IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 21 November 2024, accepted 16 December 2024, date of publication 20 December 2024, date of current version 7 January 2025.

Digital Object Identifier 10.1109/ACCESS.2024.3520877

RESEARCH ARTICLE

Fail-Safe Logic Design Strategies Within Modern FPGA Architectures

PRIYA A. BHAKTA[1], **(Student Member, IEEE), JIM PLUSQUELLIC**[1], **ANDREW SUCHANEK**[2], **(Senior Member, IEEE), AND TOM J. MANNOS**[2]
[1]Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM 87131, USA
[2]Sandia National Laboratories, Albuquerque, NM 87123, USA

Corresponding author: Jim Plusquellic (jplusq@unm.edu)

This work was supported in part by the Research and Development through the Laboratory Directed Research and Development (LDRD) Program, Sandia National Laboratories, under Grant LDRD 225963; and in part by the Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC (NTESS), a wholly owned subsidiary of Honey well International Inc., for U.S. Department of Energy's National Nuclear Security Administration (DOE/NNSA) under Contract DE-NA0003525.

ABSTRACT Fail-safe computing refers to computing systems that revert to a non-operational safe state when a fault occurs. In this paper, we investigate a circuit level technique as mitigation for single event upsets (SEUs) and fault injection attacks on field programmable gate arrays (FPGAs), and analyze the effectiveness of the technique as a fail-safe monitor for an encryption algorithm. The propagation of fault effects through FPGA primitives including lookup tables (LUTs) and programmable interconnect points (PIPs) is assessed within an FPGA architecture created using an open source tool, and validated using fault injection experiments on an FPGA. The analysis reveals additional vulnerabilities exist within reconfigurable architectures over those in equivalent fail-safe application specific integrated circuit (ASIC), thus requiring a more elaborate network of redundant circuits and checking logic. The configuration memory bits (CMBs), which configure routing and designate logic functions within the LUTs of the FPGA, add complexity to fail-safe design strategies by introducing additional fault conditions and fault propagation paths. A resource-efficient fail-safe circuit design technique called DEsign for Fail-safe in reCONfigurable systems (DEFCON) is proposed. The benefits and limitations associated with DEFCON are described in the context of fault injection experiments carried out as simulations and in FPGA hardware.

INDEX TERMS DEFCON, duplication-with-comparison, fail-safe, fault injection attacks, single-event-upsets, fault tolerance, dynamic partial reconfiguration.

I. INTRODUCTION

FPGAs are susceptible to single-event-upsets (SEUs) caused by cosmic radiation, as well as adversarial attacks, via fault injection attacks. Naturally occurring SEUs threaten the reliability of the device in carrying out critical system functions. On the other hand, adversaries can purposely introduce SEUs for the purpose of stealing sensitive information including encryption keys and inject faults in an attempt to unlock access to the implemented design. To counter these reliability issues and safeguard the data integrity of FPGAs, effective embedded reliability and security monitoring features must be integrated into FPGA designs. In this research, a fail-safe circuit technique is proposed called DEsign for Fail-safe in reCONfigurable systems (DEFCON), which is designed to detect faults and to revert the system to a safe operational state when faults occur.

Fail-safe systems need to incorporate redundancy and self-checking capability, while still optimizing the speed, power and area of a design. The default action on detection of a fault is to sound an alarm, halt the system and drive the outputs to a safe operational state. A relevant example is the control system implementing autonomous driving features in vehicles, where the detection of a fault deactivates the autonomous driving system and either brings the vehicle to a stop or hands control over to a human driver. In complex

The associate editor coordinating the review of this manuscript and approving it for publication was Gian Domenico Licciardo.

system architectures, the detection of a failure, the signaling of an alarm, and the mitigation process associated with reverting the system to a safe state nearly always requires the insertion of redundant components.

The goals of a fail-safe paradigm are distinctive from the goals of a fault tolerant paradigm. Fault tolerance refers to the ability for a system to continue operating with full functionality in the event of a single fault that causes some component(s) in the system to fail. Fault tolerant systems that meet this criteria are often referred to as fail-operational [1]. The most common fault tolerant technique uses triple modular redundancy (TMR) where the system is triplicated, and a majority voting mechanism is used to force the device outputs to the values generated by two of the redundant copies that agree, and to disregard the redundant copy that disagrees. The system in this case is able to continue operating correctly despite the occurrence of the fault. As an example, TMR is commonly used to provide protection against faults in critical components of aircraft flight control systems, enabling the aircraft to operate without mishap if one of the three copies fails.

In contrast, fail-safe design methods are typically employed in systems that have zero tolerance to faults, and which may require sophisticated failure recovery mechanisms, e.g., in control systems for nuclear power plants or nuclear weapons. Here, the default action of a fail-safe system is to sound an alarm and revert the system to a safe state. This enables corrective actions can take place from a known safe starting state, which helps guarantee reliable and correct restoration of system operation.

Redundancy is also used in fail-safe design to detect failures. However, since mitigation is typically delegated to a separate controller, and only detection is required, more resource efficient methods referred to as duplication with comparison (DWC) can be used [2], [3]. Reducing the number of redundant copies to two, in contrast to TMR, and the elimination of the majority voting component, improves resilience of the fail-safe system by virtue of its smaller footprint. Note that the mitigation component of a fail-safe system increases the size of the footprint, partially offsetting this benefit. However, in the limit, the mitigation component may be very small, e.g., a hardware reset, which would make the fail-safe system smaller compared to a TMR equivalent. Unfortunately, the actions taken by the mitigation strategy are application dependent, making it difficult to draw a conclusion with respect to the overall size of DWC versus TMR-based systems.

The goal of our work is to demonstrate the effectiveness of a fail-safe system design within FPGAs, and to minimize its footprint by optimizing the detection and alarm-generating circuit structures. Although FPGAs make the task much more difficult over an ASIC implementation, the benefit of enabling in-field updates as new technologies emerge, e.g., a new post-quantum encryption algorithm is embraced as the new standard, is a significant driver to finding a solution to fail-safe system design on an FPGA.

A key feature of the proposed DEFCON technique is the instantiation of two XOR checker circuits in one FPGA LUT, and the ability of both XOR checker gates to function properly despite a fault that may disable one of two copies. XOR checker gates are heavily used as the comparators in redundancy-based techniques including both DWC and TMR, to decide if the two redundant copies of the monitored functional unit outputs are identical. To handle faults that occur in the XOR checker network itself, two copies of an XOR checker are inserted for each pair of output bits monitored in the functional unit. Therefore, XOR checker circuit components are the dominant components in the fault detection circuitry of fail-safe systems, and techniques designed to significantly reduce the size of these bit-wise checkers, while preserving their independence with respect to faults, represents an important contribution.

The properties and requirements associated with fail-safe systems make it difficult to use automated circuit design flows to implement a design while minimizing area overhead. In this paper, we investigate fail-safe design on FPGAs at the circuit level using an open source FPGA synthesis tool called OpenFPGA [4], [5]. FPGA programming features that enable reconfigurability also increase the difficulty of ensuring the redundant components are independent, e.g., the possibility of fault propagation through unused programmable resources needs to be considered. Moreover, failure mechanisms that upset the configuration state can change logic functions and routing characteristics of the design, and therefore the self-checking capabilities of the monitoring circuit are more complex.

We first investigate fault injection and propagation using an open-source FPGA design tool because the analysis of fault propagation paths requires detailed knowledge of the underlying reprogrammable circuit structure, and such information is not available to end users of commercial FPGAs. The DEFCON technique is then implemented on a Zynq 7010 FPGA, and integrated with a DWC version of the advanced encryption standard (AES) algorithm as the function to be protected against faults. Both simulation and hardware experimental results are presented that demonstrate the benefits and limitations of the DEFCON circuit. The contributions of this work are summarized as follows.

- An implementation and analysis of a fail-safe DWC technique called DEFCON on both an open source FPGA and a commercial Xilinx Zynq architecture, in which both the monitored functional unit outputs and the DEFCON circuit itself are protected separately against single fault occurrences.
- A resource-optimized version of the XOR checker circuit, in which both copies of the redundant XOR gate are instantiated into one lookup-table (LUT) on the FPGA, significantly reducing the overhead associated with the checker circuit. The XOR checker construction technique guarantees fault independence of the two pairs of inputs used by the two XOR gates implemented within each LUT.

- A layered redundancy scheme, referred to as quad-redundancy, which enables 6-input LUT implementations of the redundant XOR gate to support recovery and continued system operation in cases where faults occur within the components of the DEFCON circuit itself. Faults which occur within the DEFCON monitor are considered non-critical faults, and therefore, a mitigation strategy that fixes them instantly is proposed.
- Simulation and hardware experimental results showing resource utilization and the effectiveness and limitations of the DEFCON technique. In addition, a FPGA implementation of the most closely related fail-safe technique is implemented and the results compared with those presented for DEFCON.

We would like to point out that the fault injection strategy used in the simulation and hardware experiments introduces faults only in the configuration memory bits (CMBs) of the FPGA, and not in the flip-flops (FFs) utilized by the functional units. This is justified because any fault that changes a stored value in a memory component of the state machine or data path of the functional units will change the result computed by one of the redundant copies, and will always be detected by the DEFCON monitor. On the other hand, a fault occurring in a CMB may change the circuit implementation characteristics of a functional unit, and can result in unexpected behaviors, as we will show. Therefore, CMB faults represent the critical components to evaluate in any type of FPGA-based fail-safe circuit design strategy.

II. BACKGROUND

When evaluating fail-safe techniques in FPGAs, two important metrics are considered: overhead costs and detection capabilities. Redundancy within FPGA modules have been shown to have better performance in detecting faults in the configuration memory, compared to error detecting code algorithms, while requiring smaller overhead [6]. The work in [6] presents 5 fault detection strategies evaluated on a substitution box (S-Box) in the AES algorithm, most notably the duplication-with-comparison (DWC) and triple modular redundancy (TMR) strategies. As expected, the DWC method uses approximately 2X the resources over the unprotected design while TMR increases resource utilization by approximately 3X. The authors conclude that DWC is the overall better-performing fault detection technique in terms of overhead and robustness.

A DWC technique blue suitable for fail-safe system design is proposed for FPGAs in [2] and [3] for detecting upsets in state, i.e., bit values stored in flip-flops (FFs), block RAM (BRAM) and configuration memory. The method duplicates the functional unit and alarm signals, and incorporates self-checking comparators that flag differences in duplicated signal components, e.g. primary outputs and those within feedback paths of the design. This fail-safe circuit design technique, referred to subsequently as the BYU method, is the most closely related technique to DEFCON. We implement the BYU method on our FPGAs and present a comparative analysis in Section V.

Published techniques that target fail-safe operation are very limited. Here, we describe the closest related FPGA-based fault tolerance techniques. As indicated earlier, fault tolerance methods can be characterized as fail-operational, where the goal of the proposed circuit design techniques is to maintain system operation despite the presence of a fault.

Several FPGA-based methods attempt to extend the mean-time-to-failure (MTTR) of a faulted system, using fault masking techniques, e.g., [7] and [8]. Similar to DEFCON, Lee et al. [7] propose encoding duplicate logic functions into a single LUT and propose using both LUT outputs. The two duplicated outputs are either ANDed or ORed together in a downstream LUT, as a means of fault masking 0-to-1 and 1-to-0 single event upsets (SEUs), respectively. An integer linear program is proposed to finalize the optimal duplication and encoding scheme that attempts to minimize the fault rate. A related scheme is proposed in [8] that additionally leverages the embedded carry chain components within configurable logic blocks. Here, logic functions are decomposed into two subfunctions, which are combined in the carry chain using AND or OR gates by controlling the carry-in to 0 and 1, respectively. The technique does not require changes to the placement and routing, unlike [7] where downstream AND and OR functions are needed, therefore routing congestion is reduced. In contrast to DEFCON, neither of these methods attempt to encode the LUT logic function to provide fault independence on the inputs to the LUT.

The authors of [9] determined that CMBs for routing resources represents 90% of the total number of CMBs in the FPGA, and therefore are highly vulnerable to SEUs. Moreover, they found that about 10% of the SEUs that upset routing CMBs produce multiple short and open faults on PIPs that cannot be corrected by TMR because the single fault assumption of TMR is violated. They propose a reliability-oriented place and route algorithm that prevents multiple errors from a single PIP fault from impacting TMR effectiveness.

The authors of [10] characterize SEUs in FPGA LUTs and their interconnects. LUT SEUs only introduce a fault when a particular cell is selected, but configuration logic blocks (CLBs) have intra-CLB routing that is fully connected and MUX-based, so SEUs can cause an irrelevant signal to be selected and a fault introduced. Inter-CLB routing is typically interconnected via bidirectional pass transistors, while SEU disconnect (open) faults are typically modeled as temporary stuck-at-0/stuck-at-1 faults, which are pulled high or low to prevent crowbar current in downstream gates. SEU short faults bridge two adjacent wires and are the most complex to characterize since they are created when two drivers oppose opposite values and downstream gates interpret them differently. Nets that fanout further complicate the model, providing the opportunity for multiple faults to be injected.

Modern FPGAs possess very large configuration bitstreams, which increases the amount of time required to find errors using traversal scrubbing techniques. A rapid scrubbing technique is proposed in [11] which utilizes the mapping between critical circuits protected with DWC and configuration frames.

The work in [12] presents challenges with hardware emulated fault injection techniques in commercial FPGAs. Hardware fault emulated techniques can be exhaustive but time consuming, as each bit in the bitstream must be flipped and then restored. In addition, there are cases where the device will lock-up, requiring a complete power cycle. The authors present an approach for building an accelerated, open source fault injection platform called ParFlip using reconfigurable SoC devices.

A fault injection-based countermeasure is proposed in [13]. The authors propose a parity-based technique to monitor critical components in an AES engine for fault injection attacks. Faults are inserted into each byte and are monitored after every operation and round of encryption, using a parity-based method. The output of each operation in the AES engine is compared with the parity outputs using XOR logic to flag any difference. This technique ensures extensive fault coverage by evaluating the outputs after each computation step within each round, making the detection robust against single faults.

Another hardware redundancy-based fault detection strategy is presented in [14]. This work proposes a fault detection strategy to enhance security for the SHA-512 hash algorithm against fault injection attacks, while maintaining area efficiency and overhead costs. In this approach, the authors use the DWC method, but instead of duplicating the entire functional unit output, they focus only on the critical components to minimize overhead. Additionally, time redundancy is applied using both normal time redundancy, where SHA-512 operations are repeated and compared at different times, and reverse time redundancy, which verifies correctness by subtracting results in subsequent steps. The proposed scheme was tested on a Xilinx Virtex-II Pro FPGA, and similar to our work, fault injection simulations were performed, where up to 20 faulty bits were randomly introduced into the input data to assess the fault detection scheme, achieving 99.99% fault coverage. The results showed a 1.39% decrease in frequency, a 2.94% increase in area, and 98.61% throughput retention.

A. A UNIFIED APPROACH TO FAIL-SAFE, FAULT TOLERANCE AND FAULT INJECTION ATTACKS

Methodologies designed to address fail-safe system design, fault tolerance and fault injection attacks individually within FPGA frameworks can be expanded in scope to address all three areas using a unified approach. For example, all of these areas are concerned with SEUs, independent of whether they occur naturally or are induced by glitching the supply voltage or clock, or utilizing electromagnetic radiation or lasers. For fail-safe and fault injection attacks, DWC methods that assert an alarm are adequate because the system needs to halt operation in either case, and remedial actions need to follow that maintain a non-operational status, restart the system and/or alert authorities.

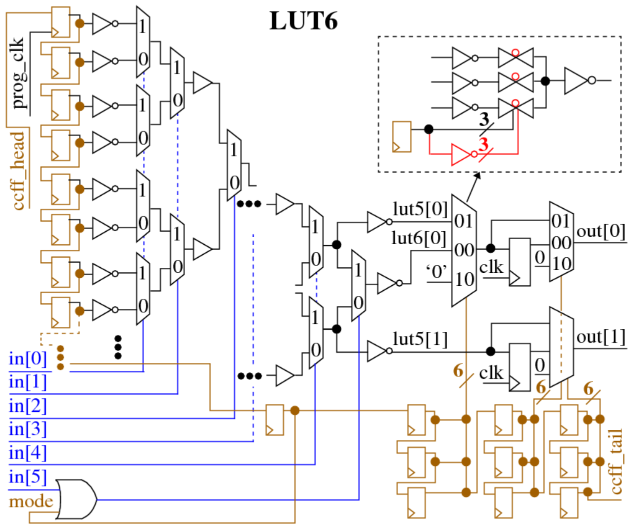

FIGURE 1. OpenFPGA 6-input LUT with register logic within the CLB.

Although fault tolerance requires the system to continue to operate, DWC techniques that flag fault conditions can be used here as well, e.g., to enable an unused redundant copy to be utilized as a means of minimizing downtime [15]. For any of these scenarios, techniques that add resilience or implement countermeasures need to have a small footprint in order to minimize their impact on the speed, power and area of the original design, and to minimize their exposure to upsets that occur in the protection circuitry itself. The primary goal of DEFCON is to achieve this objective in the context of FPGA designs.

III. OPENFPGA FRAMEWORK

The OpenFPGA synthesis framework is used in this paper to create an FPGA architecture for the simulation-based experiments and analysis [4], [5]. The python-based tool flow utilizes an XML-based architectural description to specify the details of the CLBs, LUTs, FFs, routing architecture, I/O and custom hardwired IP blocks. OpenFPGA accepts a circuit description and creates an FPGA architecture large enough to accommodate the circuit elements. The Verilog-to-routing (VTR) CAD tool [16] is used within OpenFPGA to generate the Verilog netlist and the programming bitstream that implements the circuit description. The netlist can then be processed into a layout using a standard cell library-based place-and-route (PNR) CAD tool flow.

Although the OpenFPGA synthesis tool supports a wide range of components, we utilize only the CLBs, routing components and I/O in the development of the proposed DEFCON technique. The specific implementation characteristics of the CLBs, which include LUTs and a local routing network, as well as the PIPs that define the global routing network, are needed to fully evaluate fault effects and fault propagation. This section describes the circuit structure created by OpenFPGA using an example FPGA configuration test case provided in the distribution. The test case includes ten 6-input LUTs, which are enclosed within one CLB, four connection and switch boxes and a set of 32 I/O.

A schematic of the 6-input LUT (LUT6) is shown in Fig. 1. The programming bitstream configures the column of FFs on the left side with a logic function using the configuration chain (ccff_head), and other elements highlighted in brown. A sequence of transmission-gate 2-to-1 MUXs is used to implement the look-up table, with columns of amplifying buffers inserted after every two stages of MUXing. The LUT6 can be programmed to provide an upper-half lut5 or lut6 function on out[0] and a lower-half lut5 function on out[1]. The two outputs can optionally be registered. The cone-shaped circuit structure associated with the two sequences of 2-to-1 MUXs in the upper and lower halves of the LUT6 makes the out[0] and out[1] outputs structurally independent except for the shared inputs; a property that is leveraged in the proposed DEFCON scheme.

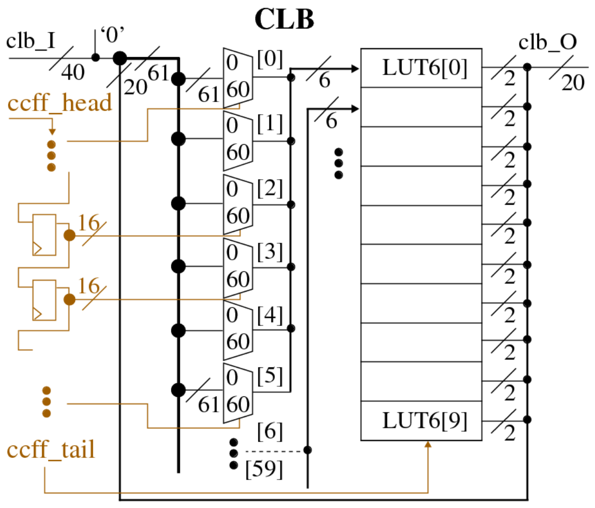

FIGURE 2. CLB architecture created by OpenFPGA, with 10 LUT6 and local routing.

The CLB architecture is shown in Fig. 2. The configurable routing network allows any of the 20 LUT6 outputs to be connected locally with any of the 60 LUT6 inputs within the CLB. The 61-to-1 MUXs, which drive the LUT6 inputs, add 40 external inputs from other CLBs and I/O, and a constant '0', as configuration options. All routing within the CLB is fan-out free, and there are no instances of reconvergent-fan-out in the circuit structures. Reconvergent-fan-out describes a circuit topology in which inputs fan-out to a logic gate network and reconverge downstream on gate inputs closer to the outputs. The lack of reconvergent-fan-out reduces the complexity of designing fail-safe circuits.

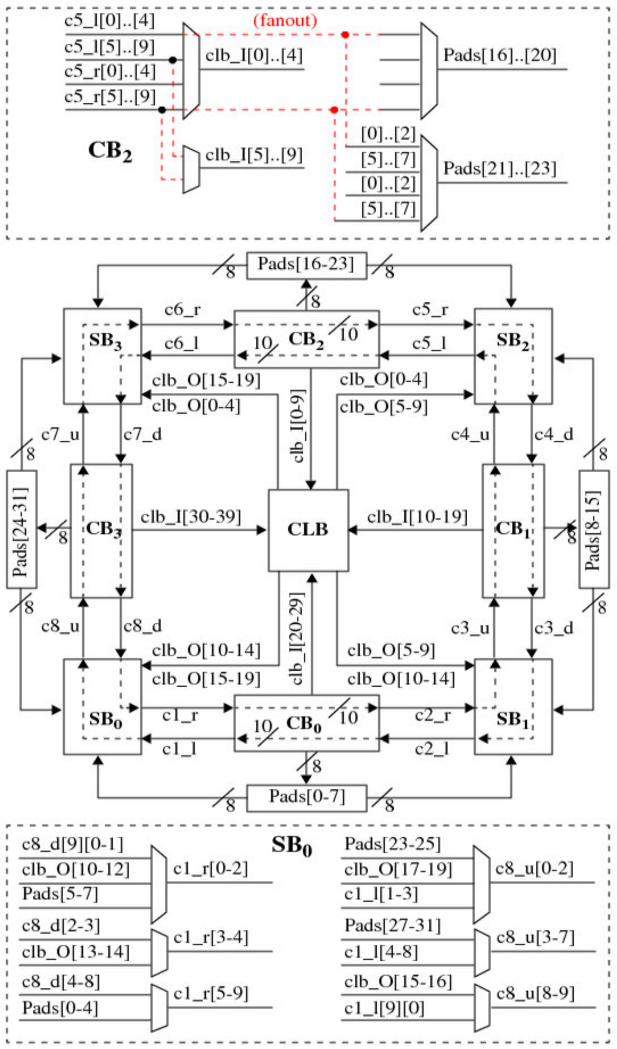

FIGURE 3. Routing architecture created by OpenFPGA.

The FPGA architecture used in our simulations is given in Fig. 3, which shows a single CLB surrounded by a set of four connection blocks ($CB_{0,3}$) and switch blocks ($SB_{0,3}$). The MUXing details of CB_2 and SB_0 shown in the call-outs illustrate that signal fan-out is implemented in the CBs, and the CB and SB provide only a partial set of routing options, in contrast to the fully interconnected routing network provided within the CLB. The I/O block includes output pads driven from the CBs and input pads, which enter the internal routing network via the SBs. The configuration chain that controls the select inputs to the MUXs is not shown for clarity nor are the implementation details of the MUXs. These architectural details are important to understanding fault effects and are covered by examples in the following section.

A. DEFCON IMPLEMENTATION ON AN OPEN-SOURCE FPGA

In this section, we describe the DEFCON circuit design and its implementation on an FPGA. We evaluate its effectiveness

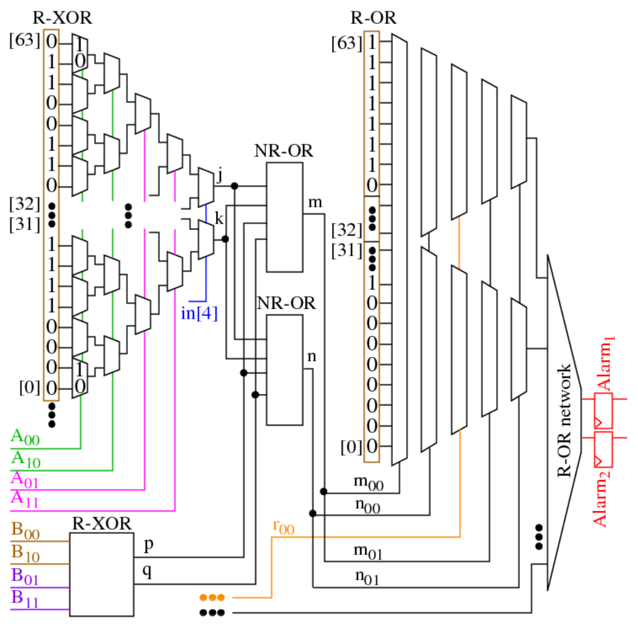

FIGURE 4. Proposed DEFCON fail-safe circuit design.

in providing fail-safe operation using simulation experiments, i.e., whether it is able to detect a fault on the outputs of the functional units it protects, and within the monitor itself. The design ensures detection of stuck-at and CMB faults that are active for one or more clock cycles, and in such cases, ensures that the alarm signal(s) will assert at most two clock cycles after fault activation.

1) XOR CHECKER COMPONENT

A schematic of the proposed DEFCON circuit is shown in Fig. 4. The DWC circuit consists of a set of redundant 2-input XOR gates (R-XOR), each designed to monitor a pair of redundant signals, A_{00} and A_{10}. The A_{00} and A_{10} connect to the same 1-bit outputs of a pair of redundant functional units, which represent the signals that DEFCON is designed to protect (not shown). The functional units might be, for example, a redundant pair of electronic control units (ECUs) in a vehicle and the output signals might be control signals to the anti-lock brake system or a driver assistance feature. The inputs A_{01} and A_{11} are the same two redundant functional unit outputs replicated again at the output of the functional units. The rational for this quad-based replication strategy is discussed in the following sections.

The R-XOR LUT6 in the upper left of Fig. 4 is programmed to implement a special version of a 2-input XOR gate. As mentioned earlier, the structure of the MUX network defines two independent logic cones for the out[0] and out[1] signals from Fig. 1, labeled here as j and k, but the LUT6 is typically programmed without regard to the fault dependence or independence of the LUT's inputs. For example, if a fault occurs on A_{00} within the LUT6, both cones of logic will be affected unless the CMB of the LUT6 is programmed in a special way.

In order to prevent any specific input from affecting both logic cones, the CMB of the LUT6 is programmed such that the 2-input XOR truth table function is replicated multiple times. Here, the upper 32 bits of the CMB repeat the 2-input XOR functional output pattern "0110". Although only the first 8 cells are shown, the pattern is in fact replicated 8 times over the upper 32 CMB as "0110 0110 0110 0110 0110 0110 0110 0110". The lower 32 bits of the CMB are also programmed to implement a 2-input XOR but locally replicate each truth table output value four times in consecutive CMB cells. The full bit sequence for the lower half of the CMBs is "0000 1111 1111 0000 0000 1111 1111 0000".

This specific configuration bit pattern removes the dependency of the two logic cone outputs on the four inputs shown in the figure, and allows a single LUT to implement a fully self-checking comparator, i.e., one LUT for each duplicated functional unit output signal that is monitored. Here, only inputs A_{00} and A_{10} affect the functional output value of the top cone, and only inputs A_{01} and A_{11} affect the output of the bottom cone. For example, if $\{A_{00}, A_{10}\}$ is "01" and $\{A_{01}, A_{11}\}$ is "01", i.e., a fault occurred on the functional unit outputs, and a second stuck-at-1 fault occurs on A_{00} (which disables the top cone because the top cone inputs are now "11"), the bottom cone will still propagate a '1' to k because A_{01} and A_{11} drive the inputs of an independent XOR function. This is true because the programming bit pattern for the lower cone replicates the '1' across all four selections associated with the A_{00} and A_{10} inputs, making these inputs irrelevant to the logic value selected by the A_{01} and A_{11} inputs. Therefore, the fault cannot be masked.

Also note that the proposed redundancy scheme also detects faults that occur in the CMB. For example, if a CMB flips from '1' to '0' in one of the XOR pattern sequences of either logic cone, and a second fault occurs in one of the monitored functional unit outputs, one of the redundant logic cones will always detect the fault and will generate a '1' on either the j or k output.

Under the single fault assumption, alternative programming patterns are possible while preserving the fail-safe property. For example, the upper half pattern "0110 xxxx xxxx 0110 0110 xxxx xxxx 0110" also works, where x denotes 'don't care', because assuming A_{01} and A_{11} are fault-free, only patterns "00" and "11" are possible. If a fault occurs on A_{00} or A_{10}, then either the first or last of the two CMB regions is selected by A_{01} and A_{11}. However, a benefit of specifying the fully replicated XOR pattern, is that it enables the detection of two faults, one on each pair of inputs.

As noted in Fig. 1, the *lut*5[0] signal passes through an additional 3-to-1 MUX. The DEFCON configuration of the *in*[5] and *mode* signals is '1' from Fig. 1, and for the select inputs of the 3-to-1 MUX, it is "01". A stuck-at-0 fault on the output of the OR gate driven by the *in*[5] or *mode* input signals is masked and made harmless by the 3-to-1 MUX configuration. A CMB fault on the 3-to-1 MUX select signals is ignored for the faulty case "00" since the *in*[5] and *mode* signal assignments select the output of the upper logic cone on both the "01" and "00" inputs. If, on the other hand, a CMB fault selects the hardwired '0' on the "10" input, the fault will be masked in the upper cone. However, the bottom cone will still detect any fault that occurs on the outputs of the monitored functional units, i.e., only a second checker fault is masked in this scenario.

The two outputs of the LUT6 are optionally registered as shown on the far right of Fig. 1. The fail-safe configuration for the 2-to-1 select MUXs is 0, which allows the lut5 signals to bypass the registers. If a fault occurs in a CMB associated with the output MUXs, forcing one of *out*[0] or *out*[1] to be registered, the alarm signal will be delayed by one clock cycle. However, similar to the masking scenario discussed above, the fault-free output will always detect a functional unit fault immediately.

2) ALARM SIGNAL COMPRESSION NETWORK

The output signals from the R-XOR gates are OR'ed together using non-redundant OR (NR-OR) gates. Two LUTs configured as NR-OR gates are shown in the center of Fig. 4. Unlike the R-XOR gate, the redundant inputs of the NR-OR gates are located in two different LUTs. This alternative form of redundancy allows the output pairs of up to three R-XOR output signals to be combined (or collected) using only two LUTs, i.e., a 3-to-2 compression ratio.

In contrast, the redundant version of the OR gate (R-OR) shown on the far right in Fig. 4 can monitor only 2.5 R-XOR gate output pairs, to provide a 2.5-to-2 compression ratio. This is true because only 5 of the 6 inputs can be used in the R-OR version because both lut5 outputs are used. Therefore, the NR-OR version reduces overall system overhead. Note, however, that both versions guarantee that at least one of the alarm output signals will be asserted if a functional fault occurs even if a second fault occurs internally within a LUT, e.g., a node is stuck-at or a CMB bit flip occurs.

The CMB programming sequence for the R-OR gate, shown on the right side of Fig. 4, is different than the R-XOR gate in two ways. First, it implements the OR logic function, and second, it is setup to handle a fifth input. The upper cone is programmed with the sequence "11111110 11111110 11111110 11111110", while the bottom cone is programmed with the sequence "11111111 11111111 11111111 00000000". A portion of both sequences is shown in the figure. These sequences implement two fault-independent functions, i.e., a 3-input OR gate in the top cone and a 2-input OR in the bottom cone.

The NR-OR and R-OR network components define a hierarchy of 3-to-2 and 2.5-to-2 compression functions, which eventually combine all of the R-XOR redundant outputs to two alarm signals, Alarm_1 and Alarm_2, shown on the far right of Fig. 4. The number of gates and levels in the hierarchy depend on the number of functional unit output signals monitored and the number of NR-OR vs R-OR gates used for compression of the alarm signals.

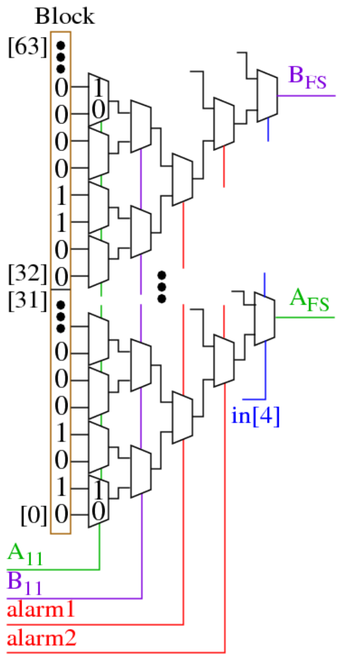

FIGURE 5. **DEFCON functional unit output blocking design.**

3) OUTPUT SIGNAL CONTROL IN DEFCON

DEFCON incorporates a mechanism to force the functional unit output signals to a fail-safe state once a fault is detected and one or both alarm signals are asserted. In the following, we assume the fail-safe state for functional unit outputs is '0,' but any bit-pattern is possible.

The output blocking circuit for two functional unit outputs, *A* and *B*, is shown in Fig. 5. The LUT6 (labeled *Block*) again makes use of the dual LUT6 outputs. One copy of the redundant functional unit output signals is routed to the LUT6 inputs along with the two alarm signals. The truth table function is coded to pass the current state of A_{11} and B_{11} under the condition that both alarms are '0', and to force the lut5 outputs to '0' otherwise. The input labeled *in*[4] is used in a quad-redundancy scheme discussed below.

As we will show in the results section, under dual-fault conditions, where one fault occurs on a functional unit output and a second occurs in the DEFCON monitor, there are a small number of cases where DEFCON fails to block the functional unit output signals. In particular, we have observed in our simulation experiments that there are a small number of DEFCON monitor faults that allow the functional unit outputs to be stuck at the non-fail-safe value ('1' in our experiments) despite the fact that one or both of the alarms are asserted. Similarly, several instances occur when a fail-safe output signal, e.g., A_{FS}, is erroneous even when neither of the alarms are asserted. Unfortunately for this second scenario, adding fully redundant versions of the Block gate does not allow a decision to be made regarding which of the redundant outputs represent the correct functional unit output value.

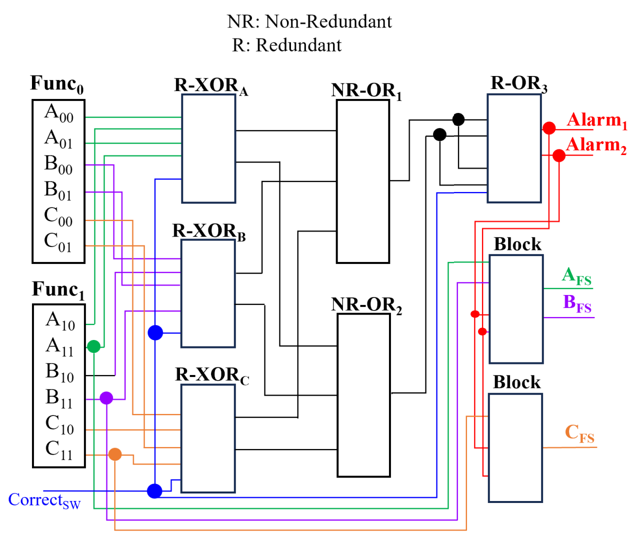

FIGURE 6. Simulation model for validation with fail-safe checks on three functional unit output signal pairs.

4) QUAD-REDUNDANCY IN DEFCON

A notable advantage of the proposed architecture over that proposed in [2] and [3] is the quad-redundancy that exists within each R-XOR LUT. Quad-redundancy allows DEFCON to mitigate SEUs at run-time that might occur in the R-XOR LUTs of the DEFCON monitor. If, for example, only one of the alarm signals is asserted, this is an indication that a fault has occurred in a CMB of the DEFCON monitor itself. The signal labeled *in*[4] in Figs. 4 and 5 can be asserted to verify whether a CMB fault has occurred by switching to the redundant copy in each of the two LUT subfunctions, i.e., by switching to CMB bits {16} through {31} in the upper cone and {48} through {63} in the bottom cone of the LUT6. If the asserted alarm signal returns to 0, then the proposed fail-safe system can continue to operate with full fail-safe detection capabilities in place and delay the repair of the CMB until a later time.

B. SIMULATION EXPERIMENT SETUP

The faults considered in our analysis are upsets in the CMB bits. Note that stuck-open and stuck-closed PIP faults are equivalent to stuck-at faults in the CMB that control the PIPs [17]. Bridging faults between independent wire segments (not connectable via a PIP) are not considered in our analysis. Moreover, the PIP components in the OpenFPGA architecture are unidirectional which eliminates the need to consider the complex fault propagation behavior that results from using bidirectional PIPs [11].

The test circuit configuration used in the simulation experiments is shown in Fig. 6. The outputs from two redundant copies of a functional unit are labeled $Func_0$ and $Func_1$. A DEFCON circuit configuration is constructed to monitor the three 1-bit redundant outputs signals from the functional units. The redundant output signals A, B and C are each replicated at the inputs of the FPGA to remove single point of failures along their routes within the FPGA. All twelve signals enter the FPGA through I/O pads as shown in the OpenFPGA architecture of Fig. 3. This models system architectures that use the FPGA as a board-level monitor for a pair of synchronized microprocessors, all as separate devices, or architectures in which the functional units are included within the FPGA architecture itself.

The DEFCON implementation utilizes six LUTs from the array of ten contained within the CLB (see Fig. 3) to create three R-XOR gates, two NR-OR gates and a R-OR gate using the CMB sequences described earlier. Two additional LUTs are used to implement the ‘Block’ gates, as shown in Fig. 6. The routing MUXs within the CLBs, SBs, and CBs, are programmed to create the connection framework as shown. The CMB chain is 2,562 bits in length, with 740 bits dedicated to configuring the 10 LUTs in the CLB, 960 bits for configuring the local routing MUXs within the CLB, 32 bits for defining the direction of the I/O pads, 400 bits for configuring the SB MUXs and 430 bits for configuring CB MUXs. The two alarm signals, $alarm_{1/2}$, represent the output of the DEFCON monitor, while A_{FS}, B_{FS} and C_{FS}, represent the fail-safe outputs of the functional units. The alarm signals can be routed to a kill switch or a module that implements a recovery procedure.

We analyze the fail-safe properties of DEFCON by simulating faults in the structural netlist representation of the OpenFPGA architecture shown in Fig. 3. Faults are introduced into the simulation model in one of two ways. First, faults are created on the outputs of the functional units by assigning opposite values to a pair of redundant outputs, e.g., $\{A_{00}, A_{01}, A_{10}, A_{11}\}$ = “0011” (note that the replicated wires, e.g., $\{A_{00}, A_{01}\}$ are always assigned the same value). There are 8 fault-free assignments in which the output pairs are assigned the same value and 24 faulty assignments where exactly one of the outputs of a pair is assigned opposite values, i.e., a single-fault model within the functional units is assumed. We refer to these signal input components of the simulation model using the term *FuncUnit*.

Faults are also introduced into the DEFCON monitor itself. For each of the *FuncUnit* test scenarios, faults are simultaneously introduced, one-at-a-time, into the CMB chain by flipping one of the 2,562 CMB bits. We use the term *FPGAUnit* in reference to these components of the simulation model. Therefore, the fault simulation experiments are carried out under a dual-fault model, with exactly one fault introduced into the *FuncUnit* and one in the *FPGAUnit*. The simulation results report the number of detections, i.e., alarm signal assertions, under this dual-fault model.

C. SIMULATION EXPERIMENT CHALLENGES

A well known challenge with simulating faults in FPGA designs is the creation of combinational loops, which occur when the fault creates a connection between an output and an input to the same combinational logic network. The flexibility within the routing architecture of an FPGA makes

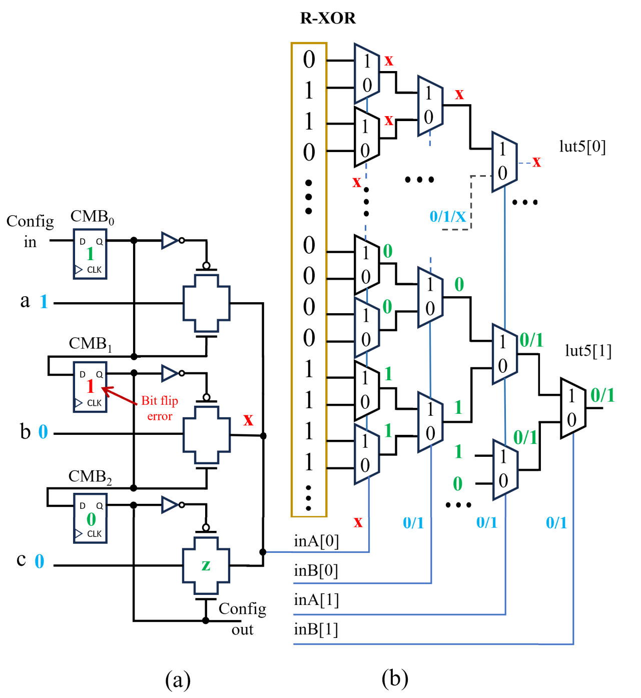

FIGURE 7. **(a) An example of a one-hot MUX implementation in which a CMB bit flip enables two transmission gates to drive the MUX output, with input *a* driving a 1 and input *b* driving a 0. The simulator is unable to resolve the logic value on the MUX output and assigns an *x*. (b) A test scenario in which the LUT input *inA*[0] is driven by an *x*, resulting in an output value of *x* on the top cone output (*lut*5[0]). The DEFCON redundancy scheme is still able to evaluate the logic function in the bottom cone with inputs *inA*[1] and *inB*[1], and produces a well-defined 0 or 1 on the output (*lut*5[1]).**

this possible. Faults that create combinational loops prevent the simulator from producing any results, and therefore are removed from the set of simulated faults in our analysis. The number of simulation failures is reported in a separate column of the results table presented in the next section.

One-hot, pass-gate MUX implementations represent a second challenge when simulating faults in FPGAs. Routing configurability within FPGAs is implemented using MUXs within CLBs, CBs and SBs. Therefore, a large number of MUXs are needed to implement the various configuration options within the routing network. One-hot, pass-gate MUX implementations are area-efficient but exhibit undesirable characteristics when faults occur, potentially creating shorts and floating output node conditions.

The left side of Fig. 7 illustrates a condition that only occurs within pass-gate implementations of MUXs. Here, a bit flip within the CMB_1 cell enables the second pass-gate, which creates a short between inputs a and b. In hardware, the logic value on the output node is defined by the resistance ratio of the two pass gates, e.g., 0.5 V for a 1.0 V supply voltage. The simulation tool handles this shorting condition by assigning an x (undefined) logic value as shown. A second fault scenario is also possible in which a CMB bit flip disables all pass gates (not shown), resulting in the output node floating to an unknown value in hardware. The simulator in this case assigns a z (high impedance) logic value. In either scenario, pass-gate MUXs introduce additional challenges over gate implementations in fault tolerant and fail-safe system designs.

Fortunately, the redundancy scheme proposed in DEFCON is able to operate correctly despite both types of fault conditions. The right side of Fig. 7 shows the behavior of the LUT outputs $lut5[0]$ and $lut5[1]$ when the $inA[0]$ LUT input is driven with an x. The output of the top cone is also x because the select inputs to the first stage of MUXs is driven with an unknown logic value, and the CMB inputs to those MUXs possess different values. The bottom cone, on the other hand, is able to operate correctly despite the fault and generates a well-defined '0' or '1' on the $lut5[1]$ output, which is determined by the values of inputs $inA[1]$ and $inB[1]$. This is true because the first (and second) stage MUX inputs are all driven with the same input value, making the actual value on the faulted input $inA[0]$ irrelevant.

Unfortunately, the simulator is not able to determine that the value on $out5[1]$ is well-defined, and will make a conservative assignment of x to both $lut5[0]$ and $lut5[1]$. The simulator propagates x through the remaining components of the netlist, resulting in a large number of x assignments to the $alarm_1$ and $alarm_2$ signals from Fig. 4.

We address this issue by adding assertions to the simulation model. The assertions monitor the input values of the six LUTs in the DEFCON design and re-assign a 0 or 1 to $lut5[0]$ or $lut5[1]$ when conditions similar to those shown in Fig. 7 are met. For example, $lut5[1]$ is re-assigned a 0 if $inA[1]$ and $inB[1]$ are the same logic value and 1 otherwise, eliminating the x assignment. By adding these assertions, we were able to reduce the number of x assignments to the downstream alarm signals significantly. For example, in the *FuncUnit* simulation with $\{A_{00/01}, A_{10/11}\} =$ "0011" and with B_x and C_x assigned "0000", the number of x assignments to $alarm_1$ without assertions is 257, which is reduced to 35 with the assertions included.

D. SIMULATION RESULTS

As indicated earlier, simulation experiments are carried out using a dual-fault model, in which single faults are introduced in the *FuncUnit* and *FPGAUnit* simultaneously. We present the results for the alarm signals and functional unit outputs separately in the following two sub-sections.

1) ALARM SIGNAL ANALYSIS

The alarm signal analysis focuses on the number of times a functional unit output fault is successfully detected. The results are shown in Table 1 for four of the thirty-two *FuncUnit* test case scenarios. The results for the remaining test case scenarios are very similar to those shown here.

The *FuncUnit* test case labeled "Fault-Free" refers to consistent assignments to the functional unit outputs, e.g., $A_{00/01}$ and $A_{10/11}$ are set to "0000" or "1111", while "Fault X" test cases refer to inconsistent assignments, e.g.,

TABLE 1. Fault detection results for DEFCON.

FuncUnit	At least one Alarm	Alarm_1	Alarm_2	Both alarms x	Missed	Loops	Correctable	Max
Fault-Free	77	69	65	12	-	0	6	2562
Fault A	2562	2511	2513	0	0	4	-	2562
Fault B	2562	2510	2512	0	0	4	-	2562
Fault C	2562	2502	2505	0	0	3	-	2562

TABLE 2. Blocked outputs results.

FuncUnit	Alarms Asserted	Total Blocked	Not Blocked A	Not Blocked B	Not Blocked C	Total	Undefined x
Fault-Free	89	2534	8	7	13	28	903
Fault A	2562	2555	2	2	3	7	251
Fault B	2562	2555	2	2	3	7	223
Fault C	2562	2555	2	2	3	7	231

"0011" and "1100". Although there are eight possible configurations for each row with three functional outputs, we show results for only one configuration, e.g., $\{A_{00/01}, A_{10/11}, B_{00/01}, B_{10/11}, C_{00/01}, C_{10/11}\}$ = "000000000000", "001100000000", "000000110000" and "000000000011", respectively.

The numbers in the columns of Table 1 represent the results across all of the 2,562 CMB fault insertion experiments, with each of the CMB bits flipped, one-at-a-time. Column 2 identifies the number of times at least one of the alarms is asserted, and Columns 3 and 4 represent the number of times $Alarm_1$ is asserted and $Alarm_2$ is asserted, respectively. Column 5 counts the number of faults in which both alarm signals are undefined (x). Column 6, labeled *Missed*, counts the number of faults that deactivate both alarm signals, which includes the faults counted in column 5 as undefined. Column 7 represents the number of test cases where the simulation failed because of combinational loops. Column 9 counts the number of faults that are correctable by asserting the $Correct_{sw}$ signal from Fig. 6.

The Fault-Free results show that only 77 faults cause one or both of the Alarm signals to be asserted. Given these alarm conditions are not linked to failures in the *FuncUnit*, but rather are associated with faults in the DEFCON monitor itself, a small number of fault assertions is a desirable feature. Of these detectable faults, $Correct_{sw}$ is able to instantly repair 6 of the faults that occur within the R-XOR gates.

The *Fault A/B/C* test results given on rows 3 through 5 reflect the effectiveness of the DEFCON monitor. The *Missed* column represents test cases in which the DEFCON monitor fails to detect a *FuncUnit* fault in the presence of a second fault in the DEFCON monitor, where 'fail' is defined as neither of the alarm signals being asserted. The DEFCON redundancy scheme was able to detect all 2,562 CMB faults in all test scenarios, with none of the CMB faults resulting in both alarms being set to 'x'. Moreover, the number of times each alarm signal is asserted is also large, indicating many faults cause both alarm signals to assert. Although a small number of test cases result in combinational loops, with results unknown, DEFCON accomplishes the goal of flagging functional unit failures under the dual-fault model.

Interestingly, the number of times both alarms are undefined for the Fault-Free test scenario is 12, despite the resolution discussed in reference to Fig. 7. The root cause for all 12 cases is related to the dominant value for the OR gates. In nearly all Fault-Free test cases, the inputs to the three OR gates are 0 and x. Logic 0 is the non-dominant value for the OR gate, so if all inputs are 0 and x, the output of the OR gate is x. The NR-OR gates are also not as robust as the R-OR gate because they have only a single output, and an x on either NR-OR output propagates an x to both alarms. Although it is possible to re-wire the inputs of R-OR to prevent this, i.e., by connecting both redundant inputs from each NR-OR gate to the same redundant OR gate inside R-OR, doing so creates combinational loops for some faults. Moreover, these cases are somewhat moot because the x on both alarms would immediately resolve to a 1 on at least one of the alarms if a fault occurs in the functional unit, because 1 is the dominant value of the OR.

2) FUNCTIONAL UNIT OUTPUT SIGNAL ANALYSIS

In this section, we analyze the effectiveness of DEFCON in blocking the functional unit output signals under both the single-fault and dual-fault scenarios described earlier. Using '0' as the fail-safe output value, we simulated the OpenFPGA design using four different test scenarios. The "Fault-Free" *FuncUnit* test case again refers to consistent assignments to the functional unit outputs, but using all '1's in this analysis, i.e., $\{A_{00/01}, A_{10/11}, B_{00/01}, B_{10/11}, C_{00/01}, C_{10/11}\}$ = "111111111111". The "Fault X" test cases refer to inconsistent assignments, namely, "001111111111", "111100111111" and "111111110011", respectively.

The data shown in the columns of Table 2, from left to right, give the *FuncUnit* test scenario, the number of times one or both alarms are asserted, the number of fault simulation cases in which the functional outputs (A_{FS}, B_{FS} and C_{FS}) are successfully blocked, the number of times DEFCON fails to block a functional unit output, the total number of unsuccessful blocks among the test cases and the number of test cases in which the output(s) is undefined.

The Fault-Free results show the largest number of unsuccessful blocks and indeterminate states, but also represent the innocuous cases because the functional unit outputs are fault-free. Here, we observe cases where neither alarm is asserted but one or more functional unit outputs are erroneous. The results shown in rows 2 through 4, and columns 4 through 6 represent DEFCON fail cases. However, the number of dual-fault test cases in which DEFCON fails is small at only 2 and 3. Here, one or both alarms are asserted but DEFCON fails to block one or more of the functional unit output signals. Unfortunately, the number of indeterminate values is significant for these rows, so it is impossible to assess the overall effectiveness of the scheme. Although not shown, DEFCON is successful in blocking all functional unit outputs when only the functional unit outputs are faulted.

E. RESOURCE UTILIZATION

The combination of both redundant (R) and non-redundant (NR) LUT instantiations makes DEFCON a compact architecture. One LUT is required for each pair of functional unit outputs that needs to be monitored. The NR-OR gates provide a 3-to-2 compression ratio, receiving input from up to 3 R-XOR gates and producing two pairs of output signals. The R-OR gates can also be used to provide compression of upstream alarm signals, but as discussed earlier, the compression ratio is somewhat less, at 2.5-to-2, requiring a larger network if used instead of NR-OR gates. However, R-OR gates are more robust to *FPGAUnit* failures because of the dual outputs.

The DEFCON monitor LUT utilization is upper-bounded by *nlogn*. For example, doubling the number of functional unit outputs from 3 to 6 doubles the number of X-OR and NR-OR gates, and triples the number of R-OR gates from 1 to 3. Here, we assume that R-OR gates are used in the final compression stages, resulting in an increase in LUT utilization from 6 to 13. Further reductions in resource utilization are possible by using NR-ORs in all stages of the alarm signal compression network.

IV. FPGA EXPERIMENTAL EVALUATION OF DEFCON

In this section, we implement and analyze the effectiveness of DEFCON on a Xilinx Zynq 7010 FPGA. The Advanced Encryption Standard (AES) algorithm is the selected functional unit for the DEFCON monitor. An open source Verilog HDL description of the AES algorithm is obtained from [18] and synthesized using Xilinx Vivado. A design is constructed in which the AES algorithm is duplicated and instrumented with DEFCON. The AES engines are configured to perform encryption with a 128-bit key.

A block diagram of the experimental design is shown in Fig. 8. The processor side (PS) contains an ARM Cortex A9, which executes a C program under the Linux operating system that serves as the fault injection manager (FIM). A fault-free version of the duplicated AES engine with DEFCON is synthesized into a partial bitstream for run time programming into a dynamic partial reconfiguration (DPR) region in the programmable logic (PL) side of the SoC. The partial bitstream is created using Vivado and transferred

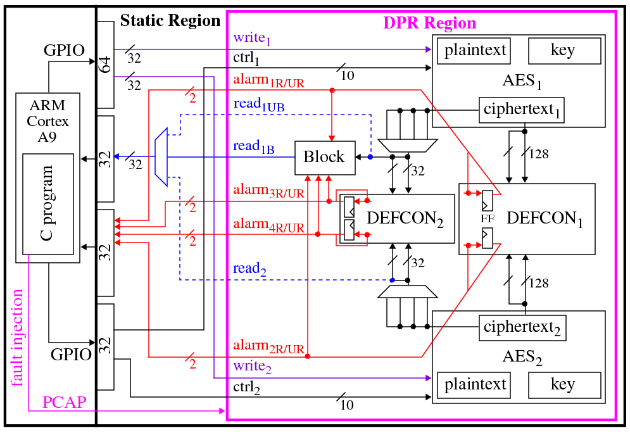

FIGURE 8. Block diagram of the processor, static and DPR programmable logic regions on the Xilinx Zynq 7010 SoC device. A C program reprograms the DPR Region through the PCAP interface, introducing one bit flip into the programming bitstream for each experiment. Two copies of the AES engine are placed and routed together into the DPR region, along with two DEFCON monitors. ***DEFCON*$_1$** compares the two 128-bit internal datapath registers that are updated on each round of encryption, while ***DEFCON*$_2$** monitors the 32-bit readout circuit. A blocking circuit (Block) forces 0's on the ***read*$_{1B}$** bus when any of the 4 registered alarm signals (***alarm*$_{xR}$**) is asserted. The static design routes control and data signals using 4 GPIO registers from the processor to the DPR region.

to a flash memory card on the Avnet ZYBO Z7-10 board [19], [20].

As shown in Fig. 8, the DPR region resources used by the two AES implementations are completely independent, with separate control signals ($ctrl_1$ and $ctrl_2$), 32-bit data input ($write_1$ and $write_2$) and data output ($read_{1B}$ and $read_2$) buses. Similarly, each copy of AES has separate key, plaintext and ciphertext round registers. The control signals and data input buses are connected to separate GPIO channels to prevent Vivado from performing routing-reduction optimizations on these signals. The data output bus is configured with a MUX to enable the $read_{1B}$, $read_2$ and $read_{1UB}$ ciphertext outputs to be read after the encryption. Here, subscripts B and UB refer to blocked and unblocked, respectively. The $read_2$ and $read_{1UB}$ are used only as information to help determine the impact of the inserted faults and would not be included in a fielded version of DEFCON. Similarly, the 4 registered ($alarm_{xR}$) and 4 unregistered ($alarm_{xUR}$) alarm signals would normally be reduced to only two registered alarms signals but are wired out separately to assist with classifying fault behaviors.

The Vivado implementation view of the design is shown in Fig. 9, which includes both a static (left) and *pblock* region (right) surrounded by a magenta rectangle. Xilinx Vivado uses the pblock construct to designate regions as reconfigurable. The annotations given as AES_1, AES_2 and *DEFCON*, illustrate only the general regions in which these elements are placed and routed, where, in reality, these components are actually interwoven with no hard boundaries between them. All routing from the static to dynamic regions and vice versa is constrained to occur only once, i.e.,

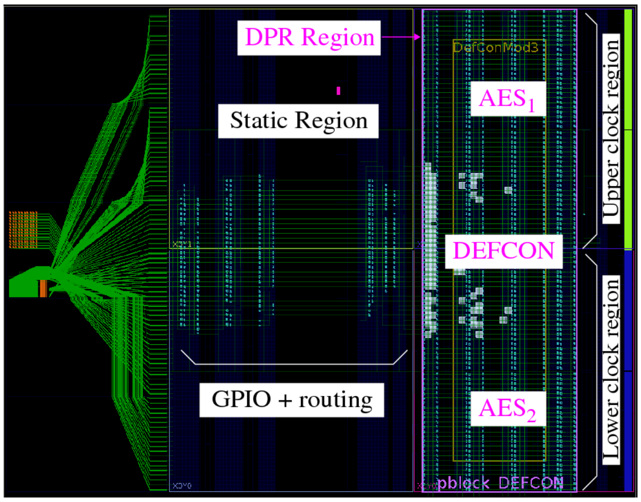

FIGURE 9. Xilinx Zynq 7010 experimental design with two copies of the AES engine and the DEFCON monitor placed in a dynamic partial reconfiguration (DPR) region, highlighted by a magenta rectangle on the right. The static design shown on the left side includes only GPIO registers that enable the C program running on the processor to deliver control and data to the programmable logic and MUXs for implementing the fanout of these signals, which are routed across separate interfaces into the DPR region.

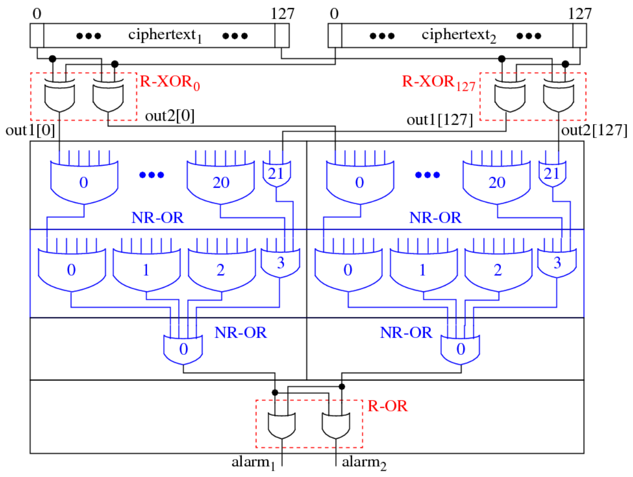

FIGURE 10. Schematic of the *DEFCON*$_1$ which monitors the two 128-bit ciphertext round registers from the two AES engines. Resources used include 128 R-XOR LUTs, 54 NR-OR LUTs and 1 R-OR LUT. The *alarm*$_1$ and *alarm*$_2$ signals connect to registers (not shown) that store '1' if a mis-match occurs during any of the 10 rounds.

no control or data signals are routed into the DPR regions and then back out again. This represents a model in which these signals route to and from I/O pads on the device, in contrast to our setup where these signals route to the processor side of the SoC through GPIO. The size of the design requires a DPR region that spans both the upper and lower clock regions on the right side of the device. Each region possesses 1,554,592 bits of programming data, yielding 3,109,184 fault emulation experiment (FEE) bits. Our FIM is able to perform approximately 6.6 fault emulation experiments per second, requiring nearly 2.7 days to complete all fault emulation experiments in each of the two clock regions.

Two DEFCON checker circuits are added to the hardware design, labeled *DEFCON*$_1$ and *DEFCON*$_2$ in Fig. 8. *DEFCON*$_1$ error checks the 128-bit ciphertext round registers from the two AES engines while *DEFCON*$_2$ error checks the two 32-bit read-out buses. The alarms for *DEFCON*$_1$ are registered to enable any mis-compares during any of the 10 rounds of encryption to record the alarm condition for readout at the end of the encryption operations. Similarly, the ciphertext read-out operation requires 4 cycles of 32-bit reads to complete, and therefore, *DEFCON*$_2$ also registers its alarm signals. The registered alarm signals ($alarm_{xR}$), as well as the end state of the unregistered alarm wires ($alarm_{xUR}$) are read by the C program after the encryption and read-out operations complete, or after a time-out in cases where the fault prevents completion. Note that when DEFCON is deployed in the field, any trigger of an alarm would immediately halt the encryption and/or read-out operations, force the outputs to a fail-safe state, and then initiate a mitigation operation for recovery. For our testing and evaluation, we allow encryption and read-out to complete independent of the state of the alarms, and read out $ciphertext_1$ using the $read_{1B}$ channel, as well as $ciphertext_2$ and the unblocked $ciphertext_1$ using the $read_2$ and $read_{1UB}$, respectively. All three copies of the ciphertext are stored along with the 8 alarm signals to a file for off-line analysis.

A. DEFCON MONITOR DESIGN

The DEFCON monitors are constructed in the same way as those shown in Fig. 4 for the OpenFPGA experiments except the number of bits monitored increases to 128 for *DEFCON*$_1$ and to 32 for *DEFCON*$_2$. Also, the depth of the NR-OR tree circuit is increased to four levels for *DEFCON*$_1$ and three levels for *DEFCON*$_2$. A schematic diagram for *DEFCON*$_1$ is shown in Fig. 10. The resources required to implement it include 128 copies of R-XOR LUTs, 54 copies of NR-OR LUTs and one copy of the R-OR LUT. *DEFCON*$_2$ requires 32 copies of R-XOR LUTs, 14 copies of the NR-OR LUTs and one copy of the R-OR LUT. In total, both DEFCON monitors utilize 230 LUTs.

The *Block* circuit is constructed with redundant deactivation signals as shown in Fig. 11. The two NOR gate outputs are routed to a set of 32 AND-gate LUTs, which gate the passage of the $read_{1UB}$ 32-bit bus to the output signal $read_{1B}$. If any of the registered alarm signals is asserted, the $read_{1B}$ outputs are forced to 0's (we use 0s as the fail-safe output values but any arbitrary assignment is possible).

B. TESTING PROCESS

The actions carried out in software by the FIM are shown by a block diagram in Fig. 12. The following summarizes the fault emulation process.

- The FIM reads the fault-free partial bitstream into a memory array.

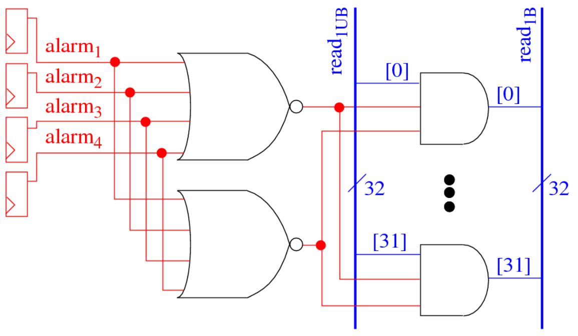

FIGURE 11. Schematic of the *Block* circuit which forces 32-bit chunks of *ciphertext*$_1$ to 0's when any of the registered alarms are asserted. The block signal is replicated to add resilience to deactivation when faults occur. Resources consist of 2 NOR LUTs and 32 AND LUTs.

- The DPR region is programmed with the fault-free partial bitstream and the GPIO control signals are used to load the key and plaintext, configure the AES engines for 128-bit encryption, and simultaneously start both AES engines to carry out a fault-free encryption operation. The fault-free ciphertext is fetched and stored in memory for use in the subsequent FEE.
- A fault emulation loop is executed in which a single bit flip error is introduced into the partial bitstream, starting at the base address of the Type 2 packets [21].
- The faulty partial bitstream is written to the PCAP interface using a system call from the FIM C program.
- The GPIO control registers are again used to load the key and plaintext, configure the AES engines for 128-bit encryption and start the AES engines in lock-step.
- The AES done signals are monitored for completion. If asserted, the data unloading procedure is commenced. A 1 millisecond time-out enables continuation in cases where the fault prevents the AES engines from asserting the done signals.
- Data unloading consists of issuing an address and start signals to the AES engines that indicate which 32-bit chuck of the ciphertext register is to be transferred to the $read_{1UB}$ and $read_2$ data buses. Ciphertext is retrieved by the FIM from the GPIO data registers and stored to a file. As indicated earlier, the data unloading process retrieves ciphertext from three different sources to assist with determining the behavior of the faulted circuit.
- The 8 alarm bit values are also read and stored to a file. The alarm status values with $correct_{sw}$ set to '0' are read first, and then read again after the $correct_{sw}$ control signal is asserted. If the $alarmx_{UR}$ bits return to '0', then the fault is classified as repairable.
- The fault emulation loop is repeated for each of the faults that can be inserted into the Type 2 data packet region of the partial bitstream. This process is repeated for both the upper and lower clock regions in the DPR region.

C. EXPERIMENTAL RESULTS

In this section, we analyze the data collected from four Xilinx Zynq 7010 SoCs. Our primary goal is to determine the number of benign faults, the number of faults that are detected by the alarms and the number of faults that are missed. From the OpenFPGA analysis, and the analysis presented in Fig. 7, faults introduced into the transmission gate MUX structures used in the switch boxes can create shorting and open conditions. Although the implementation details for the Zynq device layout are not available, we expect similar conditions can occur in the hardware. We confirm this by comparing the results across devices, and attribute any differences to within-die process variations, which act to make shorting and open fault behaviors device dependent, similar to the conclusions presented in [22].

We first describe some important limitations to our experimental setup, which uses the processor system (PS) and programmable logic (PL) side of an SoC to implement a fail-safe system. The first limitation relates to the inability of DEFCON to detect faults in circuit components that convey information after the checker itself. For example, from Fig. 8, faults that occur in the routing of the 32-bit $read_{1UB}$ bus through the *Block* circuit to the point where $read_{1B}$ crosses from the DPR region to the static region will not be detected. The redundant alarm signals and redundant implementation within the *Block* circuit itself add resiliency to the *Block* circuit to some types of faults but cannot ensure that data is not corrupted on transmission to the static region despite those cases in which the ciphertext is correct in both instances of the AES engine.

An intuitive way to fix this issue might be to connect $DEFCON_2$ to the outputs of the *Block* circuit. However, doing so requires a more complex pipeline structure to prevent corrupted data from reaching the static portion of the design (or I/O pads in actual fielded version of DEFCON). For example, if a routing fault occurs on the $read_{1B}$ data bus, $DEFCON_2$ would then assert $alarm_{3R}$ on the next rising edge of the clock, which in turn would force the *Block* circuit to produce the fail-safe values on the $read_{1B}$ data bus. However, the pipeline registers for the $read_{1B}$ data bus placed downstream at the interface to the static region just captured corrupted data, and therefore, must be reset. The pipeline registers could be instrumented with an emergency asynchronous reset signal, which would force the registers to the fail-safe values. However, this strategy is not fail-safe for faults that can now occur in the routing to and after the pipeline registers. A second strategy might make use of the unregistered $alarm_{3UR}$ signal as input to the *Block* circuit, as a means of eliminating the pipeline registers, but this design creates combinational loops and the possibility of oscillations. The best solution is to fix the placement and routing of the blocking circuit directly on the interface to the static region (or I/O pads) to eliminate downstream routing, but doing so requires the creation of a special constraints

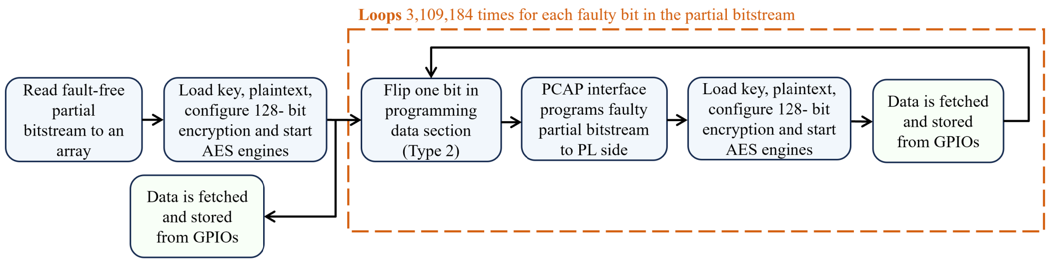
Loops 3,109,184 times for each faulty bit in the partial bitstream
Read fault-free partial bitstream to an array
Load key, plaintext, configure 128- bit encryption and start AES engines
Flip one bit in programming data section (Type 2)
PCAP interface programs faulty partial bitstream to PL side
Load key, plaintext, configure 128- bit encryption and start AES engines
Data is fetched and stored from GPIOs
Data is fetched and stored from GPIOs

FIGURE 12. Software flow diagram showing the actions carried out by the fault injection manager (FIM).

TABLE 3. Experimental results from the first Zynq 7010 device (3,109,184 total faults).

Status	Top Alarms	Bottom Alarms	Both Regions	Correctable
Benign	1,412,356	1,399,009	2,811,365	-
Active	142,236	155,572	297,808	-
At least one Alarm	141,925	153,986	295,911	324
False Positives	222	475	697	-
Missed	311	1,586	1,897	-
Untestable	0	11	11	-

file. Future work will investigate the improvements that are possible with this type of customized solution.

Bear in mind that under the single fault model, a fault that causes one of the AES engines to malfunction, will always cause both $alarm_1$ and $alarm_2$ to be asserted except in the rare case where the fault creates a short between a wire in an AES engine and one of the alarms, in which case, it may happen that one of the alarms is disabled. In either case, the FIM will record a positive detection of the fault. Therefore, all single fault occurrences within the AES engines will be detected. This assumes any single fault can cause at most two wires to be shorted together, and in such cases, it is also assumed that the two wires are not the exact same wires from both AES engines. Although possible, it is extremely unlikely that two identical wires would be routed through the same switch box. As we will show, no such instances occurred in our design.

However, there are several cases that will result in one or more alarms being asserted without any error occurring in either of the AES engines, hereafter referred to as false positive detections. From the OpenFPGA analysis, the most common false positive scenario occurs when the fault affects the DEFCON monitor itself, either in the routing of the monitoring wires or as upsets in the CMB of the DEFCON LUTs. The $correct_{sw}$ can be used to instantly repair faults occurring in the LUT CMB, but routing faults require a scrubbing operation, i.e., reprogramming the DPR region. Another scenario considers the possibility that the fault shorts one of the alarms to VCC. This latter scenario is distinguishable in our setup because only one of the 4 registered and unregistered alarms will be asserted. Other scenarios are possible, and are described in reference to the tables used to present the results.

The FIM runs a fault-free emulation as the first test and saves the correct ciphertext in an array. The fault-free ciphertext is compared with the ciphertext read from $read_{1B}$ (the ciphertext channel with the *Block* circuit in series). If a mismatch occurs and none of the 8 $alarm_x$ flags are set, the fault is classified as a miss. In our experiments, all of the observed misses occur for faults that affect the routing or LUTs after the $DEFCON_2$ circuit. We validated this by verifying the ciphertext read from the $read_{1UB}$ and $read_2$ channels is the correct ciphertext, ruling out failures in the AES engines. Moreover, in all cases, 32-bit chunks of correct ciphertext appear in the ciphertext read from the $read_{1B}$ channel. If, in contrast, either of the AES engines are affected by the fault, the AES round transformations would eliminate all matching 32-bit chunks of the correct ciphertext by virtue of the avalanche effect. Note that in a fielded version of DEFCON, we would not know the correct ciphertext, so these compare operations are done for diagnostic purposes only in this work.

For discussion of the results, we partition the 8 alarm signals into two groups of 4 bits, with the first group corresponding to alarms 1 and 2, and the second group to alarms 3 and 4. The bit values corresponding to the alarms signals are coded as hex values between 0 and F with the two high-order bits of each hex digit corresponding to the unregistered alarm signals, while the two low-order bits correspond to the registered alarm signals. For example, an alarm status code of 3 is decoded as ($alarm1_{UR}$, $alarm2_{UR}$, $alarm1_R$, $alarm2_R$) = 0011, which indicates that the registered alarm signals detect a fault, but the unregistered alarm wires are not asserted and therefore fail to detect a fault. Note that in an actual DEFCON fielded system, only the registered alarms would be used. The unregistered alarms are used in this work only for diagnostic information.

The leftmost column of Table 3 lists the status categories of the registered alarm signals, while columns 2 and 3 give the

TABLE 4. Blocking results.

Total Alarms	Full Block	Partial Block	Missed	False Positive
297,808	266,020	3,541	1,897	25,266

number of FEE classified under each of the status categories for the top and bottom halves of the DPR region, respectively. Column 4 shows the row-wise sums of Columns 2 and 3. Columns 5 counts only those faults that are correctable from the set that assert at least one registered alarm signal. We first note that the top and bottom halves each have 1,554,592 CMB, and therefore, the total number of fault emulation experiments carried out is 3,109,184. The AES instances and DEFCON circuits utilize 76.1% of the LUT resources and 34.0% of the FF resources, as reported by Vivado. The most important status conditions are listed on the rows in the table and are explained as follows:

- Benign: These faults are characterized as having the correct ciphertexts and no alarms are set. Despite the reasonably high utilization of resources used by the design, most of the CMBs are not utilized to implement the design, and therefore, faults injected into these CMBs have no effect on the functional operation. These results are consistent with previous studies which found that, on average, only 10% of the CMBs are utilized in any given design [23].
- Active: These faults are characterized as having one of the ciphertexts corrupt, either because the fault impacts one of the AES engines or because the read-out circuit is corrupted. The CMBs corresponding to the active faults are referred to in previous work (and by Xilinx) as *essential configuration bits* [22].
- At least one Alarm: These faults are characterized as being detected by one or both of the DEFCON monitors.
- False Positives: These DEFCON monitor faults are characterized as having no effect on the ciphertexts but raise one or more alarm signals. Note that these faults are also counted in the 'At least one Alarm' class.
- Missed: These faults are characterized as corrupting the $read_{1B}$ ciphertext but none of the alarm signals are asserted.
- Untestable: These faults are characterized as impacting the clock network and locking up the FPGA, making it impossible to retrieve the ciphertext and alarm results.

The following conclusions are made based on the tabulated results in Table 3 and observations of the ciphertexts and full set of alarm signals.

- Fault detection capabilities: The DEFCON technique is able to detect 99.78% of the Active faults in the Top region and 98.98% of the Active faults in the Bottom region.
- Missed faults: Using ciphertext from all three channels, we confirmed that all miss cases occurred because of faults in the routing and LUTs downstream from $DEFCON_2$, i.e., the AES engines computed the correct ciphertext, but it was not transmitted properly through the $read_{1B}$ channel.

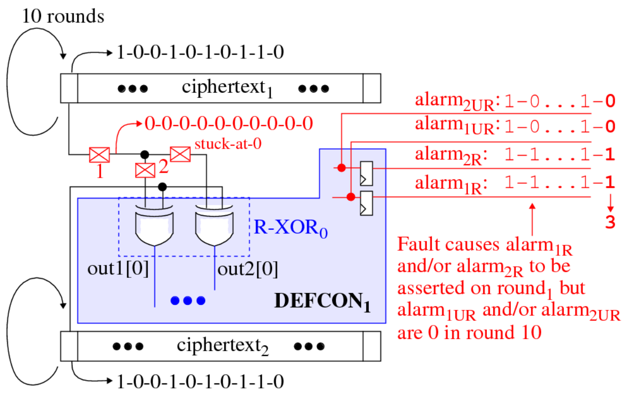

FIGURE 13. Open fault model and their impact on the registered and unregistered Alarm signals.

- Correctable faults: The number of correctable faults observed in the FEE is 324, which is consistent with the number of R-XOR and R-OR gates used in the two DEFCON monitors, i.e., 2*128 + 2*32 = 320 R-XOR and 2*2 R-NOR (as discussed earlier, two faults are correctable in each redundant LUT gate).
- Faults which assert only the registered alarms signals (Alarm code = 3): These faults occur in the routing of the ciphertext to the DEFCON checkers. Examples of faults that can cause this alarm status condition are shown in Fig. 13, which annotates a set of stuck-at-0 faults with red squares. A Stuck-at-0 fault occurs when a CMB fault creates an open circuit and presumably, the weak pull-down forces the floating net to logic '0'. Here, we show an example sequence of bit values that might be produced by both AES engines as they execute the 10 rounds of an encryption operation, given as "1001010110". In this example, the first round has a '1' in the high-order (left-most) bit position. The stuck-at-0 fault prevents the R-XOR from receiving the '1' from the top ciphertext round register and asserts an alarm on one of $out1[0]$ or $out2[0]$ depending on which stuck-at-0 fault is activated. This causes both registered alarm signal, $alarm_{xR}$, to record a '1' because of the fan-out on the input to the R-OR collector gate (see Fig. 10). Once the encryption engines finish, the ciphertext in the register has '0' for the high order bit. This causes the unregistered $alarm_{xUR}$ signals to return to '0' because the faulty value and the true value are identical. The alarm code in this case is given as "0011" indicating both unregistered alarms are deasserted and both registered alarm signals are asserted.
- Faults which assert both the registered and unregistered alarms (Alarm code = F): This is by far the most common error code, which occurs when a fault affects the routing or LUT networks in one of the AES implementations. The fault corrupts the ciphertext in one or more rounds

of the AES encryption operation and causes both of the registered and unregistered alarms to be set in both DEFCON circuits.

- Faults in the R-OR truth tables (Alarm code = 5 or A): Only one of the registered and unregistered alarm signals are asserted. This case occurs when a fault affects one of the selected '0' bits in the R-OR LUT. Here we observe the ciphertexts are correct on all three read-out channels. This condition occurs exactly 4 times given that only two R-OR gates exist in the design, one for each of the DEFCON monitors. The fault effect is illustrated in Fig.14 where a single fault in the CMB highlighted in red affects just one of the alarm signals (the upper MUX network of the LUT and alarm signal are not shown). In this case, asserting the $correct_sw$ signal, highlighted in blue, can instantly repair the fault by switching to the upper 16 bits of the redundant truth table encoding of the LUT.
- Non-determinism: We observed a small number of faults, less than 20 in any given FPGA and Top or Bottom region, that produce a constant incorrect ciphertext on all read-out channels, and therefore, none of the alarms are asserted. Moreover, the ciphertext is exactly the same across all FEE in which this occurred. We also found that the FEE that produce this strange condition are non-deterministic, i.e., by re-running the FEE a second time, the ciphertexts become fault-free on all three channels. It remains a mystery as to the root cause of this strange behavior. Our best guess is that the system command that reads the partial bitstream from the SD card and streams it into the PCAP interface from the PS side malfunctions occasionally and forces the PL side into some type of failed state. The process of re-running these FEE eliminated the errors in every instance, and therefore, we classify these faults as Benign in Table 3. Future work will further investigate the root cause.

The Block circuit results are partitioned into 4 categories as shown in Table 4, namely Full Block, Partial Block, Missed, and False Positive. When any of the alarms are asserted, the Block circuit blocks the ciphertext and forces all bytes to zero. The number of block failures, given by the Missed column, corresponds exactly to the number of missed alarm assertions given by the Missed row in Table 3. As mentioned, these misses are attributed to faults that occur after the Block component in the routing and LUTs downstream from $DEFCON_2$. We also observe cases where, despite a positive detection of a fault with one or more alarms asserted, only a portion of the ciphertext is blocked. This occurs when the fault occurs in the readout MUXing structure, where the fault is not detected by $DEFCON_2$ until the faulty input component to the MUX is accessed during the readout operation. The faults listed in the False Positive column of Table 4 indicate that the blocking circuit prevents readout even though the ciphertexts are correct. These cases are attributed

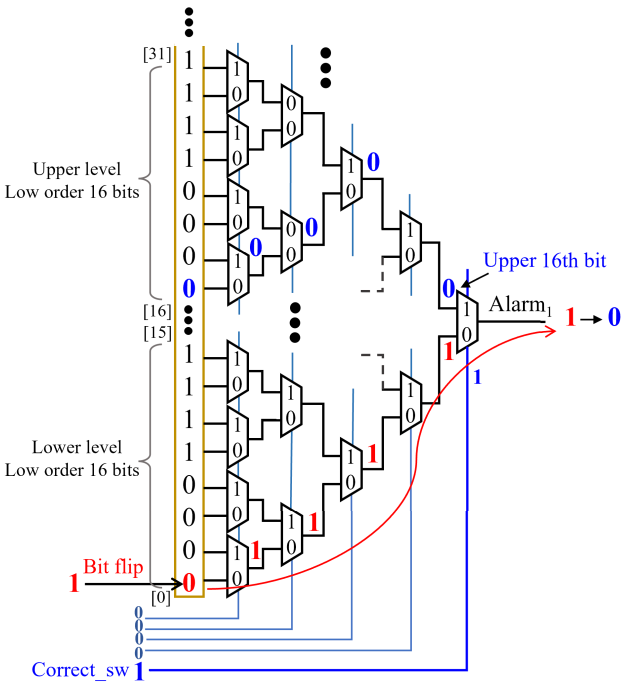

FIGURE 14. **DEFCON R-OR gate illustrating repair capability using *Correct_sw* signal.**

to experiments in which the fault occurs in the routing and LUTs of the $DEFCON_1$ or $DEFCON_2$ components.

1) FPGA DEPENDENT FAULT BEHAVIOR

The results presented in Table 3 for one of our FPGAs differs from the results obtained using data collected from three additional FPGAs. Given the same design is used in all FPGAs, the differences observed are most likely due to faults that create shorts in the routing network. The relative drive strength of the LUTs and buffers driving the shorted nodes vary because of process variation effects, which adds non-determinism to the resulting logic values interpreted by downstream LUTs.

As an example, Fig. 15 shows two LUTs, LUT_1 and LUT_2, driving two nets with opposite logic values. The fault in this case enables an N-channel switch between two output nets in a switch box. The wires and switch box MUXs define a voltage divider network where the voltage values present on the inputs of downstream LUTs, LUT_3 and LUT_4, are larger and smaller than the logic '0' and logic '1' voltage values, respectively. Although we show 0.4v and 0.6v as the downstream voltage values in this example, the actual values are dependent on the relative resistances of R_1, R_2 and R_3. The receiver LUTs are tasked with interpreting the erroneous input voltages as logic values, and may or may not produce the correct output value. It is also possible that the pass-gate MUXing network within the LUT creates additional shorts causing the output voltages on LUT_3 and LUT_4 to propagate indeterminate logic levels further downstream.

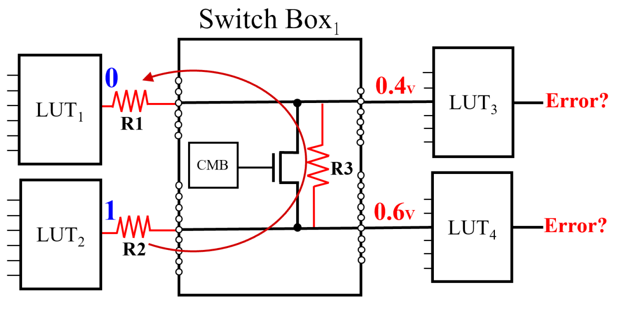

FIGURE 15. Fault model of a FE experiment in which the fault creates a short circuit between two driver LUTs, labeled LUT_1 and LUT_2, by enabling a connection between two independent nets. The fault in this example flips the CMB driving the gate of the N-channel transistor from a 0 to a 1.

TABLE 5. Counts of the number of differences observed in fault behavior of three additional FPGAs with respect to reference FPGA.

Device	Alarm	Detection	Ciphertext	Same	Total
$FPGA_1$	24,227	21,762	22,369	285,497	309,854
$FPGA_2$	20,934	18,773	19,240	286,983	308,040
$FPGA_3$	25,652	22,801	23,411	285,230	311,039

Although it is not possible to verify these conditions exist within the commercial FPGAs used in our experiments, the differences observed in fault behavior among the 4 FPGAs confirm that some type of contention exists in a subset of the FEE. The number of such instances is given in Table 5, where we tabulate differences in the alarm values and ciphertexts for each of the three additional FPGA devices listed in the first column when compared with the values from the reference FPGA presented above. Here, we report the sum of the differences observed in both the Top and Bottom regions. The second column labeled Alarm gives the number of faults where some difference exists in the 4 registered alarm values of the two FPGAs. The Detection column counts only those faults from the second column where one FPGA detects the fault (at least one alarm asserted) while the second FPGA does not (no alarms asserted). Therefore, these faults allow some FPGAs to function properly, and produce the correct ciphertext, despite the fact that the fault is active, e.g., it creates a short in the routing network.

The fourth column counts the number of faults in which the ciphertexts are different. Again, differences in the ciphertext indicate that the fault's effect is FPGA dependent. The fifth column labeled Same counts the faults that produce the same faulty behavior, i.e., the incorrect ciphertexts match as well as the alarm status values, in particular, both alarms are non-zero and equal in value. The sixth column labeled Total illustrates that the vast majority of the active faults introduce identical faulty behaviors. Here, Total represents the number of faults that generate some type of error, either in producing wrong ciphertexts and/or by asserting one or more alarms.

V. BYU VS DEFCON

In this section we analyze and compare the design of DEFCON and the DWC technique proposed by McMurtrey et al.

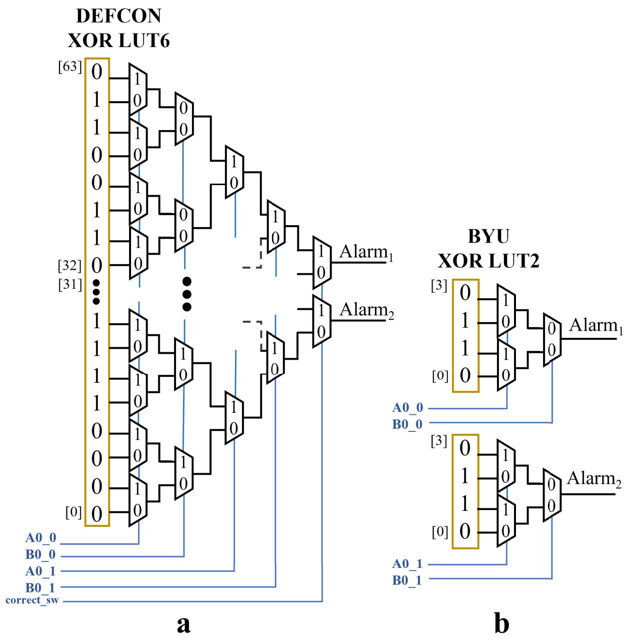

FIGURE 16. Comparison of DEFCON vs BYU XOR LUT implementations. The BYU design uses two LUT2 gates to compare the redundant output bits labeled A0_0, B0_0, A0_1 and B0_1, which is accomplished in DEFCON using only one LUT.

TABLE 6. DEFCON and BYU LUT utilization.

Design	LUTs
BYU	680
DEFCON	183

TABLE 7. BYU vs DEFCON experimental fault analysis (3,109,184 total faults).

Status	DEFCON	BYU
Benign	2,811,365	2,796,923
Active	297,808	312,243
At least one Alarm	295,911	310,381
False Positives	697	650
Missed	1,897	1,862
Untestable	11	18

[3], as well as the experimental results obtained from both designs. Here we refer to the design in [3] as the BYU circuit. The BYU technique uses two 2-input XOR LUTs to compare each redundant output bit from the 128-bit AES engines, which is twice the number utilized by DEFCON. The DEFCON and BYU circuit structures are shown in Fig. 16(a) and (b), respectively. Similar to DEFCON, the error signals are collected and drive two final alarm signals using a binary tree of 4-input OR LUTs.

The resource utilization of each technique is shown in Table 6. Here we observe that the DEFCON design offers a significant advantage over the BYU design, using a total of 183 LUTs verses the 680 used by the BYU design. Although it is difficult to accurately count, the number of routing resources used by the BYU design would also be larger by approximately the same proportion because of the additional XOR LUTs required in the BYU design.

Table 7 compares the fault detection capabilities of both designs. As expected, the number of Benign faults is slightly larger, and the number of Active faults is slightly smaller, for the DEFCON design because fewer resources are used by the DEFCON monitor circuitry. The number of faults classified as False Positives, Missed and Untestable are very similar across both designs, with the small differences likely due to optimizations carried out by the Vivado place and route tool. Therefore, the XOR packing strategy proposed for DEFCON does not present any drawbacks with respect to fault detection capabilities.

VI. CONCLUSION

In this paper, we propose a compact, fail-safe monitor implemented on an FPGA that is able to detect a difference on the outputs of two redundant functional units while being robust to a second failure that occurs in the monitor itself. We refer to the technique as DEFCON (DEsign for Fail-safe in reCONfigurable systems). A novel redundancy scheme is proposed in DEFCON in which a single LUT is able to provide two independent functions. A special encoding of the configuration memory bits in the LUT enables it to accommodate a fault(s) on one set of inputs, while correctly performing the specified function delegated to a separate set of inputs. A fifth input to the LUT enables quad-redundancy, which allows the LUT to switch instantly to a distinct set of configuration memory bits and continue operating correctly when a fault occurs in the LUT itself. Simulation results show that DEFCON can correctly assert an alarm when the monitored functional unit suffers a fault and when a simultaneous (dual) fault occurs in the configuration memory bits implementing the DEFCON monitor in the FPGA. Moreover, in nearly all dual-fault test scenarios, DEFCON is able to successfully block and force fail-safe values on all functional unit outputs. The DEFCON circuit is validated in hardware on a set of FPGAs, and compared with the most closely related technique in terms of resource utilization and detection capabilities.

ACKNOWLEDGMENT

This written work is authored by an employee of NTESS. The employee, not NTESS, owns the right, title and interest in and to the written work and is responsible for its contents. Any subjective views or opinions that might be expressed in the written work do not necessarily represent the views of the U.S. government. The publisher acknowledges that U.S. government retains a non-exclusive, paid-up, irrevocable, and world-wide license to publish or reproduce the published form of this written work or allow others to do so, for U.S. government purposes. The DOE will provide public access to results of federally sponsored research in accordance with the DOE Public Access Plan.

REFERENCES

[1] K. Szurman and Z. Kotasek, "Run-time reconfigurable fault tolerant architecture for soft-core processor NEO430," in *Proc. IEEE 22nd Int. Symp. Design Diag. Electron. Circuits Syst. (DDECS)*, Apr. 2019, pp. 1–4.

[2] J. Johnson, W. Howes, M. Wirthlin, D. L. McMurtrey, M. Caffrey, P. Graham, and K. Morgan, "Using duplication with compare for on-line error detection in FPGA-based designs," in *Proc. IEEE Aerosp. Conf.*, Mar. 2008, pp. 1–11.

[3] D. L. McMurtrey, "Using duplication with compare for on-line error detection in FPGA-based designs," Ph.D. dissertati ons, Dept. Elect. Comput. Eng., Brigham Young Univ., Provo, UT, USA, 2006.

[4] X. Tang, E. Giacomin, A. Alacchi, B. Chauviere, and P.-E. Gaillardon, "OpenFPGA: An opensource framework enabling rapid prototyping of customizable FPGAs," in *Proc. 29th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2019, pp. 367–374.

[5] X. Tang, E. Giacomin, B. Chauviere, A. Alacchi, and P.-E. Gaillardon, "OpenFPGA: An open-source framework for agile prototyping customizable FPGAs," *IEEE Micro*, vol. 40, no. 4, pp. 41–48, Jul. 2020.

[6] F. Benevenuti and F. L. Kastensmidt, "Evaluation of fault attack detection on SRAM-based FPGAs," in *Proc. 18th IEEE Latin Amer. Test Symp. (LATS)*, Mar. 2017, pp. 1–6.

[7] J.-Y. Lee, Y. Hu, R. Majumdar, L. He, and M. Li, "Fault-tolerant resynthesis with dual-output LUTs," in *Proc. 15th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2010, pp. 325–330.

[8] J.-Y. Lee, Z. Feng, and L. He, "In-place decomposition for robustness in FPGA," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2010, pp. 143–148.

[9] L. Sterpone and M. Violante, "A new reliability-oriented place and route algorithm for SRAM-based FPGAs," *IEEE Trans. Comput.*, vol. 55, no. 6, pp. 732–744, Jun. 2006.

[10] N. Jing, J.-Y. Lee, Z. Feng, W. He, Z. Mao, and L. He, "SEU fault evaluation and characteristics for SRAM-based FPGA architectures and synthesis algorithms," *ACM Trans. Design Autom. Electron. Syst.*, vol. 18, no. 1, pp. 1–18, Jan. 2013.

[11] S. Zheng, H. You, G. He, Q. Wang, T. Si, J. Jiang, J. Jin, and N. Jing, "A rapid scrubbing technique for SEU mitigation on SRAM-based FPGAs," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2019, pp. 1–5.

[12] S. T. Fleming and D. Thomas, "Injecting FPGA configuration faults in parallel," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, Dec. 2018, pp. 198–205.

[13] H. Mestiri, N. Benhadjyoussef, M. Machhout, and R. Tourki, "An FPGA implementation of the AES with fault detection countermeasure," in *Proc. Int. Conf. Control, Decis. Inf. Technol. (CoDIT)*, May 2013, pp. 264–270.

[14] F. Kahri, H. Mestiri, B. Bouallegue, and M. Machhout, "An efficient fault detection scheme for the secure hash algorithm SHA-512," in *Proc. Int. Conf. Green Energy Convers. Syst. (GECS)*, Mar. 2017, pp. 1–5.

[15] M. N. Shaker, A. H. Madian, M. B. Abdelhalim, S. H. Amer, A. S. Emara, and H. H. Amer, "Effect of open faults in FPGA switch matrices on fault detection mechanisms," in *Proc. 28th Int. Conf. Microelectron. (ICM)*, Dec. 2016, pp. 233–236.

[16] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, K. B. Kent, J. Anderson, J. Rose, and V. Betz, "VTR 7.0: Next generation architecture and CAD system for FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 7, no. 2, pp. 1–30, Jul. 2014.

[17] J. Yao, B. Dixon, C. Stroud, and V. Nelson, "System-level built-in self-test of global routing resources in Virtex-4 FPGAs," in *Proc. 41st Southeastern Symp. Syst. Theory*, Mar. 2009, pp. 29–32.

[18] M. Ehab and I. Nashaat, "Advanced encryption standard (AES128, AES192, AES256) encryption and decryption implementation in Verilog HDL," 2022. [Online]. Available: https://github.com/michaelehab/AES-Verilog

[19] *Zybo Z7 Reference Manual*, Digilent, Pullman, WA, USA, 2024.

[20] *Zybo Z7 Reference Links*, Digilent, Pullman, WA, USA, 2024.

[21] *7 Series FPGAS Configuration, User Guide*, Xilinx, San Jose, CA, USA, 2024.

[22] C. Fibich, M. Horauer, and R. Obermaisser, "Device- and temperature dependency of systematic fault injection results in Artix-7 and iCE40 FPGAs," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Feb. 2021, pp. 1600–1605.

[23] N. A. Harward, M. R. Gardiner, L. W. Hsiao, and M. J. Wirthlin, "Estimating soft processor soft error sensitivity through fault injection," in *Proc. IEEE 23rd Annu. Int. Symp. Field-Program. Custom Comput. Mach.*, May 2015, pp. 143–150.

PRIYA A. BHAKTA (Student Member, IEEE) received the B.S. and M.S. degrees in computer engineering from the University of New Mexico, Albuquerque, in 2022 and 2023, respectively, where she is currently pursuing the Ph.D. degree in computer engineering. Her research emphasis is on increasing the reliability of field programmable gate arrays (FPGAs) in high consequence systems through circuit-level fail-safe designs. Her research interests include integrated circuit logic design and embedded systems.

JIM PLUSQUELLIC received the M.S. and Ph.D. degrees in computer science from the University of Pittsburgh. He is currently a Professor in electrical and computer engineering with the University of New Mexico. He received the "Outstanding Contribution Award" from IEEE Computer Society, in 2012 and 2017, for co-founding and for his contributions to the Symposium on Hardware-Oriented Security and Trust (HOST).

ANDREW (DREW) SUCHANEK (Senior Member, IEEE) received the dual B.S. degree in applied mathematics and in computer engineering and the M.S. and Ph.D. degrees in computer engineering from the University of Arkansas, Fayetteville, AR, USA, in 2013, 2013, 2017, and 2018, respectively. Since 2017, he has been with the Sandia National Laboratories, Albuquerque, NM, USA, where he is currently a Principal Member of the Technical Staff. He leads multiple ASIC developments for various high consequence systems and is developing an automated method to ensure ASICs and circuits implemented on FPGAs are fail-safe. His research interests include radiation-hardened ICs and memories, secure and reliable FPGAs, and asynchronous digital circuits.

TOM J. MANNOS received the B.S. degree in electrical engineering from the University of Utah and the master's degree in electrical engineering from the University of New Mexico. He is currently an Integrated Circuit Designer with the Sandia National Laboratories conducting research in embedded FPGA security, security fault analysis, and superconducting electronics.