TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 10/14 pp103–116
DOI: 10.26599/TST.2018.9010136
Volume 25, Number 1, February 2020

Application Specified Soft-Error Failure Rate Analysis Using Sequential Equivalence Checking Techniques

Tun Li*, Qinhan Yu, Hai Wan, and Sikun Li

Abstract: Soft errors have become a critical challenge as a result of technology scaling. Existing circuit-hardening techniques are commonly associated with prohibitive overhead of performance, area, and power. However, evaluating the influence of soft errors in Flip-Flops (FFs) on the failure of circuit is a difficult verification problem. Here, we proposed a novel flip-flop soft-error failure rate analysis methodology using a formal method with respect to application behaviors. Approach and optimization techniques to implement the proposed methodology based on the given formula using Sequential Equivalence Checking (SEC) are introduced. The proposed method combines the advantage of formal technique-based approaches in completeness and the advantage of application behaviors in accuracy to differentiate vulnerability of components. As a result, the FFs in a circuit are sorted by their failure rates, and designers can use this information to perform optimal hardening of selected sequential components against soft errors. Experimental results of an implementation of a SpaceWire end node and the largest ISCAS'89 benchmark sequential circuits indicate the feasibility and potential scalability of our approach. A case study on an instruction decoder of a practical 32-bit microprocessor demonstrates the applicability of our method.

Key words: soft error; failure rate analysis; Sequential Equivalence Checking (SEC); application specified

1 Introduction

As technology scales and node capacitances decrease, Single Event Upsets (SEUs) induced by particle strikes from environmental radiation are increased. Other sources of transient faults are process variations or aging effects, which cause malfunctions under certain conditions. Often such faults manifest as soft errors at the functional level[1, 2]. In either case, the soft error is observed as an upset in one or more state bits. If a soft error goes undetected and finally alters the desired primary output value, then it may result in a failure.

Although several fault tolerance techniques to harden the circuits are available at the production level[3] or the design level[3–5], they impose substantial overheads, especially a power overhead, and should therefore be applied judiciously. For applications with stringent cost constraints, applying protection techniques to the most vulnerable components in a circuit, i.e., selective protection is a cost-effective reliability solution.

For selective protection, reliability evaluation is critical to find out the most vulnerable components in a given circuit. In a sequential circuit, an SEU which occurs in a Flip-Flop (FF) may either directly propagate to primary outputs and cause failure at the same cycle, or propagate to FFs repeatedly until it manifests as an error at some primary outputs and causes failure several clock cycles later, or it is logically masked on its way to primary outputs at some clock cycle. Therefore, tools that can pin-point components of a circuit that need protection would be useful to ensure reliability.

• Tun Li and Sikun Li are with the School of Computer Science, National University of Defense Technology, Changsha 410073, China. Tun Li is also with the Laboratory of Software Engineering for Complex Systems, Changsha 410073, China. E-mail: {tunli, sikunli}@nudt.edu.cn.

• Qinhan Yu and Hai Wan are with the School of Software, Tsinghua University, Beijing 100084, China. E-mail: yuqinhan10@gmail.com; wanhai@tsinghua.edu.cn.

∗ To whom correspondence should be addressed.

Manuscript received: 2018-07-15; revised: 2018-09-28; accepted: 2018-10-15

Some methods, namely, logical masking, temporal masking, and electrical masking, may determine whether an SEU will propagate to become a soft error. An SEU is logically masked because off-path gate inputs prevent a logical transition of the output of that gate. An SEU is electrically masked because the strength of the resulting pulse is reduced to become reliably latched. An SEU is temporally masked because the erroneous pulse misses the latching edge when arriving at a latch. In this study, we focus on how logical masking could be utilized to selectively protect FFs in a circuit.

Our work focuses on the failure rate of circuit components when soft error occurs. In this paper, we proposed a novel methodology to automatically assess the failure probability of FFs with respect to SEU faults while considering application behaviors. A novel failure rate measurement called Vulnerable State Set (VSS) is first defined to evaluate the contribution of each FF, and then the method to compute FF failure rate based on VSS is introduced. Finally, implementation and optimization methods for the proposed method using Sequential Equivalence Checking (SEC) techniques are presented and evaluated.

The significant contributions of this paper are in the following:

- A novel failure rate measurement called VSS, which is based on the states of circuit to measure the FF failure rate;
- A novel methodology that combines circuit states and application behaviors to practically and accurately assess the soft error failure rate of FFs;
- An automatic SEC-based soft error failure rate analysis technique with minimal requirements on users.

Our failure rate analysis can be applied to any type of circuits. We demonstrate our approach on a third-party Verilog implementation of a SpaceWire network end node and the largest ISCAS'89 benchmark. The experimental results demonstrate that our approach is feasible and effective even if most of the FFs, especially all the FFs, in a circuit were vulnerable. We also applied our methodology to a practical processor design to demonstrate the effectiveness of our methodology in selective circuit hardening.

Related work is discussed in Section 2. Section 3 introduces the motivation of our methodology and some preliminaries. The algorithms based on SEC to compute FF failure rate and optimization techniques are presented in Section 4. Section 5 provides two examples and interesting results of FF failure rate variation with application behavior. In Section 6, the algorithms are evaluated and compared on benchmark circuits. Section 7 presents a case study of our methodology on the instruction decoder of a 32-bit embedded processor. Conclusions are provided in Section 8.

2 Related Work

To evaluate the influence of a soft error on the circuit behavior, simulation, emulation, or formal methods can be applied. In all the approaches, a soft error is first injected into the circuit in various design models, and then various circuit analysis techniques are applied to evaluate the failure rate. Relying on fault injection and simulation techniques[6–9], and simulation- or emulation-based approaches can enable us to handle large systems, but can only cover a small portion of the states and the input space of a circuit, which cannot guarantee that a given erroneous behavior would never occur under the given fault model.

Formal techniques are introduced into reliability evaluation due to their advantage in completeness. For the formal method-based approaches[10–13], a soft-error model is interleaved with the original design model. Then a formal verification method such as model checking and theorem proving are applied to prove that a system failure would be avoided when a soft-error occurs.

Although formal methods can guarantee the completeness of failure rate evaluation, they cannot be applied to large-scale designs. Furthermore, the completeness is only related to the given properties under verification. If some circuit components are not considered by the given properties, the analysis will not be complete. Finally, a drawback to a prior formal method-based work is that the output is binary, indicating only whether a flip to some bit is capable of violating an assertion, and does not provide the probability of it doing so. In this study, we combine SEC with probabilistic analysis by Markov Chain (MC) analysis to investigate the probability of output inequivalence.

Fault-free simulation approaches, such as Architectural Vulnerability Factor (AVF) method[14], estimate the probability of a bit flip in various functional blocks according to the fraction of time it holds data and instructions that will affect program behavior.

Requiring detailed models of processor logic and data structures, AVF method is limited in specific applications. There is almost no related work on AVF estimation for designs that are not processor cores.

Our SEC-based failure rate analysis is similar to Ref. [15]. However, their work aims to evaluate the overall robustness of circuit and detecting bugs in fault-tolerant structures, while our work intends to evaluate the failure rate of individual FF. Furthermore, we combine circuit states and application behaviors to direct the failure rate analysis, which will serve as a more practical guide to selective protection.

Recently, some studies[16–18] have concentrated on Soft Error Rate (SER) analysis considering device and technology influences, which works on layout and transistor level. However, our method deals with influence factors on the logic level. Li and Draper[19] proposed a joint SER analysis technique for Single-Event Transients (SETs) in combinational logic and Multiple-Cell Upsets (MCUs) in sequential elements. In Ref. [20], the authors proposed a method to accelerate the SER analysis process proposed in Ref. [19]. We consider SEU in sequential elements and consider the application running on the circuits in conducting SER analysis.

Furthermore, existing robustness evaluation techniques assume that the failure rate of circuit components will not change with regard to various applications. Based on our observation, the failure rate changes for different inputs. Under certain circumstances, a soft error presented in an FF will be masked by special inputs[21]. Therefore, considering the application behaviors, we can obtain applicable and precise failure rate evaluation results, which will result in efficient selective protection.

3 Preliminaries

In Ref. [21], we found that under certain circumstances, a soft-error presented in an FF will be masked by special inputs. The observations motivated our work.

Before introducing further work, we take the ISCAS'89 benchmark circuit s27 (Fig. 1) as an example. We assume that its Primary Input (PI) a is 0. If a flip occurs in FF y_1 when the Present State (PS) of the circuit is "011", i.e., $y_1y_2y_3 = 011$, then the Primary Output (PO) o would be 1 instead of the correct value 0. Thus, the PS of s27 "011" is a vulnerable state of y_1. The set of all vulnerable states of an FF y forms the VSS of y, denoted by $VSS(y_i)$, where $i = 1, 2, 3$. In s27, only when its PS is either "001" or "101" can a flip in y_1 not affect the outputs regardless of the input values of the circuit. Thus, $VSS(y_1) = \{y_1y_2y_3|X00, X11, X10\}$, where the state bit with "X" can be both 0 and 1. Similarly, $VSS(y_2) = \{y_1y_2y_3|0XX\}$ and $VSS(y_3) = \{y_1y_2y_3|XXX\}$. Then, $|VSS(y_1)| = 6, |VSS(y_2)| = 4$, and $|VSS(y_3)| = 8$. According to this example, different FFs have different VSSs. Then, whether the VSS of an FF can be used to distinguish the vulnerability of the FF is the main objective of this study.

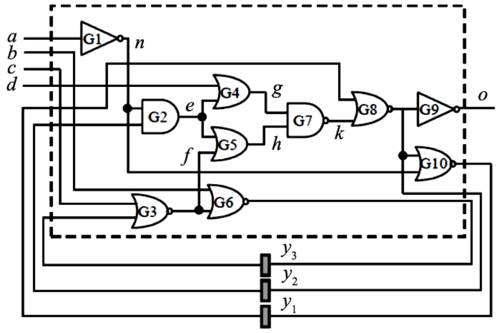


Fig. 1 An example — s27.

Here we first provide some basic definitions to understand our method.

Definition 1 Robust State Set (RSS): For an FF y in a circuit C, with the given fault model F, state s is called a Robust State (RS) of y if an arbitrary injection of a fault into y according to F cannot change the output behavior of C with any inputs presented when the current state of C is s. The set of all RS of y is called robust state set of y, denoted by $RSS(y)$.

Definition 2 VSS: For an FF y in a circuit C, with the given fault model F, state s is called a Vulnerable State (VS) of y, if at least one input can change the output behavior of C after injecting a fault into y according to F when the current state of C is s. The set of all the VS of y is called vulnerable state set of y, denoted by $VSS(y)$.

For any FF y, its current state s exclusively belongs to either $RSS(y)$ or $VSS(y)$.

Definition 3 FF soft-error Failure Rate (FFR): For an FF y in a circuit C, with the given fault model F, the probability that a soft error in y produces erroneous output is called soft-error failure rate of y, denoted by $FFR(y)$, and $0 \leqslant FFR(y) \leqslant 1$.

Definition 4 Vulnerable State Vulnerability Factor

(VSVF): For an FF y in a circuit C, with the given fault model F and a VS s_i ($s_i \in VSS(y)$ and $i = 1, 2, \ldots, |VSS(y)|$), when C is at state s_i and a fault is injected into y according to F, the probability that produces erroneous outputs is called vulnerable state vulnerability factor of s_i, denoted by $VSVF_i(y)$.

A widely accepted view is that fault-inducing particle strikes are randomly and uniformly distributed[14]. Consequently, the probability of soft error occurrence of any FF is also uniformly distributed. Therefore, for any FF y, $FFR(y)$ is the product of the ratio that y is operated at suspicious intervals and the probability (say $R(y)$) that produces erroneous output behaviors when an SEU occurs in y during these intervals:

$$FFR(y) = \frac{Total\ \ Suspicious\ \ Time(y)}{Total\ \ Runtime\ \ of\ \ Circuit} \times R(y) \quad (1)$$

For FF y, an SEU can only suspiciously affect the circuit outputs if it occurs in y when the present state of the circuit is in $VSS(y)$. That is, the suspicious intervals of y are the durations of all its vulnerable states, and therefore, $R(y)$ is $VSVF_i(y)$. We suppose that the circuit has n reachable states and that $D(s_i)$ denotes the total working time of state s_i during the runtime of the circuit. Then Eq. (1) can be refined as follows:

$$FFR(y) = \frac{\sum\limits_{i=1}^{|VSS(y)|} D(s_i) \times VSVF_i(y)}{\sum\limits_{j=1}^{n} D(s_j)} \quad (2)$$

Suppose $T(s_i)$ denotes the time working at state s_i during the runtime of circuit and c is the clock cycle duration time. Then, Eq. (2) is transformed to

$$FFR(y) = \frac{\sum\limits_{i=1}^{|VSS(y)|} c \times T(s_i) \times VSVF_i(y)}{\sum\limits_{j=1}^{n} c \times T(s_j)} = \sum_{i=1}^{|VSS(y)|} \frac{T(s_i) \times VSVF_i(y)}{\sum\limits_{j=1}^{n} T(s_j)} = \sum_{i=1}^{|VSS(y)|} (f_i \times VSVF_i(y)) \quad (3)$$

where $\frac{T(s_i)}{\sum\limits_{j=1}^{n} T(s_j)}$ is exactly the visiting probability of state s_i, which is denoted by f_i. In the Finite State Machine (FSM) of a sequential circuit, the state visiting probability varies with the workloads of the circuit. For a design in which good workload estimation is available, the visit probability of each state can be analyzed through various methods, including Markov Chain (MC) theory[22, 23]. Based on Eq. (3) and the preceding analysis, the soft-error vulnerability of an SEU is determined by its vulnerable state set and the input distribution of the circuit.

As all the visiting probabilities of all the circuit states compose the state distribution vector $V = (f_1, f_2, \ldots, f_n)$, the $FFR(y)$ is correlated to $VSS(y)$, $VSVF_i(y)$, and the steady-state probability distribution of the circuit.

4 Implementation and Optimization

According to Eq. (3), by separately computing the $VSS(y)$, $VSVF_i(y)$, and the steady-state probability distribution of each FF y, we finally obtain the $FFR(y)$. The implementation is based on SEC techniques and transient fault model.

The fault model assumes that a faulty FF behaves non-deterministically in one time step, without losing generality, usually at the first time step. The fault is injected by flipping the current value of the selected FF in one time step, i.e., from 1 to 0, or from 0 to 1.

4.1 Computing *VSS*(*y*)

The $VSS(y)$ is computed using a classic SEC technique called Partial Backward Justification (PBJ)[24]. Similar to Automatic Test Pattern Generation (ATPG)-based equivalence checking techniques, PBJ computes whether some input sequences exist in which the two circuits under verification produce different outputs. However, contrary to ATPG, PBJ is performed by a sequence of symbolic pre-image computations using an unrolled miter as computation model.

Here we briefly provide some related definitions adopted from Ref. [24].

Definition 5 Signal pair: (a_1, a_2) is called a signal pair if a_1 and a_2 are internal signals from different circuits, e.g., a_1 is from C_1 and a_2 is from C_2, or vice versa.

Definition 6 Equivalent pair: (a_1, a_2) is an equivalent (signal) pair if the binary values of signal a_1 and a_2 in response to any input vector are identical.

Definition 7 Merge operation: If (a_1, a_2) is an equivalent pair, replacing a_1 by a_2 is called a merge operation, and the signal a_2 is called a merge point.

Definition 8 Discrepancy function: An input vector v is a distinguishing vector for a signal pair (a_1, a_2) if the application of v can produce (0, 1) or (1,

0) at a_1 and a_2. The characteristic function of the set of distinguishing vectors is called the discrepancy function.

A miter of two circuits under verification is usually formed by joining the corresponding PI pairs of the two circuits and by connecting each corresponding PO pair by an XOR gate. Then the outputs of these XOR gates are generally connected to an OR gate. In this context, for the convenience and efficiency of computing the *VSS*, the outputs of the XOR gates are directly taken as the POs of the miter as shown in Fig. 2. Through the construction of miter, equivalence checking of two circuits can be reduced to verify whether the outputs of the miter are all constant 0.

We consider an arbitrary sequential circuit C and its faulty circuit C' with SEU injected. Figure 3 shows the computation model of our approach based on the unrolled miter circuit of C and C'. In the following, for a component x in C, its corresponding component in C' is denoted by x'. The same label is used for a component and its output signal. In the unrolled miter, each signal has a corresponding copy in each time step which is denoted by using the time-step index as its subscript. For example, y'_0 represents the copy of y' at time step 0. In the computation model, for an arbitrary FF y, its faulty circuit C' is a copy of C and the fault is injected by defining the value of y' at the fault cycle f as: $y'_f = !y_f$. The fault injection point is y'_0. s_i and s'_i denote the Present State (PS) of C and C' at time step i, respectively, while they are also the Next State (NS) of C and C' at time step $i-1$. s'_{id} denotes the PS set which is on the fault propagation paths in C', and s_{id} is the PS set in C corresponding to s'_{id}. s_i/s_{id} denotes the difference of set s_i and s_{id}. Likewise, o'_{id} denotes the set of POs, which are on the fault propagation paths in C'. o_{id} is the corresponding PO set of o'_{id} in C. Other POs in C and C' are omitted. As the model is constructed individually for each FF in C, the flow of our approach needs to run n times for a circuit with n FFs, and one for each FF.

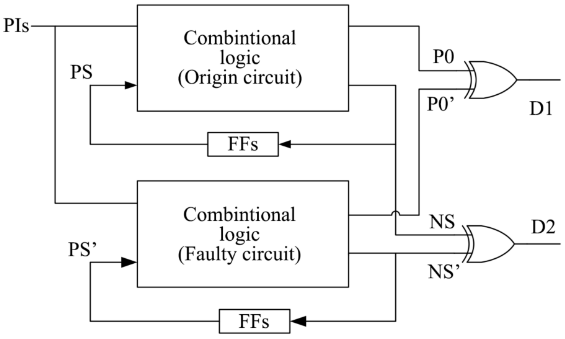

PIs
PS
Combintional logic (Origin circuit)
P0
P0'
D1
FFs
Combintional logic (Faulty circuit)
PS'
NS
NS'
D2
FFs

Fig. 2 Miter of original and faulty circuits to compute *VSS(y)*.

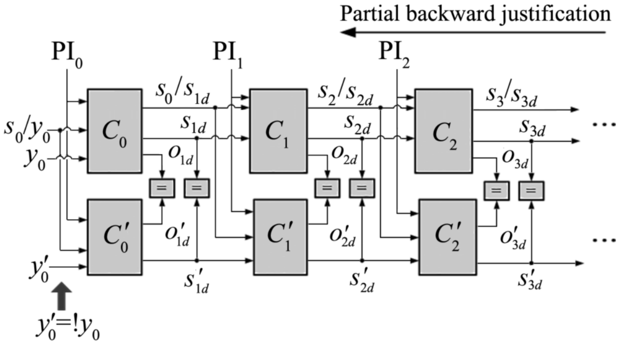


Fig. 3 Computation model to compute *VSS(y)*.

Based on the above computation model, the algorithm for computing $VSS(y)$ is shown in Fig. 4. The algorithm consists of the following phases:

(1) The miter M of C and C' is constructed and unrolled incrementally. The fault is modeled by replacing y'_0 with y_0. The set V is initialized to be empty.

(2) At a newly unrolled time step d, M is first simplified and optimized for consequent computation by detecting and merging all the internal signal pairs, PS pairs and PO pairs which are not on the fault propagation paths.

(3) The equivalence of each PO pair at time step d is checked. If not, PBJ is used to compute the necessary State Requirement Set (SRS) to differentiate each nonequivalent pair that should be satisfied by the PS of C at time step 0. For a nonequivalent PO pair, each state in its *SRS* is a vulnerable state of y. Thus, we add the *SRS* to set V. According to the definition, we obtain $VSS(y) \subseteq R$, where R is the reachable state set of the circuit. After checking one PO pair, if $V = R$, the flow for the current FF terminates with $VSS(y) = R$. Otherwise, the algorithm computes the *SRS* for the rest of the nonequivalent PO pairs. After checking all the PO pairs in time step d, the set V contains all the vulnerable states of y distributed from time step 0 to d.

(4) If $V \subset R$, some vulnerable states of y exist, which may not have been found. These will be propagated through the NS signals at time step d and cause erroneous outputs in later time steps. Thus, we assume that at least one nonequivalent NS pair exists, and PBJ is used to compute the *SRS* to differentiate each nonequivalent NS pair that should be satisfied by the PS of C at time step 0. If $SRS \subseteq V$, then the algorithm

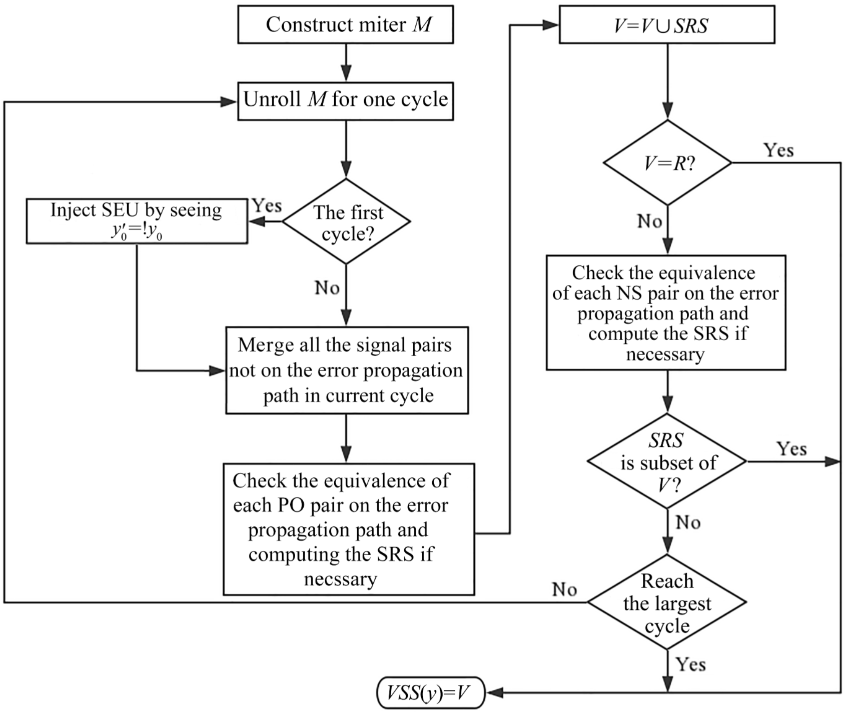

Construct miter M
Unroll M for one cycle
Inject SEU by seeing y'_0=!y_0
Yes
The first cycle?
No
Merge all the signal pairs not on the error propagation path in current cycle
Check the equivalence of each PO pair on the error propagation path and computing the SRS if necssary
V=V∪SRS
V=R?
Yes
No
Check the equivalence of each NS pair on the error propagation path and compute the SRS if necessary
SRS is subset of V?
Yes
No
Reach the largest cycle
No
Yes
VSS(y)=V

Fig. 4 Algorithm flow for computing *VSS(y)*.

terminates with $VSS(y) = R$. Otherwise, the miter is unrolled to time step $d + 1$ and the computations from phase (2) to (4) are repeated.

We can prove that to find all the vulnerable states of y, the miter is unrolled to time step $D+1$ at most, where D is the diameter of the miter. When the algorithm terminates, the set V is $VSS(y)$.

The diameter of some circuits could be infinite due to unreachable states. However, in the context of this study, a reasonable assumption is made that the circuit under consideration is functionally correct and no unreachable states are in it. Therefore, we can safely obtain D of the given circuit. In any case, to consider the infinite circuit diameter, we can settle an upper limit for unrolling by setting the largest cycle in the "Reach the largest cycle?" step in Fig. 4.

The PBJ process for an arbitrary signal pair (y_d, y'_d) is as follows:

(1) (y_d, y'_d) is first assumed to be not equivalent.

(2) A local cut set is selected, denoted by λ_d, in the fan-ins of y_d and y'_d at time step d. The signals in λ_d include PIs, merged points or other internal signals at time step d.

(3) We suppose that $Disc^d_{\lambda_d}(y_d, y'_d)$, which can be obtained by pre-image computation, denotes the discrepancy function of pair (y_d, y'_d) at time step d, i.e., the characteristic function of the set of value combinations at the cut set λ_d, which can differentiate pair (y_d, y'_d). If it is not a zero function, i.e., (y_d, y'_d) cannot be proven equivalent, then the cut set is expanded towards the PIs and PS of time step d. If (y_d, y'_d) cannot be proven equivalent until λ_d contains only PI and PS variables at time step d, then we obtain the PS requirement function of time step d to differentiate (y_d, y'_d), denoted by $SRS^d(y_d, y'_d)$, by existentially quantifying all the PIs in $Disc^d_{\lambda_d}(y_d, y'_d)$. Similarly, the PS requirement function at time step $d - 1$, will be derived by first computing the pre-image of $SRS^d(y_d, y'_d)$ followed by existentially quantifying all the PIs.

(4) Analogously, a number of time steps have to be explored backwardly to time step 0 until the following three stopping criteria of PBJ are met at time step i:

- Justified criterion: Reset state is contained in $SRS^i(y_i, y'_i)$.
- Unjustified criterion: $SRS^i(y_i, y'_i)$ is empty.
- Fixed-point criterion: $SRS^i(y_i, y'_i)$ does not contain the reset state, but is contained in the union of the state requirements derived so far, i.e., $SRS^i(y_i, y'_i) \subseteq \bigcup_{k=d}^{i+1} SRS^k(y_k, y'_k)$.

(5) If the unjustified criterion or the fixed-point criterion is met, then the target pair (y_d, y'_d) is an equivalent pair. Otherwise, it is a nonequivalent pair and its SRS at time step 0 to differentiate (y_d, y'_d) is

$SRS^0(y_0, y'_0)$, which can be derived from $SRS^i(y_i, y'_i)$ analogously.

Along the algorithm flow, the following stop criteria is checked to indicate if another time step expansion is needed:

- When V is equal to the reachable state set R of C, the flow can be stopped. We conclude that y is vulnerable and $VSS(y) = R$, because R is the maximum of a VSS.
- When every NS pair at time step d is an equivalent pair, the flow can be stopped because it indicates that the error cannot continuously propagate to the next time step.
- When M has been unrolled $d = D + 1$ time steps (D denotes the diameter of M), the flow can be stopped. We conclude that $VSS(y) = V$ and y is vulnerable if $VSS(y)$ is not empty. Otherwise, y is robust.

The complete VSS of y can be obtained when $d > D$, which can be formally described as Theorem 1 if we consider our approach in the State Transition Graph (STG) of M.

Theorem 1 If a state s is a vulnerable state of FF y, and its corresponding state in M is s', an edge exists, which outputs 1 in a trace with *length* $\leqslant D + 1$ beginning from s' in the STG of M.

To model the randomicity of SEU in time domain, the initial state of M is not constrained. In other words, the initial state of C can be an arbitrary reachable state. Thus, all the situations in which a fault may occur in FF are considered in the same computation model. This ability is the advantage of formal techniques out of fault-injection-based simulation, which has to enumerate each situation individually.

4.2 Computing $VSVF_i(y)$

We modify the computation model shown in Fig. 2 and use MC analysis to compute $VSVF_i(y)$. The new circuit is shown in Fig. 5 and denoted by M^{modified}. Logic C is used to collect the information on the correct behavior of the circuit, such as inputs PI vector (*PI*), the correct PS vector (*PS*), the correct PO vector (*PO*), and the correct next state vector (*NS*). On the other hand, circuit C' has inputs PI vector (*PI'*, where $PI' \equiv PI$), possibly erroneous PS lines (*PS'*), possibly erroneous PO vector (*PO'*), and possibly erroneous NS vector (*NS'*).

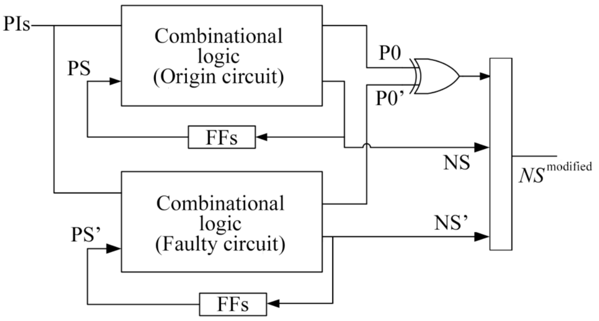

PIs
PS
Combinational logic (Origin circuit)
P0
P0'
FFs
NS
NS^{modified}
Combinational logic (Faulty circuit)
PS'
NS'
FFs

Fig. 5 Circuit model for computing $VSVF_i(y)$.

Here, we only show one pair of PO of the two logics as inputs of the XOR gate. In fact, if C has multiple POs, each pair of PO is connected to one XOR gate, and the outputs of these XOR gates become inputs of an OR gate, the output of which is denoted by ϵ. We can define the NS vectors of C and C' as: $NS = \delta = (\delta_1, \delta_2, \ldots, \delta_n)$ and $NS' = \delta' = (\delta'_1, \delta'_2, \ldots, \delta'_n)$, where n is the number of state variable. We can also define the PO vectors of C and C' as $PO = O = (o_1, o_2, \ldots, o_d)$ and $PO' = O' = (o'_1, o'_2, \ldots, o'_d)$, where d is the number of outputs. Then, we obtain $\epsilon = (o_1 \bigoplus o'_1) \vee (o_2 \bigoplus o'_2) \vee \cdots \vee (o_d \bigoplus o'_d)$, which represents possible errors in the output lines. In other words, $\epsilon = 1$ means that at the present state an error exists in the corresponding element of the output vector PO' of the circuit C', when compared with the output vector PO of the circuit C. Finally, the next state vector of the model for computing $VSVF_i(y)$ is $NS^{\text{modified}} = (\delta_1, \delta_2, \ldots, \delta_n, \delta'_1, \delta'_2, \ldots, \delta'_n)$. Consequently, we define ϵ as the state bit for erroneous outputs, and the state with $\epsilon = 1$ as output erroneous state, denoted by s_ϵ.

An SEU in FF y when at its vulnerable state means flipping the value of y and keeping the values of other FFs. Furthermore, as the SEU in y does not begin to be propagated when SEU occurs, ϵ still takes a value 0 at that time. Therefore, according to Definition 2, each vulnerable state of C has a corresponding state in M^{modified}, and vice verse.

For example, we suppose that circuit is at state “001”, which is the vulnerable state of FF y, and an SEU occurs in y, where y is the left most state bit. Then, the corresponding state in M^{modified} is “001 101 0”.

After constructing M^{modified}, we can obtain its STG of it. Figure 6 shows an example circuit, its STG and that of M^{modified}, where the “0” or “1” labeled on each out-going edge is the input values that cause the state transition. As the inputs of M^{modified} and C are the same, for the given input probability distribution, we can easily build the MC for M^{modified}.

As the STG of M^{modified} contains state transitions

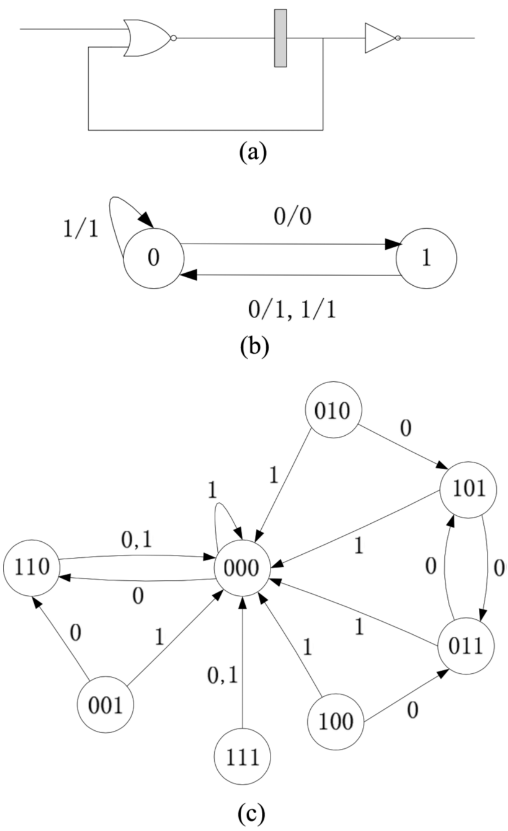


Fig. 6 (a) An example circuit, (b) STG of the example, and (c) STG of the M^{modified} of the example.

starting from s_ϵ states, the probability of these outgoing edge may be greater than 0. For example, in Fig. 6, the states with erroneous output are 001, 011, 101, and 111; from any one of them, other reachable states can be reached. However, in the original circuit, the circuit fails to work whenever an error occurs at primary outputs. Therefore, in the original circuit, the probability of outgoing edges from s_ϵ states is always 0.

In our method, the signal ϵ, which indicates output errors, is at the right most bit of the state encoding. Therefore, all the states with error outputs are encoded with "1" at the right most bit. In other words, the states with error outputs are encoded as odd numbers. Consequently, to precisely model the behavior of soft errors, we can fill the lines of even numbers in the MC matrix with "0". For the sample circuit in Fig. 6, the corresponding matrix of M^{modified} is as follows:

$$P = \begin{bmatrix} p_{00} & p_{01} & \cdots & p_{06} & p_{07} \\ p_{10} & p_{11} & \cdots & p_{16} & p_{17} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ p_{60} & p_{61} & \cdots & p_{66} & p_{67} \\ p_{70} & p_{71} & \cdots & p_{76} & p_{77} \end{bmatrix} = \begin{bmatrix} p_{00} & p_{01} & \cdots & p_{06} & p_{07} \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ p_{60} & p_{61} & \cdots & p_{66} & p_{67} \\ 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \tag{4}$$

Theoretically, for any FF y and any of its *VS* s_i, $VSVF_i(y)$ is the sum of all transition probabilities which start from s_i and reach any state s_j with error outputs in h steps in MC M^{modified}, that is

$$VSVF_i(y) = \sum_{h=1}^{\infty} \sum_{s_j=1} p_{ij}^{(h)} \tag{5}$$

According to Eq. (5), to precisely compute $VSVF_i(y)$, it must satisfy $h \to \infty$. However, since the earlier the FF injected with soft error causes error outputs, the more vulnerable it is[25]. Consequently, the larger the state transition step h is, the less influence it has on the precision of $VSVF_i(y)$. In our method, the longest transition steps can be defined by designers (such as k). Then Eq. (5) is refined as follows:

$$VSVF_i(y) = \sum_{h=1}^{k} \sum_{s_j=1} p_{ij}^{(h)} \tag{6}$$

By analyzing the characteristic of computation of $VSVF_i(y)$, we can further reduce the state space of M^{modified}, and so does the size of the corresponding MC matrix. According to the above computation process of $VSVF_i(y)$, we found that for each state, the process focuses only on the n step transition probabilities from it to the s_ϵ states. However, in the computation model, we also consider the transition probabilities that start from s_ϵ states to other states, which brings a large portion of useless storage cost into the computation. Furthermore, in the computation model, the states that cannot reach any s_ϵ states, and the robust state that cannot be reached from any states have no effect on the computation of $VSVF_i(y)$. Consequently, all these states can be deleted from the STG of M^{modified} to reduce the size of the MC matrix.

Therefore, the optimization can be conducted in following steps:

(1) All the outgoing edges of s_ϵ states are deleted.

(2) All s_ϵ states that cannot be reached from any states without error outputs are deleted.

(3) All the rest s_ϵ states are merged into one state, denoted by s_m, and all the edges from any state without error outputs are redirected to s_m.

(4) All the states that cannot reach s_m and the robust states that cannot be reached from any states are deleted.

Figure 7a shows the results of applying the first three optimization steps to M^{modified} shown in Fig. 6, which can reduce the number of states from 8 to 5. Figure 7b shows the final result after finding that states "110" and "000" cannot reach s_m and then be deleted, which can finally reduce the number of states to 3. Through these optimizations, the storage and computation for the MC matrix can be dramatically reduced.

Therefore, after optimizing the STG of M^{modified}, Eq. (6) is transformed to $\sum_{h=1}^{k}\sum_{s_j=1} p_{im}^{(h)}$. That is, the state transition probabilities for any state that reaches s_m in h steps will occupy the $(j-1)$-th column of the corresponding MC matrix. Therefore, for any FF, a corresponding $VSVF_i(y)$ exists in column $(j-1)$ for any transition steps from 1 to h.

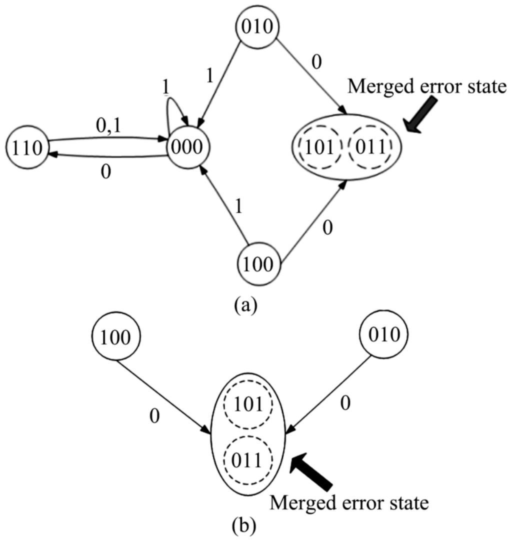


Fig. 7 (a) STG after optimization by the first 3 steps, and (b) final STG after application of last optimization step.

4.3 Computing steady-state probabilities

To compute steady-state probabilities, we directly apply the methods and optimization techniques proposed in Refs. [22, 23]. For the given initial distribution of circuit, which is input vector probability distribution, we can easily compute the steady-state probabilities by using MC techniques.

4.4 Putting everything together

As shown in Fig. 8, the proposed method is integrated into one flowchart. The method can be divided into 4 stages. After computing VSS, VSVF, and runtime state distribution, we sort the resulting FFR of each FF. The sorting result is used to select FF to be protected.

The complexity of computing VSS is $O(C^2 \times |FF|)$, where $|FF|$ is the number of FFs in a circuit, while C is the upper bound of unrolling the miter, which could be a value k set by users or the diameter D of a circuit. However, the relation $k \leqslant D$ exists. Therefore, we take the complexity of computing VSS as $O(D^2 \times |FF|)$. We find the complexity of stage 2, which is computing runtime state distribution, and of stage 3, which is computing VSVF. Equation (6) determines the complexity of stage 3, which is $O(k^2)$, where $k \leqslant D$. Therefore, we take $O(D^2)$ as the complexity of stage 3. The algorithm used in stage 2 is from Refs. [22, 23], which also uses MC for computation. Therefore, the

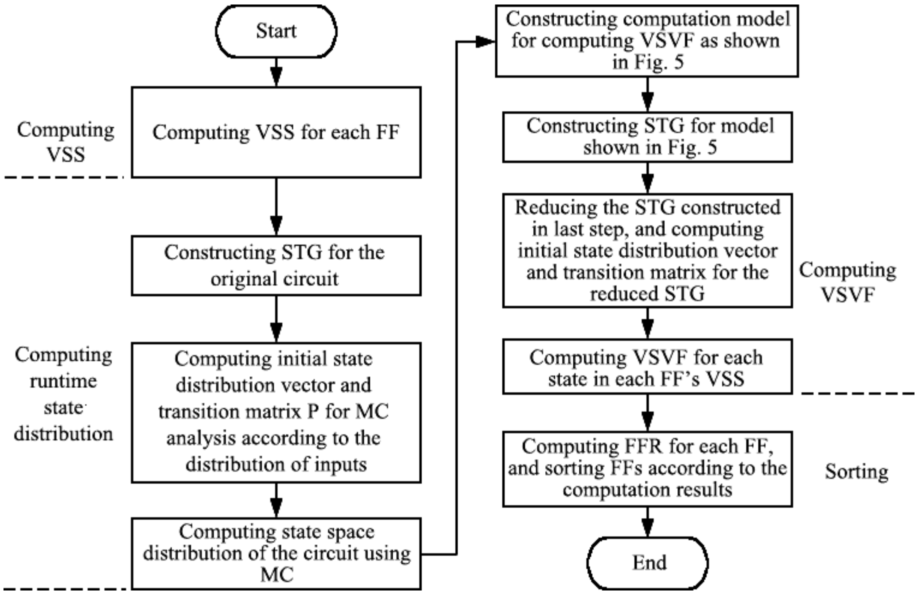

Start
Computing VSS
Computing VSS for each FF
Constructing STG for the original circuit
Computing runtime state distribution
Computing initial state distribution vector and transition matrix P for MC analysis according to the distribution of inputs
Computing state space distribution of the circuit using MC
Constructing computation model for computing VSVF as shown in Fig. 5
Constructing STG for model shown in Fig. 5
Reducing the STG constructed in last step, and computing initial state distribution vector and transition matrix for the reduced STG
Computing VSVF
Computing VSVF for each state in each FF's VSS
Computing FFR for each FF, and sorting FFs according to the computation results
Sorting
End

Fig. 8 Whole algorithm.

complexity of stage 2 is also $O(D^2)$ too. In summary, we find that the complexity of our method is $O(D^2 \times |FF|)$.

5 Examples

In this section, two circuits are used to demonstrate the proposed soft-error failure rate analysis methodology. The interesting phenomena of FF failure rate with special application behaviors are also presented.

The first example is circuit s27 in ISCAS'89 benchmarks. For the given initial distribution of s27 and input vector probability distribution, Fig. 9 shows the curve of FFR of each FF changing along time steps. The vertical axis is FFR, and the horizontal axis is the time step. We can find that FFR of each FF finally becomes steady since s27 is a circuit with steady-state probability. In the duration indicated by the two dotted lines, the order of the FFs' FFR is y_2, y_1, and y_3. Thereafter, y_2 is still the FF with the largest failure rate.

The second circuit SBA2[26] is a synchronous bus arbiter that contains two cells. It has two inputs (Req$_0$ and Req$_1$), four FFs (W_0, W_1, T_0, and T_1), and eight reachable states. Its STG is shown in Fig. 10a, and the initial state is "0010". We suppose that the input vector probability distribution is uniform, and the curve of FFR of each FF changing along time step is shown in Fig. 10b. We can find that the FFR of T_0 and T_1 is always 1, which means that whatever states SBA2 is in, any soft error that occurs in T_0 or T_1 will cause the circuit to fail. The other two FFs' (W_0 and W_1) FFR are changed periodically because SBA2 has no steady-state probabilities and the state probabilities are periodically distributed.

Based on Figs. 9 and 10, the following conclusions are obtained:

- For circuits with steady-state probability and steady distributed inputs, the FFR of each FF finally becomes steady.

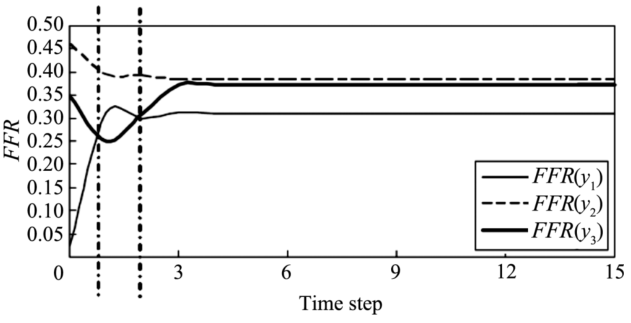
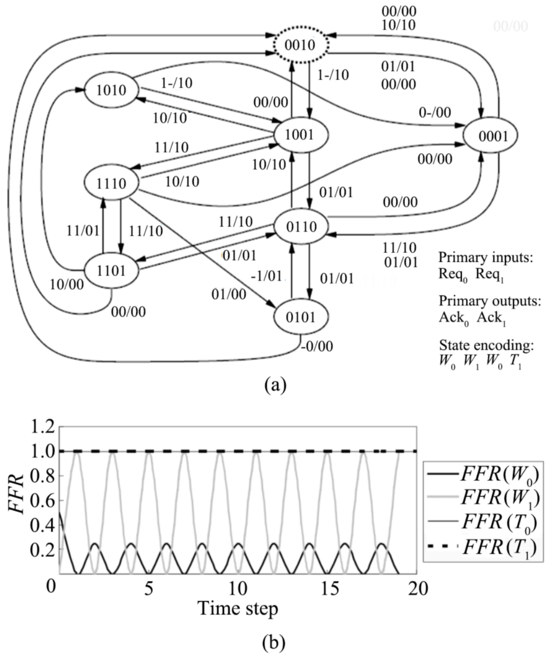


Fig. 9 FFR curve of each FF in circuit s27.



(a)

(b)

Fig. 10 (a) STG of circuit SBA2[26], and (b) FFR changing curve of each FF in circuit SBA2.

- For circuits with unsteady-state probability, some FFs' FFR will not be steady and will change periodically.

6 Experimental Results

We have implemented the proposed method for soft-error failure rate analysis. The tool is implemented in Java and takes gate-level netlist as inputs, in which JavaBDD package[27] is used for Boolean function manipulation. The SEC is based on the techniques proposed in Ref. [28]. All experiments are conducted on a workstation equipped with two 4-cores 2.66 GHz Intel Xeon processors, 16 GB memory, and 12 MB Cache. All the designs are synthesized to gate-level netlist using Synopsys Design Compiler 2007 with ARM SMIC 130 nm Logic013G standard cell library. We demonstrate our approach on a third party Verilog implementation of a SpaceWire network end node and the largest ISCAS'89 benchmarks.

To evaluate the accuracy of our method in selecting FF for protection, we use Built-In Soft-Error Resilience (BISER)[5] to harden the selected FFs. The evaluation metrics for the quality of FF selection is error coverage. A reliability evaluation tool is built based on our previous work[29]. The structure of the tool, as shown in Fig. 11, consists of an automatic simulation stimulus generator, a fault injector, two simulators, and a comparator. The fault injector is used to inject transient faults into the circuit. During each run of the

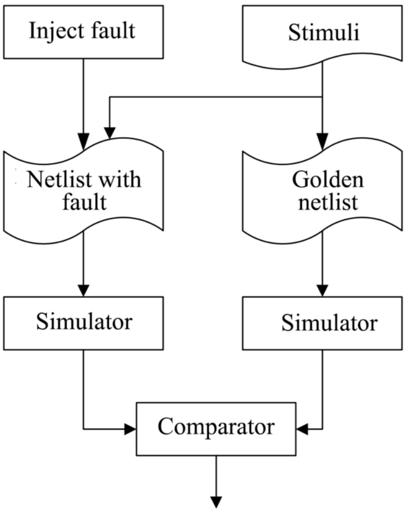


Fig. 11 Circuit reliability analysis based on fault simulation used in the experiments.

experiment, 100 000 moments are randomly selected for fault injection. The same stimuli are applied to two copies of circuits (with or without SEU) after injecting one fault. The simulations on two copies of circuits are run in parallel and the corresponding outputs and state variables are compared cycle by cycle. Nonequivalence means that the injected fault is not tolerant by hardening the selected FFs. Then, the error count is increased by 1, and the states of the circuit with fault are recovered by those of the golden model. Thereafter, the simulation continues and new fault is injected at the next moment.

In our study, each experimental circuit is analyzed twice with the reliability evaluation tool. The first pass is run on the golden model to determine the error count that without hardening. The second pass is run on the hardened circuit to obtain the error count after selective protection. More precisely, all the moments to inject fault are identical for the two passes. Finally, the error coverage is calculated using following formula, where EODAP means "error outputs detected after protection" and EODWP means "error outputs detected without protection":

$$Error\ Coverage = 1 - \frac{EODAP}{EODWP} \times 100\% \qquad (7)$$

The experimental results conducted on the largest ISCAS'89 benchmarks are shown in Table 1. The second and third columns show the number of FFs and the number of robust FFs in each circuit, respectively. Columns 4 to 7 show the error coverage with four sets of stimuli for the largest 20%, 40%, 60%, and 80% failure rate FFs being protected.

We also compared our method with that proposed in Ref. [21]. The remaining columns of Table 1 show the error coverage with four sets of stimuli for the largest 20%, 40%, 60%, and 80% failure rate FFs being protected and those by the method in Ref. [21].

Table 2 shows the memory and time cost of the proposed method and those in Ref. [21]. The fourth and fifth columns show the time and memory cost for failure rate analysis by the proposed method. We can conclude that according to the behaviors of applications, the failure rate analysis of FFs is more accurate with minor overhead analysis cost.

The SpaceWire network end node design, which is downloaded from opencore.org[30], is the source and the sink of packets. It consists of a receiver, a transmitter, and a state machine FSM for protocol control. In the experiment, the functions of the receiver and the transmitter are as follows: the FSM module is responsible for generating and sending control signals to the transmitter and the receiver. The transmitter sends the packets across the link. The receiver buffers data and detects various errors that might occur such as disconnection and parity errors. Furthermore, the receiver reports errors to the FSM module. The synthesized netlist of this circuit contains 145 FFs.

The vulnerability evaluation resulting from our approach shows that 110 FFs in the SpaceWire node design are robust, while the remaining 35 FFs are

Table 1 Experimental results of ISCAS'89 circuits.

Circuit	Number of FFs	Number of robust FFs	Error coverage				Error coverage[21]			
			20%	40%	60%	80%	20%	40%	60%	80%
s4863	104	0	0.261	0.463	0.670	0.868	0.165	0.361	0.560	0.732
s5378	179	23	0.762	0.863	0.972	0.990	0.632	0.754	0.851	0.928
s3384	183	0	0.333	0.459	0.527	0.805	0.229	0.349	0.413	0.648
s9234	211	33	0.296	0.453	0.778	0.981	0.179	0.371	0.690	0.893
s15850	534	0	0.459	0.573	0.778	0.986	0.353	0.568	0.754	0.916
s38584	1426	0	0.422	0.617	0.760	0.959	0.316	0.501	0.659	0.877
s38417	1636	72	0.463	0.645	0.781	0.972	0.362	0.540	0.694	0.901
s35932	1728	0	0.512	0.755	0.919	0.997	0.421	0.635	0.819	0.920

Table 2 Experimental results of ISCAS’89 circuits.

Circuit	Number of FFs	Number of robust FFs	Time (s)	Memory (MB)	Time (s)[21]
s4863	104	0	202.2	37.319	129
s5378	179	23	338.6	40.205	241
s3384	183	0	679.8	67.801	417
s9234	211	33	1441.9	94.493	614
s15850	534	0	2055.8	137.180	1231
s38584	1426	0	5561.6	371.201	4305
s38417	1636	72	7242.3	402.729	6276
s35932	1728	0	10 023.7	401.815	9164

vulnerable, among which the VSSs of 16 FFs are the reachable state set of the design, which is the FFs without immunity. Then various percentages of the FFs with the largest failure rate are selected for protection. Four sets of stimuli with various input distribution are used to evaluate the error coverage after circuit protection. The experimental results are shown in Fig. 12, where the horizontal axis is the percentage of FFs to be protected, while the vertical axis indicates the error coverage for the corresponding protections. We can observe from the figure that when 11% of the FFs (16 FFs without immunity) are selected for protection, the average error coverage of the four sets of stimuli is approximately 77%. After the protected FFs are increased to 24% (35 FFs), the error coverage approaches 100%.

Based on the ARM SMIC 130 nm Logic013G standard cell library, the extra power and area overhead introduced for various selective protections using the BISER technique is also evaluated, as shown in Table 3. The last line shows that without protection, no power and area overheads are introduced. The second line shows the extra overheads obtained from full protection for all the 145 FFs. The third line shows that if all the 35 vulnerable FFs are protected, then the result is an extra 14.8% power and 0.36% area penalty. The fourth line shows when the 16 FFs without immunity are selected for protection, extra 6.5% power penalty and 0.15% area penalty are introduced. Accordingly, fault tolerance guided by our approach can cut 56% power penalty and 58.3% area penalty from the overall penalty of protecting all the vulnerable FFs if the error coverage (77%) is acceptable.

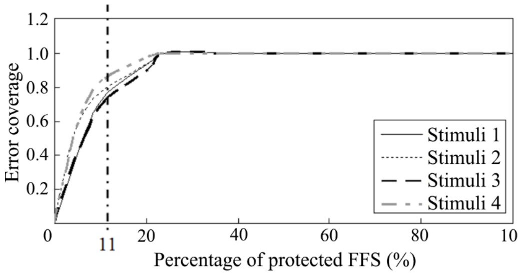


Fig. 12 Error coverage changing curve according to percentage of protected FFs.

Table 3 Extra power and area overhead for different degrees of protection.

Degree of protection	Power overhead (%)	Area overhead (%)
Full protection	58.9	1.39
All vulnerable FFs protection	14.8	0.36
11% vulnerable FFs protection	6.5	0.15
No protection	0	0

7 Case Study

In this case study, we applied our application-specified failure-rate analysis methodology to evaluate the instruction decoder of Estar2 for Quantum Geography Information System (QGIS) application. Estar2 is a 32-bit embedded microprocessor that contains a high performance 32-bit microprocessor core. The instruction decoder circuit of the core is the most complex component, which consumes the largest area and power cost of the entire design. The instruction decoder circuit consists of 369 FFs, 3439 combinational logic gates, and 485 output signals. As the outputs of the decoder impose the greatest influence on functions of the following pipeline, it is the most vulnerable to soft error. QGIS is an open source digital map software that can perform various operations on maps.

In the case study, the QGIS source code is first compiled into an executable “el” file using “arm-elf-gcc”. Then, the executable elf file is run on an Estar2 instruction set simulator based on SimpleScalar[31], and the execution trace is recorded. The recoded trace is analyzed to obtain the input vector probability distribution, which is one of the inputs of our method.

After analysis, we find that all the FFs in the circuit are vulnerable to soft error. The time spent to analyze the failure rate is approximately 5329 seconds. Then, by full protection, approximately extra 45.4% power and 1.18% area overheads are introduced. Considering the design constraints that extra power and area overhead do not exceed 22.7% and 0.59%, respectively, we select the first 189 FFs with highest failure rate to be protected. The error coverage approaches 91% after hardening.

8 Conclusion

We have presented a novel methodology for performing soft-error failure rate analysis of arbitrary sequential circuit designs, which relates the failure rate of FFs with the behaviors of applications run on it. The methodology is implemented using sequential equivalence checking techniques, which has minimal requirements on designers and is an automatic and complete evaluation technique. With the optimization techniques, our method can be applied to large-scale designs. The experimental results and case study show the effectiveness of our methods. In the near future, we will extend our work to deal with soft error in combinational component and multiple soft-error analysis, and also mitigate our work to Boolean SATisfiability problem (SAT)-based techniques.

Acknowledgment

The study was supported by the National Key Basic R&D Program (973) of China (No. 2017YFB1001802).

References

[1] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvis, Modeling the effect of technology trends on the soft error rate of combinational logic, in *Proc. 32nd Int. Conf. on Dependable Systems and Networks*, Bethesda, MD, USA, 2002, pp. 389–398.

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

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

[4] A. Pan, O. Khan, and S. Kundu, Improving yield and reliability of chip multiprocessors, in *Proc. 16th Int. Conf. Design, Automation and Test in Europe*, Nice, France, 2009, pp. 490–495.

[5] M. Zhang, S. Mitra, T. M. Mak, N. Seifert, Q. Shi, K. Kim, N. Shanbhag, N. Wang, and S. Patel, Sequential element design with built-in soft error resilience, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 12, pp. 1368–1378, 2006.

[6] E. Böhl, W. Harter, and M. Trunzer, Real time effect testing of processor faults, in *Proc. 5th IEEE Int. On-Line Testing Workshop*, Rhodes, Greece, 1999, pp. 39–43.

[7] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, Statistical fault injection: How much is sufficient? in *Proc. 16th Design, Automation and Test in Europe*, Nice, France, 2009, pp. 502–506.

[8] L. Berrojo, I. González, F. Corno, M. Reorda, G. Squillero, L. Entrena, and C. Ongil, An industrial environment for high-level fault-tolerant structures insertion and validation, in *Proc. IEEE VLSI Test Symposium*, Monterey, CA, USA, 2002, pp. 229–236.

[9] 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 *Proc. 13th Conf. Design, Automation and Test in Europe*, Munich, Germany, 2006, pp. 176–181.

[10] S. Seshia, W. Li, and S. Mitra, Verification-guided soft error resilience, in *Proc. 14th Conf. Design, Automation and Test in Europe*, Nice, France, 2007, pp. 1442–1447.

[11] A. Darbari, B. Al-Hashimi, P. Harrod, and D. Bradley, A new approach for transient fault injection using symbolic simulation, in *Proc. 14th IEEE Int. On-Line Testing Symposium*, Rhodes, Greece, 2008, pp. 93–98.

[12] L. Pierre, R. Clavel, and R. Clavel, ACL2 for the verification of fault-tolerance properties: First results, in *Proc. 8th Int. Workshop on the ACL2 Theorem Prover & Its Applications*, Boston, MA, USA, 2009, pp. 90–99.

[13] N. Miskov-Zivanov and D. Marculescu, Modeling and optimization for soft-error reliability of sequential circuits, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 5, pp. 803–816, 2008.

[14] S. S. Mukherjee, C. T. Weaver, J. S. Emer, S. K. Reinhardt, and T. M. Austin, A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor, in *Proc. 36th Annual Int. Symposium on Microarchitecture*, San Diego, CA, USA, 2003, pp. 29–40.

[15] G. Fey, A. Sulflow, S. Frehse, and R. Drechsler, Effective robustness analysis using bounded model checking techniques, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 8, pp. 1239–1252, 2011.

[16] Y. Du and S. Chen, A novel layout-based single event transient injection approach to evaluate the soft error rate of large combinational circuits in complimentary metal-oxide-semiconductor bulk technology, *IEEE Transactions on Reliability*, vol. 65, no. 1, pp. 248–255, 2016.

[17] B. Liu, S. Chen, B. Liang, Z. Liu, and Z. Zhao, The effect of re-convergence on SER estimation in combinational circuits, *IEEE Transactions on Nuclear Science*, vol. 56, no. 6, pp. 3122–3129, 2009.

[18] S. Kiamehr, T. Osiecki, M. Tahoori, and S. Nassif, Radiation-induced soft error analysis of SRAMs in SOI FinFET technology: A device to circuit approach, in *Proc. 51st Annual Design Automation Conference*, San Francisco, CA, USA, 2014, pp. 1–6.

[19] J. Li and J. Draper, Joint soft-error-rate (SER) estimation for combinational logic and sequential elements, in *Proc. 2016 IEEE Computer Society Annual Symposium on VLSI*, Pittsburgh, PA, USA, 2016, pp. 737–742.

[20] J. Li and J. Draper, Accelerated soft-error-rate (SER) estimation for combinational and sequential circuits, *ACM Transactions on Design Automation of Electronic Systems*, vol. 22, no. 3, pp. 420–441, 2017.

[21] D. Zhu, T. Li, and S. Li, An approximate soft error

reliability sorting approach based on state analysis of sequential circuits, in *Proc. 25th IEEE Int. Symposium on Defect and Fault Tolerance in VLSI Systems*, Kyoto, Japan, 2010, pp. 209–217.

[22] G. Hachtel, E. Macii, A. Pardo, and F. Somenzi, Markovian analysis of large finite state machines, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 12, pp. 1479–1493, 1994.

[23] D. Marculescu, R. Marculescu, and M. Pedram, Trace-driven steady-state probability estimation in FSMs with application to power estimation, in *Proc. 5th IEEE Conf. Design, Automation and Test in Europe*, Paris, France, 1998, pp. 774–781.

[24] S. Y. Huang and K. T. Cheng, *Formal Equivalence Checking and Design Debugging*. Norwell, MA, USA: Kluwer Academic Publishers, 1998.

[25] G. Asadi and M. B. Tahoori, Soft error modeling and protection for sequential elements, in *Proc. 20th IEEE Symposium on Defect and Fault Tolerance in VLSI Systems*, Monterey, CA, USA, 2005, pp. 463–474.

[26] K. L. McMillan, *Symbolic Model Checking*. Norwell, MA, USA: Kluwer Academic Publishers, 1994.

[27] JavaBDD, https://sourceforge.net/projects/javabdd/, 2018.

[28] D. Zhu, T. Li, and S. Li, 2D decomposition sequential equivalence checking of system level and RTL descriptions, in *Proc. 9th IEEE Int. Symposium on Quality of Electronic Design*, San Jose, CA, USA, 2008, pp. 637–642.

[29] T. Li, Y. Guo, G. Liu, and S. Li, Functional vectors generation for RT-level Verilog descriptions based on path enumeration and constraint logic programming, in *Proc. 8th Euromicro Conference on Digital System Design*, Porto, Portugal, 2005, pp. 17–25.

[30] SpaceWire, https://opencores.org/projects/spacewire_light, 2018.

[31] SimpleScalar, http://www.simplescalar.com, 2018.

Tun Li is a professor in the College of Computer, National University of Defense Technology. He received the PhD degree in computer science from National University of Defense Technology in 2003. His research interests include electronic design automation and SoC design verification.

Hai Wan received the BS degree from National University of Defense Technology, Changsha, China, in 2003 and PhD degree from Tsinghua University, Beijing, China, in 2011. From 2011 to 2012, he was a postdoctor researcher at INRIA-Nancy, France. From 2013, he is on the faculty at Tsinghua University. He is currently a research associate professor at the School of Software, Tsinghua University. His research interests include real time systems and embedded systems.

Qinhan Yu received the BS degree from Tsinghua University, Beijing, China, in 2014, where he is currently working toward the PhD degree in software engineering in the School of Software. His current research interests include real-time systems, network scheduling, and embedded systems.

Sikun Li is a professor in the College of Computer, National University of Defense Technology, China. He received the BS degree from PLA Military Institute of Engineering in 1960. His research interests include VLSI design methodology, virtual reality technology, and computer aided design.