Evaluating Architectural, Redundancy, and Implementation Strategies for Radiation Hardening of FinFET Integrated Circuits

Evaluating Architectural, Redundancy, and Implementation Strategies for Radiation Hardening of FinFET Integrated Circuits

Samuel Pagliarini, *Member, IEEE*, Luis Benites, Mayler Martins, *Member, IEEE*, Paolo Rech, *Senior Member, IEEE,* and Fernanda Kastensmidt, *Member, IEEE*

***Abstract*—In this paper, authors explore radiation hardening techniques through the design of a test chip implemented in 16 nm FinFET technology, along with architectural and redundancy design space exploration of its modules. Nine variants of matrix multiplication were taped-out and irradiated with neutrons. The results obtained from the neutron campaign revealed that the radiation hardened variants present superior resiliency when either local or global TMR schemes are employed. Furthermore, simulation-based fault injection was utilized to validate the measurements and to explore the effects of different implementation strategies on failure rates. We further show that the interplay between these different implementation strategies is not trivial to capture and that synthesis optimizations can effectively break assumptions about the effectiveness of redundancy schemes.**

I. Introduction

Integrated circuits (ICs) are prone to faults due to an extensive array of adverse environmental elements. Amongst those elements, highly energetic particles are of particular interest since their interaction with silicon is effectively capable of inducing faults in both memory and logic resources of an IC. The physics behind the phenomena is well studied, both at transistor [1] and circuit-level [2]. When combinational elements of a circuit are hit, transient and short-lived perturbations are created. If perturbations reach a primary output of a circuit – often by being latched by a flip-flop along the way – then the perturbation becomes an error. Sequential elements of a circuit are also threatened by the same particles. These elements can be directly affected as the information they carry is flipped from 0-to-1 (or vice versa) and remains corrupted until the element is overwritten with new information, which is called a single-event upset.

There are numerous approaches to harden an integrated circuit against radiation-induced errors, each having different overheads and relative efficiency. Some strategies apply to both sequential and combinational cells, while others are targeted at one or the other. Radiation hardening by design is often employed since it does not rely on any foundry-level process options. Instead, the design is modified to leverage logical, electrical, and temporal masking. Mitigation strategies include cell upsizing [3][4], reliability-aware cell replacement [4], guard-gates [5], triple modular redundancy (TMR) [6], and many others [7].

For a given target circuit, designers can calculate how likely a fault is to become an error. Mitigation techniques are then utilized to bring this number down to acceptable levels, whichever level might be suited/recommended for the application domain being targeted. Mitigation techniques can be applied at different abstraction levels and stages of the design flow. In this paper, we study mitigation strategies that fall into three distinct categories: architectural, redundancy, and implementation. On the architectural side, we explore how the error rates of a matrix multiplication (MM) circuit change when parallelism and pipelining are employed. For redundancy, we consider different granularity levels of TMR, namely global and local. A test-chip was fabricated in a 16nm FinFET technology in order to investigate the effectiveness of architectural and redundancy strategies. Seven optimization options available in a commercial logic synthesis tool are considered during the implementation of the studied circuits. The optimization options are evaluated applying fault injection by simulation.

When an IC is being designed, the circuit description goes through a series of transformations, some of which can have a significant impact on the error rate of the circuit being implemented. At the start of this transformation chain, we often employ logic synthesis to translate an RTL description of the circuit into a netlist. Further, the netlist goes through physical synthesis, in which steps such as placement, routing, clock tree synthesis, timing optimization, and signoff are performed.

Several hardening approaches can be employed by analyzing and modifying the generated netlist that is the output of logic synthesis. Examples are cell replacement [4], cell sizing [8], use of a hardened standard-cell library, and use of the dual interlock cell (DICE) technique [8]. The design description can also be modified to incorporate a TMR strategy, parity bits, and error detection and correction (EDAC) blocks [10]. Combinations of the aforementioned techniques are also possible.

However, strategies that modify or evaluate the actual synthesis behavior [11] are less frequently seen. There are approximate logic synthesis techniques available [12][13][14],

Manuscript submitted 04/Sep/2020.

S. Pagliarini is with Tallinn University of Technology (Tallinn, Estonia). He was previously with Carnegie Mellon University (Pittsburgh - PA, USA).

L. Benites, P. Rech, and F. Kastensmidt are with Universidade Federal do Rio Grande do Sul (Porto Alegre, Brazil). P. Rech is also with Politecnico di Torino (Torino, Italy).

M. Martins is with Siemens EDA, a part of Siemens Digital Industries Software (Fremont – CA, USA). He was previously with Carnegie Mellon University (Pittsburgh - PA, USA).

where the area penalty of a TMR scheme is reduced considerably through the logic simplification of the redundant blocks while making (some) input vectors unprotected. These approximations are a good fit for error-tolerant circuits, such as video/image processing circuits.

In physical synthesis, the placement step is highly relevant for hardening as the cells' spatial location affects masking. For instance, in [15], a careful analysis of the effect of placement with respect to multiple faults was performed. A placement engine that is fault-aware is proposed in [16], albeit presenting heavy overheads. An improved version is presented in [17].

In this paper, however, we focus on the effects of logic synthesis by studying how optimization decisions influence the error rate, i.e., how implementation strategies promote (or impede) fault propagation, often in non-obvious ways. Neutron testing was performed at the Los Alamos Neutron Science Center (LANSCE) facility with the intent to investigate architectural and redundancy strategies. All the mitigations implemented at architectural, logical redundancy, and synthesis levels were evaluated by simulation-based fault injection in order to explore design space trade-offs.

This paper is organized as follows: in Section II, we present the test chip specification and architectures that were implemented in silicon. In Section III, we discuss implementation strategies and their trade-offs. Our results are given in Section IV and Section V. A discussion follows on Section VI. Finally, our conclusions are drawn in Section VII.

II. Test-Chip Specification and Architectural/Redundancy Strategies

In this section, we present the specifications of the test chip and the architectural choices to allow multiple implementations to be tested.

A. Test Chip Specification

An IC was designed to perform matrix multiplication using three different architectures and three different strategies for redundancy. Therefore, nine unique modules were implemented. The circuits were described in Verilog language and synthesized using Cadence Genus and Innovus tools following a typical standard cell-based flow.

The nine MM variants were all taped out in the same 16nm FinFET die. A serialized interface is utilized to control which MM variant is currently active. Output signals of the modules are multiplexed to keep the pin count reasonably under control. A 175MHz clock is generated externally and delivered to the chip via a dedicated pin.

Input signals (operands of the multiplication) are common for all MM modules and come from embedded memories. Six ROMs were utilized for this purpose, three for operand A and three for operand B of the matrix multiplication M = A x B. Each ROM has 64 addresses of 8 bits each. Pre-initialized ROMs were employed, so the multiplication result is known and easy to verify externally.

B. Architectural and Redundancy Strategies

On the architectural side, a designer can reason about pipelining strategies and parallelism for accelerating a given computation. These strategies generally increase throughput and frequency at the expense of area resources, but that is often a worthwhile trade-off in IC design.

Our first case studied circuit performs matrix multiplication of 6x6 input matrices of 8-bit vectors. Our design space exploration led to three architecturally different versions that were generated using a High-Level Synthesis (HLS) tool from Xilinx [18]. HLS, also known as behavioral or algorithmic synthesis, is a method that has as input a high level/functional description of a design and compiles it into an RTL implementation that is later synthesized to the gate level using a logic synthesis tool. The high level description is often done in C/C++/SystemC, and the behavior is usually untimed. One of the main advantages of HLS is the automatic scheduling and loop optimizations performed. The scheduling creates a finite state machine and can inherently pipeline the design, selecting the best number of stages. The loop optimization in HLS checks for logic dependencies and unroll loops when possible, where parallelism can be increased to obtain a reduced latency, or parallel units can be merged to reduce area.

The first MM version was generated without employing any HLS optimization directives and resulted in a non-pipelined version with a latency of 733 clock cycles (v0). Alternatively, two optimized versions were also generated considering different pipelining choices. The second version has a 21-stage pipeline and presents a latency of 112 clock cycles (v3). The third version exhibits a 7-stage pipeline that corresponds to a latency of 133 clock cycles per multiplication (v6).

Each one of the three architecturally different versions was hardened by the use of a TMR technique. In this work, two variants of the TMR approach were considered according to the granularity level employed when generating redundant structures: coarse grain and fine grain. The coarse grain TMR (global) is the same as conventional modular TMR, where a module is considered as the entire design, and the modules' outputs are voted out. On the other hand, fine-grain TMR (local) replicates sub-modules instead of the whole design. One case is the replication of sequential cells, while the combinational paths remain the same.

Moreover, a single voter is placed at the output of every triplicated sequential cell. For this purpose, we have utilized a tool previously developed in [19] to generate the TMR variants automatically. Table I presents a summary of the three architectures, each one having three variants.

III. Implementation Strategies and Evaluation Methodology

In this section, we present the logic synthesis optimizations typically chosen to trade-off area, power and maximum frequency of operation. Logic synthesis is the process that translates an RTL description into a mapped netlist. The process is understandably heavily influenced by the standard cell library composition and design constraints [11][20]. However, specific optimizations enabled by the user also affect the final result.

A. Logic synthesis optimizations

In our study, we have explored a total of seven optimizations available in Genus, which are listed below:

1. Clock gating (*opt_cg*)
2. Ungrouping (*opt_ung*)
3. Datapath analytical (*opt_dpa*)
4. Bubble pushing (*opt_bubp*)
5. Tighten max transition (*opt_maxt*)
6. Retiming for delay (*opt_retd*)
7. Retiming for area (*opt_reta*)

Optimizations from 1 to 5 are enabled by different attributes and their settings (given in parenthesis), namely *lp_insert_clock_gating* (true), *auto_ungroup* (both), *dp_analytical_opt* (extreme), *lbr_seq_in_out_phase_opto* (true), and *max_transition* (100ps). The retiming optimizations are executed with the commands *retime –min_delay* and *retime –min_area.*

Many of these optimizations are not enabled by default in a reference synthesis flow but are often employed by IC designers in order to optimize certain aspects of a block (or chip). Next, we discuss each technique briefly.

Clock gating is typically employed for power saving reasons. It consists of adding logic that prevents flip-flops from seeing clock toggles unless a specific enable signal is asserted. Synthesis tools can infer enable signals automatically from RTL. **Ungrouping**, when allowed**,** implies that the synthesis engine may flatten the design hierarchy and consider optimizations that traverse boundaries. This optimization has implications for verification and impacts runtime considerably, but typically brings area savings. **Datapath analytical** enables aggressive datapath optimization that knowingly trades area for timing improvement – this type of specific strategy is often proprietary, and the user of the synthesis tool is not in control of their specific usage. When **bubble pushing** is enabled, inverters are pushed in and out of flip-flops by switching between the use of $Q/\overline{Q}$ or $D/\overline{D}$ pins. This technique is also called (output) inversion of sequential cells. **Tightening max transitions** forces buffering as signals are prevented from having slow transitions, even for paths where this behaviour would cause no timing violation. This artificial tightening helps with issues of crosstalk and to achieve timing closure. **Retiming** changes the boundaries of sequential and combinational logic by moving cells from one stage of logic to another. Retiming can be targeted for delay – which is the typical use case. Retiming can also target area savings, but never at the expense of delay, i.e., delay remains the primary optimization target.

In most experiments reported later on, one and only one optimization is enabled at a time. When two optimizations are considered concurrently, we utilize a plus sign and label them "*opt_abc + opt_xyz*". For all cases, the synthesis effort was kept *high* for fairness.

B. Fault injection methodology:

Next, we detail our methodology for injecting faults, which is the approach we have followed for: a) cross validation of the architectural/redundancy strategies under neutrons, and b) evaluating the many synthesis optimizations. We will use this fault injection approach and terminology on the remainder of this paper.

First, we clarify that all faults are injected at **gate level** and only **single faults** are injected. The studied circuits are reset every time a computation is finished, thus preventing the accumulation of faults. All circuits are synthesized with (and only with) reset-capable flip-flops, thus guaranteeing that a reset flushes any lingering fault or error.

TABLE I
CHARACTERISTICS OF THE MATRIX MULTIPLICATION MODULES

MM Version	TMR?	Cell count	Area (um^2)
v0	No	670	253.0
v1	Local	2054	782.2
v2	Global	2212	880.8
v3	No	13674	5076.4
v4	Local	42559	15841.9
v5	Global	44035	17170.4
v6	No	2391	981.2
v7	Local	7253	2956.9
v8	Global	7717	3325.4

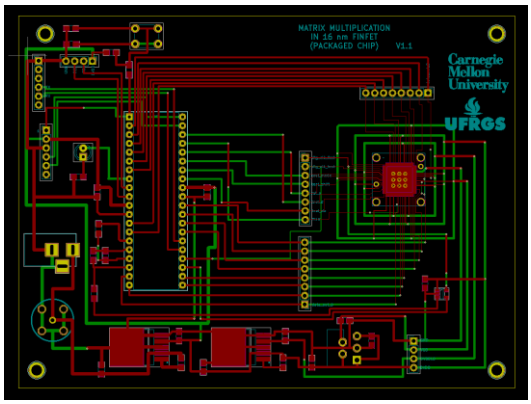
For each injected fault, we observe the outputs of the circuit for the entire duration of the computation taking place, which may span many clock cycles. If, at any given clock cycle, any output differs from the expected, we say this scenario **leads to a 'fail'**. If no output differs from the expected at the end of the computation, we say this scenario **leads to a 'pass'**.

Fault injection targets are random, i.e., **gates are selected randomly**. Fault injection campaigns targeting combinational logic and sequential logic are always executed separately for clarity, so in any given campaign all targets are of the same type. Only one fault is injected per computation and the fault injection time is also random. For instance, for the MM v0 circuit, there is one fault every 733 cycles, where the injection can occur in a random cycle between 1 and 733. Input patterns for data signals are randomized (i.e., clock and reset are not randomized). However, for experiments that compare two variants of the same circuit, we guarantee that the random input patterns are the same for the different variants.

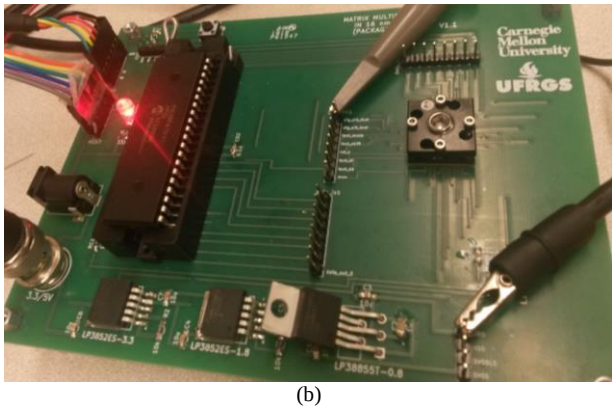
IV. RESULTS FOR THE NEUTRON TEST CAMPAIGN

The IC containing the 9 MM modules was fabricated in a commercial 16nm FinFET technology. The die size is 1.8mm x 1.8mm due to packaging restrictions – the actual MM circuitry takes only a fraction of the available area. Area and cell count for each MM variant are detailed in Table I. The nominal voltage of the core of the IC is 0.8V, the intermediate voltage is 1.8V for ESD structures and IO ring, while the output voltage is 3.3V. The maximum frequency of the slowest MM variant on the worst/worst corner is 1.26 GHz. This value was then adopted during timing closure for the entire chip.

The fabricated chip was mounted on the PCB detailed in Fig. 1 and radiated with neutrons at LANSCE. The board utilizes an 8-bit microcontroller (seen on the left), three voltage regulators for 3.3V, 1.8V and 0.8V (bottom), a controllable oscillator that provides a 17-170 MHz clock, the socket where the fabricated chip sits (upper right), pin headers for exposing chip internals for debug, and a range of decoupling capacitors for power distribution. The microcontroller is purposefully placed 5 cm away from the socket to avoid being affected by radiation and cause false positives. The microcontroller can communicate with a host computer through a USB to serial cable, using the RS232



(a)



(b)

Fig. 1. (a) Schematic of the PCB. (b) PCB being utilized during bring-up tests in the lab. The same PCB was later utilized during the neutron campaign.

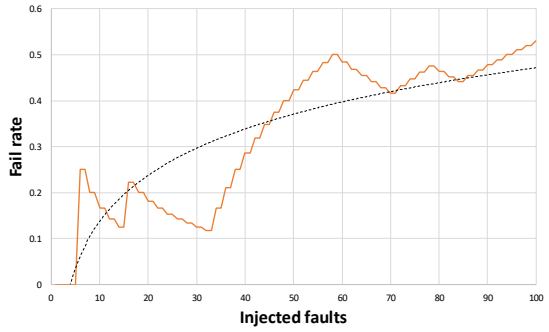
standard. A simple protocol was implemented to allow the user to send *multiply* commands remotely. The microcontroller is also in charge of sampling the results generated by the chip under test. In order to check the results in real-time, all possible multiplication results are stored in a ROM-based look-up table. The chip can also be reset on demand (e.g., after finding an error).

The campaign lasted for a period of seven consecutive days. The software running on the microcontroller kept alternating between MM modules, continuously monitoring the outputs for erroneous bits. The error count for the overall campaign is given in Table II.

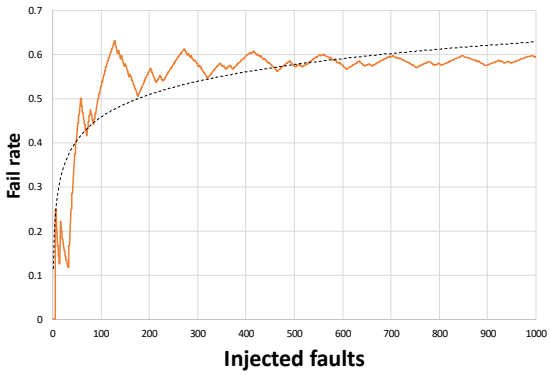
While the results are logical and in line with circuit size (i.e, v0 is the smallest version and also has the smallest number of errors) and redundancy strategies (i.e., no errors in TMR-protected versions), the error count is rather too low for drawing meaningful cross-sections. We highlight that in the last 36h of beam time, our setup was modified to activate only the largest of the hardened versions (v5), in an attempt to see an error in a TMR-protected variant. Still, no errors were detected for this variant. We proceed to further study the circuits in detail through simulation-based fault injection.

TABLE II
ERRORS DETECTED DURING NEUTRON CAMPAIGN

MM version	Pipeline stages	TMR?	Errors
v0	N/A	No	2
v1	N/A	Local	0
v2	N/A	Global	0
v3	21	No	27
v4	21	Local	0
v5	21	Global	0
v6	7	No	14
v7	7	Local	0
v8	7	Global	0



(a)



(b)

Fig. 2. Fail rate trend for the MM v0 circuit. A total of 1000 faults is injected in order to obtain a stable trend. The dashed line represents a logarithmic trendline.

A. Validation against Fault Injection Campaign

Our first aim was to validate that the results presented in Table II match with simulation-based fault injection. In order to perform this crosscheck, however, we first established how many faults have to be injected into the circuit to gain a reasonable insight into its reliability (i.e., how often faults become errors vs. how often faults are masked). For this purpose, we took the same MM v0 netlist that was utilized in the tapeout and proceeded to inject faults on it.

TABLE III
FAULT INJECTION CAMPAIGN FOR THE MATRIX MULTIPLICATION MODULES, TARGETING SEQUENTIAL ELEMENTS ONLY

MM version	TMR?	Pass	Fail	Fail rate (%)
v0	No	4167	5833	58.33
v1	Global	10000	0	0.00
v2	Local	10000	0	0.00
v3	No	8633	1367	13.67
v4	Global	10000	0	0.00
v5	Local	10000	0	0.00
v6	No	4802	5198	51.98
v7	Global	10000	0	0.00
v8	Local	10000	0	0.00

TABLE IV
FAULT INJECTION CAMPAIGN FOR THE MATRIX MULTIPLICATION MODULES, TARGETING COMBINATIONAL ELEMENTS ONLY

MM version	TMR?	Pass	Fail	Fail rate (%)
v0	No	75642	24358	24.35
v1	Local	97845	2155	2.15
v2	Global	97909	2091	2.09
v3	No	92557	7443	7.77
v4	Local	99809	191	0.19
v5	Global	99812	188	0.18
v6	No	71017	28983	28.98
v7	Local	99086	914	0.91
v8	Global	99837	163	0.16

Initially, for the analysis of the MM v0 circuit, we selected only flip-flops as targets and executed a fault injection campaign according to the methodology previously described. In Fig. 2, we plot the fail rate trend for this campaign. When 100 faults are accounted for in Fig. 2 (a), the fail rate does not appear to stabilize, meaning that it is possible we have not injected enough faults to see the circuit's fail rate saturate. When 1K faults are accounted for in Fig. 2 (b), a noticeably more stable profile appears that indeed captures the circuit's architectural vulnerability factor. Erring on the side of caution, we then performed 10K injections on each MM variant. Still, only sequential elements were considered as targets. The results are given in Table III. Not surprisingly, all TMR-protected variants have registered no errors.

Non-protected variants (v0, v3, and v6) have error rates that, at first glance, do not support the number of errors reported in Table II. However, it must be noted that the studied circuits have different sizes and architectures, and that the values reported in Table III ought to be derated over area for a fairer comparison. We later introduce a figure of merit (FoM) that considers this aspect and that can be used to compare same-architecture circuits.

In Table IV, we again perform fault injection, but this time focusing only on the combinational cells of the MM variants. A total of 100K faults[1] were injected per circuit. As expected, all variants now present at least some fail scenarios. All TMR-protected versions have output voters that are not themselves protected, and we made sure that voters were also targeted in our campaign. As a general trend, the non-protected variants v0, v3, and v6 still present much higher error rates than their protected counterparts.

[1] We have performed the same trend analysis shown in Fig. 2 for combinational logic, but omit it from the paper. The trend stabilizes before 10K samples.

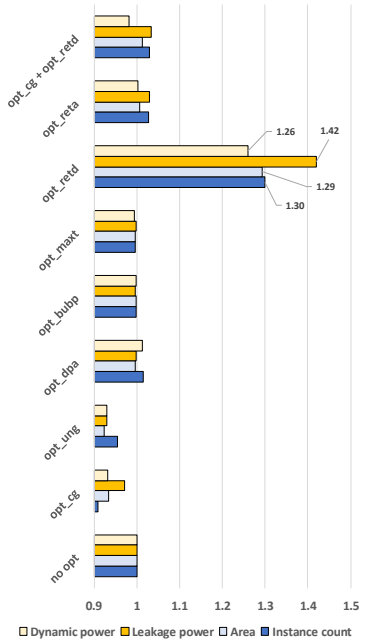


Fig. 3. Profile of the studied implementation variants for the ALU circuit. All values are normalized with respect to the baseline implementation.

V. RESULTS FOR THE SYNTHESIS OPTIMIZATION EXPLORATION

We are also interested in analyzing how implementation decisions influence the resiliency of circuits. More specifically, in this section, we investigate how logical synthesis and related optimizations promote changes in the circuit's structure that translate to changes in its resiliency. The circuits analysed include a 32-bit arithmetic logic unit (ALU), an AES crypto module [20], a PID industrial controller [22], a floating-point unit obtained from [23], and the MM v8 module previously utilized in our tapeout. For all circuits reported in this section, the technology (16nm) and standard cell library of choice (3rd party, commercial library) is the same utilized in our tapeout. Synthesis scripts also match, except for the optimizations that were purposefully turned on or off across the experiments.

1) 32-bit Arithmetic Logic Unit

We synthesized a relatively small 32-bit ALU that performs only sum and multiplication operations. The synthesis of the baseline ALU – when none of the studied optimizations is enabled – results in a circuit with 1503 cells that occupies 745um^2 of area, burns 1.80uW of leakage power, and 2.51mW of dynamic power when running at 1GHz. Next, we enabled the studied optimizations one by one and proceeded to measure the number of instances, area, leakage and dynamic power. For the sake of comparison, clock frequency was not changed for each variant, even for retimed variants that could benefit from it. The normalized results are shown in Fig. 3, where *no opt* stands for no optimization performed.

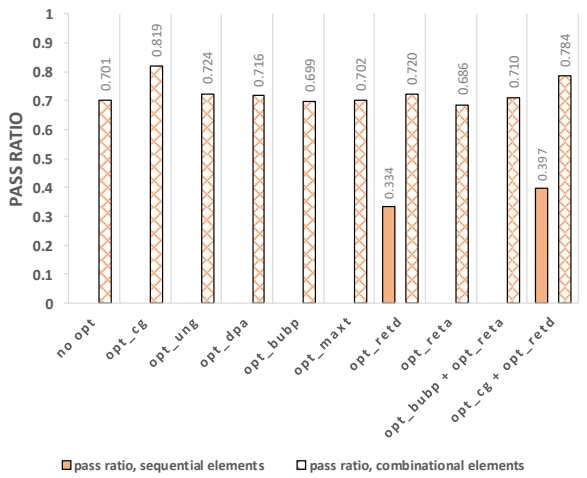


Fig. 4. Pass rates for sequential logic and combinational logic of different implementations of the ALU circuit.

For this specific ALU circuit, optimizations *opt_cg*, *opt_ung*, and *opt_retd* stand out. The first two clearly enable synthesis to generate more optimized circuits that are approximately 7% smaller and 3-7% more power efficient. Our observation is that clock gating logic brings small improvements because it makes use of integrated clock gating cells that otherwise are ignored by the synthesis. Even if counterintuitive, the use of clock gating integrated cells reduces area in this experiment. On the other hand, *opt_retd* increased all measured circuit characteristics by 26-42% (see data callouts in Fig. 3). One of the reasons for such increase is that retiming (both *opt_retd* and *opt_reta*) may, when deemed advantageous, clone flip-flops to reduce load on certain circuit nodes. For the studied circuit, it was verified that the number of flip-flops increases significantly, from 128 to 194, when *opt_retd* is applied. This, in turn, increases the area and power consumption considerably.

We have also studied pairs of optimizations to see if the results would change promptly (with respect to applying one optimization at a time). For most pairs, surprisingly, the changes in circuitry were minimal and could not be discerned from inherent "noise" that exists in the synthesis process. The pair *opt_cg* + *opt_retd*, however, stands out as it always produced a result that is a middle point between *opt_cg* and *opt_retd* – it has both the increased frequency enabled by retiming as well as the power savings of clock gating. For this reason, we also consider this pair of optimizations in Fig. 3.

While Fig. 3 would give a designer enough information to pick the best design point for the studied circuit, it does not give away any information on the circuit reliability – or how it shifts from one version to another. We resorted to fault injection once again to analyze if the circuit variants present different fail rates. The result is summarized in Fig. 4, where we differentiate sequential from combinational logic in order to better understand the effects of the chosen optimizations. On the leftmost portion of Fig. 4, we have the baseline circuit for which no optimizations were enabled (*no opt*).

It is important to note that the ALU circuit is an ideal candidate for this investigation: it is a small single cycle circuit with virtually no control path. This circuit was chosen

TABLE V
CHARACTERISTICS OF DIFFERENT IMPLEMENTATIONS OF THE AES CIRCUIT

Version	Cell count	Flops	Area (um^2)
no opt	10633	693	3716.30
opt_cg	10512	693	3646.06
opt_ung	10420	597	3533.72
opt_dpa	10675	693	3715.58
opt_bubp	10618	693	3662.34
opt_maxt	10625	693	3721.96
opt_retd	10867	**814**	**3858.87**
opt_reta	10890	782	3813.35
opt_bubp + opt_reta	**11198**	782	3855.29
opt_cg + opt_retd	10552	698	3657.00

precisely because it allows for direct correlation between any optimization choice and changes in reliability. Since the ALU circuit is so simplistic, any flip-flop injected with a fault leads to an error at the circuit output (i.e., there is no masking, so the pass bars are zeros, thus not shown). This type of circuit allows us to conclude that *opt_retd* brings changes that break this assumption quickly; now, masking does take place. This result is **evidence that certain synthesis optimization strategies can disrupt architectural design choices**. As previously mentioned, cloning of flip-flops takes place during *opt_retd*, which creates a scenario in which faults do not translate into errors since newly-created shadow flip-flops do not always have an active path to drive a primary output. This specific optimization reduces the fail rate by ~33% for the ALU circuit, which is explained by the fact that also ~33% of the flip-flops in the circuit are clones (66 out of 194).

Regarding combinational logic, we have observed fail rates in the range of 18-31%, whereas the baseline circuit displays a fail rate of 29.9%. The optimization with the highest pass rate is clock gating (*opt_cg)*. By definition, *opt_cg* targets sequential logic. However, faults in combinational gates are more frequently masked since gated flip-flops cannot latch an error if the enable signal is not asserted at the time of the fault injection.

Conversely, the optimization with the lowest pass rate is retiming for area (*opt_reta)*. We conjecture that retiming for area promotes changes in the circuit where distinct paths have more cells in common with one another. The cells that are "shared" between distinct paths become critical nodes that reach primary outputs in many ways, thus reducing the probability that injected faults are logically masked. In summary, when the studied circuit is synthesized with *opt_reta*, it is more likely that errors will occur than when the same circuit is synthesized with *opt_cg* only. However, when we take into account that the *opt_cg* version is smaller (see Fig. 3), the relative disparity between both versions is even more pronounced. This degree of disparity between commonly employed strategies is not specific to the ALU circuit. We have seen similar trends for other circuits, including a PID industrial controller, especially for strategies that affect the sequential elements of the circuit. In particular, the PID contains similar dataflow structures that are present in the ALU, but the PID controller has twice more flops than s ALU and effectively has more control logic than datapath logic. Nevertheless, we suppress the results for the PID controller, as they are very similar: the number of flip flops changes by ~50

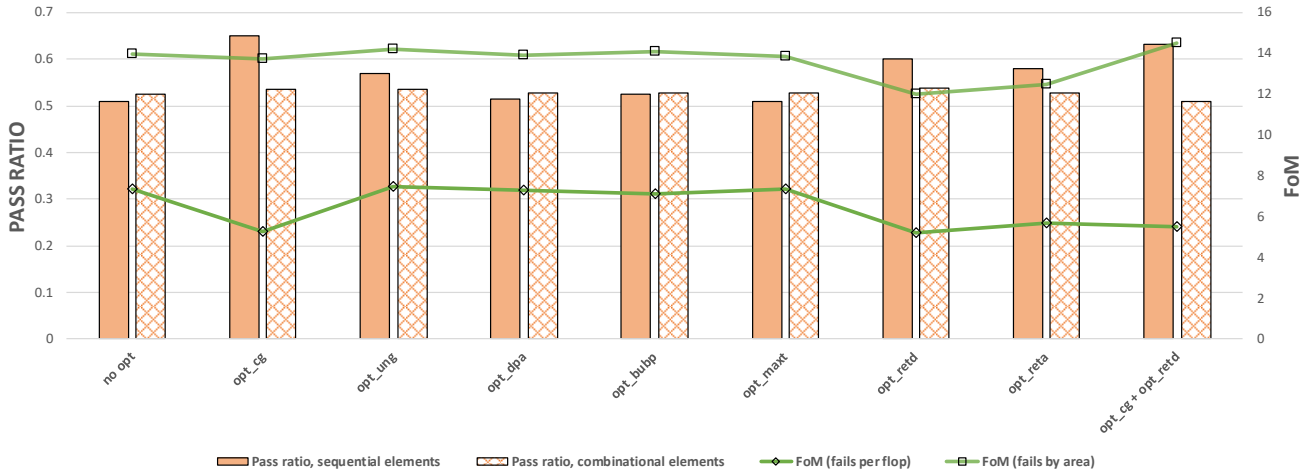


Fig. 5. Pass ratio for sequential logic and combinational logic of different implementations of the AES circuit. Figures of Merit for the fail rates per flop and by area are drawn as green lines.

when *opt_reta* is applied, *opt_cg* and *opt_ung* also produce small reductions in area.

2) AES Crypto Engine

For the AES crypto engine, we performed a nearly identical experiment where we synthesized many versions of the same circuit for a 1GHz frequency while enabling one optimization at a time. The results for each of these runs are given in Table V, where we bold the highest recorded values for the number of cells, the number of flops, and area. We highlight that this circuit is about one order of magnitude larger than the previously utilized ALU. It is clear that *opt_retd*, here, also leads to the highest area overhead as it adds more than 100 flip-flops that were not part of the original design. Conversely, *opt_ung* has the opposite effect, actually reducing the number of flops in the design from 693 to 597 and, consequently, the overall area.

Regarding combinations of optimizations, most pairs we studied revealed results that were barely discernible. In theory, two optimizations could present a linear superposition property. In practice, once one optimization is performed, it prevents others, most likely because they affect the same elements in the circuit. Nevertheless, we show the results for the *opt_cg* + *opt_retd* combination, as well as for the *opt_bubp* + *opt_reta* pair. This last one, in particular, increases the cell count by 5.3% and the circuit area by 3.7%.

Next, we performed a fault injection campaign utilizing the many variants of the AES circuit. Each variant was injected with 100K faults at random locations and at random times. The simulation testbench was carefully crafted such that faults are only injected when the circuit is actively computing, i.e., after reset and load operations already took place, but one of the encryption rounds is still being executed. We have also controlled the seed of simulation to make sure that identical (but randomized) input vectors are applied to all variants. The results of the campaign are shown in Fig. 5. Bars are utilized to display the combinational and sequential pass ratios.

We have previously alluded that the effectiveness of a given optimization should be a function of the area overhead as well as a function of the fail rate. In order to account for these two quantities at the same time, we have developed two figures of merit: *fails per flop* and *fails by area*. The former is calculated by dividing the number of injected faults that led to fail scenarios by the number of flops in the design. The latter is calculated by diving by the circuit area instead. Both metrics are drawn as green lines in Fig. 5. The lower the FoM, the better.

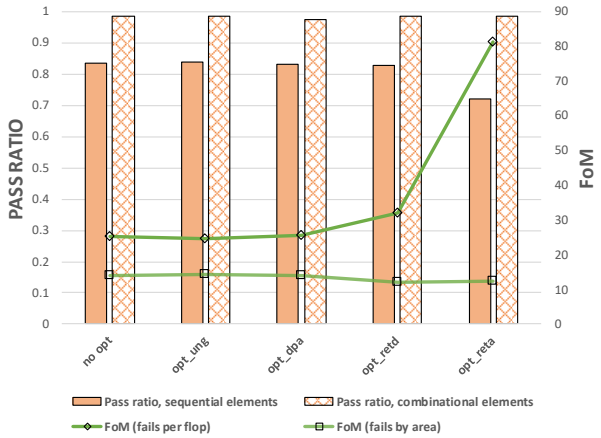


Fig. 6. Pass ratio for sequential and combinational logic of different implementations of a floating point unit. FoM for the fail rates per flop and by area are drawn as green lines. For clarity, optimizations with small changes are not shown.

When analyzing the results for the *fails per flop* FoM line in Fig. 5, it becomes clear that *opt_retd* is the better candidate (even if it has the third highest pass ratio at 0.601). The second best candidate is *opt_cg*, even if this optimization leads to a pass ratio of 0.65. Regarding the *fails by area* FoM, the best candidate is also associated with *opt_retd*, while the second best candidate is *opt_bubp* + *opt_reta*. It should be evident from the assessment above that the use of retiming is beneficial for the reliability of a circuit. However, the AES circuit is extremely regular, and it is conceivable that other circuits would not show the same trend. In order to investigate this hypothesis, we have performed an identical fault injection campaign into a single-precision floating-point unit. The results are shown in Fig. 6 and reveal that *opt_retd* is the best candidate when assessing *fails by area*. Regarding fails per

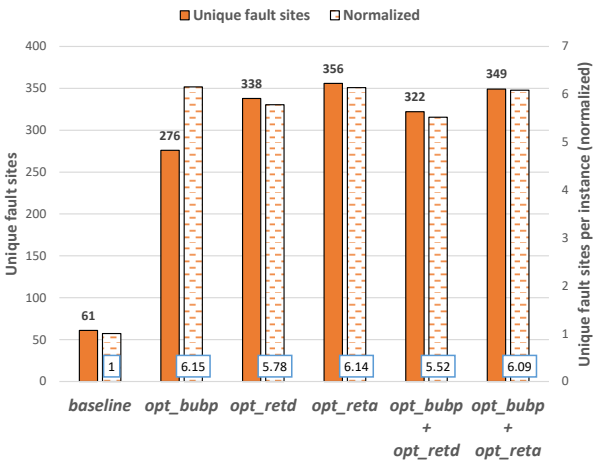


Fig. 7. Number of unique fault sites for variants of the MM v8 circuit (primary vertical axis). Unique fault sites per instance, normalized by the baseline (secondary vertical axis). Each variant was injected with 100K faults targeting combinational logic only.

flop, then *opt_ung* becomes the best candidate, while retiming strategies appear to be detrimental to reliability. In fact, *opt_reta* is an extreme case for this circuit. Here, the number of flip flops decreases by approximately 300 but every flop becomes more susceptible to faults: the pass rate drops from 83% to 72%. Therefore, in practice, circuits have to be analyzed individually, and **no one-size-fits-all recipe seems to exist for reliability improvement**.

3) Matrix multiplication (MM v8)

The one question we have not answered is whether the MM variants that are hardened can also be broken by the synthesis optimizations studied in this paper. First, we clarify that we disable flip-flop redundancy removal that the synthesis tool does by default: this is crucial; otherwise, any TMR scheme is optimized away. Next, we have synthesized the MM variants while enabling different synthesis optimizations. We made use of the exactly same tapeout netlists as an input to the synthesis tool, which was asked to rework them with the selected optimizations now enabled. Our interest is in showing whether a result like the one highlighted in Fig. 4 also happens for MM.

In order to do so, we have identified how many combinational cells, when injected with faults, are able to generate errors at the circuit output. We term these cells unique fault sites and we do not distinguish how often an injected fault at a site is able to turn into an error – one time is already sufficient. This result is shown in Fig. 7 for MM v8 variants. We highlight that this circuit has 58 outputs and that there are 61 unique fault sites for the *baseline* version of it. These are all related to voter logic that is often implemented by one MAJ cell. On the other hand, all variants that implement some form of logic restructuring led to hundreds of unique fault sites. Therefore, most of these sites are **not related to voter logic**. A normalized result is also plotted in Fig. 7. First, we divided the number of unique fault sites in each circuit variant by its number of instances. The value obtained for the baseline is set as the reference; all other values are normalized by it. From the plot, it is possible to appreciate that the variants have 5-6x more unique fault sites, even when instance counts are taken into account during normalization.

We have also performed a similar analysis for sequential logic. As expected, the *no opt* variant has zero fault sites as TMR works as expected. The *opt_bubp* variant does not change the number of flops of this circuit, thus the number of faults sites remained zero. The retiming variants, i.e., *opt_retd* and *opt_reta*, both led to the creation of a single fault site. In summary, retiming strategies can create fault scenarios in TMR-protected circuits that were not originally present. This can happen in both sequential and combinational logic.

Finally, in Table VI, we show the required execution times for the fault injection campaigns for the ALU and FPU circuits. These campaigns were executed on an Intel(R) Xeon(R) Gold 5122 CPU @ 3.60GHz with 20 concurrent threads; thus, the execution times are relatively modest even for 100K faults per variant and grow approximately linearly with circuit size.

TABLE VI
EXECUTION TIMES FOR FAULT INJECTION CAMPAIGNS

Circuit	Time (s)	Circuit	Time (s)
ALU (*no opt*)	671	**FPU** (*no opt*)	5555
ALU (*opt_cg*)	676	**FPU** (*opt_cg*)	5498
ALU (*opt_ung*)	670	**FPU** (*opt_ung*)	5495
ALU (*opt_dpa*)	669	**FPU** (*opt_dpa*)	4118
ALU (*opt_bubp*)	663	**FPU** (*opt_bubp*)	5551
ALU (*opt_maxt*)	668	**FPU** (*opt_maxt*)	5495
ALU (*opt_retd*)	677	**FPU** (*opt_retd*)	5407
ALU (*opt_reta*)	679	**FPU** (*opt_reta*)	5507

VI. DISCUSSION

In all the results presented in this paper, the same synthesis engine was utilized. It is possible, if not likely, that a tool from a different vendor could have led to different results. In a future work, we will make use of a "dynamic pipelining" strategy that is offered by another vendor. While this strategy is meant for use in datapaths, we believe it would have similar consequences to any TMR-protected logic. Furthermore, we cannot ignore the fact that synthesis tools are proprietary and that we do not have full access to their inner workings. It is therefore hard to separate noise from signal in some of our experiments. Nevertheless, for those experiments were counterintuitive results appear, we provide supporting arguments based on our collective experience in IC design.

In [11], the authors study the effect of synthesis on a reliability metric termed error propagation probability (EPP in their paper). Their results suggest that combinational logic becomes more susceptible to errors with an increase in synthesis effort. Here, we set the effort to high in all cases and proceed to selectively turn optimizations on and off. This makes a direct comparison very difficult.

Perhaps the most important takeaway message of this study is that the changes in circuit structure that are promoted by retiming appear relatively tame and constrained. Since simulation-based fault injection can target gates individually and precisely, it becomes possible to analyze the effects in a decoupled fashion. This same analysis would be order of

magnitudes harder in a neutron campaign. For instance, among the MM v8 circuit variants shown in Fig. 7, the error rate differs by a tenth of a percent. This is due to the fact that only 3-4% of the gates in the circuit get to become unique fault sites. Once again, this result would be relatively hard to show with accelerated particle testing where individual gates cannot be easily targeted at will.

It is worth noting that we cannot apply the FoM to compare/validate simulation to test data. This is because the circuits are implemented with different architectures and the FoM does not consider masking. Nevertheless, by looking at data in Table II w.r.t. Table III and IV, we can see that there is little agreement and the number of observed events might be too low to draw meaningful simulation to test data comparisons.

VII. Conclusion

Designing ICs is a challenging task, where engineers are asked to explore a vast optimization surface in order to reach the best trade-offs for area, power, and timing. When we consider reliability as part of this optimization problem, designers are now asked to further explore different strategies for architecture, redundancy schemes, and implementation options. In this paper, we have shown different approaches to account for each strategy. A thorough simulation-based fault injection campaign enabled us to compare different strategies in a matter of a few hours (see Table VI), a comparison that is not trivial to perform even in an accelerated test environment. While TMR-based schemes are highly effective and are often employed, we have shown that it is necessary to make sure that the redundancy strategy is not disturbed during synthesis runs. We have shown that retiming strategies can effectively increase the number of unique fault sites by as many as 6 times.

Acknowledgements

This work was supported in part by DARPA contract HR0011-16-C-0038 – Circuit Realization at Faster Timescales (CRAFT). This work was also supported in part by the EC through the European Social Fund in the context of the project “ICT programme” and by the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 886202. The authors would also like to acknowledge LANSCE for the valuable beam time.

References

[1] P. E. Dodd and L. W. Massengill, "Basic mechanisms and modeling of single-event upset in digital microelectronics," in *IEEE Trans. on Nuclear Science*, vol. 50, no. 3, pp. 583-602, June 2003.

[2] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," in *IEEE Trans. on Device and Materials Reliability*, vol. 5, no. 3, pp. 305-316, Sept. 2005.

[3] Quming Zhou and K. Mohanram, "Gate sizing to radiation harden combinational logic," in *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 1, pp. 155-166, Jan. 2006.

[4] D. B. Limbrick, N. N. Mahatme, W. H. Robinson and B. L. Bhuva, "Reliability-Aware Synthesis of Combinational Logic With Minimal Performance Penalty," in *IEEE Trans. on Nuclear Science*, vol. 60, no. 4, pp. 2776-2781, Aug. 2013.

[5] A. Balasubramanian, B. L. Bhuva, J. D. Black and L. W. Massengill, "RHBD techniques for mitigating effects of single-event hits using guard-gates," in *IEEE Trans. on Nuclear Science*, vol. 52, no. 6, pp. 2531-2535, Dec. 2005.

[6] P. K. Samudrala, J. Ramos and S. Katkoori, "Selective triple Modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs," in *IEEE Transactions on Nuclear Science*, vol. 51, no. 5, pp. 2957-2969, Oct. 2004.

[7] R. C. Lacoe, "Improving Integrated Circuit Performance Through the Application of Hardness-by-Design Methodology," in *IEEE Transactions on Nuclear Science*, vol. 55, no. 4, pp. 1903-1925, Aug. 2008, doi: 10.1109/TNS.2008.2000480.

[8] S. N. Pagliarini et al. "Exploring the feasibility of selective hardening for combinational logic," in *Microelectronics Reliability*, vol. 52, no 9-10, pp. 1843-1847, 2012.

[9] T. Calin, M. Nicolaidis, R. Velazco, "Upset Hardened Memory Design for Submicron CMOS Technology," in *IEEE Trans. Nucl. Science*, Vol. 43, No. 6, pp. 2874-2878, December 1996.

[10] M. Portolan and R. Leveugle, "A highly flexible hardened RTL processor core based on LEON2," in *IEEE Transactions on Nuclear Science*, vol. 53, no. 4, pp. 2069-2075, 2006.

[11] D. B. Limbrick, S. Yue, W. H. Robinson and B. L. Bhuva, "Impact of Synthesis Constraints on Error Propagation Probability of Digital Circuits," *2011 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, Vancouver, BC, 2011, pp. 103-111.

[12] D. Shin and S. K. Gupta, "Approximate logic synthesis for error tolerant applications," 2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010), Dresden, Germany, 2010, pp. 957-960, doi: 10.1109/DATE.2010.5456913.

[13] I. A. C. Gomes, et al. "Exploring the use of approximate TMR to mask transient faults in logic with low area overhead," in *Microelectronics Reliability*, vol. 55, no. 9-10, pp. 2072-2076, 2015.

[14] I. Albandes, et al. "Building ATMR circuits using approximate library and heuristic approaches," in *Microelectronics Reliability*, vol. 97, pp. 24-30, 2019.

[15] L. Entrena et al., "Constrained Placement Methodology for Reducing SER Under Single-Event-Induced Charge Sharing Effects," in *IEEE Trans. on Nuclear Science*, vol. 59, no. 4, pp. 811-817, Aug. 2012.

[16] S. N. Pagliarini and D. Pradhan, "A placement strategy for reducing the effects of multiple faults in digital circuits," *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*, Platja d'Aro, Girona, 2014, pp. 69-74, doi: 10.1109/IOLTS.2014.6873674.

[17] M. I. Bandan, S. Pagliarini, J. Mathew and D. Pradhan, "Improved Multiple Faults-Aware Placement Strategy: Reducing the Overheads and Error Rates in Digital Circuits," in *IEEE Transactions on Reliability*, vol. 66, no. 1, pp. 233-244, March 2017, doi: 10.1109/TR.2016.2643010.

[18] Xilinx, “Vivado High-Level Synthesis”. [Online] Available: https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html

[19] L. A. C. Benites and F. L. Kastensmidt, "Automated design flow for applying triple modular redundancy (TMR) in complex digital circuits," *IEEE 19th Latin-American Test Symposium* (LATS 2018), March 2018.

[20] S. Pagliarini, M. Martins and L. Pileggi, "Virtual characterization for exhaustive DFM evaluation of logic cell libraries," *2017 18th International Symposium on Quality Electronic Design (ISQED)*, Santa Clara, CA, 2017, pp. 93-98, doi: 10.1109/ISQED.2017.7918299.

[21] R. Usselmann, “AES (Rijndael) IP Core”. [Online] Available: https://opencores.org/project,aes_core. Accessed on: Dec. 5, 2020.

[22] T. Zhu, "PID controller". [Online] Available: https://opencores.org/projects/pid_controller. Accessed on: Dec. 5, 2020.

[23] R. Usselmann, "Floating Point Unit". [Online] Available: https://opencores.org/projects/fpu. Accessed on: Dec. 5, 2020.