Enhancing Robustness of Sequential Circuits Using Application-specific Knowledge and Formal Methods

Sebastian Huhn*† Stefan Frehse† Robert Wille†‡ Rolf Drechsler*†

*Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
†Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
‡Institute for Integrated Circuits, Johannes Kepler University Linz, 4040 Linz, Austria

{huhn,drechsle}@informatik.uni-bremen.de stefan.frehse@dfki.de robert.wille@jku.at

Abstract—**Due to shrinking feature sizes, integrated circuits are getting more vulnerable against transient faults. Methods increasing the robustness of circuits against these faults already exist for a long period of time but either introduce huge additional logic, increase the latency of the circuit, or are applicable for dedicated circuits such as microprocessors only. This work proposes an alternative hardening method which requires only a slight increase in additional hardware, does not influence the timing behavior, and is automatically applicable to arbitrary circuits. To this end, application-specific knowledge of the considered circuit is exploited, analyzed by a dedicated orchestration of formal techniques, and, eventually, used to synthesize a fault detection mechanism enhancing the robustness of the circuit. Experimental evaluations show that the proposed solution leads to a significant increase in the robustness, while the hardware overhead is kept moderate.**

I. Introduction

The complexity of *Integrated Circuits* (ICs) is steadily increasing. In particular, the number of sequential elements, i.e., *Flip Flops* (FFs), raises. These FFs are characteristically vulnerable to transient faults. Such a fault appears in form of a toggled bit for a short period of time and is not logically, electrically, or temporarily masked. Transient faults may occur at any FF which is not explicitly protected. This potentially affects the functional behavior of the circuit and, therefore, may result in invalid output values. Especially in case of safety critical systems, these violations can lead to disastrous consequences. Furthermore, other specific application fields, e.g., aeronautic systems, have environmental influences like high-energetic radiation, which facilitate transient faults even more [1], [2]. Thus, the development of mechanisms, which are capable to detect and react on such faults, is a highly relevant, but also challenging task.

In this context, the *robustness* of a given circuit is an important metric, which can be derived from the number of non-robust FFs that are vulnerable to transient faults. In order to increase the robustness of a sequential circuit, the number of vulnerable (non-robust) FFs needs to be decreased. To this end, the corresponding FFs are *hardened* by extending the *Circuit-under-Hardening* (CuH) so that the respective values are recomputed and, in case of faults, faulty signals can be replaced. These recomputations are usually conducted either by additionally employing redundant hardware or redundant time. More precisely, existing methods which are currently applied in order to increase the robustness of a given sequential circuit can roughly be categorized into the following three schemes [3]:

- *Space-based approaches*, which embed additional logic blocks to generate certain redundancy in order to enhance their robustness. Approaches such as *Triple Modular Redundancy* (TMR) [4] or *Error-Correction Code* [5]–[7] constitute representatives of this scheme.
- *Timing-based approaches*, which influence the timing behavior of the considered circuit in order to guarantee correct output values at the FFs. Representative candidates of this scheme have been are proposed in [8], [9].
- *Application-specific approaches*, which only consider dedicated parts of a circuit for which a robust solution is explicitly derived. Examples of this scheme include, e.g., dedicated fault-tolerance control flows for microprocessors [10]. Other application-specific approaches exploits invariants automatically to ensure correctness, e.g., hardware assertions [11]. Those assertions are used to uncover violations of the specific functional behavior during verification. However, those assertions do not focus on any dedicated fault model considering transient faults and, hence, no compact realization is given a priori.

However, all these schemes come with significant shortcomings: Space-based approaches introduce huge additional logic into the circuit – often caused by naive multiplication of sequential elements and, hence, a redundancy which is not necessarily needed for the functional behavior. Timing-based approaches suffer from the fact that they potentially increase the latency and, hence, cause restrictions to the actual design of the circuit. Application-specific approaches are limited to dedicated parts of a circuit (e.g., the considered microprocessor) and, hence, are not applicable for sequential circuits in general.

In this work, we are aiming to address these shortcomings by proposing a solution, which does not simply recomputes FFs values of the original design, but instead collects application-specific knowledge of their behavior. More precisely, the proposed methodology learns the relations of FFs and stores the conditions under which FFs assume the same logic value. This knowledge allows to employ a dedicated logic block which compares all corresponding FF values and, hence, can detect if one of them inherits a fault.

Following the methodology in this work, only a moderate hardware overhead in terms of gate count compared to the space-based approaches is required while, at the same time, the timing behavior is not affected at all. The resulting flow is automatically applicable to any sequential circuit and, hence, not limited to dedicated circuits as the existing applications-specific solutions. Since the computation and compaction of the FF relations are computationally hard tasks, a dedicated orchestration of formal methods such as *Bounded Model Checking* (BMC, [12]), powerful solvers for the *Boolean Satisfiability Problem* (SAT problem, [13]), and compact data structures for circuit representation involving *Binary Decision Diagrams* (BDDs, [14]) is proposed which are capable of coping with this complexity. Finally, this approach is flexible in the sense that the designer can easily configure the trade-off between the hardware overhead and the desired enhancement in robustness.

Experimental results confirm the benefits of the proposed methodology. In fact, robustness of a circuit can be increased to approx. 90% in most of the cases, while the circuit size increases only by a factor of approx. 1.13 on average. Compared to established methods (e.g., TMR, where indeed 100% robustness is achieved, but at the expense of more than tripling the circuit size), this provides a suitable alternative and a trade-off between robustness and hardware overhead.

The proposed methodology is described in the remainder of this paper as follows: Section II provides the background on sequential circuits, transient faults, and robustness. Afterwards, the general idea and the overall flow are described in Section III. A detailed description of the implementation is presented in Section IV and the obtained results are summarized in Section V. Finally, conclusions are drawn in Section VI.

II. Background

This section briefly introduces the relevant terminologies used in the remainder of this paper.

A. Sequential Circuits

A sequential circuit Φ is given as a commonly known gate level representation that consists of *Primary Inputs* (PIs), *Primary Outputs* (POs), combinational gates G, and *sequential elements* (SE) such as FFs, i.e., $\Phi = (\mathsf{IN}, \mathsf{OUT}, \mathsf{G}, \mathsf{SE})$. The sequential elements are assumed to be synchronous to (at least) one clock domain.[1] The FFs of a given sequential circuit can be grouped by a *hierarchical levelizing* procedure. Two FFs FF_i and FF_j are contained in the same group, if the number of FFs in both fan-in cones are the same on the shortest path towards the PIs.

Alternatively, a sequential circuit can also be represented by a *Finite State Machine* (FSM). An FSM is defined by a tuple $M = (I, S, T)$, where I describes the set of initial states, S represents the state space of the circuit, and T defines the *transition relation*. A *transition relation* $T(s, s')$ evaluates to true, if there is at least one transition from state s to state s'. The *set of reachable states* $S^* \subseteq S$ contains those states that are reachable from an initial state in an arbitrary number of steps.

B. Transient Faults

The shrinking feature size leads to an increased vulnerability of circuits against *Single Transient Faults* (STFs), which are typically caused by *Single Event Upsets* (SEUs), e.g., electrical noise, particle strikes, or other environmental effects [1], [2]. Typically, the influence of a transient fault occurring at a FF is modeled as an unintended toggled output value. This influence can possibly cause an invalid and unintended behavior of the circuit Φ for a short period of time. Based on this vulnerability, a circuit Φ is called robust if no fault exists such that the input/output behavior is affected. In order to increase the robustness of a circuit Φ, a *Fault Detection Mechanism* (FDM) can be applied that handles cases in which a single transient fault occurs at a FF, e.g., to realize precautions.

C. Assessing Robustness

To consider the vulnerability of sequential circuits against transient faults, a metric for robustness has been introduced, which measures the fault tolerance (i.e., the robustness) with respect to a fault model [15], [16]. More precisely:

Definition 1. *Let $\Phi = (\mathsf{IN}, \mathsf{OUT}, \mathsf{G}, \mathsf{SE})$ be a sequential circuit. A FF is considered to be non-robust, if there is at least one reachable state and one transient fault such that the output behavior of Φ is tampered. Let N be the* set of non-robust FFs *with $N \subseteq \mathsf{SE}$. Then, the robustness of Φ can be determined by $\mathcal{R} = 1 - \frac{|N|}{|\mathsf{SE}|}$ [17].*

In order to determine the robustness of a given sequential circuit the non-robust FFs $\mathbb{N}$ can be computed by either formal methods [18] or simulation-based techniques [19]–[22]. In general, the robustness can be determined using a simulation-based approach as follows:

1) Define a number l of simulation cycles to be considered while adjusting the state of the circuit which is finally used for fault injection. Beside this, define a number k of cycles to be simulated for fault propagation.
2) The PIs of a given sequential circuit Φ are stimulated up to $l-1$ cycles using random values.
3) In cycle l, the state s_l is extracted from the simulation environment. A copy $\widehat{s_l}$ of s_l is modified so that an STF is injected at a randomly chosen FF $f \in \mathsf{SE}$ and, hence, the output value of f is toggled.
4) The circuit Φ is simulated twice for up to k cycles: One simulation starts from the healthy state s_l and one from faulty state $\widehat{s_l}$ containing the injected STF. During both simulation runs and for every clock cycle, the same PIs values are driven.
5) In cycle $l+k$, all POs of both simulation runs are compared. If at least one of the POs values differs, the f is non-robust unless the circuit contains an FDM reporting a fault.
6) This procedure is repeated from Step 3 until all FFs are covered by fault injection.

Due to the nature of random simulation, the number of covered states depends on the chosen parameters for l and k.

III. General Idea

The general idea of the proposed methodology rests on the following observations:

- Today's circuits usually contain a huge number of FFs, which can store at least a single bit, i.e., '0' or '1'. If a single FF is affected by a transient fault, this bit is toggled. Existing approaches insert redundant logic into the design, e.g., to recompute the correct value, which causes a significant hardware overhead.
- At the same time, the value of an observed single FF is often equal to the value of many other FFs. Moreover, since the behavior of the circuit is known, it is possible to determine the relation between them, i.e., the states in which certain FFs assume the same value.
- Hence, instead of introducing redundancy for recomputations, we propose to simply compare the value of a FF to the values of other FFs from which it is known that, for the respectively considered state, they are supposed to generate the same value.

In order to realize this idea, we need a formalism that states whether a partition of non-robust FFs assumes the same value for given reachable states. In the following, this is formally described in terms of an equivalence property.

Definition 2. *Let $P_j \subseteq N_i$ be a partition of at least two non-robust FFs and $\widehat{S} \subseteq S^*$ be a set of reachable states. Then, an* Equivalence Property *(EP) is defined by*

$$\mathsf{EP}(\widehat{S}, P_j) := \forall f_n, f_m \in P_j : f_n(s) = f_m(s) \; \forall \; s \in \widehat{S}$$

and evaluates to true if all combinations of FFs $f_n, f_m \in P_j$ assume the same output value in all of these states $\widehat{S} \subset S^*$.

Example 1. *Consider the circuit shown in Fig. 1 which is composed of five FFs distributed in two hierarchical circuit levels 1 and 2. If both FF_1 and FF_2 (level 1) are set to '0', then FF_3, FF_4, and FF_5 (level 2) are assumed to have the same output value '0' after a single clock cycle. This scenario is represented by an $\mathsf{EP}(\widehat{S}, P_j)$ with the partition $P_j = \{FF_3, FF_4, FF_5\}$ and the state $s_j \in \widehat{S}$ being defined by $FF_1 = 0$ and $FF_2 = 0$, i.e., $\mathsf{EP}(\widehat{S}, P_j) = 1$ holds.*

[1]In order to ease the following descriptions, we will assume a single clock domain. However, the proposed methodology can be extended to further clocks domains as well.

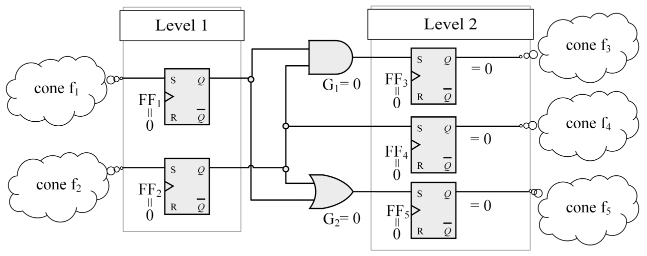

Fig. 1: A non-robust sequential circuit

Using the Equivalence Property and the general idea sketched above, robustness of sequential circuits is enhanced as follows:

1) Determine the set N of non-robust FFs of the given sequential circuit. The assessment of robustness as reviewed in Section II-C can be utilized.
2) Consider all non-robust FFs N and determine subsets $N_1 \cup N_2 \cup \cdots \cup N_L = N$ according to their hierarchical circuit levels (L being the total number of hierarchical levels in the circuit). This clustering is required to avoid masking effects of FFs that are located in different hierarchical circuit levels affecting each other.
3) For each level $1 \leq l \leq L$ and all subsets of non-robust FFs $N_i \subseteq N$, determine suitable partitions $P_j \in \mathcal{P}(N_i)$ and a set of reachable states $\widehat{S} \subseteq S^*$ such that all FFs in P_j are supposed to generate the same value, i.e., determine P_js and corresponding $\widehat{S}$s for which $\mathsf{EP}(\widehat{S}, P_j) = 1$ holds.
4) Using the knowledge from the obtained EPs, synthesize a *Fault Detection Mechanism* (FDM). To this end, realize the following logic blocks:

- *Activator* $\mathcal{A}$: Generates a signal $\mathcal{A}$ (supposed to trigger the FDM) stating whether ($\mathcal{A} = 1$) or not ($\mathcal{A} = 0$) the FFs in P_j are supposed to generate the same value under the current state $s \in \widehat{S}$.
- *Comparator* $\mathcal{C}$: Generates a signal $\mathcal{C}$ stating whether ($\mathcal{C} = 1$) or not ($\mathcal{C} = 0$) all FFs in a partition P_j to be hardened actually assume the same output value.
- *Detector*: Generates a fault signal $\mathcal{F}$ reporting the detection of a fault. A fault is detected, if not all FFs in a partition P_j assume the same output value (i.e., $\mathcal{C} = 0$), although they are supposed to do that for the current state (i.e., $\mathcal{A} = 1$), i.e., $\mathcal{F} = \neg\mathcal{C} \wedge \mathcal{A}$.

This proposed FDM detects transient faults occurring in FFs of the considered circuit. If a fault is detected, an introduced fault signal $\mathcal{F}$ is driven. This enables the realization of precautions against faulty behavior at the POs, e.g., by resetting the circuit or masking the affected POs. Overall, this leads to an enhanced robustness. The ratio of the enhancement can thereby be controlled, e.g., by adjusting the number knowledge collected through the EPs.

IV. Implementation

This section provides details on how the general ideas discussed above have been implemented. The *bottleneck* of the proposed methodology obviously is the determination of as much as possible of the application-specific knowledge in terms of EPs. Doing this in a complete fashion would require the consideration of all possible partitions $P_j \in \mathcal{P}(N_i)$ of all non-robust FFs located in the same hierarchical circuit level – leading to an exponential complexity, which is not feasible for practical applications. Moreover, most of the partitions P_j are likely to be not suitable for an EP anyway, since no state s_j may exists for them so that all assume the same value. In order to address these issues, we propose a scheme which does not consider all possible partitions, but aims for efficiently determining *good* partitions.

Algorithm 1: Partition Enumeration procedure

```
   Data: set of non-robust FFs: $N_i$, upper-bound partition size: $p_s$
   Ensure: $0 < p_s \leq |N_i|$
1  Container $\mathcal{E} = \emptyset$     /* Data container for EPs */
2  while $p_s > 1$ do
3      Let $P_j \in \mathcal{P}(N_L)$ such that $|P_j| = p_s$
4      if $P_j = \emptyset$ then
5          $p_s = p_s - 1$
6          continue
7      $\widehat{S} = \mathsf{StateCollector}(P_j)$
8      if $\widehat{S} \neq \emptyset$ then
9          $\mathcal{E} = \mathcal{E} \cup \mathsf{EP}(\widehat{S}, P_j)$
10         $N_i = N_i \setminus P_j$
11     else $N_i = N_i \setminus \{f\}$ with $f \in N_i$ (chosen after analysis)
```

In addition to that, we heavily exploit formal methods such as *Bounded Model Checking* (BMC, [12]), powerful solvers for the *Boolean Satisfiability Problem* (SAT problem, [13]), and compact data structures for circuit representation involving *Binary Decision Diagrams* (BDDs, [14]).

Eventually, this leads to a methodology composing:

1) A *Partition Enumerator* selects suitable partitions $P_j \in \mathcal{P}(N_i)$ which have not been considered before.
2) A *State Collector* determines the states $\widehat{S}$ under which all FFs in the selected partition P_j assume the same value and, hence, determines all $\mathsf{EP}(\widehat{S}, P_j)$ evaluating to true.
3) An *FDM Synthesizer* takes the obtained knowledge, realizes the FDM, and, eventually, embeds the resulting logic into the original circuit.

In the following, each step is described in more detail and illustrated by means of the circuit considered in Fig. 1.

A. Partition Enumerator

In order to determine suitable partitions, first all non-robust FFs (given in N) are distinguished according to their hierarchical circuit level. Then, for each subset $N_i \subset N$, *good* partitions $P_j \in \mathcal{P}(N_i)$ are enumerated as shown in Algorithm 1.

Algorithm 1 receives a set of non-robust FFs N_i and an upper-bound for the maximum partition size p_s to be considered (provided by the designer). First, a possible partition P_j is chosen as shown in Line 3. If no partition of size p_s is left to be considered anymore, p_s is decreased, and the loop continues with the new partition size (Lines 4 to 4) until the partition size deceeds the lower bound. Afterwards, the partition P_j is passed to the State Collector (Line 7) that computes states $\widehat{S}$ in which all FFs in P_j assume the same value (described in detail in Section IV-B).

If at least one state $s \in \widehat{S}$ exists (Lines 8 to 8), an EP is created using the states $\widehat{S}$ determined by the State Collector as well as the currently considered partition P_j. Then, the resulting EP is stored within a global data container $\mathcal{E}$ (used later by the FDM Synthesizer) and the partition P_j is excluded from any further consideration since at least one state is covered. In contrast, if no state exists, i.e., $\widehat{S} = \emptyset$ (Lines 8 to 11), one of the non-robust FFs $f \in N_i$ is removed from the further consideration in order to increase the chance to determine a suitable partition in the next iterations. The FF f to be removed is determined by an analysis based on a greedy algorithm.

Example 2. *Consider again the circuit shown in Fig. 1 and all non-robust FFs in the hierarchical circuit level $L = 2$, i.e., $N_2 = \{FF_3, FF_4, FF_5\}$. The Partition Enumerator starts with a maximum partition size set to $p_s = 3$ and, hence, first considers $P_j = \{FF_3, FF_4, FF_5\}$. This partition is validated by the State Collector as described in the next subsection.*

Algorithm 2: State Collecting procedure

```
Data: enumerated partition: $P_j$, max. number of states: $u$
Data: unrolling depth: $l$
1 $\widehat{S} = \emptyset$                       /* stored as BDD */
2 for $k = 1$ to $l$ do
3     $F = \mathsf{SFind}(P_j, k)$
4     repeat
5         if $|\widehat{S}| > u$ then return $\widehat{S}$
6         $\widehat{S} = \widehat{S} \cup s_{i+1}$         /* collects state */
7         $F = F \wedge \neg s_{i+1}$        /* blocks solution */
8     until $\mathsf{SAT}(F)$
9 return $\widehat{S}$
```

B. State Collector

The main task of the State Collector is to determine reachable states $\widehat{S}$ such that the EP holds for a partition P_j obtained by the Partition Enumerator. To this end, solutions for the BMC problem [12] are used to determine these states. In general, BMC is about determining a path of states $s_0 \dots s_l$ from the initial state s_0 so that, eventually, the terminal state s_l violates or satisfies a certain property. For a sequential circuit represented by the FSM $M = (I, S, T)$, this can be formulated as

$$\mathsf{BMC}(l) = I(s_0) \wedge \bigwedge_{0 \leq i < l} T(s_i, s_{i+1}) \wedge P(s_l)$$

whereas P is a logical formula for the considered property to be verified.

For our purposes, we revise this BMC formulation in order to determine a path of states so that, eventually, the EP holds for the currently considered partition P_j. More precisely,

$$\mathsf{SFind}(P_j, l) = I(s_0) \wedge \bigwedge_{0 \leq i < l} T(s_i, s_{i+1}) \wedge \mathsf{EP}(s_l, P_j)$$

is employed. The formula SFind is satisfiable, if there is at least one path $s_0 \dots s_l$ such that all FFs in the currently considered partition P_j assume the same output value at state s_l. The number l of transitions to be considered, i.e., the unrolling depth of the circuit, can be iteratively increased until either a state s_l has been determined or the maximal unrolling depth (defined by the designer) is reached. If no path can be determined, the partition P_j has been found unsuitable as no state could have been determined in which all FFs in P_j assume the same output value. In addition to that, another parameter $u > 0$ is added which prevents the State Collector from determining too many states (significantly increasing the complexity of the FDM, while hardly improving the achieved robustness anymore).

Overall, this leads to the State Collecting method as summarized in Algorithm 2. The algorithms receives the partition P_j from the Partition Enumerator, the maximum number u of states to be generated, as well as the unrolling depth l for the underlying BMC problem. The collected states $\widehat{S}$ are compactly represented by means of BDDs [14] and initialized by the empty set in Line 1.

As long as the maximum number u of states to be determined is not reached (Line 5), further states are computed. This is done by formulating the BMC problem (Line 3) for the currently considered unrolling depth k ($0 < k < l$). Afterwards, the resulting formulation (denoted by F) is solved by a SAT solver (Line 8). As long as a satisfying solution is determined, the corresponding states are added to $\widehat{S}$ (Line 6) and, afterwards, blocked in the BMC formulation F so that new states can be determined (Line 7).

Example 3. *Given the partition $P_j = \{FF_3, FF_4, FF_5\}$ as provided by the Partition Enumerator (see Example 2), states shall be determined so that all FFs in P_j assume the same output value. Assuming that the both cones f_1 and f_2 in the circuit from Fig. 1 do not contain any FFs, a State Collector instance according to* SFind *is formulated. Solving this instance yields a satisfying solution where all FFs $\{FF_3, FF_4, FF_5\}$ have the same output value '0'. From that, it is shown that $\mathsf{EP}(\widehat{S}, P_j)$ holds under the state $s \in \widehat{S}$ defined by $FF_1 = 0$ and $FF_2 = 0$. Consequently, the partition P_j is valid. In the Partition Enumerator (Algorithm 1), this* EP *is later stored in the container $\mathcal{E}$.*

C. FDM Synthesizer

The methods from above yield a data container $\mathcal{E}$ including all application-specific knowledge which has been obtained in terms of EPs, i.e., all valid partitions P_j and corresponding states $\widehat{S}$ which satisfy the equivalence property. This knowledge is now utilized in order to synthesize an FDM. More precisely, for each determined $\mathsf{EP}(\widehat{S}, P_j) \in \mathcal{E}$, a fault signal $\mathcal{F}$ is to be generated which is set to '1' whenever the circuit is in a state $s \in \widehat{S}$ (checked by the activator) and, at the same time, the FFs in the partition P_j do not assume the same value (checked by the comparator). In the following, details on the realization of this FDM are provided.

1) Activator $\mathcal{A}$: For a given $\mathsf{EP}(\widehat{S}, P_j)$, a signal $\mathcal{A}$ has to be created which is set to '1' iff the circuit is in a state $s \in \widehat{S}$. As mentioned before in Section IV-B, all currently relevant states $\widehat{S}$ are stored in terms of a BDD. Hence, corresponding logic triggering the signal $\mathcal{A}$ can easily be derived from the BDD, by replacing all BDD nodes with a corresponding MUX gate (as, e.g., shown in [23]).

Besides that, the timing of $\mathcal{A}$ has to be properly adjusted. Transient faults are assumed to occur in the transition between two consecutive states. This is why the states $s \in \widehat{S}$ are collected for state s_{l-1} (assuming their effects manifest in state s_l). Consequently, the check for states has to be conducted one state before the values of all FFs in P_j are to be compared, i.e., the activator signal $\mathcal{A}$ has to be generated one state before the comparison is conducted. This requires signal $\mathcal{A}$ to be buffered for one cycle, which is accomplished by introducing an additional FF L-Act$_1$.

However, since L-Act$_1$ is vulnerable against transient faults, robustness is not guaranteed anymore. Hence, a second FF L-Act$_2$ which also receives the value of signal $\mathcal{A}$ is introduced. After one cycle, the output values of both FFs L-Act$_1$ and L-Act$_2$ are checked for equivalence. If the two value are not equal, a fault is reported (by setting the fault signal $\mathcal{F}$ to '1').

Example 4. *Consider again the running example with the circuit from Fig. 1 and the determined* $\mathsf{EP}(\widehat{S}, P_j)$. *Fig. 2 shows the resulting circuit created by the FDM scheme proposed in this work. The bottom left corner of Fig. 2 sketches the resulting* Activator *logic. More precisely, using $\widehat{S}$ represented as BDD, a MUX circuit is created which generates the signal $\mathcal{A}$ (sketched by the block* State Collector*). Then, the resulting signal is passed to the two FFs L-Act$_1$ and L-Act$_2$. Finally, the equivalence check in order to make these newly inserted FFs robust is conducted using an XOR gate which triggers the fault signal $\mathcal{F}$.*

2) Comparator $\mathcal{C}$: For a given $\mathsf{EP}(\widehat{S}, P_j)$, a signal $\mathcal{C}$ has to be created which is set to '1' iff the FFs in a partition P_j assume same values. This can easily be realized by connecting the corresponding FF outputs by XNOR gates and comparing their result. The example illustrates the resulting logic.

Example 5. *Consider again the running example with the circuit from Fig. 1 and the determined* $\mathsf{EP}(\widehat{S}, P_j)$. *The bottom middle part of Fig. 2 sketches the resulting* Comparator *logic. Here, the outputs of all FFs $\{FF_3, FF_4, FF_5\} \in P_j$ are compared by XNOR gates. Afterwards, the outputs of these*

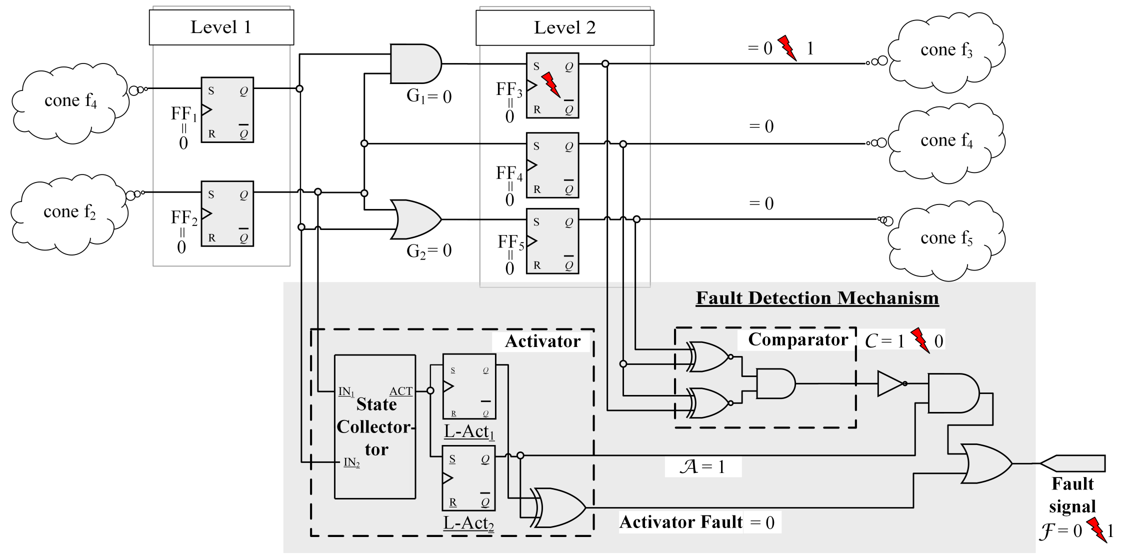

Fig. 2: Applying the proposed methodology to the circuit from Fig. 1

XNOR gates are passed to an AND gate. If all FFs assume the same value, this AND gate evaluates to '1'.

3) Generating the Fault Signal $\mathcal{F}$: Finally, the signals $\mathcal{A}$ and $\mathcal{C}$ are assembled into a single FDM that generates the fault signal $\mathcal{F}$. Recall, that $\mathcal{F}$ is set to '1' iff a fault has been detected. For a given $\mathsf{EP}(\widehat{S}, P_j)$, this is the case, if the circuit just left a state $\widehat{S}$ (buffered in $\mathcal{A}$) and all FFs in $\mathcal{P}_|$ are not equal, i.e., $\mathcal{F} = \neg\mathcal{C} \wedge \mathcal{A}$, which can be realized easily.

Example 6. *Consider again the running example and the resulting circuit shown in Fig. 2. As can be seen in the bottom right corner, the signal $\mathcal{C}$ is first inverted and, afterwards, ANDed with the $\mathcal{A}$ signal. The resulting value is additionally ORed with the fault value from the robustness check of the* Activator *– eventually resulting in the desired signal $\mathcal{F}$. Consider now the entire circuit and, e.g., a transient fault in FF_3 (denoted by the red strike symbol). This causes the FFs $\{FF_3, FF_4, FF_5\} \in P_j$ to not assume the same value anymore in states $\widehat{S}$ where this is supposed to happen, i.e., the $\mathsf{EP}(\widehat{S}, P_j)$ fails. This case is propagated through the FDM (see annotations in Fig. 2) which, eventually, sets the fault signal $\mathcal{F}$ to '1', and, by this, detects the fault.*

Logic as described above is, of course, added for all $\mathsf{EP} \in \mathcal{E}$. Overall, this leads to a circuit which has a slightly increased number of gates, but substantially improved robustness. This has been confirmed by experimental evaluations whose results are summarized next.

V. Experimental Results

The proposed methodology has been implemented in C++. To determine the non-robust FFs of the circuit, a simulation-based robustness checker has been implemented which transforms the given circuits (provided in *Verilog* and parsed by *Verific*) into a compiled simulation model (to this end, *LLVM* [24] *IR code* is generated by the simulation environment). In order to conduct the respective BMC task (cf. Section IV), *MiniSAT* [13] on top of *metaSMT* [25], together with the *X-value abstraction* as described in [26] has been utilized. The BDD package *CUDD* has been used to generate the MUX circuits.

Afterwards, the resulting flow has been evaluated using *ITC'99* benchmark circuits. In order to determine the set of all non-robust FFs (cf. Section II-C), the parameters $l = 500$ and $k = 5$ have been found suitable for these circuits. The *Partition Enumerator* (cf. Section IV-A) considered different partition sizes, i.e., $p_s \in \{4, 8, 16\}$. Finally, the *State Collector* (cf. Section IV-B) always assumed an unrolling depth of $l = 10$ and a bounded number $u = 1024$ of states to be collected per partition P_j.[2] All evaluations have been conducted on an *Intel Xeon E3-1230v2 3.3 GHz* processor with 32GB system memory. The obtained results are summarized as follows:

TABLE I: Run time for different $p_s \in \{4, 8, 16\}$

circ.	**#gates**	**#FFs**	**run time [s]**		
			$p_s = 4$	$p_s = 8$	$p_s = 16$
b05	608	66	7.71	1.42	1.42
b06	66	9	0.11	< 0.10	< 0.10
b07	382	51	34.83	10.78	10.77
b08	168	21	0.23	0.23	0.23
b09	131	28	1.61	0.66	0.66
b10	172	17	3.95	1.05	1.50
b11	366	30	60.80	1.02	0.46
b12	1000	121	238.62	75.35	69.06
b13	309	53	16.29	5.40	6.65
b14	3461	247	1287.13	341.21	105.38
b15	6931	447	28787.10	5115.23	917.77

- Table I provides details on the considered benchmark circuits, i.e., its respective name, number of gates, and number of FFs, as well as the run time (in CPU seconds) required by the proposed methodology when partition sizes of $p_s \in \{4, 8, 16\}$ are applied.
- Fig. 3 shows the hardware overhead (in terms of a gate count factor) caused by applying the proposed methodology for the considered partition sizes $p_s \in \{4, 8, 16\}$.[3]
- Fig. 4 shows the robustness of the original circuit as well as the robustness after applying the proposed methodology (again for different partition sizes $p_s \in \{4, 8, 16\}$).

First, the results nicely show the effect of different partition sizes p_s. In almost all cases a larger p_s leads to a smaller hardware overhead. This is because larger partitions cover more FFs and, hence, require the consideration of a smaller total number of partitions leading to less FDM logic. At the same time, this reduces the required run time since less BMC checks have to be conducted.

However, more important is the overall performance. In this regard, the proposed methodology provides a suitable alternative to previously proposed solutions such as discussed in Section I. Although space-based approaches such as TMR [4] can guarantee 100% robustness, they usually require more

[2]Note that, in most cases, this bound was not exceeded.

[3]If no bar is shown, a hardware overhead of 1 or close to 1 is measured.

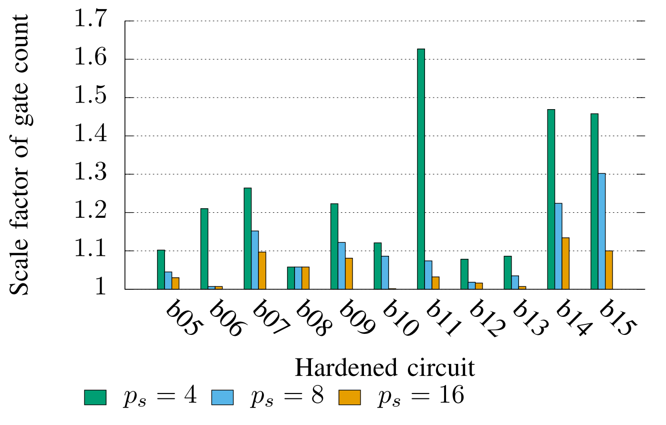

Fig. 3: Hardware overhead for $p_s \in \{4, 8, 16\}$

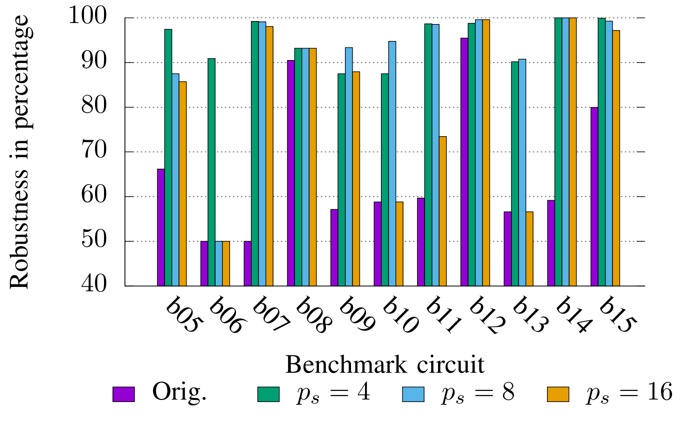

Fig. 4: Robustness for original/improved circuits

than thrice the amount of hardware (i.e., yielding a scaling factor of > 3.0). In contrast, the solution proposed in this work is capable of always improving the robustness to more than 90% (in some cases even close to 100%), while only a factor of approx. 1.13 more hardware is required for this on average. As already discussed before, the proposed solution also outperforms timing-based and application-specific approaches, since timing is hardly affected at all in the proposed solution and the methodology can be applied to arbitrary sequential circuits. By this, a suitable trade-off between enhancing the robustness and keeping the hardware overhead small is achieved.

VI. Conclusions & Future Work

In this work, we proposed an approach for improving the robustness in sequential circuits. The main idea is to avoid the addition of huge additional hardware, e.g., caused by recomputing possibly faulty signals, and instead exploit application-specific knowledge about the FFs in each state. To this end, a methodology is introduced which gains the corresponding knowledge and, afterwards, utilizes them for a fault detection mechanism. To cope with the underlying complexity, a dedicated orchestration of formal techniques is employed. This results in a hardening method which requires only a slight increase in additional hardware, does not influence the timing behavior, and is automatically applicable to arbitrary circuits. Experimental evaluations confirmed these benefits: Robustness can be increased to approx. 90%, while the circuit size increases only by a factor of approx. 1.13 on average. Future work will focus on developing a technique for not only detecting the respective faults but also correcting them with the proposed methodology. Besides this, a fast preprocessing step will be developed which allows to determine the most promising partition size for arbitrary circuits.

VII. Ackknowledgement

This work was supported by the University of Bremen's graduate school SyDe, funded by the German Excellence Initiative. We would like to thank Verific Design Automation Inc. for providing the SystemVerilog front-end used for the implementation of our technique.

References

[1] T. Heijmen and A. Nieuwland, "Soft-error rate testing of deep-submicron integrated circuits," in *IEEE European Test Symp.*, 2006, pp. 247–252.

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

[3] F. L. Kastensmidt, L. Carro, and R. Reis, *Fault-Tolerance Techniques for SRAM-Based FPGAs*. Springer, 2006.

[4] C. E. Stroud and A. E. Barbour, "Design for testability and test generation for static redundancy system level fault-tolerant circuits," in *Int'l Test Conf.*, 1989, pp. 812–818.

[5] Z. Luo, "ECC, an extended calculus of constructions," in *Symp. on Logic in Computer Science*, 1989, pp. 386–395.

[6] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.

[7] A. Sanchez-Macian, P. Reviriego, and J. A. Maestro, "Hamming SEC-DAED and extended hamming SEC-DED-TAED codes through selective shortening and bit placement," *IEEE Trans. Device and Materials Reliability*, vol. 14, no. 1, pp. 574–576, 2014.

[8] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge, "Razor: a low-power pipeline based on circuit-level timing speculation," in *Microarchitecture*, 2003, pp. 7–18.

[9] D. Blaauw, S. Kalaiselvan, K. Lai, W. H. Ma, S. Pant, C. Tokunaga, S. Das, and D. Bull, "Razor II: in situ error detection and correction for PVT and SER tolerance," in *IEEE Int'l Conf. on Solid-Sate Circuits*, 2008, pp. 400–622.

[10] N. Farazmand, M. Fazeli, and S. G. Miremadi, "FEDC: control flow error detection and correction for embedded systems without program interruption," in *Int'l Conf. on Availability, Reliability and Security*, 2008, pp. 33–38.

[11] S. Hertz, D. Sheridan, and S. Vasudevan, "Mining hardware assertions with guidance from static analysis," *IEEE Trans. on CAD*, vol. 32, no. 6, pp. 952–965, 2013.

[12] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 1999, ch. Symbolic Model Checking without BDDs, pp. 193–207.

[13] N. Eén and N. Sörensson, *Int'l Conf. on Theory and Applications of Satisfiability Testing*. Springer, 2004, ch. An Extensible SAT-solver, pp. 502–518.

[14] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. on CAD*, vol. C-35, no. 8, pp. 677–691, 1986.

[15] U. Krautz, M. Pflanz, C. Jacobi, H. W. Tast, K. Weber, and H. T. Vierhaus, "Evaluating coverage of error detection logic for soft errors using formal methods," in *Design, Automation and Test in Europe*, vol. 1, 2006, pp. 1–6.

[16] L. Doyen, T. A. Henzinger, A. Legay, and D. Nickovic, "Robustness of sequential circuits," in *Int'l Conf. on Application of Concurrency to System Design*, 2010, pp. 77–84.

[17] G. Fey, A. Sülflow, S. Frehse, and R. Drechsler, "Effective robustness analysis using bounded model checking techniques," *IEEE Trans. on CAD*, vol. 30, no. 8, pp. 1239–1252, 2011.

[18] G. Fey and R. Drechsler, "A basis for formal robustness checking," in *Int'l Symp. on Quality Electronic Design*, 2008, pp. 784–789.

[19] Shi-Yu-Huang, K.-T. Cheng, K.-C. Chen, and J. Y. J. Lu, "Fault-simulation based design error diagnosis for sequential circuits," in *Design Automation Conf.*, 1998, pp. 632–637.

[20] N. Miskov-Zivanov and D. Marculescu, "Multiple transient faults in combinational and sequential circuits: A systematic approach," *IEEE Trans. on CAD*, vol. 29, no. 10, pp. 1614–1627, 2010.

[21] M. Bozzano, A. Cimatti, and F. Tapparo, *Symbolic Fault Tree Analysis for Reactive Systems*. Springer, 2007, ch. ATVA, pp. 162–176.

[22] D. Nayak and D. M. H. Walker, "Simulation-based design error diagnosis and correction in combinational digital circuits," in *IEEE VLSI Test Symp.*, 1999, pp. 70–78.

[23] R. Drechsler, J. Shi, and G. Fey, "Synthesis of fully testable circuits from BDDs," *IEEE Trans. on CAD*, vol. 23, no. 3, pp. 440–443, 2004.

[24] C. Lattner and V. Adve, "LLVM: a compilation framework for lifelong program analysis transformation," in *Int'l Symp. on Code Generation and Optimization*, 2004, pp. 75–86.

[25] H. Riener, F. Haedicke, S. Frehse, M. Soeken, D. Große, R. Drechsler, and G. Fey, "metaSMT: focus on your application and not on solver integration," *Int'l Journal on Software Tools for Technology Transfer*, pp. 1–17, 2016.

[26] O. Grumberg, *3-Valued Abstraction for (Bounded) Model Checking*. Springer, 2009, pp. 21–21.