critical bits number of neighboring frames are usually similar to each other, according to our observation. Thus, the strategy of Max-TH algorithm is to group frame with higher critical bits and its surrounding frames with similar critical bits number and form one partition. As shown in algorithm 1, among all frames, the frame which has maximum critical bits number (i.e. f_j) is selected and forms one new partition. Then, its adjacent frames are added to this partition one by one till their critical bits number is equal or higher than a predefined threshold i.e. $\alpha h(f_j)$ (α is constant, $0 < \alpha \leq 1$). Then second new partition is formed by finding the next frame with maximum critical bits number among the rest of unpartitioned frames. Similar to the first partition, the second partition includes some consecutive neighboring frames that have critical bits number more or equal to the threshold. This procedure is continued until all frames are partitioned.

ALGORITHM 1: Max-TH algorithm

```
INPUT: 𝔽 = {f_i : 1 ≤ i ≤ N}, h(f_i), α
OUTPUT: p_j, 1 ≤ j ≤ M
A = 𝔽 ;
while A ≠ φ do
    Find f_j in A with maximum h(f_j);
    p = Create empty partition;
    Add f_j to p; A = A − f_j; k = j + 1;
    while h(f_k) ≥ αh(f_j) and k ≤ N do
        Add f_k to p; A = A − f_k; k++;
    end
    k = j − 1;
    while h(f_k) ≥ αh(f_j) and k ≥ 1 do
        Add f_k to p; A = A − f_k; k−−;
    end
    Store p;
end
```

2) Merging Partitions : By comparing the initial partitions derived from Algorithm 1 and partitions obtained from accelerated brute-force, it is observed that merging three adjacent partitions i.e. p_j, p_{j+1}, p_{j+2} may result in some additional reduction in MTTR if they have following characterization: 1) maximum critical bits of frames in middle partition p_{j+1} (i.e. $Max(p_{j+1})$) is less than each of two other partitions, 2) the ratio of maximum critical bits number of p_j to p_{j+2} is either less than or equal to α, or larger than or equal to $\frac{1}{\alpha}$. The whole partitions are traversed once to find partitions with aforementioned property as shown in algorithm 2. Then if merging them leads to an additional reduction in MTTR, they are merged. Note that MTTR is calculated for the obtained partition combination while partitions are ordered based on the proposed ordering method introduced in Sec.IV-A.

After that, the whole partitions are traversed once again except this time two consecutive partitions are merged if only

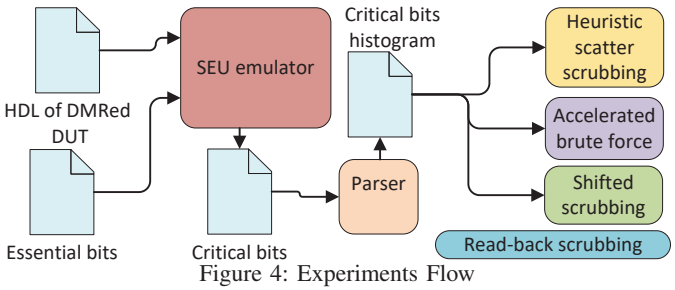

Figure 4: Experiments Flow

Table II: DMR implementation characterizations of DUTs

DUT	CM Frames	LUTs	FFs	Slices
Adder16	35	452	1073	344
FFT4	159	753	2042	1291
Ripple carry	35	290	556	233
Brent kung	35	566	1072	415
FFT8	271	1356	3881	2196
FIR8	71	520	1053	355
ITC99b04	107	393	590	362
Kinematic	505	5069	2026	1598
Multiplier 32	414	4096	1072	1352
Kugge stone	70	715	1073	454

it leads to a reduction in MTTR. This process is continued until no reduction is achieved during a whole trace through the partitions.

ALGORITHM 2: Merging algorithm

INPUT: P=Max-Th($\mathbb{F}$); min=MTTR(P);
OUTPUT: $p_j,\ 1 \leq j \leq M$ which leads to minimum MTTR
// Merge 3 partitions :
for $p_j \in P,\ j : 1 \rightarrow M$ **do**
  if $Max(p_{j+1}) \leq Max(p_j),\ Max(p_{j+2})$ *and* $\frac{Max(p_{j+2})}{Max(p_j)} \leq \alpha$ *or* $\frac{Max(p_{j+2})}{Max(p_j)} \geq \frac{1}{\alpha}$ **then**
    if *MTTR after merge*$(p_j,\ p_{j+1}, p_{j+2}) <$ *min* **then**
      merge $p_j,\ p_{j+1},\ p_{j+2}$; update P; min=MTTR(P);
    end
  end
end
// Merge 2 partitions :
while *MTTR is reduced* **do**
  for $p_j \in P,\ j : 1 \rightarrow M$ **do**
    if *MTTR after merge*$(p_j,\ p_{j+1}) <$ *min* **then**
      merge $p_j,\ p_{j+1}$; update P; min=MTTR(P);
    end
  end
end

Final partition combination is investigated for adjacent partitions that have consecutive order. Extra jumping overhead in between two adjacent partitions that have consecutive order can be eliminated, because the scrubber does not have to jump to different addresses. These two partitions are naturally combined to one partition.

V. Experimental Results

In our experiments, Xilinx FPGAs is used as testing benchmark. The essential bits are provided by Xilinx Vivado tool. However, an injection-based emulator is required in order to extract the critical bits and distribution of them within the frames of a design. The emulator is designed based on the work proposed in [12], implemented on a Zynq board. The emulator consists of a controller executed on an ARM processor and Single Event Mitigation (SEM) IP core implemented on FPGA (i.e. XC7Z020). SEM IP is activated by the controller in order to inject an error into a specific configuration bit. Then, the critical bit is recognized by comparing the output of Design Under Test (DUT) after error injection with the expected output. Injection is accomplished in which every bit of design under test is flipped and for each bit, several random inputs are fed into the design.

Fig.4 depicts general flow of our experimental process. In an initial step, the DUT is equipped with an error detection technique which is in our experiments DMR. In the next step, the DMRed version of DUT is integrated into the emulator. The DMRed DUT is connected into the controller through the AXI interface. Then, the controller is configured to inject SEUs only into frames of DMRed DUT. The critical bits list generated by emulator is fed into a parser implemented with C++ which produces a list of critical bits number that belongs to the design frames i.e. critical bits histograms. The critical bits histograms are used to generate and schedule partitions. Finally the MTTR of proposed methods including Both heuristic and brute-force and existing scrubbing including read-back and shifted scrubbing are calculated for a fair comparison.

A set of 15 benchmarks including small synthetic designs and real designs are used as DUT. Synthetic designs consist of small number of frames (between 15 to 20) so that accelerated brute-force is applicable for them in a feasible time. The real benchmarks size varies from tens to several hundreds of CM frames. Table.II provides characterizations of DMR implementation of real benchmarks.

Fig.5 shows the obtained MTTR for each DUTs when each of scrubbing methods listed in the Fig.5 is used. As expected, using read-back scrubbing leads to maximum MTTR among the other methods, because scrubbing always starts from the frame with the lowest address without taking the critical bits histogram into account. Our proposed heuristic scatter scrubbing achieves the most reduction in MTTR compared to the other methods. Note, our MTTR improvements range from a few microseconds till several hundreds. This is very relevant, considering circuit cycles times are in the range of nanoseconds. Using heuristic scatter scrubbing, on average 40% and 25% reduction in MTTR is achieved compared to read-back and shifted scrubbing respectively. The amount of reduction percentage in MTTR due to using heuristic scatter scrubbing compared to shifted scrubbing is varied for different applications as shown in Fig.5. The more critical bits locality exists within the application frames, the more MTTR reduction can be obtained using our method. On the other hand, not much reduction is achievable with our method for the application with uniform critical bits distribution. For example, about 64% reduction is obtained for Fast Fourier Transform (FFT) due to the existence of many consecutive frames with similar critical bits, mainly because of the modular design of FFT. However, since Brent Kung has interwoven and uniform circuit, MTTR is not improved much using our method. MTTR is shown in Fig.5 for when scatter scrubbing is used and jump overhead is assumed to be zero. It provides a good metric that indicates maximum potential reduction is achieved for

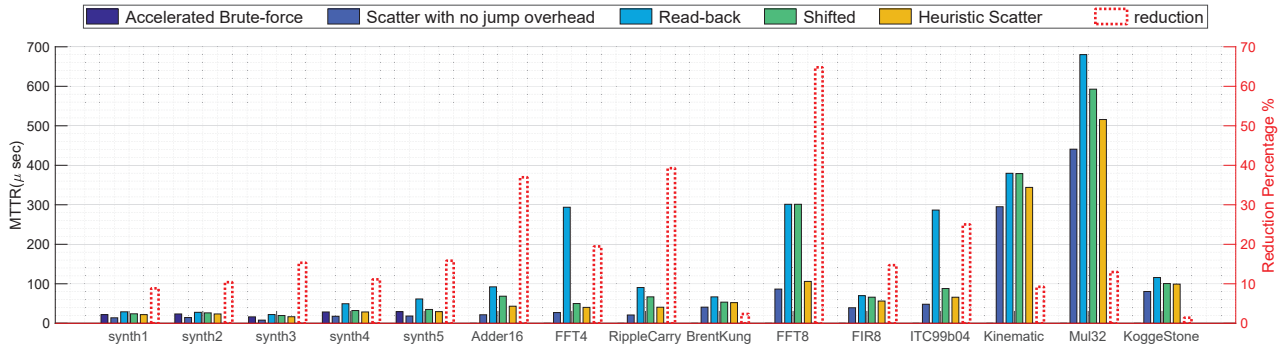

Figure 5: MTTR Comparison for various scrubbing methods, red dash line bars are MTTR reduction percentage of heuristic scatter scrubbing ce is only shown for synthetic DUTs

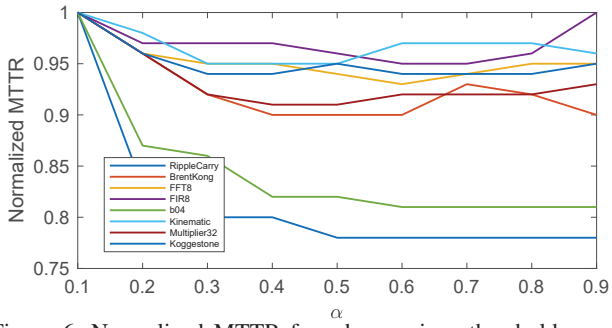

Figure 6: Normalized MTTR for when various thresholds are used to generate initial solution in heuristic scatter scrubbing

a specific critical bit distribution by using scatter scrubbing. The less difference between conventional scrubber MTTR and the heuristic scatter scrubber MTTR with zero jump overhead assumption, the less reduction is expected to be achieved by heuristic scatter scrubber with non-zero jump overhead.

As discussed in Sec.IV, heuristic scatter scrubbing results in a near-optimal solution. However, for small synthetic circuits, optimal MTTR is achievable using accelerated brute-force method as shown in Fig.5. The results show that the MTTR obtained using heuristic scatter scrubbing is equal to optimal MTTR obtained by accelerated brute-force while the execution time of the former is much less than the latter.

Effect of choosing different thresholds in order to generate an initial solution using Max-TH algorithm is evaluated when α is varied between zero and one. Fig.6 shows obtained normalized MTTR using various α. The results show that, minimum MTTR is obtained when $0.4 \leq \alpha \leq 0.6$. However, considering α out of this range does not lead to more than 25% increment compared to minimum MTTR is achieved when $0.4 \leq \alpha \leq 0.6$. This is because even the MTTR of the initial solution becomes worse by taking $\alpha < 0.4$ *or* $\alpha > 0.6$, but MTTR is reduced eventually through algorithm 2.

The scatter scrubber hardware is the same as the state-of-the-art *shifted scrubber*, except additional memory is required to store the starting frame and size of each partition, and the ability to switch among partitions. Since the number of partitions is in the order of 100, not much extra memory is needed.

VI. Conclusion

A new scrubbing technique is proposed in this paper. Our proposed scrubber is initiated only when an error is detected using DMR. It is stopped directly after repairing the infected bit. Our contribution is to minimize the time between detecting an error and repairing it. Taking the critical bits histogram into account, we proposed methods for 1) partitioning, and for 2) scheduling the frames. In a given FPGA design, configuration frames are partitioned by grouping number of consecutive frames including approximately similar critical bits. Those partitions are then scheduled for scrubbing based on proposed ordering method such that at the end of scrubbing scheme MTTR is minimized. Experimental results show that scatter scrubber reduces MTTR on average 40% and 25%, with a maximum of 86% and 64%, compared to existing read-back scrubbing and shifted scrubbing, respectively.

References

[1] C. Bolchini, A. Miele, and C. Sandionigi, "A Novel Design Methodology for Implementing Reliability-Aware Systems on SRAM-based FPGAs," in IEEE Trans. on Computers, 2011.

[2] R. Santos, Sh. Venkataraman, A. Kumar, "Generic scrubbing-based architecture for custom error correction algorithms," International Symposium on Rapid System Prototyping (RSP), 2015.

[3] G. Nazar, L. Santos, and L. Carro, "Accelerated FPGA Repair Through Shifted Scrubbing", in IEEE International Conference on Field Programmable Logic and Applications (FPL'13) , 2013.

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

[5] 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.

[6] 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.

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

[8] S. Venkataraman, R. Santos, S. Maheshwari, and A. Kumar, "Multi-directional error correction schemes for SRAM-based FPGAs," in Field Programmable Logic and Applications (FPL), 2014 24th International Conference on , 2014.

[9] Xilinx, " Soft Error Mitigation Controller v4.1 Product Guide ", www.xilinx.com, 2018.

[10] Xilinx, " 7 Series FPGAs Configuration User Guide ", www.xilinx.com, 2018.

[11] Smith, W., " Various optimizers for single-stage production" , Naval Research Logistics, 1956.

[12] M. Mousavi, H. Pourshaghaghi, M. Tahghighi, R. Jordans, H. Corporaal, "A generic methodology to compute design sensitivity to SEU in SRAM-based FPGA", Proceedings of the 21st Euromicro Conference on Digital System Desgin, 2018.