Design and Verification of SEE-Tolerant ASICs at CERN: Methodologies and Challenges

Adithya Pulli, Matteo Lupi, Stefano Esposito, Simone Scarfì, Szymon Kulis, Xavier Llopart Cudie
CERN, Geneva, Switzerland (*adithya.pulli@cern.ch*)

***Abstract*—The radiation environment of the detectors at the CERN Large Hadron Collider (LHC) presents an unprecedented challenge for electronic system design. Since the early 1990s, CERN has been developing custom Application Specific Integrated Circuits (ASICs) tailored to the unique requirements of LHC experiments. As ASIC complexity increases, the tolerance to Single Event Effects (SEE) emerges as a significant design and verification challenge. This article discusses the distinctive challenges in designing SEE-tolerant ASICs for high-energy physics (HEP) experiments. We provide an overview of methodologies used for the design and verification of radiation-tolerant ASICs at CERN, along with examples of SEE vulnerabilities discovered during verification.**

Keywords—SEE, Soft Errors, Fault Injection, TMR

I. ASICs for High Energy Physics Experiments

High-energy physics experiments demand specialized electronic components with functionalities precisely tailored to experimental needs. Off-the-shelf commercial components often fall short of meeting the stringent requirements of these experiments, such as radiation tolerance, fixed latency data transmission, low noise, and dense feature integration. Hence, since the early 1990s, CERN and other HEP laboratories have been designing and deploying custom ASICs to precisely meet these requirements [1]. Initially, ASICs primarily comprised analog readout circuits with minimal digital logic. However, recent ASICs designed for the high luminosity upgrades of the Large Hadron Collider (LHC) at CERN contain complex digital circuits to accommodate higher particle flux and finer detector granularity.

When high-energy particles interact with semiconductor materials, they can disrupt device operation, leading to cumulative and single-event radiation effects. Cumulative effects, such as total ionizing dose (TID) and displacement damage (DD), are mitigated using physical design techniques. Single Event Effects (SEEs), manifest as Single Event Upsets (SEUs) in memory elements or Single Event Transients (SETs) on design nodes. Various micro-architectural techniques, including triple modular redundancy (TMR), triple time redundancy (TTR), and error correction codes (ECC), are often employed to mitigate the effects of SEEs [2]. However, exhaustive protection against SEEs incurs significant area and power overheads. Therefore, ASIC designers apply fault tolerance techniques to critical portions of the design. This process of selective hardening and implementation is error prone.

This article focuses on SEE mitigation and verification techniques used in ASICs designed at the microelectronics group at CERN. An overview of SEE mitigation techniques, with an emphasis on Triple Modular Redundancy (TMR), the preferred design strategy for HEP ASICs, is provided in Section II. Section III discusses the SEE verification methodology using simulation and formal techniques. The design and verification techniques have been successfully applied to several ASICs designed at CERN. Typical examples of bugs found in these designs are elaborated in Section IV. SEE verification remains a complex topic and a fundamental concern for ASIC designs in the HEP community. Present and future efforts to enhance design and verification techniques are outlined in Section V.

II. SEE mitigation techniques

Several techniques have been proposed to safeguard circuits against SEE induced by ionizing particles. In Figure 1, the most commonly used SEE mitigation techniques are summarized. While SEE mitigation at the technology and cell level are popular choices in the automotive and aerospace industries, system-level techniques

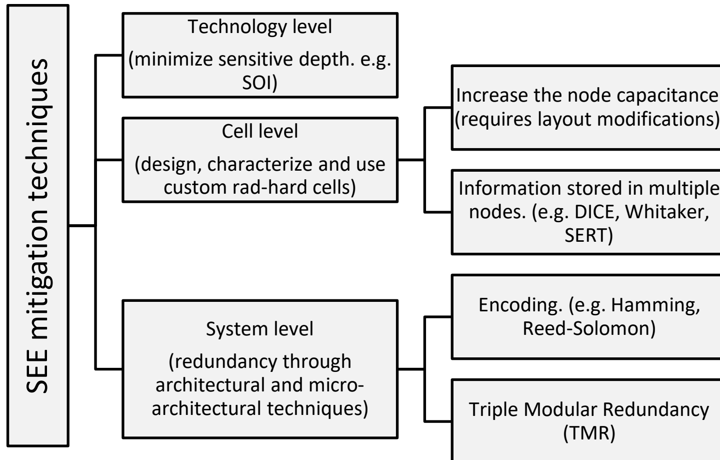

Figure 1 SEE mitigation techniques.

emerged as a popular choice of SEE mitigation in the HEP community. This preference is mainly attributed to the unique radiation environments of HEP experiments. The high-energy particle flux in HEP experiments is several orders of magnitude higher than that of automotive and aerospace environments. Therefore, ASICs for HEP experiments require several orders of magnitude better protection against single event and cumulative effects compared to ASICs for automotive and aerospace applications [1]. SEE mitigation at the system level can be achieved either by encoding system states and other memory elements using error correction codes or through triple module redundancy (TMR). Protection using error correction codes is highly architecture dependent and falls short of protecting the circuits against SETs. Conversely, TMR is a generic technique that can be applied to any kind of design.

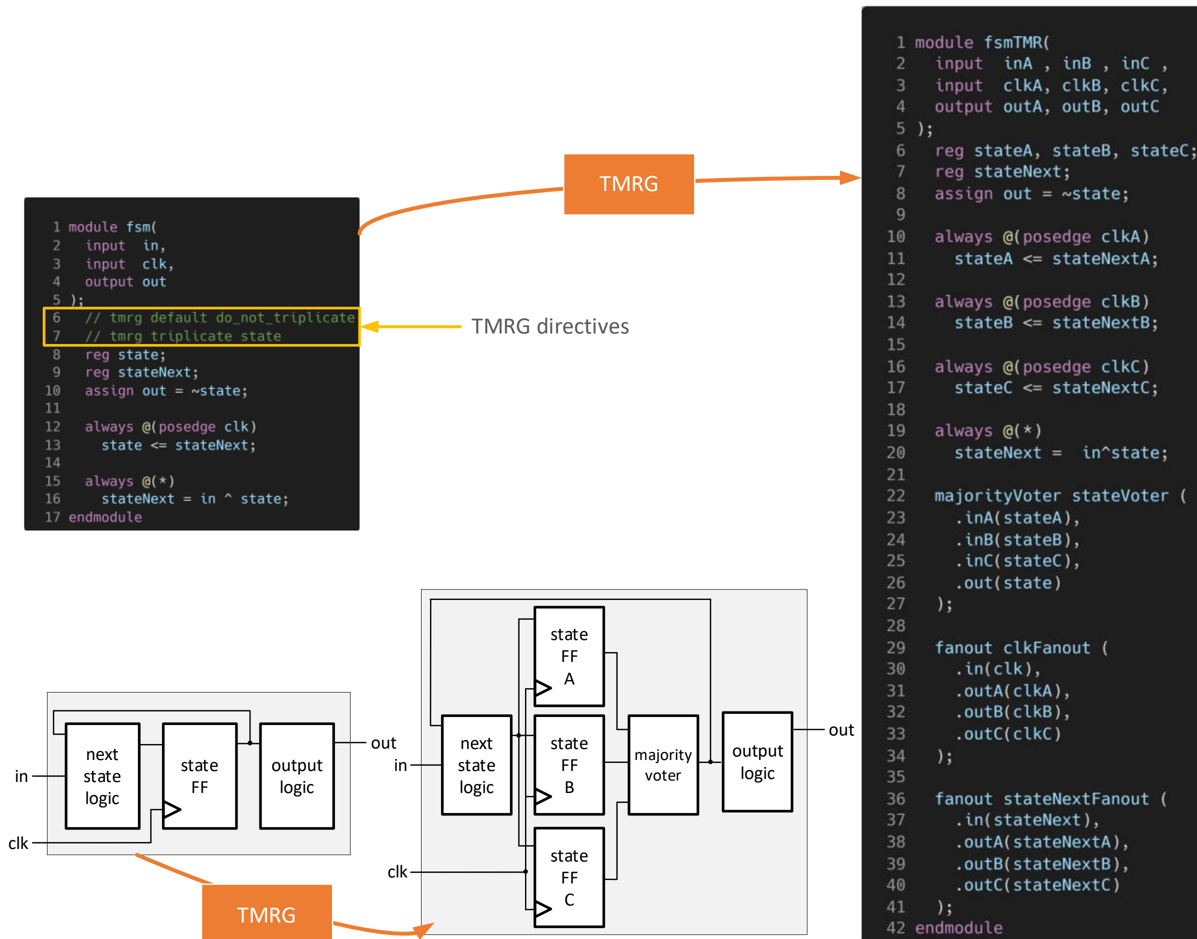

Figure 2 FSM triplication using TMRG. Notice that only the state is triplicated. This circuit is not protected against SETs in combinatorial logic.

TMR can be implemented either at the block level or at a lower granularity. In block-TMR implementations, the block, typically a sub-system in the ASIC, that must be protected is replicated as three block boxes, and the majority voter is placed at the outputs of the triplet. If one of the triplicated copies of the block produces an incorrect output due to SEE, the other two correct outputs will outvote the erroneous one. The main drawback of block-TMR is the absence of feedback to the internal states of the block. This can lead to divergence of the internal states of the three blocks over time due to error accumulation. While this scheme works well at low upset rates, it is not suitable for HEP ASICs where upset rates are very high. TMR is applied at a much lower granularity, typically at the logic level in HEP ASICs.

Manually incorporating TMR elements into RTL is time-consuming and error prone. The Triple Module Redundancy Generator (TMRG) tool developed at CERN automates the process of triplicating digital circuits freeing the designer from introducing the TMR code at the RTL coding stage. TMRG is an open-source tool that can be used to triplicate Verilog designs. To ensure maximum flexibility TMRG tool allows the designer to decide which blocks and signals should be triplicated. The directives to the TMRG tool are placed as comments in the RTL code. An in-depth description of TMRG and its capabilities can be found in [3].

Figure 2 and Figure 3 show how TMR is applied to a generic finite state machine using TMRG. Error accumulation is avoided by using a majority-voted state to determine the next state of the FSM. While the scheme shown in Figure 2 is sufficient to protect the design against SEUs in the sequential cells, it does not protect the design against SETs in the combinatorial logic or the voter. Since SETs are of concern for HEP ASICs, the full TMR solution, as shown in Figure 3, is adopted, where combinatorial logic, sequential logic, and clocks are triplicated. This full triplication guarantees that all the nodes in the design are robust to SEUs and SETs.

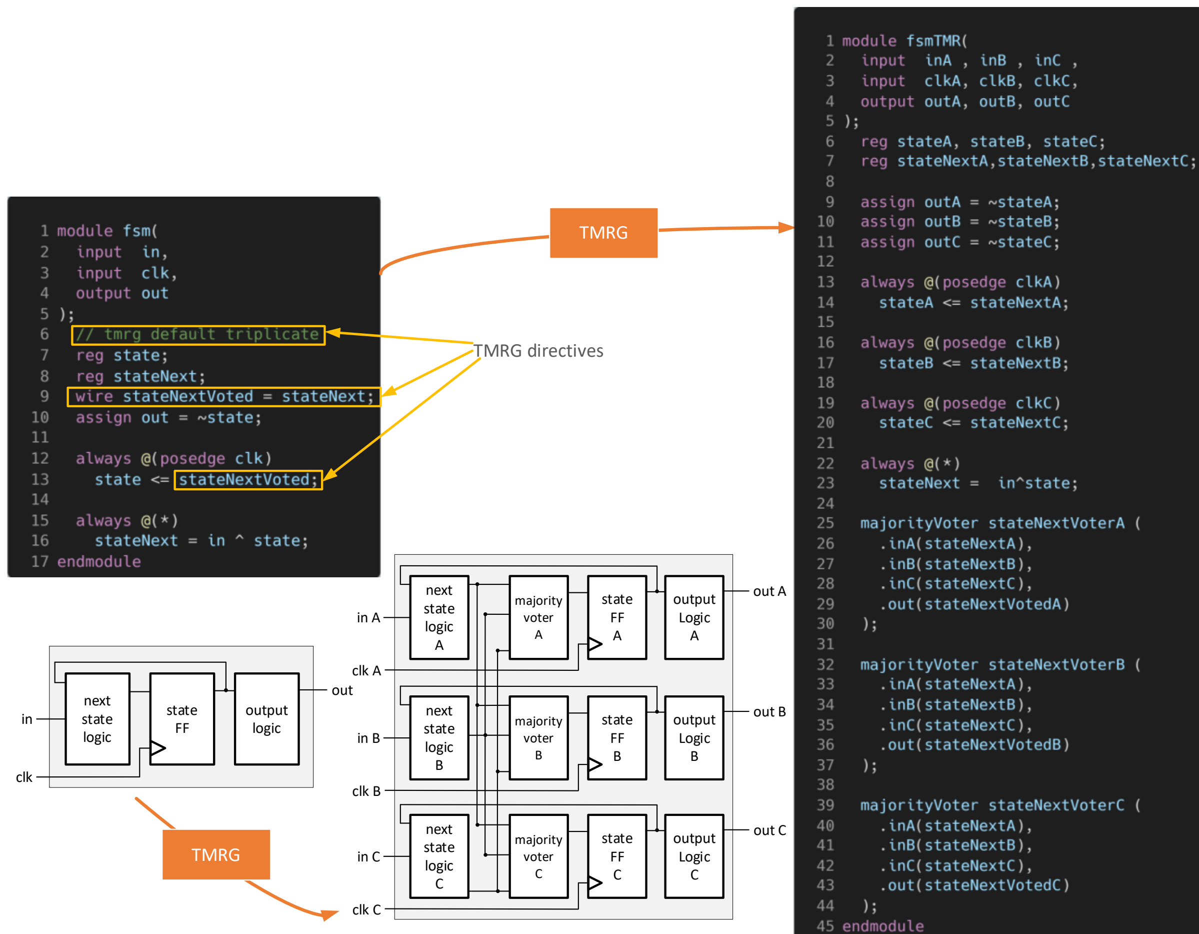

Figure 3 Full TMR using TMRG. All the combinatorial logic, clocks and sequential cells are triplicated. This circuit is fully protected against all SEEs.

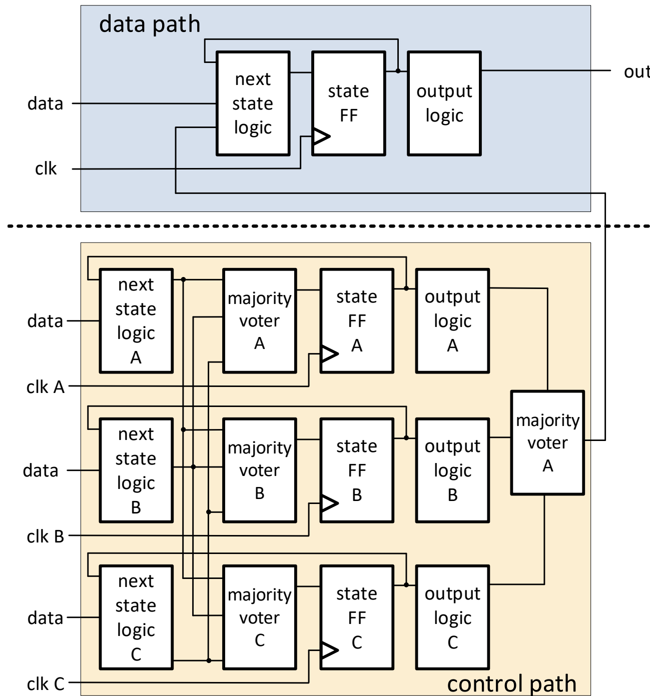

Figure 4 Selective triplication using TMR.

Applying the TMR technique for the entire design is prohibitively expensive. In typical HEP ASICs, a subset of logic that must be triplicated is identified by the designer based on the requirements of the ASIC. A simple example of selective triplication is when the control path of the logic is triplicated while the data path is left unprotected, as shown in Figure 4. The process of selective triplication is error-prone and therefore requires elaborate verification to prove the correctness of TMR intent and TMR implementation. SEE verification plays a crucial role in HEP ASIC development and these techniques are described in section III.

III. SEE Verification Techniques

Tolerance to Single Event Effects (SEEs) is one of the most crucial aspects for HEP ASICs to ensure a reliable and robust operation. The verification of SEE is an integral part of the functional verification process and is incorporated into the overall verification strategy and plan. Both simulation and formal verification techniques are used to verify the design for SEE robustness. During verification, two major SEE verification goals must be met:

1. *Correctness and completeness of TMR intent*: It must be ensured that all parts of the design that need to be triplicated for the reliable operation of the ASIC are correctly identified.

2. *Correctness of TMR implementation*: It must be ensured that all parts of the design identified for triplication are correctly triplicated.

Simulation methods for SEE verification address both the verification goals. Presently, formal methods are used only to address the correctness of TMR implementation. In section III.A, the SEE verification framework used to implement the simulation based SEE verification strategy is detailed. Section III.B elaborates on how model checking is used to verify TMR correctness.

A. Simulation methods

To achieve comprehensive verification sign-off for SEE tolerance, a robust strategy toward SEE verification must be defined. The SEE verification requirements of the project are considered while building the functional verification framework. SEE tolerance requirements vary depending on the ASIC's application in the HEP experiment. For instance, ASICs used to implement high-speed links in the detector have stringent data integrity requirements compared to front-end readout ASICs where data losses may be tolerated. These requirements are built into the scoreboards used in functional verification. Early planning for comprehensive SEE verification improves verification productivity by reducing the need for extensive refactoring of verification frameworks at later stages. Additionally, automatic scoreboarding of SEE tolerance enables the implementation of highly constrained random tests that include fault injections.

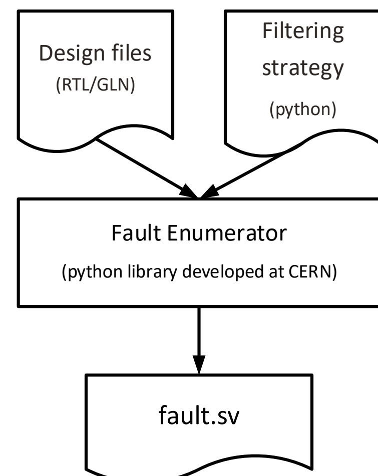

Figure 5 Fault generation flow.

SEE verification starts with a fault generation flow, shown in Figure 5. The fault enumerator is a python library developed at CERN. It takes the design files, either in RTL or Gate Level Netlist (GLN) format, and a filtering strategy as inputs and generates a filtered list of fault nodes in System Verilog format (fault.sv). The output file generated by this flow is utilized by SEE UVC to inject constrained random faults (Figure 5). In the absence of a filtering strategy, the fault enumerator extracts an exhaustive list of design nodes for SEU and SET injection. A filtering strategy can optionally be implemented using the Python APIs of the fault enumerator. This strategy helps users select a subset of design nodes for SEE injection based on test intent. For instance, it may be desirable to inject SEUs only in TMR nodes to prove the correctness of TMR implementation.

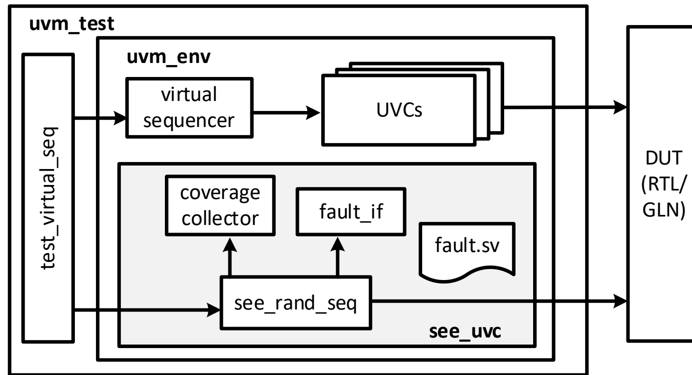

Figure 6 SEE UVM Verification Component.

Fault injection is performed using SEE UVC, which is integrated into the UVM verification environment as shown in Figure 6. SEE UVC consists of a random SEE injection sequence (*see_rand_seq*), a coverage collection component, and a fault debug interface (*fault_if*). The *see_rand_seq* randomly injects either an SEU or a SET, multiple times at randomly chosen moments during the simulation, on a randomly selected node. The sequence implementation assures random stability of simulation with and without fault injection. This enables users to repeat simulations with the same functional stimuli by either enabling or disabling fault injection. In addition to assuring random stability, *see_rand_seq* also allows users to easily constrain and control fault type, fault time, fault node, and fault duration (for SETs).

During fault injection, SEU is modeled as a deposit of the inverted value on a memory element, and SET is modeled as a temporary glitch on a net using force and release mechanisms. The implementation of deposit, force, and release utilize *uvm_hdl_** methods which enable efficient simulator interoperability of SEE UVC. The pseudocode to model SEU and SET injection is shown in Figure 7.

```
1 task inject_seu(input string node, int node_idx);
2   logic val;
3   uvm_hdl_read(node, val);
4   uvm_hdl_deposit(node, ~val);
5 endtask
6
7 task inject_set(input string node, int duration_ps,
  int node_idx);
8   logic val;
9   uvm_hdl_read(node, val);
10  uvm_hdl_force(node, ~val);
11  #(duration_ps*1ps);
12  uvm_hdl_release(node);
13 endtask
```

Figure 7 Methods for SEU and SET injection.

SEE UVC also includes a coverage collection component that samples a coverpoint on every fault injection. This data when collected over several fault simulations provides a useful metric of the number of faults injected per faultable node. Fault coverage information in conjunction with functional and code coverage data is used to gain insights into how much of the functionality was exercised with fault injections. In addition to the coverage collection component, SEE UVC also includes *fault_if* which provides high-level insight into the status of fault injection. This is an extremely useful debug feature for interactive debugging that enables users to quickly identify which fault in the design caused a specific behavior of the circuit.

Various fault campaigns are included in the verification plan. A fault campaign is defined as a regression consisting of constrained random tests, which are run multiple times to achieve a specific SEE verification goal. Pass rate and coverage are tracked for each fault campaign and are used as indicators to assess the completeness of SEE verification. Some typical examples of fault campaigns are summarized in Table 1.

Table 1 Examples of fault campaigns.

Name	Goal	Method	Abstraction
TMR	Verify that all nodes that are intended to be triplicated are correctly triplicated	Two simulations are run in parallel with the same random seed: one with fault injections and the other without. All the outputs of the DUT are recorded during simulations and compared at the end. If TMR is correctly implemented, then outputs of both the simulations must match	RTL, GLN
Non-TMR	Verify the TMR intent is correct	Inject SEEs in non-TMR nodes and check that the fault tolerance requirements are preserved	RTL
Random	Verify the SEE robustness of the DUT	Inject random SEEs in the DUT at various test phases. Interleave test phases with SEE injection and without SEE injection. Enable SEE error tolerance in the checkers during SEE injection and use strict checkers when SEE injection is disabled.	RTL, GLN

B. Formal methods

While simulation based SEE verification is the main workhorse for verifying SEE robustness, formal verification techniques are also incorporated. Formal property verification is used to check the correctness of TMR implementation at GLN level [5]. Due to the very low-level nature of the TMR strategy, the triplicated parts of digital circuits recover from SEUs in a predictable time (typically one clock cycle). This observation is used to formally verify that a few cycles after one of the triplicated nodes is upset by an SEU, all the triplicated copies of that node are equal. The SEUs are modeled by modifying the standard cell primitives for sequential cells. An internal signal (*q_seu*) is added to the primitive of the cell, which is left unconnected, allowing the model-checking tool to control it. When this signal is asserted the content of the sequential cell is toggled (see Figure 8), emulating the effect of an SEU.

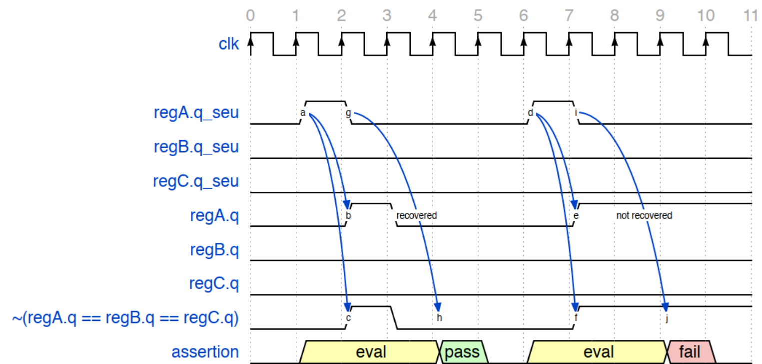

Figure 8 Example of TMR assertions.

The fault enumerator python library described in section III.A is used to generate a list of System Verilog assertions and assumptions for all the TMR nodes in the design (Figure 9). Additionally, it also generates TCL files needed to steer the formal verification tool. For each of the sequential logic elements identified, the framework generates a set of assumptions and assertions. Even though the number of properties is considerable for larger

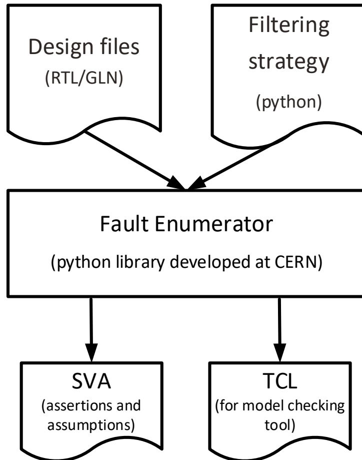

Figure 9 SVA generation flow.

designs, their computational complexity is low given their limited duration. The pseudocode of generated assertions and assumptions is shown in Figure 10.

The assumptions ensure that the SEUs injected are not producing false positives. The first assumption (line 7 in Figure 10) allows injecting at most only one SEU per triplet. The second assumption (line 8 in Figure 10) ensures that each SEU has a duration of one clock cycle. This does not lose in generality since the goal of the proof is to check that an SEU is corrected within a certain number of clock cycles. The number of SEU injected in the whole design is not limited, meaning that the tool can inject one SEU in each triplet of the design. Despite this being an unrealistic condition, it does not affect a correctly triplicated design and it allows speeding up proof.

```
1 //Assertions
2 regA.q_seu |-> ##2 (regA.q == regB.q == regC.q);
3 regB.q_seu |-> ##2 (regA.q == regB.q == regC.q);
4 regC.q_seu |-> ##2 (regA.q == regB.q == regC.q);
5
6 //Assumptions
7 $onehot0({regA.q_seu,regB.q_seu,regC.q_seu});
8 |{regA.q_seu,regB.q_seu,regC.q_seu} |=>
  ({regA.q_seu,regB.q_seu,regC.q_seu}==3'b000)[*1];
```

Figure 10 Pseudocode for the generated SVA.

The three assertions (lines 2-4 in Figure 10) verify that an SEU is corrected within two clock cycles, for each of the three SEU in a triplet. During the first clock cycle, the SEU is injected; in the second, the SEU affects the output of the D-flipflop (DFF); and finally, the DFF is corrected in the third clock cycle. These properties should hold at any point in time for any correctly triplicated DFF in the design, requiring no additional information on the design functionality.

IV. ANATOMY OF SEE VULNERABILITIES (BUGS)

SEE design and verification techniques discussed in sections II and III have been successfully applied to various ASICs designed for high luminosity upgrades of LHC at CERN [6][7]. Over time, these techniques have matured and become widely accepted methodologies for designing SEE-tolerant ASICs within the HEP community. In this section, examples of some typical SEE bugs are provided.

A. *Logic optimizations*

Modern synthesis tools implement several logic optimization steps to achieve Power-Performance-Area (PPA) goals. Since the SEE mitigation using TMR relies on redundancy, the synthesis optimization steps often remove

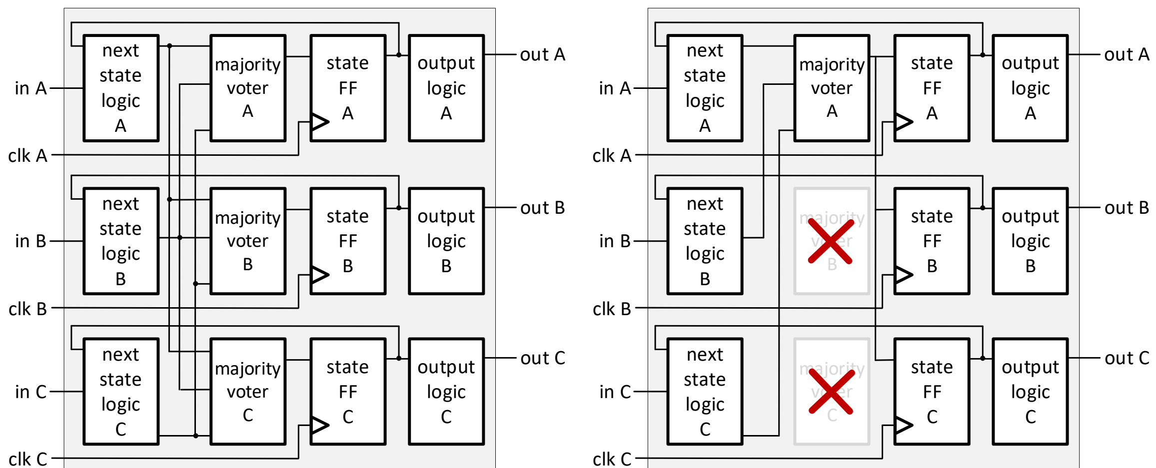

Figure 11 Illustration of logic optimization by the synthesis tool potentially compromising the SEE robustness of a design. The circuit on the left represents the intended design. In the absence of "do not touch" constraints on the voter, the synthesis tool may optimize the voters.

redundant logic. As shown in Figure 11, the design intent requires triplication of voters. The synthesis tool may optimize the logic and remove two of the voters. The synthesized design is therefore no longer protected against SETs in voters. To avoid such optimizations, the TMRG tool generates a set of design constraints to prevent synthesis optimization of voters.

B. *Using non-voted signals*

TMRG serves as a convenience tool that offloads the time-consuming and error-prone task of inserting TMR code from the RTL designer. However, it must be noted that incorrect use of TMRG directives can often result in unintended outcomes. One such incorrect TMRG directive (Figure 12) is the use of a non-voted signal to determine the next state in the FSM.

Figure 12 Illustration of a bug caused by the incorrect use of a non-voted signal instead of a voted signal by the designer. The circuit and code on the left display the design with the bug (marked with a red X). The circuit and code on the right show the correctly triplicated design.

C. *Missing else*

Another typical error observed in RTL is the omission of the “else” in a conditional statement within a sequential process (see Figure 13). This missing else condition results in missing feedback from the voters thus leading to error accumulation in the triplicated registers.

V. Challenges and Future Work

Selective hardening of ASICs used in high-energy physics experiments is a time-consuming and error-prone process. The verification of SEE tolerance in such ASICs presents significant challenges. This paper has presented an overview of SEE mitigation and verification techniques used in the design of HEP ASICs. Based on our experience, SEE verification is resource-intensive in terms of engineering resources, compute resources, licenses, and time. The key to successful SEE verification lies in early planning. SEE verification must be considered while defining the verification strategy and implementing the functional verification framework. Although our SEE verification framework addresses several challenges, there are areas for potential improvement:

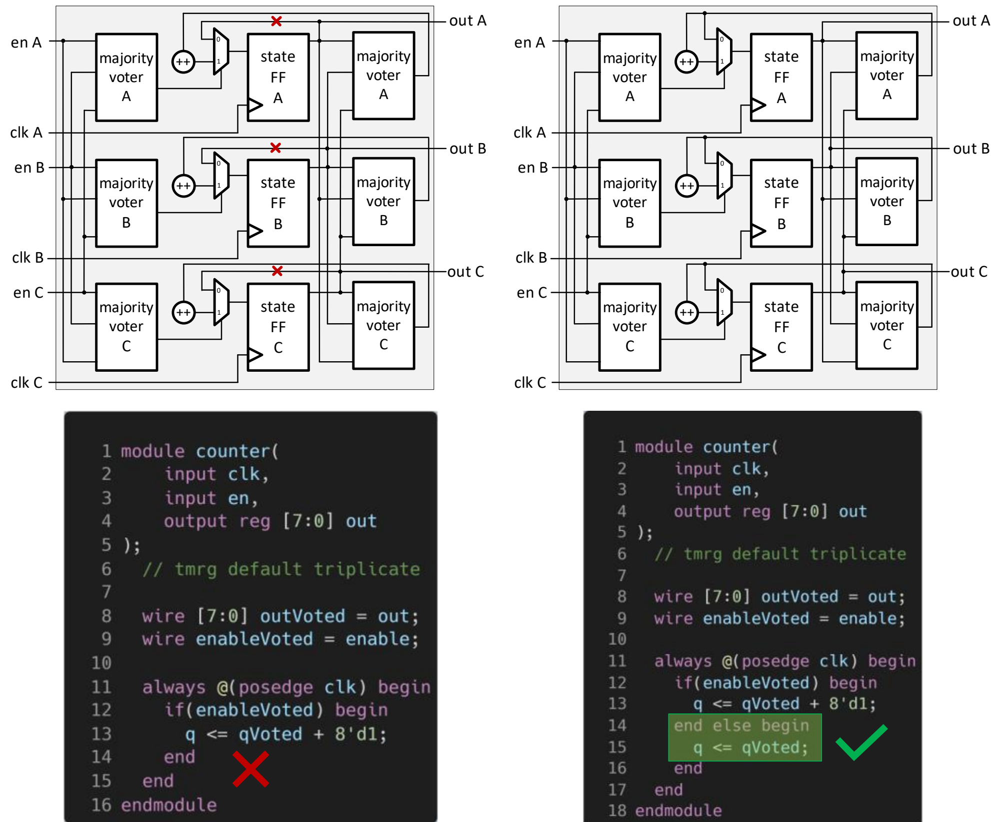

Figure 13 Illustration of a bug in a simple counter due to the omission of an else condition by the designer is shown on the left. Although this circuit produces functionally correct output under normal conditions, it lacks protection against SEUs. When the enable signal to the counter is held low, an SEU in one of the state flip-flops cannot be corrected. If another SEU occurs in a different state flip-flop while the enable signal remains low, the circuit will produce an incorrect output.

- Currently, the formal verification described in section III.B is limited to checking TMR implementation. An extension of formal verification to include verifying TMR intent is desired. This would involve identifying properties of the design that must be maintained despite SEUs and then using these properties, together with generated SVAs, in model checking. This approach would enable the use of model-checking for both design space exploration and the reduction of fault campaigns.

- Another area not currently explored is SET verification using formal techniques. Modeling SET behavior for the model-checking tool is non-trivial. One potential approach to explore involves transforming SETs into multi-bit SEUs in the design and then using our formal SEU injection methods. While this approach works well for most standard cells in the design, it is not applicable for SETs in clock buffers and asynchronous logic, which requires further investigation.

- Although TMRG is a valuable convenience tool, it is not free from bugs. To enhance confidence in the TMR RTL generated by TMRG, the implementation of a formal equivalence-checking tool that can account for TMR insertion is desired. Off-the-shelf Logic Equivalence Check (LEC) tools may not work out of the box for such equivalence checking, necessitating further research in this direction.

REFERENCES

[1] Faccio, F., ASIC survival in the radiation environment of the LHC experiments: 30 years of struggle and still tantalizing, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment Volume 1045 (2023).

[2] Caratelli, A., et al, Low-power SEE hardening techniques and error rate evaluation in 65 nm readout ASICs, PoS TWEPP 2019 (2020) pg. 015.

[3] Kulis, S., Single Event Effects mitigation with TMRG tool, JINST 12 (01) (Jan. 2017) pg. C01082–C01082.

[4] Pulli, A., Lupi, M., A simulation methodology for verification of transient fault tolerance of ASICs designed for high-energy physics experiments, JINST 18 (2023).

[5] Lupi, M., Pulli, A., SEU injection framework for radiation-tolerant ASICs, a formal verification approach, JINST 18 (2023)

[6] Pulli, A., Kremastiotis, I., Kulis, S., Verification methodology of a multi-mode radiation-hard high-speed transceiver ASIC, JINST 17 (2022).

[7] Scarfi, S., Verification Environment for ALTIROC ASIC of the ATLAS High Granularity Timing Detector, TWEPP (2023).