# Non-interfering On-line and In-field SoC Testing

Tobias Strauch
R&D, EDAptix e.K.
Munich, Germany
Email: tobias@edaptix.com

*Abstract*—With increasing aging problems of advanced technologies, in-field testing becomes an inevitable challenge, on top of the already demanding requirements, such as the ISO26262 for automotive safety. SOCs used in space, automotive or military applications in particular are worst affected as the in-field failures in these applications could even be life threatening. We focus on on-line and in-field testing for Single Event Upsets (SEU, caused by a single ionizing particle) and aging defects (such as delay variation and stuck-at faults) which may appear during normal operation of the device. Interrupting normal operations for aging defects testing is a major challenge for the OS. Additionally, checkpointing with rollback-recovery can be costly and mission critical data can be lost in case of an SEU event. We eliminate many of these problems with our non-interfering in-field testing and recovery solution.

We apply a hardware performance improvement technique called System Hyper Pipelining (SHP), which combines well-known context switching (Barrel CPU) and C-slow retiming techniques. The SoC is enhanced with an SEU detection and ultra-fast recovery mechanism. We also use an RTL ATPG framework that enables the generation of software-based self-tests to achieve 100% coverage of all testable stuck-at-faults. The paper finishes with very promising performance-per-area and test-cycles-per-net results. We argue that our robust system architecture and EDA solution, designed and developed primarily for in-field testing of SoCs, can also be used for production and on-line testing as well as other applications.

*Index Terms*—In-field testing, on-line testing, SEU detection and recovery, aging related device failures, RTL ATPG, non-interfering testing, interleaved multi-threading

## I. Introduction

Chips used in aerospace, automotive, and military applications are subject to in-field failures that can be extremely mission costly or even life-threatening.

Cosmic ray phenomena such as solar particle events cause high radiant flux that lasts for hours to days, increasing the likelihood of single-event upsets (SEUs) by several orders of magnitude. With the advent of nanoscale (high-)performance computing, soft errors that impact the reliability of modern electronic systems even at ground level have become one of the most challenging issues for the semiconductor industry.

All parts of a design can be affected, including neural networks, where Failure In Time (FIT) rates can exceed safety standards, e.g. ISO 26262 for the automotive industry, by orders of magnitude, as shown in [1].

There are also device defects that can occur during in-field operation of the device and are mainly due to latent faults that may not be obvious or readily detectable during production or on-line testing but may develop over time under real-time applications in the field due to environmental conditions.

The industry is responding to these challenges with standardizations such as ISO 26262. These new requirements must coexist with existing applications and testing must be carefully scheduled to avoid impacting the applications on the device. Efficient scheduling for on-line and in-field testing can be a major challenge for the operating system as [2] and [3] clearly demonstrate. Additionally, checkpointing with rollback-recovery can be costly (power, timing, . . . ) [4] and mission critical data can be lost in case of an SEU event when a system rollback must be initiated.

In this paper, we introduce a robust SoC architecture and EDA software solution to cope with the aforementioned challenges. The main goal is to continuously test for SEU and aging faults during on-line testing and in-field operation without interfering with the normal operation as well as to recover from an SEU detection very efficiently.

In order to provide a self-contained work, we start the paper with a list of short introductions to the respective techniques on which this work is based, such as

- an interleaved multithreading technique (Section II),
- functional redundancy and failure recovery (Section III),
- aging-related failure detection (Section IV) and
- a gate inherent fault based RTL ATPG (Section IV).

Our work is introduced in Section VI and compared to related work in Section VII, before results are presented in Section VII.

## II. Barrel CPU and C-slow Retiming

Fig. 1a shows the basic structure of a sequential circuit with its combinational logic (CL) and original design registers (DR). Clock, in- and outputs are not shown for the sake of simplicity. The sequential circuit processes a single thread T(0) running at what we define here as macro-cycle speed.

*1) Barrel CPU:* A barrel processor is a CPU that switches between threads of execution every cycle. The design technique is also known as "interleaved" or "fine-grained" temporal multithreading. A modern example of a barrel RISC-V CPU is shown in [5].

Fig. 1b gives an abstract view of a design based on the barrel technique. The DRs are now replaced by memories (Mem) and the design is extended by a thread controller (TC). D is the number of threads the memory can hold (memory depth). The executed thread can now be freely selected within D threads (read pointer) and saved at the corresponding address (write pointer) using the thread index. The individual threads still run at macro-cycle speed.
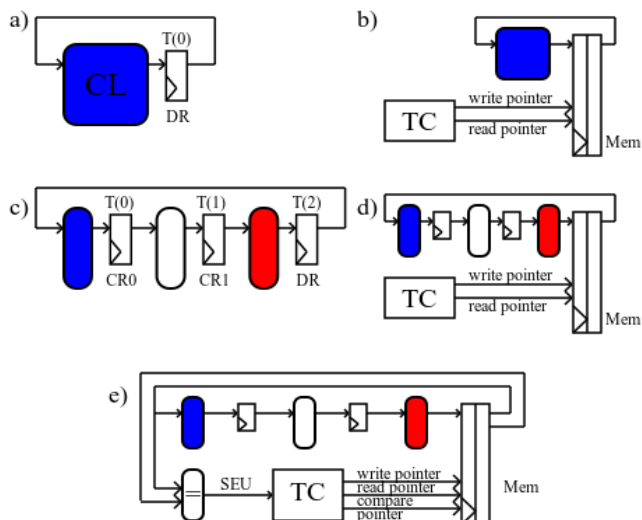
Fig. 1. a) Simplified single clock design. b) Applying barrel technique c) Applying C-slow retiming d) Applying System-Hyper-Pipeling (SHP) e) SEU detection and recovery based on C-slow retiming and applyed on SHP
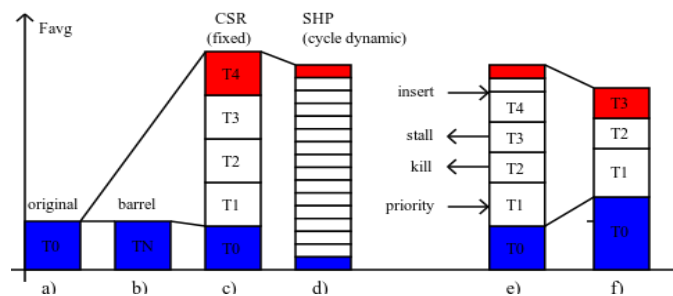


Fig. 2. Average thread performance (Favg) of different scenarios running a) Original design, b) Design with barrel, c) C-slow retiming and d-f) SHP technique .

*2) C-slow retiming (CSR):* The C-slow retiming (CSR) technique provides C copies of a given design by inserting an equal number of registers into each combinatorial path and therefore reusing the logic in a time sliced fashion [6].

Fig. 1c outlines the CSR technique. The original logic is sliced into C (here C=3) sections, and each original path now has C-1 additional registers running at micro-cycle speed. This results in C functional independent design copies T(0, ..., C-1) which use the logic in a time sliced fashion. Each thread has its own thread index. For each design copy it now takes C micro-cycles to achieve the same result as in one cycle of the original design (macro-cycle). The implemented register sets are called "CSR Registers" (CR).

*3) System Hyper Pipelining (SHP):* System hyper-pipelining (SHP) is a technique introduced in [7] that combines the barrel and C-slow retiming techniques mentioned above. Fig. 1d shows the modifications towards an SHP-ed design, which can run any number of threads ($T <= D$) in any possible interleaved order.

*4) Thread mixing:* When applying SHP on an SoC, the number of individual threads (D, barrel technique) as well as the number of interleaved executed design copies C for individual subblocks can vary. For example, the less timing critical Ethernet design does not need to use C-slow retiming to achieve the required performance and therefore only the barrel technique needs to be applied. Accelerators, on the other hand, are usually time-critical and only the C-slow retiming technique might be relevant. The CPU is based on SHP to achieve the best possible performance-per-area trade-off. This approach allows for an optimal thread mixing and best serves our purpose while providing an optimal performance-per-area trade-off.

*5) Load balancing:* Fig. 2 shows the advantages of the aforementioned techniques compared to the original design. The x-axis of the histogram shows different scenarios/solutions, the y-axis the system performance. Assuming a thread (T0) on

the original CPU runs at e.g. 80MHz on an FPGA (Fig. 2a).

The barrel CPU version allows context switching between multiple threads, but does not improve CPU performance as such (Fig. 2b) as it still runs at macro-cycle speed.

It can be seen how CSR improves the system performance of the original system implementation (Fig. 2c). System performance is no longer necessarily limited by the critical path of the original design or external memory access, but rather, for example, by the switching limit of the FPGA (e.g. 600 MHz). The design runs at micro-cycle speed. When using CSR, all threads run at the same speed and load balancing is not possible.

For executing multiple programs on multiple CPUs (symmetrical multi-processing), SHP allows a more efficient usage of the system resources (Fig. 2d to 2f). It adds the possibility to distribute the system performance over a minimum (C, Fig. 2c) and a maximum set of threads (D, Fig. 2d), whereas any solution in-between can be realized. Fig. 2e) shows a random example. This load balancing is handled by a TC and can be dynamically modified during runtime. Fig. 2f refers to more advanced SHP techniques as shown in [7], where more system performance is given to specific threads.

## III. REDUNDANCY AND FAILURE RECOVERY

So far we have briefly described well-known digital design concepts such as a barrel CPU and C-slow retiming as well as their combined application (SHP). These concepts can now be extended to detect and to recover from SEUs.

An SEU is a change of state caused for example by a single ionizing particle (ion, electron, photon...) hitting a sensitive node in a design. The change of state is a result of the free charge created by ionization in or near an important node of a logic element (e.g. register). The failure in device output or operation caused by the strike is called SEU.

The main techniques to detect an SEU are either based on spatial redundancy or temporal redundancy. Spatial redundancy is based on the replication of n-times the original module building n+1 identical redundant modules, where outputs are merged into a majority voter. Time redundancy is based on capturing the states multiple times to vote out a transient fault. The values are shifted by a delay. The idea is to be able to capture a majority of upset free values to be able to mask the fault. We define the level of redundancy as R.

Some approaches based on time redundancy use interleaved-multi-threading to detect and to recover from such an SEU. In a recent publication [8] the aforementioned barrel technique is applied on a RISC-V processor on selected CPU elements, such as program counter, register file, etc.. Identical threads are executed, and the results are compared. If a mismatch is detected, a recovery mechanism restores the system, using an auxiliary thread as reference.

The same basic idea is shown in [9], based on designs which use the aforementioned C-slow retiming technique. C-slow retiming inserts the same number of registers into each path to use the logic in a time-sliced fashion. It is demonstrated, how to enhance such a design with an SEU detection logic, how identical threads can be executed on such a design and how a design can recover from an SEU fault within a limited number of cycles. Fig. 1e shows the basic concept of our work (inspired by [9]) with the extension that the design registers are replaced by memories and the TC controls the recovery sequence after an SEU detection.

## IV. DETECTION OF AGING-RELATED FAILURES

*1) Using timing critical path measurement:* Aging related Bias Temperature Instability (BTI) and Hot Carrier Injection (HCI) faults affect the delay of individual cells and the overall path timing, as shown in [10] and [11]. An experiment using ring oscillators to demonstrate these aging effects on an FPGA is shown in [12].

Aging introduced faults can be modeled on RTL [13] and can already be considered during logic synthesis [14] by shortening critical paths or during the design, place and route steps when FPGAa are used [15].

Testing delay faults in functional mode is shown in [16]. The selection of timing critical paths for this purpose can be achieved by assertion guided SBST [17] during RTL verification with the help of statistical timing models (such as [18]) or on gate level using gate level timing information [19].

The hardware (HW) can be enhancement for critical delay measurement as shown in [20] or through robust and in-situ self-testing techniques outlined in [21].

Since traditional test methods interfere with normal operation, the issue of scheduling test tasks becomes very critical. Calculating periodic testing in embedded processors is discussed in [22] and at-speed tests using functional tests for delay faults is evaualted in [23]. Also power related issues during on-line and in-field testing [24] as well as OS related challenges [25] must be considered.

*2) Using transistor activation and propagation:* Aging-related delay variations can also affect hold time, pulse width, and other timing requirements that certainly cannot be continuously measured. Another aging-related problem is the time-dependent dielectric breakdown (TDDB) of transistors [26], which ultimately leads to a fatal failure of the transistor. We argue that activation and comparison with a reference value (after propagation) of all testable signals (also known as stuck-at testing), helps to identify aging-related transistor TDDB failures that cannot be detected by critical path measurement.

## V. GATE INHERENT FAULT (BASED RTL ATPG)

The term "RTL ATPG" defines the methodology for generating stuck-at faults (SAF) test patterns based on RTL design descriptions. This can be done using dedicated functional tests which are executed on the device. An overview of software based self-tests (SBST) is given in [27]. A promising Gate Inherent Fault (GIF) RTL ATPG model was presented in [28].

The GIFs are extracted from each complex RTL primitive (multiplier, adder, shifter, case-statement, etc.) of the RTL source code individually. They are related to the internal logic paths of a complex gate. They are not related to any net/signal or gate in the gate level design. It is observed that when all GIFs on RTL are covered (100%) and the same stimulus is applied on gate level, then all testable gate level SAFs of the netlist are covered (100%) as well. The GIF model is therefore synthesis independent.

The GIF model can be applied on any alternative language construct (multiplier, etc.) or any combination of language constructs as well. The key point is, that the GIF model is related to internal paths of complex gates and not to signals in the given RTL design nor to nets in its gate level representation.

### A. The GIF-GO model definition

Under the proposed GIF gate output (GIF-GO) model, a GIF is described by a quadruple (gi, go, i, $\alpha$) where gi is a gate input, go is a gate output, i is an index and $\alpha \in \{0,1\}$. The fault (gi, go, i, $\alpha$) is detected by a test t that satisfies the following conditions:

1. The test t detects the path fault gi to go with index i (gi-go-i).
2. The fault free value of **gate output go** under t is $\alpha$.
3. In the presence of the fault gi-go-i, the output value go = !$\alpha$.

In other words, t propagates the effects of a gi-go path fault with index i to the gate output go. The output's value is $\alpha$ in the fault free circuit and !$\alpha$ in the presence of the fault.

An alternative view on this argues that the functionality of each (complex) gate can be defined by a Karnaugh map. The GIF model now states, that each '1' and '0' entry of the gate's Karnaugh map must be sensitized and propagated to primary outputs.

### B. Logic duplication

An important element of RTL synthesis is logic duplication. Duplicated logic can generate net faults which are not detected when a test set is used that is based on the GIF-GO model. Therefore the final RTL fault model needs to consider logic duplication. All outputs of a design are called primary outputs (PO). In case of a sequential netlist, register data inputs are considered as PO as well.

### C. The GIF-PO model definition

Under the proposed GIF-PO model, a GIF is described by a quintuple (gi, go, i, j, $\alpha$) where gi is a gate input, go is a gate output, i is an index, j is a primary output and $\alpha \in \{0,1\}$. The fault (gi, go, i, j, $\alpha$) is detected by a test t that satisfies the following conditions:

1. The test t detects the path fault gi to go with index i (gi-go-i).
2. The fault free value of **primary output j** under t is $\alpha$.
3. In the presence of the fault gi-go-i, the primary output value j = $!\alpha$.

In other words, t propagates the effects of a gi-go path fault with index i to the primary output j. The primary output's value is $\alpha$ in the fault free circuit and $!\alpha$ in the presence of the fault.

## VI. OUR WORK

Efficient scheduling for on-line and in-field testing can be a major challenge for the operating system as [2] and [3] clearly demonstrate. Additionally, checkpointing with rollback-recovery can be costly (power, timing, . . . ) [4] and mission critical data can be lost in case of an SEU event when a system rollback must be initiated.

The unique contribution of our work is that we demonstrate how an interleaved multithreaded SHP architecture can be utilized for non-interfering on-line and in-field testing. Without interrupting normal operations, we demonstrate

- how to detect and recover from SEU faults,
- how to detect faults generating functional mismatches and
- how to detect delay faults caused by aging.

As far as the authors are aware, this simultaneous approach has not been proposed before. The following steps are executed:

### A. Our work: hardware related

*1) SoC specification and preparation:* For each element of the SoC, such as the CPU, communication, and acceleration peripherals, etc., we individually specify the parameters C and D, where C refers to the number of design copies we achieve by applying C-slow retiming and D refers to the number of threads we want to store (barrel technique, memory depth).

*2) Applying barrel technique (manually):* We then manually improve the design by replacing registers with a set of registers (or memory bits) and by adding the appropriate read and write logic to the design for individual thread execution.

*3) Applying C-slow retiming technique (automatically):* The design is automatically improved by incorporating the C-slow retiming technique. This timing driven automatic register insertion technique is performed on RTL as presented in [29].

*4) Inserting SEU detection and recovery logic:* The design is further manually optimized to support the SEU detection and recovery mechanism similar to the concept shown in [9].

Memory read port: Based on the modification to support the barrel technique, design states of individual threads are stored in memories or small register sets, depending on the number of maximum threads (memory depth, D). Redundant threads may be stored in locations as far apart as possible. Additionally, a 2nd read port is added to the memories.

Thread controller: The TC drives the write port to store a design state (or not). It also controls the read ports to a) start execution of a thread cycle and b) to compare its state with a second (redundant) thread at the beginning of a cycle execution (see Fig. 1e).

Comparison logic: Additional comparison logic detects mismatches between the two selected threads. This logic can be pipelined similar to the scheme used for C-slow retiming.

Algorithm: The algorithm for detecting and recovering from an SEU is based on the concept that redundant thread cycles are only completed (stored) when all threads start with identical state values. All threads start from individual memory locations. These starting state values are then compared, while the threads are propagated through the C-slow retimed logic.

If no mismatch is discovered, the resulting state is written R times and normal operation continues. If a mismatch is detected, no state is overwritten and the cycle is repeated. The TC recognizes the results of the majority voting and replaces the start conditions of faulty threads with one of the correct threads.

Enhancements: If the SEU detection period takes longer than the execution, then the executed threads can be stored in alternating memory locations to avoid overwriting valid thread states. Another improvement is to replace the additional read port with a pipelined state capture register and to update the associated comparison logic accordingly. The TC then ensures that two consecutive threads can be compared. These registers can be the same registers inserted for the C-slow retiming technique.

*5) HW adaption for RTL ATPG:* In order to continuously test for SAFs certain HW related optimizations are required. These can be features like loop-back logic or an overwriting mechanism to set a counter into a defined state by software. This will become clearer in the next section.

### B. Our work: EDA software related

In this section, we present a framework based on an advanced RTL simulator and a coverage database viewer. The goal is to generate functional tests that can be run on the device during on-line testing or in-field operation and collect the maximum number of GIFs when executed. The RTL simulator recognizes all relevant GIFs of the source code and passes their coverage throughout the logic during functional simulation. Sequential functional tests typically stimulate and propagate GIFs over many execution cycles until they can be observed at relevant registers by the application running on the device. In other words, the test result should be different in the presence of a fault compared to the fault free behavior.

On the SHP-based HW, GIF-related threads do not interfere with normal operation and can be scheduled to run in parallel. It can be beneficial to test more safety-critical logic, such as control logic, more frequently than less critical sections, such as an FPU for instance.

Due to the complexity of the GIF test pattern generation on RTL, it is almost imperative to split the overall task into multiple test sets, most likely related to individual sub-designs. For each test, the GIF coverage characteristic is stored in a database and the results of a single test or multiple test runs can be analyzed using a database viewer. When a GIF cannot be covered, it is usually an indicator of redundant logic.

An example of a database viewer is given in Fig. 3. The SoC design contains a CPU, an SDRAM controller as well as

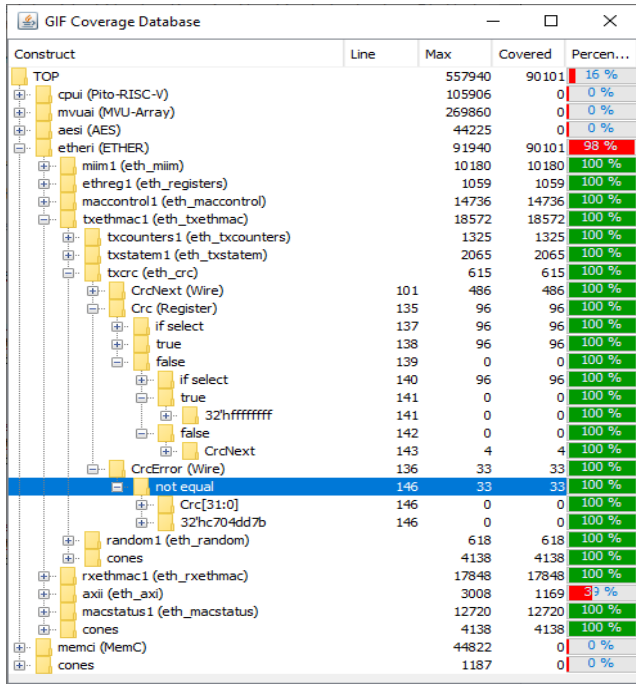| Construct | Line | Max | Covered | Percen... |
|---|---|---|---|---|
| TOP | | 557940 | 90101 | 16 % |
| cpui (Pito-RISC-V) | | 105906 | 0 | 0 % |
| mvuai (MVU-Array) | | 269860 | 0 | 0 % |
| aesi (AES) | | 44225 | 0 | 0 % |
| etheri (ETHER) | | 91940 | 90101 | 98 % |
|   miim1 (eth_miim) | | 10180 | 10180 | 100 % |
|   ethreg1 (eth_registers) | | 1059 | 1059 | 100 % |
|   maccontrol1 (eth_maccontrol) | | 14736 | 14736 | 100 % |
|   txethmac1 (eth_txethmac) | | 18572 | 18572 | 100 % |
|     txcounters1 (eth_txcounters) | | 1325 | 1325 | 100 % |
|     txstatem1 (eth_txstatem) | | 2065 | 2065 | 100 % |
|     txcrc (eth_crc) | | 615 | 615 | 100 % |
|       CrcNext (Wire) | 101 | 486 | 486 | 100 % |
|       Crc (Register) | 135 | 96 | 96 | 100 % |
|         if select | 137 | 96 | 96 | 100 % |
|         true | 138 | 96 | 96 | 100 % |
|         false | 139 | 0 | 0 | 100 % |
|           if select | 140 | 96 | 96 | 100 % |
|           true | 141 | 0 | 0 | 100 % |
|             32'hffffffff | 141 | 0 | 0 | 100 % |
|           false | 142 | 0 | 0 | 100 % |
|             CrcNext | 143 | 4 | 4 | 100 % |
|       CrcError (Wire) | 136 | 33 | 33 | 100 % |
|         not equal | 146 | 33 | 33 | 100 % |
|           Crc[31:0] | 146 | 0 | 0 | 100 % |
|           32'hc704dd7b | 146 | 0 | 0 | 100 % |
|     random1 (eth_random) | | 618 | 618 | 100 % |
|     cones | | 4138 | 4138 | 100 % |
|   rxethmac1 (eth_rxethmac) | | 17848 | 17848 | 100 % |
|   axii (eth_axi) | | 3008 | 1169 | 39 % |
|   macstatus1 (eth_macstatus) | | 12720 | 12720 | 100 % |
|   cones | | 4138 | 4138 | 100 % |
| memci (MemC) | | 44822 | 0 | 0 % |
| cones | | 1187 | 0 | 0 % |

Fig. 3. Snapshot of the coverage viewer GUI.

some peripherals such as an Ethernet core (here shown partly unfolded). All testcases related to this core are selected and their accumulated GIF coverage is displayed. It can be seen that some GIFs are covered (e.g. if-then-else or not-equal construct) and that the relative coverage on the Ethernet core itself reaches 98%. The other cores have low GIF coverage because only Ethernet core related testcases are merged in this example.

Hard to cover faults are the reason why the test pattern generation process is usually accompanied by HW adjustment efforts. This includes the possibility of setting counter registers via SW. Also, including loopback functionality in communication peripherals is quite common for GIF testing purposes if it is not already present in the design.

## VII. RELATED WORK

In [8], a SEU detection and recovery mechanism is proposed based on the concept we referred to as the barrel technique in Section II. Our work follows the concept outlined in [9], which uses C-slow retiming for interleaved multithreading.

In [8], redundant threads are executed and once a mismatch is detected, an auxiliary thread is used for recovery, which in turn may be subject to an SEU fault. In our work, no fixed auxiliary thread is used. It follows the rule that the states of all redundant threads are only overwritten when their start conditions (register values) are identical. Once a mismatch is detected, the thread controller replaces the failing thread with one of the remaining correct threads (not necessarily a single very specific thread) and initiates an ultra-fast recovery mechanism.

Approaches solely based on the barrel technique [8] reduce the system performance with each additional redundant thread due to insufficient logic sharing. In contrast, the advantage of the C-slow retiming approach is that there is only a small degradation in the maximal thread performance (due to register insertion) when running C threads on the system, while dramatically increasing the performance-per-area factor at the same time [9].

It is not clear to the authors how the comparison logic proposed in [8] can be fast enough to detect SEU faults at the end of a single cycle. No performance results are presented in [8]. In our work, the SEU detection logic compares register values at the start of a cycle and the comparison logic can be pipelined following the C-slow retiming technique (Fig. 1e). In an extended version, threads can be continuously stored in an alternating register bank to be used for normal operation but also to have a backup version after completion of the comparison task to be used by a simple rollback mechanism.

In [30] Riefert et al. demonstrate the use of SAT solvers within an RTL ATPG framework for SBST of in-field testing of a processor. The flow still depends on gate level faults and repetitive gate level fault simulation steps, which makes its usage for large SoCs questionable. In the GIF model-based solution the test pattern are generated entirely on RTL to generate test pattern for in-field execution with 100% coverage of all testable SAFs on gate level.

Table II shows the SAF coverage (SAFC) reported in the literature [31]–[39] for various IP blocks, which are used for SBST based SAF detection. Only one work reports 100% SAFC. It is based on an AES example [31]. With our demonstrated framework tests can be generated for 100% coverage of all testable SAFs on the complete SoC for in-field testing guided by the database viewer in an interactive process.

Gao et al. [40] propose a Time-Multiplexed Online Checking (TMOC) scheme using embedded blocks for checker implementation, which enables various system parts to be checked dynamically during in-field operation in a time-multiplexed fashion. Also, a reliability analysis for optimal periodic testing of intermittent faults that minimizes the test cost was introduced by Kranitis et al. in [41]. It can be argued that with our interleaved and non-interfering solution task scheduling for online and in-field testing becomes less challenging.

## VIII. RESULTS

*1) Design preparation, applying SHP:* For our SoC reference design we use BARVINN [5], which is based on a barrel CPU (RISC-V) and a set of Matrix Vector Units (MVUs) optimized for AI algorithms. We also added a cryptographic (AES) and a communication (Ether) peripheral as well as an SDRAM memory controller (MemC). Table I shows the GIF number for each module.

We apply CSR (C=4) on the CPU and all peripherals. 33% of the MVU designs can be removed as the remaining MVU blocks can now run in a time-sliced fashion. We also apply the barrel technique on the Ethernet core.

We base our results on FPGA technology (AMD, Kintex) as FPGAs are used in space, automotive and military applications and use the term area synonymously with LUTs. Our reference design is also implemented with ASIC technology (Sky130). Here, the term area includes the standard cell area as well as the additional area resulting from the use of small memory cells

TABLE I

SoC MODULE PERFORMANCE-PER-AREA RESULTS AND TEST-CYCLE-PER-NET (TCPN) FOR FPGA AND ASIC IMPLEMENTATIONS.

| | GIF [k] | FPGA, original | | | FPGA, optimized | | | | | ASIC, original | | | ASIC, optimized | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | LUT [k] | Perf [M Hz] | PpA [MHz /k] | LUT [k] | Perf [M Hz] | PpA [MHz /k] | PpAr [%] | TCPN | Area [k μm²] | Perf [M Hz] | PpA [MHz/ kμm²] | Area [k μm²] | Perf [M Hz] | PpA [MHz/ kμm²] | PpAr [%] | TCPN |
| CPU | 106 | 10.4 | 250 | 23.9 | 13.0 | 675 | 51.8 | 2.16 | 0.97 | 65.3 | 155 | 2.38 | 117 | 385 | 3.27 | 1.37 | 2.31 |
| MVU | 270 | 31.7 | 250 | 7.87 | 41.8 | 598 | 14.3 | 1.81 | 0.87 | 186 | 150 | 0.80 | 297 | 337 | 1.13 | 1.41 | 1.61 |
| AES | 44.2 | 8.46 | 310 | 36.6 | 9.86 | 781 | 79.2 | 2.16 | 0.18 | 50.7 | 329 | 6.50 | 84.6 | 667 | 7.89 | 1.21 | 0.36 |
| Ether | 91.9 | 12.9 | 449 | 34.7 | 15.8 | 862 | 54.6 | 1.57 | 0.72 | 85.5 | 341 | 3.99 | 169 | 739 | 4.36 | 1.09 | 1.97 |
| MemC | 44.8 | 6.75 | 296 | 43.8 | 7.98 | 795 | 99.6 | 2.27 | 0.32 | 45.6 | 247 | 5.43 | 92.8 | 556 | 5.99 | 1.10 | 1.05 |

TABLE II

SAF COVERAGE (SAFC) AND TEST-CYCLES-PER-NET (TCPN) NUMBERS.

| | FPGA | ASIC | [31] | [32] | [33] | [34] |
|---|---|---|---|---|---|---|
| Source | SoC | SoC | AES | SoC per. | VLIW | MIPS |
| SAFC [%] | 100 | 100 | 100 | 94.92 | 98.3 | 97.46 |
| TCPN | 0.61 | 1.46 | n.a. | n.a. | 0.024 | n.a. |
| | [35] | [36] | [37] | [38] | [39] | |
| Source | Processor | | | | | |
| SAFC [%] | 92.3 | 92.7 | 90.03 | 93.74 | 92.2 | |
| TCPN | 9.19 | 0.13 | n.a. | 0.18 | 0.10 | |

required to support the SHP technology. For FPGA technology we chose D=16 and for ASIC technology D=8.

Table I shows the original and the optimized area as well as the respective performance. Based on that, the performance per area factor (PpA) is listed. The increase of the relative PpA number (PpAr) is also given for each design block.

In a multiple core lockstep configuration, the PpAr remains constant, whereas in our proposed system architecture, the PpAr improves significantly for both FPGAs and ASICs. The idea of the overall concept is to use this performance gain for non-interfering testing. The workload of the test application can then be adapted to suit on-line and in-field test requirements.

*2) SEU detection and Recovery:* All design blocks are capable of interleaved multi-threading supporting multiple identical subsequent threads. We chose to execute three redundant threads (R=3) and added a TC as well as the SEU detection logic mentioned above to the SoC. Since the number of redundant threads is less than the number of executed threads (R<C) and because the SEU detection logic is fast enough, no intermediate thread context storing is necessary.

Due to the SEU detection and recovery logic insertion, the FPGA SoC LUT count increased by 0.5% in average and the average area increase for the ASIC is 0.6%), which is not shown in Table I. Since our methodology is based on C-slow retiming, we expect the same advantages in power consumption compared to alternative approaches, as reported in [9].

*3) Stuck-at detection:* We generated non-interfering test-cases for each individual SoC block using the EDA software presented in Section VI-B. We achieve 100% SAF coverage of all testable faults on gate-level. The area impact of the HW enhancements is neglectable.

Table I shows the test-cycles-per-net (TCPN) for each individual SoC block. SBST based SAF coverage is shown in

Table II for crypto-devices [31] (100% stuck-at-fault coverage, (SAFC)), SoC communication peripherals [32] (95% SAFC) and processors [33]–[39] (92.2% - 98.2% SAFC). We calculate an average TCPN for our FPGA implementation of 0.61 and 1.46 for the ASIC implementation respectively (listed in Table II). Alternative work reported here with lower TCPN [33], [36], [38], [39] do not reach 100% SAFC and the design flow reported in [33] is also highly optimized.

*4) Fault injection simulation:* In our demonstrated methodology, SEU events and aging-induced errors that result in a functional sequential mismatch are detected through design state comparison. Any SAF caused by production or aging issues is detected through a comprehensive functional testing program, resulting in 100% SAF coverage of all testable faults on gate level. Fault injection simulation does not provide any meaningful results in this context and is therefore not used.

## IX. SUMMARY

To meet increasingly challenging safety requirements, SoCs must be designed to carry out in-field testing (ISO26262). Interrupting normal operations for aging defects testing is a major challenge for the OS. Additionally, checkpointing with rollback-recovery can be costly and mission critical data can be lost in case of an SEU event. To drastically reduce these problems, we use a robust system architecture based on an interleaved multi-threaded HW concept (system-hyper-pipelining, SHP), which combines the advantages of context switching (barrel technique) and C-slow retiming. We also enhance this structure by an SEU detection and fast recovery mechanism.

In this paper we concentrate on SEU detection and recovery as well as on delay measurement and SAF testing during normal in-field operation. The area overhead for inserting the SEU detection logic is extremely low and the recovery period is ultra-fast. Our proven RTL ATPG flow enables the generation of 100% SAF tests of all testable faults and the area impact to perform these software-based tests is negligible. The tests do not interfere with normal operation and can be dynamically scheduled depending on the application's workload and safety requirements.

## REFERENCES

[1] G. Li et al., "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC), Nov. 2017.

[2] Y. Li, O. Mutlu, S. Mitra, "Operating System Scheduling for Efficient On-line Self-Test in Robust Systems", IEEE/ACM Inter. Conf. on Computer-Aided Design - Digest of Technical Papers 2009, 2-5 November 2009, San Jose, CA, USA, pp . 1-8

[3] N. Bartzoudis, V. Tantsios, and K. McDonald-Maier, "Dynamic Scheduling of Test Routines for Efficient Online Self-Testing of Embedded Microprocessors", 14th IEEE International On-Line Testing Symposium 2008, 7-9 July 2008, Rhodes, Greece, pp. 1-4

[4] V. Izosimov, P. Pop, P. Eles, and Z. Peng, "Synthesis of Fault-Tolerant Embedded Systems with Checkpointing and Replication", Third IEEE Inter. Workshop on Electronic Design, Test and Applications, DELTA 2006, 17-19 January 2006, Kuala Lumpur, Malaysia, pp. 1-8.

[5] M. Askarihemmat, S. Wagner, O. Bilaniuk, Y. Hariri, Y. Savaria, and J. David, "BARVINN: Arbitrary Precision DNN Accelerator Controlled by a RISC-V CPU", 28th Asia and South Pacific Design Automation Conf., ASP-DAC 2023, 16-19 Jan. 2023, Tokyo, Japan, pp. 483 – 489.

[6] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry", 631 Algorithmica, vol. 6, nos. 1–6, pp. 5–35, Jun. 1991.

[7] T. Strauch, "Connecting Things to the IoT by Using Virtual Peripherals on a Dynamically Multithreaded Cortex M3", IEEE Trans. on Circuits and Systems I: Regular Papers, Vol. 64, Issue 9, Sep. 2017, pp. 2462-2469.

[8] M. Barbirotta, A. Cheikh, A. Mastrandrea, F. Menichelli, M., and M. Olivieri, "Evaluation of Dynamic Triple Modular Redundancy in an Interleaved-Multi-Threading RISC-V Core", Journal Low Power Electron. Appl. 2023, Vol. 13, Issue 2, pp. 1 - 13.

[9] T.Strauch, "Using C-Slow Retiming in Safety Critical and Low Power Applications", FPGAs and Parallel Architectures for Aerospace Applications, Springer Inter. Publishing Switzerland 2016, DOI 10.1007/978-3-319-14352-1, Chapter 12.

[10] M. Ebrahimi, F. Oboril, S. Kiamehr, and M. Tahoori, "Aging-aware logic synthesis", 2013 IEEE/ACM Intern. Conf. on Computer-Aided Design (ICCAD), 18-21 Nov. 2013, San Jose, CA, USA , pp. 61-68.

[11] A. Baba, and S. Mitra, "Testing for Transistor Aging", 27th IEEE VLSI Test Symposium, 3-7 May 2009, Santa Cruz, CA, USA, pp. 215-220.

[12] A. Amouri, F. Bruguier, S. Kiamehr, P. Benoit, L. Torres, and M. Tahoori, "Aging effects in FPGAs: an experimental analysis", 24th International Conference on Field Programmable Logic and Applications (FPL), 2-4 September 2014, Munich, Germany, pp. 1-4.

[13] N. Koppaetzky, M. Metzdorf, R. Eilers, D. Helms, and W. Nebel, "RT level timing modeling for aging prediction", Design, Automation & Test in Europe Conference & Exhibition (DATE), 14-18 March 2016, Dresden, Germany, pp. 297-300.

[14] Y. Lu, S. Duan, and T. Kazmierski, "A New Ageing-Aware Approach Via Path Isolation", Forum on Specification & Design Languages (FDL), 10-12 September 2018, Garching, Germany, pp. 1-5.

[15] A. Amouri, and M. Tahoori, "High-level aging estimation for FPGA-mapped designs", 22nd Intern. Conf. on Field Programmable Logic and Applications (FPL), 29-31 Aug. 2012, Oslo, Norway, pp. 284-291.

[16] V. Singh, M. Inoue, K. Saluja, and H. Fujiwara, "Instruction-Based Self-Testing of Delay Faults in Pipelined Processors", IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol. 14, Issue: 11, Nov. 2006, pp. 1203 - 1215.

[17] Y. Oddos, K. Morin-Allory, and D. Borrione, "Prototyping Generators for on-line test vector generation based on PSL properties", IEEE Design and Diagnostics of Electronic Circuits and Systems, 11-13 April 2007, Krakow, Poland, pp. 1-6.

[18] L. Wang, J. Liou, and K. Cheng, "Critical path selection for delay fault testing based upon a statistical timing model", IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 23, Issue: 11, Nov. 2004, pp. 1550 - 1565.

[19] X. Fu, H. Li, and X. Li, "Testable Path Selection and Grouping for Faster Than At-Speed Testing", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 20, Issue: 2, Feb. 2012, pp. 236 - 247.

[20] E. Stott, Z. Guan, J. Levine, J. Wong, and P. Cheung, "Variation and Reliability in FPGAs", IEEE Design & Test, Vol. 30, Issue 6, Dec. 2013, pp. 50 - 59.

[21] J. Li, and M. Seok, "Robust and In-Situ Self-Testing Technique for Monitoring Device Aging Effects in Pipeline Circuits", 51st ACM/EDAC/IEEE Design Automation Conference (DAC), 1-5 June 2014, San Francisco, CA, USA, pp. 1-6.

[22] N. Kranitis, A. Merentitis, N. Laoutaris, G. Theodorou, A. Paschalis, D. Gizopoulos, C. Halatsis, "Optimal Periodic Testing of Intermittent Faults In Embedded Pipelined Processor Applications", Proceedings of the Design Automation & Test in Europe Conference (DATE), 6-10 March 2006, Munich, Germany, pp. 1-6.

[23] M. Kakoee, V. Bertacco, L. Benini, "At-Speed Distributed Functional Testing to Detect Logic and Delay Faults in NoCs", IEEE Trans. on Computers, Vol. 63, Issue 3, March 2014, pp. 703 - 717.

[24] M. Haghbayan, A. Rahmani, M. Fattah, P. Liljeberg, J. Plosila, Z. Navabi, and H. Tenhunen, "Power-aware online testing of manycore systems in the dark silicon era", Design, Automation & Test in Europe Conference & Exhibition (DATE), 9-13 March 2015, Grenoble, France, pp. 435-440.

[25] "Y. Li, O. Mutlu, S. Mitra, "Operating system scheduling for efficient online self-test in robust systems", 2009 IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers, 2-5 November 2009, San Jose, CA, USA, pp. 201-208.

[26] M. Choudhury, V. Chandra, K. Mohanram, and R. Aitken, "Analytical model for TDDB-based performance degradation in combinational logic", Design, Automation & Test in Europe Conference & Exhibition (DATE), 8-12 March 2010, Dresden, Germany, pp. 423–428.

[27] M. Psarakis, D. Gizopoulos, E. Sanchez, and and M. Reorda, "Microprocessor Software-Based Self-Testing", IEEE Design & Test of Computers, Vol. 27, Issue 3, May-June 2010, 22ns Jan. 2010, pp. 4-19.

[28] T. Strauch, "A Novel RTL ATPG Model Based on Gate Inherent Faults of Complex Gates", Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen, MBMV 2017, Bremen, Germany, February 8-9, 2017, pp. 117-128

[29] T.Strauch, "Timing Driven C-Slow Retiming on RTL for MultiCores on FPGAs", ParaFPGA2013, 10-13 September 2013, Munich, Germany, pages 1-6, also at Cornell University Library, 14th July 2018, https://arxiv.org/abs/1807.05446

[30] A. Riefert, R. Cantoro, M. Sauer, M. Reorda, and Bernd Becker, "On the Automatic Generation of SBST Test Programs for In-Field Test", Design, Automation & Test in Europe Conference & Exhibition, DATE 2015, Grenoble, France, 9-13 March 2015, pp. 1186-1191.

[31] G. Di Natale, M. Doulcier, M.-L. Flottes, and B. Rouzeyre, "Self-Test Techniques for Crypto-Devices", IEEE Trans. in VLSI, vol. 18, no. 2, Feb. 2010, pp. 329-333.

[32] A. Apostolakis, D. Gizopoulos, M. Psarakis, D. Ravotto, and M. Reorda, "Test Program Generation for Communication Peripherals in Processor-Based SoC Devices", IEEE Design & Test of Computers, vol 26, no. 2, March-April 2009, pp. 52-63.

[33] D. Sabena, M. Reorda, and L. Sterpone, "On the Automatic Generation of Optimized Software-Based Self-Test Programs for VLIW Processors", IEEE Trans. on VLSI, vol. 22, no. 4, April 2014, pp. 813-823.

[34] A. Riefert, R. Cantoro, M. Sauer, M. Reorda, and B. Becker, "On the Automatic Generation of SBST Test Programs for In-Field Test", Design, Automation & Test in Europe Conference & Exhibition, DATE 2015, Grenoble, France, 9-13 March 2015, pp. 1277-1280.

[35] Y. Zhang, H. Li, and X. Li, "Automatic Test Program Generation Using Executing-Trace-Based Constraint Extraction for Embedded Processors", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 21, Issue 7, July 2013, pp. 1220-1233.

[36] N. Kranitis, A. Paschalis, D. Gizopoulos, and G. Xenoulis, "Software-Based Self-Testing of Embedded Processors", IEEE Transactions on Computers, Vol. 54, Issue 4, April 2005, pp. 461-475.

[37] D. Gizopoulos, M. Psarakis, M. Hatzimihail, M. Maniatakos, A. Paschalis, A. Raghunathan, and S. Ravi, "Systematic Software-Based Self-Test for Pipelined Processors", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 16, Issue 11, Nov. 2008, pp. 1441-1453.

[38] C. Chen, C. Wei, T. Lu, and H. Gao, "Software-Based Self-Testing With Multiple-Level Abstractions for Soft Processor Cores", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 15, Issue 5, May 2007, pp. 505-517.

[39] N. Kranitis, G. Xenoulis, D. Gizopoulos, A. Paschalis, and Y. Zorian, "Low-Cost Software-Based Self-Testing of RISC Processor Cores", Design, Automation and Test in Europe Conference and Exhibition (DATE), 7th March 2003, Munich, Germany, pp. 1-6.

[40] M. Gao, H. Chang, P. Lisherness, and K. Cheng, "Time-Multiplexed Online Checking", IEEE Trans. on Computers, Vol. 60, No. 9, Sep. 2011, pp. 1300-1312.

[41] N. Kranitis, A. Merentitis, N. Laoutaris, G. Theodorou, A. Paschalis, D. Gizopoulos, and C. Halatsis, "Optimal Periodic Testing of Intermittent Faults In Embedded Pipelined Processor Applications", Proc. of the Design Automation & Test in Europe Conference, DATE 2006, 6-10 March 2006, Munich, Germany, pp. 1-6.