2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines

Improving the Effectiveness of TMR Designs on FPGAs with SEU-Aware Incremental Placement

Matthew Cannon, Andrew Keller and Michael Wirthlin
NSF Center for High-Performance Reconfigurable Computing (CHREC)
Brigham Young University
Provo, UT USA
{matthew.cannon, andrewmkeller, wirthlin}@byu.edu

***Abstract*—TMR combined with configuration scrubbing is an effective technique to mitigate against radiation-induced CRAM upsets on SRAM-based FPGAs. However, its effectiveness is limited by low-level common mode failures due to the physical mapping of a design to the FPGA device. This paper describes how common mode failures are introduced during the implementation process and introduces an approach for resolving them through a custom incremental placement tool for Xilinx 7-Series FPGAs. Multiple designs across multiple generations of devices are shown to be sensitive to common mode failures. Applying the incremental placement technique yields an improvement of 10,721x over an unmitigated design through fault-injection testing. Radiation testing is then performed to show that the MTTF of this technique is 91,500 days in GEO orbit, a 367x improvement over the unmitigated design and a 5x improvement over baseline TMR.**

***Keywords*-Field Programmable Gate Array (FPGA), Triple Modular Redundancy (TMR), reliability, common mode failures, Single Event Effect (SEE), Single Event Upset (SEU), fault-injection, radiation testing**

I. Introduction

SRAM Field Programmable Gate Arrays (FPGAs) are increasingly used for high-reliable and safety-critical applications. SRAM FPGAs, however, are sensitive to ionizing radiation and can experience single-event upsets (SEUs) within the internal state of the FPGA, including the configuration RAM (CRAM) [1]. FPGAs used in high radiation environments, such as space or high-energy physics environments, must consider the effects of SEUs on the behavior of the device. Further, safety critical and high-reliability terrestrial applications must consider the effects of terrestrial neutrons by anticipating, and possibly providing mitigation for SEUs within the FPGA device.

In an FPGA, the CRAM bits control the functionality of the device, such as the LUT contents or routing information. An SEU in one of these bits could alter the behavior of the device. For example, if an SEU occurred within the LUT contents, it would change the logic function the LUT was implementing. When the design next uses this bit, the wrong value will be propagated out of the LUT and could cause an incorrect computation or circuit failure.

A popular technique to mitigate against the effects of ionizing radiation is triple modular redundancy (TMR), which triplicates the circuit and places a majority voter on the outputs. When implemented on an FPGA, the use of TMR can successfully mask failures caused by radiation. TMR has been shown to provide significant improvements in reliability and SEU sensitivity through CRAM fault injection, radiation testing, and even with FPGAs deployed in space. The effectiveness of TMR is limited, however, due to the presence of common mode failures (CMF) in the TMR implementation. In spite of TMR, some single CRAM bits can overcome the spatial redundancy of TMR by impacting more than one TMR domain. The presence of CMFs significantly reduces the effectiveness of TMR by placing an upper limit on the total achievable reliability improvement.

This paper presents a technique for improving the effectiveness of TMR implemented on an FPGA by identifying and removing CMFs, using an incremental placement technique. This new mitigation strategy was compared against a design with no mitigation and baseline TMR to measure the increase in the mean time to failure (MTTF). Testing through fault-injection also shows a significant improvement of $10,721\times$ over no mitigation. Through radiation testing this technique increased the MTTF from $19,000$ days to $91,500$ days, an improvement of almost $5\times$ over TMR and $367\times$ over no mitigation.

The rest of this paper will continue as follows. A background section will provide a brief overview of TMR and configuration scrubbing. A motivation section will state the need for and importance of the proposed tool. Previous work will then be mentioned. The theory behind CMF and how to remove them will be proposed, followed by specific algorithms to perform this task. Finally results from both fault injection and radiation testing will be shown.

II. Background

TMR is an SEU mitigation technique that uses three redundant copies of a module to mask failures. When a module (i.e., the circuit to be protected by TMR) is triplicated, three separate domains are created: TMR_0, TMR_1, and TMR_2, as shown in Figure 1. All three domains are driven by the same input stimulus and under normal operating conditions should yield identical outputs. If one of the domains becomes corrupted, its outputs may not match those of the other two domains. An erroneous output is masked by voting on the outputs from each domain so that only the majority vote is propagated. For example, if an SEU occurred within a LUT, the error would be propagated to the majority voter. The other two domains would be unaffected, so the majority voter would propagate the correct value, instead of the erroneous one produced by the compromised LUT. Voters can be placed throughout a module to synchronize internal signals between domains and increase reliability (often referred to as partitioning). The voting mechanism is often triplicated as well, which prevents the introduction of single-point failures and allows a voter to fail without compromising the integrity of TMR. TMR is able to mask any error that is limited to a single domain between voter insertion points. There are many variations of implementing TMR [2], [3].

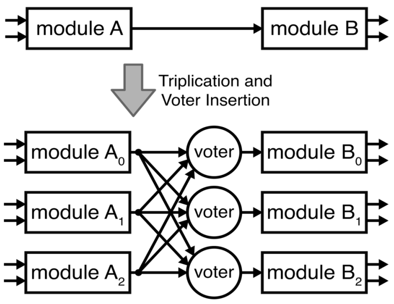

Fig. 1: Triple Modular Redundancy

Configuration memory repair is often coupled with TMR to prevent the accumulation of errors that would break TMR and is often implemented with configuration scrubbing on FPGAs. Configuration scrubbing is usually performed by partially reconfiguring the device with the original bitstream to "scrub" incorrect values [4], [5], [6]. Repair is also needed for the state of the circuit. If the design state becomes corrupted (e.g., counters, state machines, status registers), there needs to be a method to clear the error. Some errors will naturally flush out of the design (i.e., the state is not used in next state logic), or can be manually flushed out of the design on reset. To allow self synchronization, voters need to be placed along feedback paths throughout the design [7]. Scrubbing can even be implemented on BRAM by reading the ECC and correcting any errors, if present [8].

Coupling TMR with repair mechanisms significantly improves the MTTF of the design. TMR by itself improves the reliability of a design early on in its operation, but over time, as errors are allowed to accumulate in a TMR'd design, its reliability can actually become worse than that of the design with no TMR at all [9]. This is because TMR will increase the circuit size, effectively creating a bigger target to be hit. This leads to the TMR system having a lower MTTF than the unmitigated design. Thus, it is essential to include a repair mechanism with TMR if higher reliability is desired over long operating times.

While there are many ways to apply TMR, applying it manually through hardware description language is error prone and synthesis tools are likely to remove redundancy through optimizations, so automated approaches are preferred. The approach used in this paper is to apply TMR to the netlist of a design after logic synthesis (see Figure 2). Once the netlist has been modified for TMR, the updated netlist is then used in implementation. The tool used in this work is described in [10].

TABLE I: TMR Improvement

Design	Sensitivity (Bits)	Sensitivity Reduction	Impr.	MTTF - GEO (days)
Unmitigated	258,440	N/A	1.0×	255
TMR only	63,813	194,627	4.0×	1,031
TMR/Scrubbing	5,038	58,775	51.3×	13,058
TMR/Scrubbing & CMF Removal	$\sim$ 500	4,538	517×	131,579

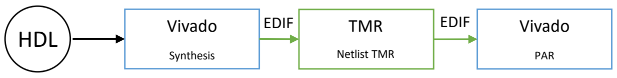

Fig. 2: Netlist-based TMR flow

III. Motivation

Based on well established reliability models [9], the MTTF of a TMR system with repair should be very high as the repair rate approaches infinity (and is only limited by other events such as functional interrupts). In other words, if a fault affects only one TMR domain and it is repaired before a different domain fails, then there is no limit to the reliability improvement provided by TMR, when considering only single bit upsets. While infinite repair rate is impossible to achieve, high repair to failure rates are possible. For an Artix-7 XC7A200T part with a scrub rate of 2.5 seconds (conservative estimate), the repair rate is $38,422\times$ faster than the CRAM upset rate in GEO orbit [11].

In practice, implementing single-device TMR with repair has not yielded near infinite improvement in reliability [12], [13]. During fault-injection [14], the upset rate and repair rates are carefully controlled by only allowing one fault to exist in the design at a time. In one experiment [13], a 50x improvement was demonstrated in reliability through fault injection of a LEON3 processor with feedback TMR and configuration scrubbing compared to the design without TMR. Unlike the reliability model suggests, applying TMR with repair still yielded failures, proving the existence of CMF bits. Additional mitigation factors, on top of TMR with repair, are needed to remove such bits from the design.

Previous studies have concluded that CMFs are caused by the low-level implementation of the TMR circuit on the FPGA and depend on subtle FPGA architecture-specific features of an FPGA family [15]. Some individual CRAM cells may impact the behavior of several logic resources and have an effect that spans resources that may be in more than one TMR domain.

Removing CMF bits can yield significant reliability improvements. As shown in Table I, TMR can improve the MTTF of a design by $4\times$ (MTTF calculated for GEO orbit) [13], removing many sensitive bits. Adding configuration scrubbing to the technique improve the MTTF by almost $50\times$ over the unmitigated version. The final row of the table shows the improvement we hope to achieve with this work. Even though this proposed technique will only protect a fraction of the bits that TMR protects (a few thousand compared to a few hundred thousand) this will provide a significant reliability improvement, which we predict could be greater than $10\times$ better than TMR with scrubbing and $500\times$ better than no mitigation.

IV. Previous Work

In [16], the authors performed an in depth study on domain crossing errors (DCE) on Virtex-II devices. DCEs are considered to be multi-cell upset (MCU) events which trigger multiple domain errors. The authors test a variety of circuits and report on interesting characteristics of the events when the circuits fail as well as the probability of errors. While related to this work, and certainly complementary, it is fundamentally different in that the authors are primarily concerned with multiple upsets, rather than single bits which cause multiple domain failures. This work showed that a majority of errors occur within the configurable routing blocks (CLBs) routing structure.

There have been a number of studies to address the problems of CMFs in TMR design. A reliability-oriented place and route algorithm (RoRA) to address the bits that are not covered through advanced partitioning is proposed in [17], [18]. In this study, the authors identify CMF as routing faults that can lead to three possibilities: a short between two nets, an open (disconnect) of two nets and a short and an open on two nets. Using this information, they propose the RoRA algorithm as a separate design flow from the typical Xilinx PAR flow. Their router will introduce "forbidden" vertices (wires in the device) to prevent nets in other domains from using them. Their algorithm produces a more reliable circuit, but takes a performance hit and does not completely mitigate against all failures. This method is similar to the technique we will propose, in that we seek to make alterations to the low level implementation.

It is worth mentioning that this problem can be solved in other ways than addressing CMF directly. It is possible to apply more advanced techniques, such as Quintuple Modular Redundancy (QMR) to reduce the issue [19]. In QMR the module is replicated 5 times (opposed to 3 in TMR), which means 3 domains have to simultaneously fail for QMR to fail, meaning that CMF will not occur if only 2 domains fail. However, QMR is usually not preferred as there is a high resource penalty, which may or may not be acceptable depending on the particular mission.

Our work expands on previous work in a number of ways. First our work identifies a new cause of CMF in TMR and we present on the background for the root cause of routing CMF. Second, our work shows a significant increase in reliability (in terms of sensitive bits and percent sensitivity) while also allowing fault-injection to target the entire design. We also believe our technique to be less "invasive" than previous techniques. It only requires a small change in the placement and allows the tools to route the design (without allowing them to introduce more CMF).

V. CMFs in Xilinx 7-Series FPGAs

Through fault-injection, we have found that the majority of CMF bits affect the routing in a CLB switchbox (fault-injection results can be found in Section VII). The standard design tools (Vivado) enforce DRC checks which prevents any shorts in the golden bitstream. However, an SEU can introduce shorts by changing the logical value of a bit. To understand how this is possible, it is helpful to understand the low-level details of how a routing mux operates.

Routing is performed on an FPGA by setting a series of programmable interconnect points (PIPs) from the source to sink nodes. A PIP acts as a switch between a source and sink wire on the device. When "on", the source and sink wires are connected and when "off", the two wires are disconnected. A collection of PIPs that drive a wire form a routing mux (also referred to as a PIP junction in Vivado). A routing mux is programmed by setting a row/column bit, as shown in a Xilinx patent [20] (see Figure 3). For convenience, the routing mux has been redrawn in a form showing the row/column structure in Figure 4, which also shows how a sample configuration would be programmed in the mux. When a PIP is turned "on" by the configuration, it is programmed by setting the corresponding row/column in the routing mux while the other row/column bits are unset (not active). In a valid design, only one PIP is turned "on" in the routing mux (otherwise shorts would be present). While the other PIPs are turned "off", some of the source wires on those PIPs are driving other nets in the design while other source wires are unused. Adjacent to the logic tiles (CLBs, BRAMs, DSPs, etc.) are routing switchboxes that contain many of these muxes that either route to the adjacent logic tiles, or other switch boxes.

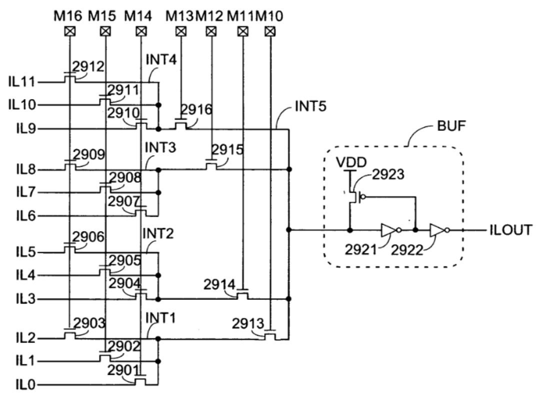

Fig. 3: Routing mux from Xilinx Patent

To understand the failure mechanism, it is necessary to define certain short configurations. When two nets are shorted, it is likely that only one of the nets is comprised, or in other words, that one net drives the other net. In order to describe these situations we use the following terminology for shorted nets:

- **win-win** - both nets retain their correct value. This can happen if both nets are driving the same value or if the short is not strong enough for one net to drive the other.

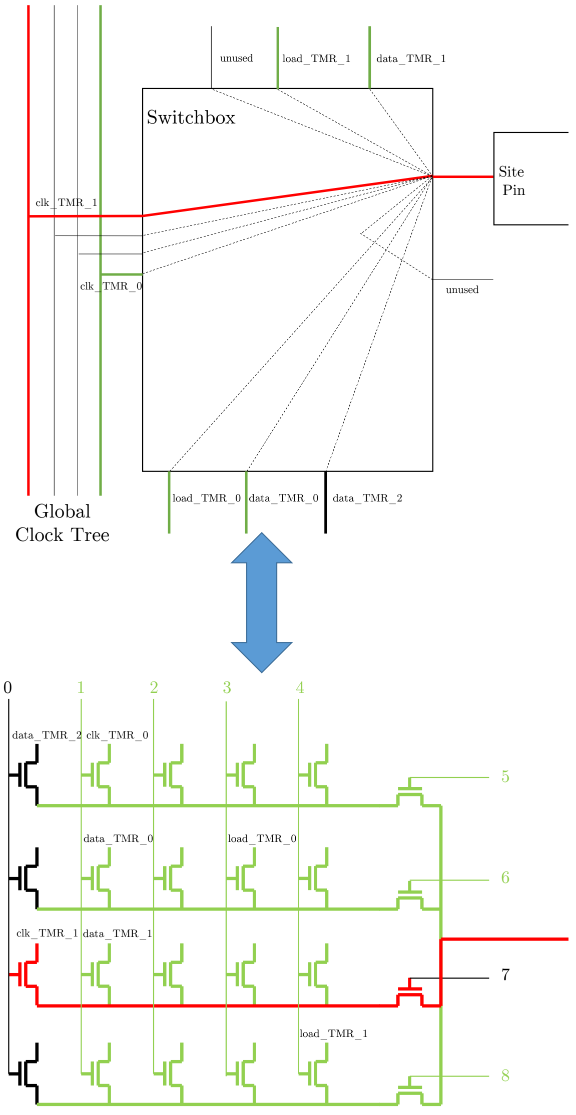

Fig. 4: Example configuration of a routing mux

- **win-lose** - the most likely scenario. This is the case where one net wins over the other net and drives it (i.e. both nets go to VDD or GND).
- **lose-lose** - both nets lose, i.e. they are both driven to the incorrect value.

When a routing configuration bit is upset, two things can happen: a PIP or multiple PIPs are turned "on" or a PIP is turned "off". If an upset occurs in a set column or row bit, then a PIP is turned "off", which will disconnect a net in the design. Since only *one* net is affected, only one domain will break and TMR will not fail. Thus, opens will not cause CMF.

When an upset occurs in an unset column or row bit, one PIP or multiple PIPs will be turned "on". When a second row bit becomes set, it will turn "on" one PIP in the mux and create a short with the row that was already set in the mux. As an example of this case, consider the configuration shown in Figure 4 where row 7 is the set bit. If row 5 is activated, it would short the nets clk_TMR_1 and data_TMR_2 together because column 0 is set. If a lose-lose scenario occurs, then this could cause TMR to fail (if these nets drive cells that are in the same partition). However, we have yet to observe such a scenario through testing and we will instead assume that a win-lose scenario is likely occurring. Note that in a win-lose scenario, TMR would not fail since only one of the nets would be compromised (and only one domain).

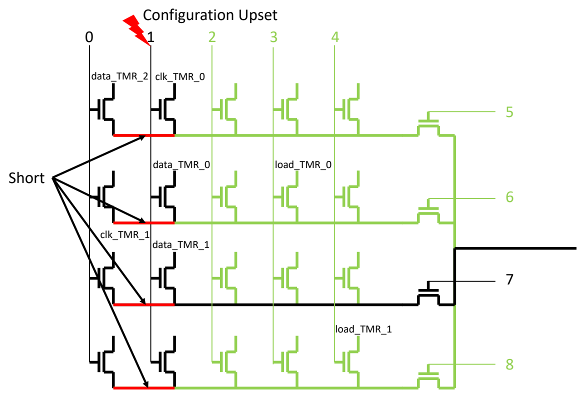

Fig. 5: Example of multiple shorts in a routing mux

Now we will consider what happens when a second column bit is set. In this scenario, multiple PIPs are turned "on" which can lead to multiple shorts, as shown in Figure 5. When a second column bit becomes set, it can create multiple shorts between multiple nets. In Figure 5 column 0 is set. If the column 1 bit experienced an SEU and became set, then this would short the nets clk_TMR_1 and data_TMR_1 together, but would also short the nets clk_TMR_0 and data_TMR_2 together. In a win-lose situation, two domains can be simultaneously affected and break TMR. This would happen if both the clk_TMR_1 and clk_TMR_0 nets failed (assuming that they drove cells belonging to the same partition downstream). We have observed these situations to be the root cause of routing CMF.

Furthermore, through fault injection, we have only observed this CMF case to affect clock nets.[1] Instead of considering all nets, only routing muxes with clock nets need to be considered. We have several hypotheses for why this occurs only on clock nets (but we have yet to verify any of these possibilities). First, clock nets have a higher fanout and when disrupted will affect a higher number of cells. When multiple clocks short together (or with other nets) there is a greater chance for CMF because more cells can fail. Second, clock nets are more sensitive to slight disruptions. If the short puts enough load on the net, it can negatively affect the timing to that cell leading to failure. Even if the clock still drives the correct value, the signal may arrive later than is acceptable. Third, there is some unknown

[1]This might affect other nets, but is rare enough that we have not yet observed it through our experimentation of 8,000,000 randomly injected bits across multiple designs.

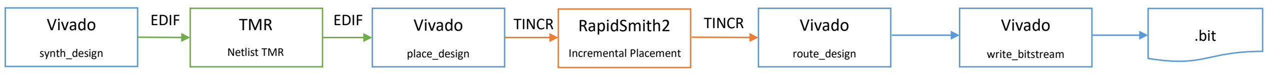

Fig. 6: Design flow for removing CMF from a TMR design.

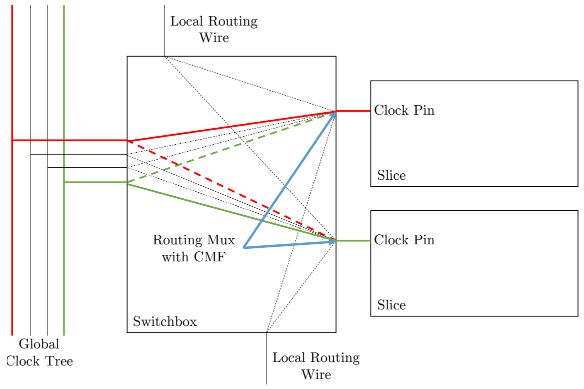

Fig. 7: Local clock routing. Not all wires/muxes are shown.

architectural feature associated with clock specific wires that makes them more susceptible to shorting.

Within the Xilinx 7-Series architecture, there are two routing muxes in every CLB switchbox where this routing CMF can occur. The global clock tree is routed through these muxes when driving a slice, as shown in Figure 7. Each CLB tile contains two slices, each slice containing a handful of look-up tables (LUTs) and flip-flops (FFs) (among other primitive cells). The clock, reset/set and chip enable lines are shared for each cell in a slice. Thus, at most, two unique clock nets can be present in a switchbox in a CLB tile. The basic idea behind our algorithm is to ensure one of several possibilities. First, there is up to one clock in the CLB tile. Second, if there are two clocks, they are from the same domain. Third, if there are two clocks from multiple domains, their downstream cells are not from the same partition. If one of these cases is true for every CLB tile in the device, then this type of CMF will not exist.

For our tool, after applying TMR to the netlist, we allow Vivado to place the design. We then make our changes after placement (but before routing). Our algorithm is to check every tile for one of the three cases. If one of those cases does not exist, then CMF is possible and one of the slices in that CLB tile is swapped with a slice in a neighboring CLB tile. Both tiles involved in the swap are checked to make sure one of the three conditions exist before the swap is finalized. All of this type of CMF will be removed after performing swaps for each tile with CMF in the original design. Once the incremental placement is completed, the design implementation is completed by running route design in Vivado followed by bitstream generation. The proposed flow is shown in Figure 6.

While we only show the results and a detailed analysis for 7-Series devices, other generations should have similar causes of CMF and similar solutions. The routing mux we have shown is from a Xilinx patent, and we assume that many generations of devices use a similar structure. The Virtex-4,5,6 and UltraScale families can all have multiple clocks in the same tile, which leads us to believe that this particular CMF will be present across all of these devices. Current efforts are underway to port this technique to the UltraScale family.

VI. Automated CMF Identification & Removal

A custom tool was created to automatically identify and remove CMFs in a placed design. To aid this process, two research CAD tools are used to manipulate the design outside of the typical flow, Tincr [21] and RapidSmith2 [22], [23]. Tincr is a library of TCL commands containing many functions, including those to export and import designs into the Vivado tool suite (these designs are known as Tincr Checkpoints). These checkpoints store the netlist, constraints, placement and routing information of a design. The exported design from Tincr can be imported into the RapidSmith2 tool, an open-source CAD tool for Xilinx FPGAs that allows low-level manipulations of FPGA designs.

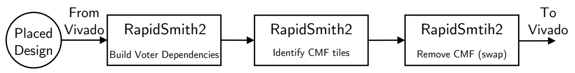

Fig. 8: Designs steps for removing CMF in RapidSmith2

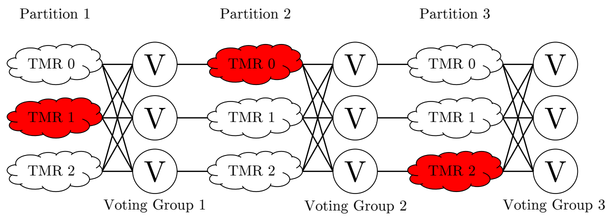

Fig. 9: More frequent TMR voting example.

Figure 8 shows the steps that need to be taken in order to remove CMF from a design. The first step is to identify the voter dependencies which relates to feedback TMR. With feedback TMR [7], groups of logical cells are divided into partitions, as shown in Figure 9, with voters being placed on feedback paths. We will refer to the triplicated voter as a voter group. The figure shows clean partitions, i.e., the cells in each partition only drive 1 voter group. In real designs, this does not have to be the case. An easy example is a global clock buffer (BUFG) cell, which will drive all the flip-flops of a single domain, naturally driving many voter groups.

Partitioning helps improve the effectiveness of TMR, as the design can tolerate multiple failures in multiple domains, as long as those failures occur in different partitions. In Figure 9, the red cells could represent failures. Even though they occur in multiple domains, they are protected by voting groups and will not cause TMR failure. However, if multiple errors occur in multiple domains of the same partition, then TMR can fail. Understanding the partitioning of a design is important for determining fail sets, i.e., sets of cells that can cause TMR failure.

Through reverse traversal of the design graph, the voting group each cell drives can be determined. This creates a property for each cell, that we will call $dependentVoters$. This will be a set containing the ID of each voter group the cell drives, e.g. (1,3) or (2).

The next step of the flow, shown in Figure 8, is to identify CMF tiles. To detect CMF in a CLB tile, all the cells with clocks[2] of each slice in a tile must be checked to determine if they form a fail set or not. In TMR, a fail set is any set of cells that will cause the design to fail if each of those cells fails simultaneously. Using the voter dependencies of each cell, built in the first step of the flow, it is possible to determine fail sets, as demonstrated by Algorithm 1.

For this work we will take a conservative approach and assume that any set which contains cells from multiple domains which drive the same voter group will cause a failure. The algorithm then performs the following steps: each $cell$ from the input group of $cells$ is analyzed for its dependent voters (voter groups) and domain, which is added to a map to record the information. After each $cell$ has been analyzed it can be determined whether these cells form a fail set by looking at the number of domains associated with each $voterID$. If there are more than two domains, that signifies that there is a voter group that would experience two or more domain failures if every $cell$ in $cells$ were to fail. Thus the algorithm is to iterate over $voterID$ and check the size of the set of domains.

Algorithm 1 Identify Fail Set

```
procedure ISFAILSET($cells$)
  Initialize map<voterID, set<Domain>> $possFailSets$
  Initialize collection<voterID> $dependentVoters$
  for each $cell$ in $cells$ do
    $dependentVoters \leftarrow cell$.getDependentVoters()
    for each $voter$ in $dependentVoters$ do
      $possFailSets$.add($voter$, $cell$.domain)
    end for
  end for
  for each $voter$ in $possFailSets$.keys() do
    if $possFailSets$.get($voter$).size() $\geq 2$ then
      return $True$
    end if
  end for
  return $False$
end procedure
```

Now that CMF has been identified, the next step is to remove it, as shown in Algorithm 2. This will be done with a post-placement pass on the design. Recall that we have proposed to swap all the cells in a site (i.e. a packed slice) with another slice in nearby tile, as shown in Figure 10. To avoid creating another tile with CMF, the potential new configuration is checked for CMF before the swap is finalized. In the rare case that no compatible sites are found, tiles that are further away (instead of immediate neighbors) are checked. After iterating over all the tiles with CMF in the original design, the new design is imported back into Vivado, and the normal flow is continued from the routing step.

[2]Usually this will just be the flip-flops, but there are other cells that can be driven by a clock, such as when a SLICEM LUT is configured in RAM mode.

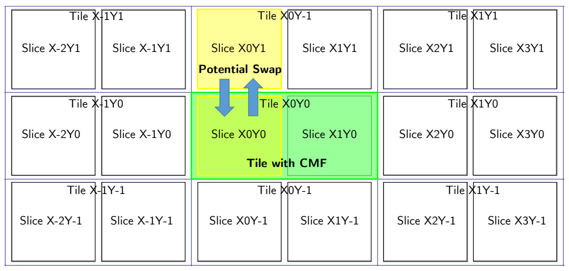

Fig. 10: Swaps can be made with any site in a neighboring tile. Note row/column references are relative.

Algorithm 2 CMF Removal

```
procedure ISFAILSET($slice1$, $slice2$)
  /* .clks means $slice$.getCellsWithClocks() */
  return ISFAILSET($slice1$.clks + $slice2$.clks)
end procedure

procedure REMOVECMF($design$)
  Initialize collection<Tile> $tilesWithCMF$
  $tilesWithCMF \leftarrow$ IDENTIFYCMF( )
  for each $curTile$ in $tilesWithCMF$ do
    Initialize $resolved \leftarrow False$
    Initialize collection<Tile> $nearbyTiles \leftarrow curTile$.neighbors
    Initialize $curSlices \leftarrow curTile$.getUsedSlices()
    while !$resolved$ do
      Initialize set<Tile> $newTiles$
      for each $tile$ in $nearbyTiles$ do
        if not ISFAILSET($curTile$.slice1, $tile$.slice1) and not
                    ISFAILSET($curTile$.slice2, $tile$.slice2) then
          $resolved \leftarrow True$
          swap($curTile$.slice2,$tile$.slice1)
        else if not ISFAILSET($curTile$.slice1, $tile$.slice2) and not
                    ISFAILSET($curTile$.slice2, $tile$.slice1) then
          $resolved \leftarrow True$
          swap($curTile$.slice1,$tile$.slice1)
        else
          $newTiles$.addAll($tile$.getNeighbors())
        end if
      end for
      if !$resolved$ then
        $nearbyTiles \leftarrow newTiles$
      end if
    end while
  end for
end procedure
```

VII. Results

Several designs were analyzed and tested for CMF (but the CMF removal technique was not applied). These include the

TABLE II: Circuit Implementation Properties - b13

TMR Type	f_{max}	Number of Routing Nodes	Number of Cells	Number of Sites	Number of Tiles
Unmitigated	82.4 MHz	$233,780$	$25,542$	$3,688$	$2,033$
TMR	67.7 MHz	$1,373,762$	$104,066$	$19,096$	$9,803$
PCMF	64 MHz	$1,434,069$	$103,297$	$19,096$	$9,806$

Note: PCMF has less cells than TMR because all VCC and GND cells are compounded in to a single cell when importing into RapidSmith2

TABLE III: Fault Injection Results - b13

TMR Type	Number of Injections	Number of Faults	Percent Sensitivity	Confidence Intervals	Number of Sensitive Bits	Improvement
Unmitigated	$2,193,073$	$29,436$	1.342%	1.327 to 1.357%	$784,860$ to $802,876$	$1\times$
TMR	$2,351,568$	43	$1.8 \times 10^{-3}\%$	1.3 to $2.4 \times 10{-}3\%$	758 to $1,405$	$734\times$
PCMF	$2,396,265$	3	$1.3 \times 10^{-4}\%$	0 to $2.7 \times 10^{-4}\%$	0 to 158	$10,721\times$

TABLE IV: Radiation Test Results - b13

TMR Type	Fluence	Number of Failures	Cross Section (n/cm^2)	95% Confidence	MTTF - GEO (days)	Improvement
Unmitigated	1.70×10^{11}	314	1.85×10^{-9}	2.06×10^{-9}	2.49×10^{2}	$1\times$
TMR	2.48×10^{11}	6	2.42×10^{-11}	4.39×10^{-11}	1.90×10^{4}	$76\times$
PCMF	3.98×10^{11}	2	5.03×10^{-12}	1.21×10^{-11}	9.15×10^{4}	$368\times$

Note: MTTF calculated for GEO orbit using the bit cross-section presented in [11] and likely over estimates the real MTTF.

TABLE V: Detected CMF In Circuits

Circuit	Number of CMF Tiles	Number of CLB Tiles	Number of CMF Bits
b13	2,471	9,728	43
md5	6,821	15,697	56
sha3	89	8,209	14
aes	99	13,814	2
leon3	205	3,068	not tested
b13 (UltraScale)	7,515	29,509	24

Note: Some designs were only analyzed for the CMF issue

b13, md5, sha3, aes128 and leon3 circuits. The number of tiles with CMF as well as the number of used CLB tiles are reported in Table V. The number of detected CMF bits (from fault-injection) is also reported for some designs. A majority of bits found through fault-injection correlate to the routing CMF issue, with a small number (less than 10%) due to the SLICEM issue (discussed later). Due to limited resources for fault-injection and limited beam availability, only the b13 design was selected for a comprehensive study for how this technique effects reliability.

The b13 design comes from the ITC'99 benchmark suite and is a simple finite state machine that interfaces with a weather station [24]. Test vectors used in the test to simulate and compare the results are taken from an automatic test pattern generation. It has been used by a mitigation working group to test benefits of TMR [12]. The design was replicated 256 times in order to generate a circuit that utilized more of the device's resources. The circuit properties of the b13 are reported in Table II for each TMR type: Unmitigated, Netlist-TMR and this technique, dubbed PCMF for placement CMF. There is a hefty price in applying TMR to a design that is well known (trade-off with increased reliability). The PCMF technique adds some overhead on top of TMR in terms of routing cost. We believe that this overhead could be reduced by considering swaps that would minimize the increase to the half perimeter wirelength (HPWL) (or another metric) instead selecting the first swap that meets that criteria, but leave this as future work.

The design has been compared with other types of TMR application for comparison in improvement using both fault-injection and radiation testing, shown in Tables III and IV, respectively. Confidence intervals are shown with 95% confidence. Our tool took .9 seconds to analyze the design for CMF and 180 seconds to remove CMF from the design.

Our fault-injection infrastructure consisted of a custom setup using Nexys Video Artix-7 FPGA boards available from Digilent. Each setup consists of 2 boards, one master and one device under test (DUT), connected via the FMC card slot. The master operates with a golden copy of the design (i.e. no CRAM fault injections) in lockstep with the DUT running the same design, but subject to CRAM upsets. After fault-injection, the design was allowed to run for a period of time to flush out any faults in the system, before being scrubbed and repeating the process. After finding a fault the device was reconfigured and the bit was injected again to verify the upset. Fault-injection was preformed via JTAG.

The most important result from Table III is that our new technique shows a $10,721\times$ improvement in design sensitivity over the unmitigated design and a $14.6\times$ improvement over netlist TMR. Three failures were observed during fault injection. These errors are not related to routing CMF, but is another, more rare, type of CMF. This CMF appears to be related to SLICEM sites and future work will look into resolving this type of CMF.

Radiation testing was performed using a neutron beam at the Los Alamos Neutron Science Center (LANSCE) at Los

Alamos National Laboratory (LANL) [25]. This testing shows that our technique improves the MTTF over baseline TMR from $19,000$ to $91,500$ days, an improvement of about 5x. The failures observed were directly attributable to multi-cell upsets (MCUs), i.e., the simultaneous changing of two or more bits. No single bit upset failures were observed.

VIII. Conclusion and Future Work

An automated tool was developed to identify CMFs which limit the effectiveness of netlist-based TMR tools for SRAM FPGAs. In addition, an incremental placement tool was developed in this work and was successfully able to remove CMF by changing the placement of a few cells. Extensive fault injection results demonstrate that this technique successfully identified and removed most CMF. The estimated reliability of a benchmark design was significantly improved by supplementing TMR with this technique.

Results from this initial experiment suggest that this technique is a promising approach for improving the effectiveness of TMR in FPGA designs. Likewise, initial radiation results also suggest this technique is promising. With a $5\times$ improvement in MTTF, the vast majority a failures are now comprised of multiple upsets (instead of single upsets). Future work will investigate the effectiveness of this technique on a wide variety of designs as well as testing those designs in other radiation sources (such as heavy ions). It will also be interesting to investigate the if there is any correlation between the MCUs observed and if there is any possible technique to address them to further reduce the design's sensitivity. While currently only applicable to 7-Series devices, the UltraScale family also contains a similar structure (2 clocks in the same tile). Efforts are already underway to apply this technique to the UltraScale devices.

As an additional technique for improving the reliability of SRAM-based FPGAs in the presence of ionizing radiation, this technique will allow SRAM-based FPGAs to be increasingly considered for use in spacecraft and other environments with high levels of radiation. Applying TMR to an FPGA design and modifying the implementation of the design to remove CMF are two complementary techniques that facilitate the reliable use of FPGA designs in even the harshest radiation environments. We anticipate that these techniques will be used in future space missions employing SRAM-based FPGAs.

Acknowledgments

This work was supported by the I/UCRC Program of the National Science Foundation under Grant No. 1265957. This work was also supported by the Utah NASA Space Grant Consortium.

References

[1] Xilinx, "Device reliability report," Xilinx, Tech. Rep. UG116, 2017.

[2] A. Jacobs *et al.*, "Reconfigurable fault tolerance: A comprehensive framework for reliable and adaptive FPGA-based space computing," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 5, no. 4, pp. 21:1–21:30, Dec. 2012.

[3] M. Berg and K. LaBel, "New developments in error detection and correction strategies for critical applications," 2017. [Online]. Available: https://ntrs.nasa.gov/search.jsp?R=20170004736

[4] I. Herrera-Alzu and M. Lopez-Vallejo, "Design techniques for Xilinx Virtex FPGA configuration memory scrubbers," *Nuclear Science, IEEE Transactions on*, vol. 60, no. 1, pp. 376–385, Feb 2013.

[5] M. Berg *et al.*, "Effectiveness of internal versus external SEU scrubbing mitigation strategies in a Xilinx FPGA: Design, test, and analysis," *IEEE Transactions on Nuclear Science*, vol. 4, no. 55, pp. 2259–2266, 2008.

[6] A. Stoddard *et al.*, "A hybrid approach to FPGA configuration scrubbing," *IEEE Transactions on Nuclear Science*, vol. 64, no. 1, pp. 497–503, 2017.

[7] J. M. Johnson and M. J. Wirthlin, "Voter insertion algorithms for FPGA designs using triple modular redundancy," in *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '10. New York, NY, USA: ACM, 2010, pp. 249–258.

[8] G. R. Allen *et al.*, "Single-event upset (SEU) results of embedded error detect and correct enabled block random access memory (block RAM) within the Xilinx XQR5VFX130," *IEEE Transactions on Nuclear Science*, vol. 57, no. 6, pp. 3426–3431, Dec 2010.

[9] D. P. Siewiorek and R. S. Swarz, *Reliable computer systems*. A. K. Peters.

[10] B. Pratt *et al.*, "Improving FPGA design robustness with partial TMR," in *2006 IEEE International Reliability Physics Symposium Proceedings*, March 2006, pp. 226–232.

[11] D. S. Lee *et al.*, "Single-event characterization of the 28 nm Xilinx Kintex-7 field-programmable gate array under heavy ion irradiation," in *2014 IEEE Radiation Effects Data Workshop (REDW)*, July 2014, pp. 10–14.

[12] H. Quinn *et al.*, "Using benchmarks for radiation testing of microprocessors and FPGAs," *IEEE Transactions on Nuclear Science*, vol. 62, no. 6, pp. 2547–2554, Dec 2015.

[13] A. M. Keller and M. J. Wirthlin, "Benefits of complementary SEU mitigation for the LEON3 soft processor on SRAM-based FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 64, no. 1, pp. 519–528, Jan 2017.

[14] J. Arlat *et al.*, "Fault injection for dependability validation: A methodology and some applications," *IEEE Transactions on Software Engineering*, vol. 16, no. 2, pp. 166–182, Feb 1990.

[15] F. L. Kastensmidt *et al.*, "On the optimal design of triple modular redundancy logic for SRAM-based FPGAs," in *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 2*, ser. DATE '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 1290–1295.

[16] H. Quinn *et al.*, "Domain crossing errors: Limitations on single device triple-modular redundancy circuits in Xilinx FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 54, no. 6, pp. 2037–2043, Dec 2007.

[17] L. Sterpone and M. Violante, "A new reliability-oriented place and route algorithm for SRAM-based FPGAs," *IEEE Transactions on Computers*, vol. 55, no. 6, pp. 732–744, June 2006.

[18] M. S. Reorda, L. Sterpone, and M. Violante, "Multiple errors produced by single upsets in FPGA configuration memory: A possible solution," in *European Test Symposium (ETS'05)*, May 2005, pp. 136–141.

[19] S. Trimberger, "Quintuple modular redundancy for high reliability circuits implemented in programmable logic devices," Apr. 13 2004, uS Patent 6,720,793. [Online]. Available: http://www.google.tl/patents/US6720793

[20] S. Young, "Integrated circuit having fast interconnect paths between memory elements and carry logic," May 15 2007, US Patent 7,218,143. [Online]. Available: https://www.google.com/patents/US7218143

[21] B. White and B. Nelson, "Tincr – a custom CAD tool framework for Vivado," in *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*, Dec 2014, pp. 1–6.

[22] T. Haroldsen, B. Nelson, and B. Hutchings, "RapidSmith 2: A framework for BEL-level CAD exploration on Xilinx FPGAs," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: ACM, 2015, pp. 66–69.

[23] T. Townsend, B. Nelson, and M. Wirthlin, "An XDL alternative for interfacing RapidSmith and Vivado," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2016, pp. 1–1.

[24] F. Corno, M. S. Reorda, and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," *IEEE Design Test of Computers*, vol. 17, no. 3, pp. 44–53, Jul 2000.

[25] P. W. Lisowski and K. F. Schoenberg, "The Los Alamos neutron science center," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 562, no. 2, pp. 910–914, 2006.