ACSEM: Accuracy-Configurable Fast Soft Error Masking Analysis in Combinatorial Circuits

Florian Kriebel, Semeen Rehman, Duo Sun, Pau Vilimelis Aceituno, Muhammad Shafique, Jörg Henkel
Chair for Embedded Systems (CES), Karlsruhe Institute of Technology (KIT), Germany
Corresponding Author's Email: muhammad.shafique@kit.edu

***Abstract*—Small feature sizes and associated low-operating voltages have led to radiation-induced soft errors as a major source of unreliability in modern circuits. As not all errors propagate to the final output of a combinatorial circuit (e.g., because of logical masking effects), an analysis of the error masking characteristics is required to evaluate and enhance the quality of a reliable processor design. State-of-the-art gate-level soft error masking techniques require a significant amount of analysis time due to their inherent nature of parsing and analyzing the complete processor's netlist, which may take up to several days. In this paper, we present a fast and Accuracy-Configurable Soft Error Masking analysis technique (ACSEM) that performs error probability analysis on parts of netlist within the user-provided masking accuracy range. To enable this, we theoretically derive the maximum number of steps in the netlist graph that has to be processed to reach the required masking accuracy level. This significantly reduces the analysis time by orders of magnitude compared to traditional state-of-the art approaches that process all logic gate paths in a given combinatorial circuit.**

I. Introduction and Related Work

The technical scaling in the nano-era has enabled smaller, faster, and low-power transistors that provide high performance-per-power efficiency. However, ultra-small dimensions and low operating-/threshold voltages have led to various reliability threats like *soft errors* and aging [1, 2]. Soft errors are transient bit flips in the logic gates or memory cells (e.g., due to high-energy particle strike) that propagate to the software layers and jeopardize correct program execution. Due to their transient nature, soft errors are hard to detect in complex circuits (e.g., processor pipeline) in a cost-effective way.

Experimental studies by [3–6] have shown that various soft errors are masked at different system layers, ranging from circuit-level to the application software-level. In this paper, we focus on the circuit-level soft error masking analysis considering the variations in the input stimuli (e.g., due to varying applications' workload characteristics) as this is of more importance from the reliable hardware architecture (e.g., processors, accelerator-based systems) perspective [3, 7].

Circuit-Level Soft Error Masking: Soft errors can occur at any gate inside a circuit and their effects can propagate to multiple successor gates and may finally appear as a user-visible error. However, the further propagation of soft errors in logic circuits may be prevented at a certain gate in the forward (logic gate) path due to the 'gate type' and 'values of other input signals'. This effect is called *circuit-level logical masking*[1] [9–12]. Hence, *a soft error may not lead to an incorrect result*. Ignorance of the circuit's error masking property may lead to over-design of a reliable processor or application-specific hardware accelerator circuit that may incur significant power and area overhead[2]. Therefore, *a fast analysis of the error masking characteristics is required* to evaluate and enhance the quality of a reliable processor design in a cost-effective way, while curtailing the analysis and design time to ensure fast time to market.

State-of-the-Art Soft Error Masking Analysis Techniques and Their Limitations: Several state-of-the art approaches for soft error estimation have been proposed, widely ranging from statistical to empirical approaches, e.g., fault-injection [6, 13], analytical vulnerability estimations techniques [3–6], etc. Monte-Carlo based statistical fault-injection techniques typically perform numerous fault-injection experiments on the gate-level or at the architecture-level [6, 13]. In [6], an emulation-based fault injection technique on an FPGA is presented, where two circuit-level models (i.e. gate and register-transfer level) are integrated.

These fault-injection based techniques may require long simulation time in order to reach the required degree of accuracy. Therefore, such techniques are not suitable for large-sized circuits [3, 9]. To alleviate the time overhead, probabilistic/analytical approaches caught attention which have shown to be faster than Monte-Carlo based fault-injection [7, 9, 14, 15]. The Soft Arch approach [7] presents an architecture-level model and tool that is used to quantify mean time to failure (MTTF) of a processor. [9] considers both signal and error correlation for soft error analysis. However, these techniques still demonstrate their feasibility and applicability on small-sized circuits and would require a significant analysis time for large-sized netlists due to their design complexity and significant increase in the combinatorial design space [10, 16]. The primary reason of high complexity of these techniques is that they rely on full/exhaustive exploration and analysis of the complete processor's netlist, multi-level correlation analysis from input to output, etc. Moreover, for high coverage, the soft error masking analysis is typically performed for various input stimuli resulting in a proportional increased in the analysis time.

When applying state-of-the-art full netlist analysis techniques (like [17, 18]), we observed that, due to the exploration of full netlist, these techniques requires $2-3$ weeks for performing the soft-error masking analysis for the LEON-3 processor netlist[3] for a single application and single stimulus. This time overhead linearly increases with the set of input stimuli and benchmark applications, as a different input results in varying signal probabilities, thus a different soft error masking distribution. This illustrates that such techniques do not scale with the increasing complexity of processors and application-specific circuits in the nano-era. Some techniques aim at restricting the parts of netlist in form of cone [9], but still explore the full netlist from root to the leaf nodes, which may contribute to millions-billions possible combinations of logic gate paths; as we will show in our results (Section IV-B). In our experiments, we observed that a significant amount of errors near the circuit input were 100% masked before arriving the circuit output. This observation lays one of the most important foundations of our work. Typically, in a real-world system design depending upon the target applications, a user of the soft error analysis tool[4] specifies a target system reliability level (e.g., mean-time-to-failure as the device error rate) with a probability of

[1]There are also other types of masking effects like *electrical masking* and *latch window masking* within the gate/SRAM cell [8]. In this paper our scope is on *logical masking effect* that span large-sized netlists and exhaustive analysis of logical masking effects is extremely time consuming [6, 9, 10].

[2]This can potentially also increase the verification and test costs.

[3]Synthesized using Synopsis Design Compiler with TSMC 45nm technology; Masking analysis performed on an Intel Core-i7 processor with 8GB RAM, See detailed experimental setup in Section IV-A

[4]For instance, system designer, system architecture, verification/test engineers, etc.

tolerate errors, since ensuring a perfectly (100%) reliable system is only cost-wise feasible in mission-critical systems. Therefore, *enabling a tradeoff between the analysis accuracy and the analysis time is a highly desirable feature*.

In summary, there is a need for fast soft-error masking analysis technique that can curtail the analysis time for large-scale netlists from days/weeks to minutes/seconds. Moreover, such a technique needs to provide a tradeoff between analysis accuracy and analysis time in order to facilitate fast analysis for a multitude of diverse applications ranging from highly soft-error tolerant to mission-critical.

Our Novel Contributions: In order to address the above-discussed scientific challenges, we propose a *fast and Accuracy-Configurable Soft Error Masking analysis technique (ACSEM)* that performs masking probability analysis on parts of a netlist within the user-provided accuracy range of error probabilities. For that, we theoretically derive the maximum number of steps in the circuit netlist graph, such that any soft error at this point or onwards may reach the circuit output signals with a probability more than the user-specific non-tolerable error range. In other words, any error beyond (i.e., from this point towards the inputs) will have a masking probability greater than the user-specific masking constraint, thus does not need to be evaluated. Afterward, ACSEM employs a Backward Depth-First Search to perform on-the-fly logic gate path extraction and (soft-error) masking probability computation (i.e., starting from the leaf node in the netlist graph) under the constraint of maximum step size. Our technique speeds up the soft error analysis by orders of magnitude, e.g., for a LEON-3 embedded processor, ACSEM requires only 54 seconds to perform a soft error masking analysis with a maximum masking probability of > 0.999, while fully-exploring the netlist requires > 3 weeks.

Open-Tool Release: We will make our soft error masking analysis tools available for download at http://ces.itec.kit.edu/download/.

II. Background: Circuit-Level Logical Masking

In order to illustrate how a bit flip can be logically masked at the circuit level, we present an example in Fig. 1 that shows an excerpt of a logic gate path. The paths is a chain of connected logic gates starting with an input gate $AND1$ and ending with an output gate $AND3$. As originally the two input signals of gate $AND1$ are 0 and 1, the final output of the path is 0. A particle strike happening at one of the inputs of gate $AND1$, or in some preceding logic gate whose output is fed into $AND1$ as an input, might result in a bit flip, e.g., from 0-to-1 (see Fig. 1). This error will be propagated at the output of the gate $AND1$. Now the erroneous output of $AND1$ will become one of the inputs of gate $AND2$. Afterwards, the error will further propagate to the input of gate $AND3$. Since its second input will determine the final output of the path, the erroneous input of gate $AND3$ will be masked once operated with other input 0. In case the second input has a value 1, the error will propagate to the output of gate $AND3$.

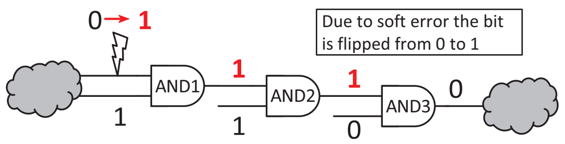


Fig. 1: An Example of gate-level logical masking in combinatorial circuits

III. ACSEM: Accuracy-Configurable Soft Error Masking Analysis

Fig. 2 shows the flow of our Accuracy-Configurable Soft Error Masking Analysis (ACSEM) scheme. It consists of three main components (explained in the subsequent sections)

1) *Masking Probability of Individual Gate Type* (Section III-A) analyzes the potential of a gate type to mask an erroneous input considering its functionality and input signal probabilities as a basis for the following masking probability calculation.
2) *Accuracy-Aware Step Calculation* (Section III-B) provides the maximum path depth to be extracted and analyzed to satisfy the user-provided accuracy range, reducing the analysis time.
3) *On-the-fly Path Extraction and Masking Probability Calculation* (Section III-C) extracts the logic gate paths of a circuit based on its netlist and simultaneously performs the masking probability calculation until the maximum step (Section III-B) is reached.

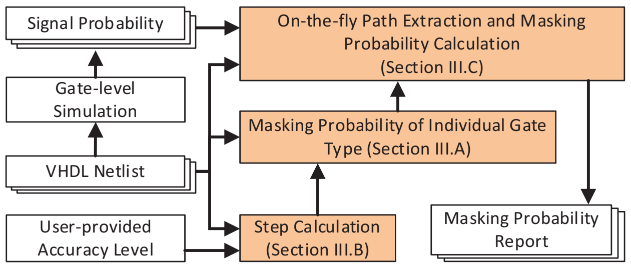

Signal Probability
Gate-level Simulation
VHDL Netlist
User-provided Accuracy Level
On-the-fly Path Extraction and Masking Probability Calculation (Section III.C)
Masking Probability of Individual Gate Type (Section III.A)
Step Calculation (Section III.B)
Masking Probability Report

Fig. 2: Overview of our Accuracy-Configurable Soft Error Masking Analysis (ACSEM) scheme

The key scientific challenge is how to find a maximum number of steps in the circuit netlist graph such that any soft error at this point or onwards may reach the circuit output signals with a probability more than the user-specific non-tolerable error range.

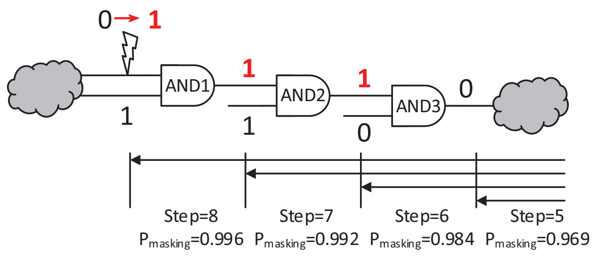

0→1
AND1
1
AND2
1
AND3
0
1
1
0
Step=8 $P_{masking}$=0.996
Step=7 $P_{masking}$=0.992
Step=6 $P_{masking}$=0.984
Step=5 $P_{masking}$=0.969

Fig. 3: Example circuits with different masking probabilities depending on the gate path length from the output

TABLE I: Masking probabilities for different *step* values for the circuit in Fig. 3

Step	1	2	3	4	5
$P_{Masking}$	0.5	0.75	0.875	0.9375	0.9688
Step	**6**	**7**	**8**	**9**	**10**
$P_{Masking}$	0.9844	0.9922	0.9961	0.9981	0.9990

Motivational Case Study: As shown in Section II, the effect of logical masking at gate level is important to be considered for an *accurate* analysis of the susceptibility of a circuit towards soft errors. As the basis of a *fast* analysis approach, two important observations have to be highlighted:

1) The logical masking property of a circuit depends on the *types of gates* it consists of. For instance, an AND gate has the ability to mask a bit flip as shown in Fig. 1 (with an average-case probability of 0.5). However, this property does not hold for all gate types, e.g., an XOR gate does not mask an error.
2) The potential of a bit flip to be masked depends on the *number of gates* with a masking potential that are passed before the final output of the circuit is reached.

As it can be seen in Fig. 3 and Table I for a circuit only consisting of AND gates, the masking probability increases significantly for the first *step* values and is already 0.9922% after a *step* representing 7 concatenated AND gates. A further analysis of the masking probability reveals that only a slight additional increase is observed for $step$ 8 (0.996%). However, as not all gates have the potential to mask errors and the masking of a gate also depends on its input, finding an appropriate value for $step$ that assures a sufficiently high masking while preventing from unnecessary analysis time is a challenge for different circuits. This can be avoided by considering that only the output of a circuit needs to be correct as internal signals are not exposed to other components. Therefore, our analysis starts from the leaf nodes and performs calculations in a backward fashion as only errors that propagate to the output and are visible in the final result affect the circuit reliability.

A. Calculating Masking Probabilities of Individual Gates

Model of the Circuit Netlist: The netlist graph of a combinatorial circuit is given as a set of $\mathscr{C} = (\mathscr{G}, \mathscr{L})$, where $\mathscr{G} = \{g_1, g_2, ..., g_{nG}\}$ is the set of logic gates connected with links given in the set $\mathscr{L} = \{l_1, l_2, ..., l_{nL}\}$. We consider logic gates with multiple inputs and single output, i.e., gate $g \in \mathscr{G}$ has a set of input signals $\mathscr{I}_g = \{I_{(g,1)}, I_{(g,2)}, ..., I_{(g,n)}\}$ and an output signal O_g. Each input/output signal has a particular value 0 or 1 that depends upon the functionality of the gate and an application‘s input data for the processor/accelerator. For gate $g \in \mathscr{G}$, we define $\mathscr{X}_g = \{x_{(g,1)}, x_{(g,2)}, ..., x_{(g,n)}\}$, s.t. $x_{(g,i)} \in \{0, 1\}$, as a set of input value combinations and $\mathscr{Y}_g = \{y_g\}$ as the potential output value combination, s.t. $y_g \in \{0, 1\}$.

Estimating Gate-Level Masking Probabilities: The masking probability of a gate highly depends upon the values of its input signals, which vary depending upon the data properties that the processor/circuit is processing. Therefore, first, we obtain signal probabilities (through gate-level simulations, using *ModelSim* in our case).

For an input $I_{(g,i)} \in \mathscr{I}_g$, the input signal probabilities are denoted as $P_{s0}(I_{(g,i)})$ and $P_{s1}(I_{(g,i)})$ for a signal value of 0 and 1, respectively. Similarly, signal probabilities for the gate output O_g are denoted as $P_{s0}(O_g)$ and $P_{s1}(O_g)$. For a gate g and a given input value combination $\mathscr{X}_g$, the output signal probability of O_g can be estimated as:

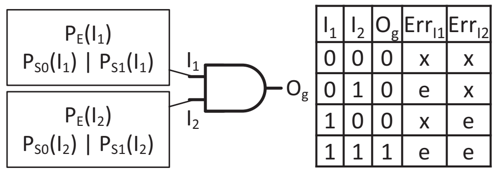
$$P_{sy_g}(O_g) = \prod_{i=0}^{n} P_{sx_{(g,i)}}(I_{(g,i)}), \quad x_{(g,i)} \in \mathscr{X}_g, y_g \in \mathscr{Y}_g \tag{1}$$

For $k \in [0, n^2 - 1]$ different input combinations of $\mathscr{X}_g$, the output O_j can be generated as an output combination $\mathscr{Y}_g = \{\mathscr{Y}_{(g,1)}, \mathscr{Y}_{(g,2)}, ..., \mathscr{Y}_{(g,k)}\}$.

Using these signal probabilities, we derive the individual soft error masking probabilities $p_M(g)$ for each gate $g \in \mathscr{G}$. In the following, we explain the procedure, how masking probability for an AND gate is derived[5].

An Example: Let us assume a gate g as a 2-input AND gate with inputs I_1 and I_2, and one output O_g. For both inputs, the input signal probability for signal 0 can be denoted as $P_{S0}(I_1)$ and $P_{S0}(I_2)$. And the input signal probability for signal 1 can be denoted as $P_{S1}(I_1)$ and $P_{S1}(I_2)$. For both inputs, the *error probability* can be represented as $p_e(I_1)$ and $p_e(I_2)$. The truth table of a 2-input AND gate is shown in Fig. 4. Err_{I_1} and Err_{I_2} denote that whether there will be an error occurring at output O_g, when one of the two inputs is faulty. The mark x denotes that when there is a single-bit error at this input, it would not affect the final output O_g. The mark e denotes that, if there is a single-bit error at this input, the final output O_g will also be faulty. For example, when $I_1 = 0$ and $I_2 = 0$ and in case a fault occurs on one of the inputs, the output O_g will not be affected. However, when $I_1 = 0$ and $I_2 = 1$, if I_1 is faulty and changes from 0 to 1, the output O_j will change from 0 to 1 as well, i.e. a fault appears at the gate output. For a 2-input AND gate there are total four faulty cases out

[5]detailed derivations for all other gate types are omitted due to space limitations



I_1	I_2	O_g	Err_{I1}	Err_{I2}
0	0	0	x	x
0	1	0	e	x
1	0	0	x	e
1	1	1	e	e

Fig. 4: Masking probability analysis of an AND gate

of eight cases. Therefore, the native (i.e., average-case) output error probability of a 2-input AND gate is given as $p_{e_{nat}}(O_g) = 0.5$. Depending upon the input signal probabilities, the final output error probability for a 2-input AND gate is represented as:

$$\begin{aligned} p_e(O_g) = 0.5 \times (P_{S0}(I_1) \times P_{S1}(I_2) + P_{S1}(I_1) \times P_{S0}(I_2)) \\ + P_{S1}(I_1) \times P_{S1}(I_2) \end{aligned} \tag{2}$$

And the masking probability is complementary with output error probability, so the masking probability of a 2-input AND gate can be represented as: $p_M(O_g) = 1 - p_e(O_g)$.

B. Calculating the Maximum Number of Steps

Based on the masking probabilities of individual gates, we derive the maximum number of steps S_{MAX} to be traversed in the circuit graph, such that the error probability p_e becomes equal to or greater than the user-provided (i.e., system designer) masking/accuracy range (p_{mR}). For instance, a system designer is given with a target system reliability to tolerate errors with a probability of 0.05, since ensuring a 100% safe operation (i.e., $p_E = 0$) may be too costly in terms of area/power; in this case, the $p_{mR} = 0.95$.

As discussed above, certain gate types (e.g., AND, OR gates) have the inherent ability to mask errors, while for others (e.g., XOR gates) an error always propagates to its output. Furthermore, an error can affect several successors in case the affected gate output is used multiple times (i.e., an error is propagated to multiple dependent gates). It is assumed that an error can occur at any possible point within the circuit. For a circuit with n_G total gates having n_{OR} number of OR gates and n_{AND} number of AND gates, the masking probability p_M is estimated as:

$$p_M = \Pr\left[Mask\, in\, 1\, Step\right] = 0.5 \times \frac{n_{OR} + n_{AND}}{n_G} \tag{3}$$

Therefore, the probability of an error being masked in N steps in the combinatorial circuit is derived as follows:

$$\Pr\left[Mask\, in\, S\, Steps\right] = \sum_{i=0}^{S-1} p_M \times (1 - p_M)^i = 1 - (1 - p_M)^S \tag{4}$$

In order to take into account that one error may affect multiple outputs, the *expected number of errors* arising from a single fault is given below and it depends upon the (expected) number of times that an output of a gate is reused.

$$p_E = E\left[Error\, Propagates\right] = \frac{Output\, reuse}{Total\, ouputs\, used} = \frac{n_I}{2 \times n_O} \tag{5}$$

n_I and n_O are the *total* number of inputs and outputs in all gates of the complete circuit without considering the ones from non-masking gates. The factor 2 considers that every gate (AND/OR) has 2 inputs and 1 output.

Combining Eq. 4 and 5, the expected number of errors after s steps is obtained, which represents the amount of errors present at a distance of N gates from the point where the error occurred.

$$E\left[errors\, at\, s\, steps\right] = \left(\frac{1 - p_M}{p_E}\right)^s \tag{6}$$

This represents the expected number of erroneous values after N steps from the point where the error occurred. It can be > 1 which means that in average, an error will not be masked, rather, it is likely to propagate. If the expected number of errors is < 1, the error will probably be masked. Based on Eq. 6, the average number of steps after which an error will be masked can be calculated by:

$$E\left[Steps\,til\,Mask\right] = \sum_{s=0}^{N-1} s \times \left(\frac{1-p_M}{p_E}\right)^s = S1_N\left(\frac{1-p_M}{p_E}\right) \quad (7)$$

The function $S1_N$ is given as[6]:

$$S1_N(a) = \sum_{s=0}^{N-1} s * a^s = a * \sum_{s=1}^{N-1} s * a^{s-1} + 1 = \frac{a}{(1-a)^2} + 1 \quad (8)$$

Since only the errors visible at the output matter, N is given as the maximum number of gates between the point where the error occurred and the output.

The variance is computed as follows:

$$\begin{aligned} Var\left[Steps\,until\,masking\right] &= \sum_{s=0}^{N-1} s^2 * E\left[Errors\,at\,s\,Steps\right] \\ &= \sum_{s=0}^{N-1} s^2 * \left(\frac{1-p_M}{p_E}\right)^s = S2_N\left(\frac{1-p_M}{p_E}\right) \end{aligned} \quad (9)$$

The function $S2_N$ is given as[7]:

$$S2_N(a) = \sum_{s=0}^{N-1} s^2 * a^s = \frac{a}{(1-a)^2} + 1 - \frac{2*a}{(1-a)^4} \quad (10)$$

Given the variance and the average number of steps until an error is masked, the Chebyshev theorem (see Eq. 11) can be used to obtain the maximum number of steps (S_{MAX}) needed to mask an error with a user-defined error probability $p_{eR} = 1 - p_{mR}$. σ is the standard deviation which can be derived from the variance and K is a constant.

$$\Pr[|A - E[A]| \geq K \times \sigma] \leq \frac{1}{K^2} = p_{eR} \quad (11)$$

$$K = \frac{1}{\sqrt{p_{eR}}} \Rightarrow \Pr\left[S \geq \frac{1}{\sqrt{p_{eR}}} \times \sqrt{Var[S]} + E[S]\right] \leq p_{eR} \quad (12)$$

where S is the number of steps or levels of gates until an error is masked. By substitution we obtain

$$S_{MAX} \geq \sqrt{\frac{S2_N\left(1 - {p_M}/{p_E}\right)}{p_{eR}}} + S1_N\left(\frac{1-p_M}{p_E}\right) \quad (13)$$

This implies that, an error at more than S_{MAX}) steps in the circuit netlist graph from the output has a probability smaller than p_{eR} to be visible at the output, while any error within the range may propagate. In results (Section IV-B), we will show that S_{MAX}) is several cases is around below 10 steps to achieve a $p_{eR} = 0.001$, which helps in significantly speeding up large-sized circuits where the total number of steps may exceed 100 or so.

C. Masking Probability Extraction

The algorithm 1 presents the procedure for on-the-fly logic gate path extraction and (soft-error) masking probability computation under the constraint of maximum step size (S_{MAX}) as computed in Section III-B. The algorithm first estimates the masking probabilities for all individual gates using the method described in Section III-A (line 2). Afterwards, it employs a *Backward Depth-First Search* technique (lines 7-38) the extracts the logic gate paths and estimates the masking probabilities in a backward fashion, i.e., starting from the leaf nodes (i.e., circuit outputs) first and moving towards the root nodes (i.e., circuit inputs). The algorithm then inserts one of the leaf nodes into a stack data structure St and uses another stack Mt to keep track of the masking probability of the currently traversed path. Afterwards, it checks whether the current leaf node has been fully traversed or not (lines 8-13). If yes and if there as still other lead nodes available in the circuit to be analyzed, the next leaf node is inserted into the stack St and the *Backward Depth-First Search* is performed again. Afterwards, the algorithm loops through all input edges of the current top element of stack St for calculating the masking probabilities (line 17-38). The traversed edge are marked as “deactivated”, so the algorithm will not traverse the same edge again. A loop path check mechanism is implemented to prevent a cyclic graph generating endless traversals (line 20). If the path reaches the maximum step size (S_{MAX}), the search for the path under consideration stops, the current partial path is saved, and the algorithm continues searching other paths linked to the leaf node under consideration, unless all the paths linked to the current leaf node has been processed. Finally, the masking probabilities

Algorithm 1 Masking probability calculation

```
Input: Circuit netlist $\mathscr{C} = (\mathscr{G}, \mathscr{L})$ with a set of gates as nodes ($\mathscr{G}$) and connections as edged ($\mathscr{L}$), a set of inputs of all gates ($I_g, \forall g \in \mathscr{G}$)set $I$ of inputs, signal probability of all gates ($p_S = [P_{s0}(I_g), P_{s1}(I_g)], \forall g \in \mathscr{G}$), maximum step size computed in Section III-B ($S_{MAX}$).
Output: set $\mathscr{P}$ of paths with corresponding masking probabilities $p_M(P_k), \forall P_k \in \mathscr{P}$.
1: for $g \in \mathscr{G}$ do                    ▷ calc. masking probability for all gates
2:   $g.p_M = calculateGateMasking(g.p_S)$;
3: $o = O.top$;
4: $St, Mt$;
5: $St.push(o); Mt.push(o.p_M)$;
6: $O.pop$;
7: while $!O.empty$ do        ▷ loop until all output nodes are processed
8:   if $St.empty$ then
9:     if $O.empty$ then
10:      $o = O.top; St.push(o); O.pop; top = St.top$;
11:      $Mt.clear; Mt.push(o.p_M)$;
12:    else
13:      $break$;
14: $top = S.top$;
15: if $!top.l$ then
16:   $St.top; Mt.pop$;
17: while $top.l$ do                    ▷ loop for all input edges of top
18:   if $top.l.active$ then
19:     $top.l.active = false$;
20:     if $!FindPathLoop(St, top.predecessor)$ then
21:       if $St.size + 1 == S_{MAX}$ then        ▷ path length reaches step
22:         $Mt.push(calculatePathMasking(Mt.back, top.p_M))$;
23:         $St.push(top.predecessor)$;
24:         $outputPathMasking(St, Mt.back)$;             ▷ output path
25:         $St.pop; Mt.pop; top.l.next$;         ▷ move to next input edge
26:       else if $find(I, top.predecessor)$ then
27:         $outputPathMasking(St, Mt.back); top.l.next$;
28:       else
29:         $Mt.push(calculatePathMasking(Mt.back; top.p_M)$;
30:         $St.push(top.predecessor); break$;
31:     else
32:       $top.l.next$;
33:   else
34:     $top.l.next$;
35:     while $top.l$ && $top.l.active$ do              ▷ get next valid edge
36:       $top.l.next$;
37:     if $!top.l.next$ then
38:       $\mathscr{C}.resetAllEdgesOf(top); St.pop; Mt.pop; break$;
```

[6]The complete derivation is omitted due to space limitations

[7]The complete derivation is omitted due to space limitations

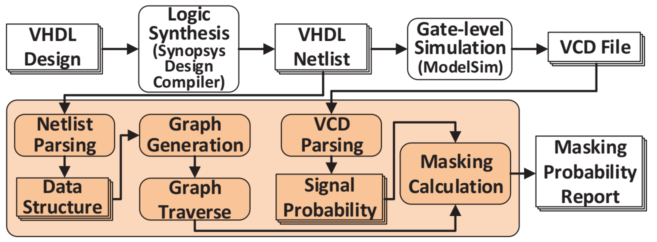


Fig. 5: Tool flow for logic circuit masking calculation

TABLE II: Summary of Synthesized Circuits

	# of gates	# of paths	# of nodes	# of edges	# of gate types
quadsub	98	3.6×10^{2}	168	236	6
sav	58	1.6×10^{4}	194	234	8
collapseadd	77	1.6×10^{4}	157	284	8
transform	514	6.5×10^{4}	594	1330	18
pointfilter	883	1.2×10^{8}	963	2321	48
lfbs4	333	4.4×10^{5}	413	960	23
clip3	791	2.3×10^{6}	871	2069	29
c7552	3512	6.8×10^{5}	3827	6046	14
Leon-3	4203	$> 10^{10}$	4544	12826	124

for each extracted path are returned.

IV. Results

A. Experimental Setup

In order to acquire the masking probability of different gate paths of a logic circuit, we use the tool flow illustrated in Fig. 5. First, the target design, which is given as a VHDL behavioral description, is synthesized by a logic synthesis tool. In our case, the Design Compiler from Synopsys is used which takes the VHDL Design, a technology library (TSMC 45nm Standard-Cell Library) and synthesis constraints (set_max_area=0) as inputs. The operating condition is thereby set to *worst case*, the operating voltage is 0.81V, the operating temperature is 125°C and the process corner used is *SS*. The logic synthesis generates a corresponding netlist file, which can be used to execute a gate-level simulation. With proper inputs, a Value Change Dump (VCD) file, which contains all signal changes, can be generated. Both synthesis netlist file and VCD file are inputs to our ACSEM. It uses its netlist parser to parse the netlist and generates a corresponding data structure, which stores all the necessary information in the netlist file such as entities, components, gates and interconnections. Those information is then used to generate a directed graph, in which the nodes represent the gates and input/output ports, and the edges represent the interconnections. In the meantime, ACSEM parses the VCD file, and calculates the signal probability for all signals. A Backward Depth First Search (BDFS) algorithm is applied to the generated graph to obtain all valid paths from an output node of the graph until all output nodes are traversed. During the path extraction process, the backward masking calculation is employed, which calculates the masking probability step by step. After all output nodes in the graph have been traversed, the algorithm terminates, and writes the masking probability report to a file.

B. Masking Analysis of Different Circuits

For evaluation, circuits (several hardware accelerators and a larger circuit from the ISCAS'89 benchmark suite) with different functionalities and complexities (e.g., in terms of number of gates) ranging from *sav* with 58 gates to *c7552* with 3512 gates are analyzed.

Signal probabilities changing with input: As pointed out in Section III-A the signal probabilities change depending upon the input values. Fig. 6 shows the distribution of signal probabilities of the

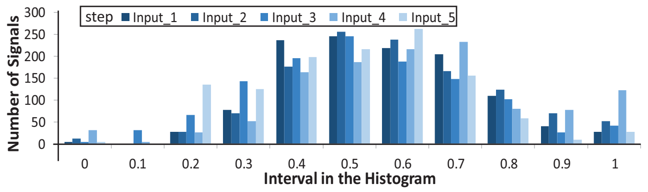


Fig. 6: Signal probabilities for different inputs for circuit *quadsub*

TABLE III: Maximum masking probability for circuit *quadsub* for different input sets at different step values

	Input1	Input2	Input3	Input4	Input5
5	0.611104	0.603385	0.722303	0.722201	0.688395
6	0.726369	0.705812	0.775312	0.846211	0.873076
7	0.790742	0.800997	0.841413	0.896351	0.924773
8	0.844847	0.899130	0.878285	0.959686	0.935928
9	0.881365	0.927786	0.908335	0.969143	0.950168
10	0.920309	0.946514	0.932034	0.978188	0.969303

example circuit *quadsub* for different input vectors. In total 5 different input sets each consisting of different input vectors are applied to the circuit and the signal probabilities of all its gates are recorded. Afterwards, the individual signal probabilities are grouped into 11 different classes. The x-axis shows the interval range $[0, 1]$, which indicates the actual signal probability (note that 0 and 1 are reported separately). The results illustrate that the signal probabilities vary depending on the input value set provided even for the same circuit.

The changing signal probabilities for different input sets directly translate to a difference in the maximum masking probability shown in Table III. In order to avoid an overestimation of the masking probability of the circuit the minimum value for the different input sets is considered as the final masking probability of the circuit.

Speedup: As one of the main contributions of ACSEM is the reduction in the analysis time, we analyzed different circuits for different *step* values and calculated the speedup compared to the full netlist. The speed up is therefore calculated as:

$$Speedup = \frac{Execution\ time\ without\ step}{Execution\ time\ with\ step} \tag{14}$$

As shown in Fig 7(a), the speed up of the algorithm decreases when the step value increases as longer gate paths have to be extracted and analyzed. When the circuit is small enough the speed up value is even close to 1 as the step value increases.

For example, the *quadsub* circuit has only 98 gates and total 360 paths are extracted from the graph, which is the lowest one among all tested circuits. So the speedup of ACSEM on *quadsub* is quite low, since the step value is very close to or even larger than the average length of paths in the circuit graph. Therefore, the BDFS algorithm with the step mechanism will be degraded to a full BDFS algorithm. On the other hand, the *pointfilter* circuit has the highest number of paths, and also the highest speedup among all circuits. The *pointfilter* circuit has a medium number of gates compared to the c7552 benchmark circuit, which has the largest number of gates and interconnections and nodes in the graph. However, the *pointfilter* circuit has a larger number of total paths extracted from the graph, which means that *pointfilter* circuit is more complex than the c7552 circuit. The step mechanism successfully prevented the BDFS algorithm from going further when the current precision of calculation for masking probability has already reached user's requirement, which saves a lot of computation time.

Masking Calculation Time and Total execution time: Fig. 7(b) and Fig. 7(c) present the masking calculation time and the total execution time of ACSEM, respectively. The evaluation shows that

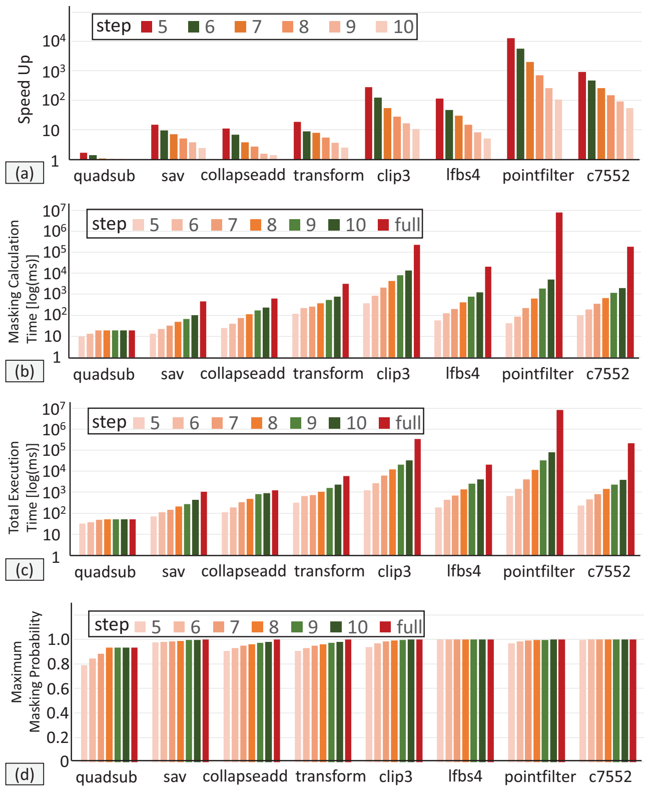


Fig. 7: Evaluation of Accuracy Configurability with Different Derived Steps: (a) Speedup; (b) Masking Calculation Time; (c) Total Execution of the Analysis; (d) Maximum Masking Probability at S_{MAX}

our algorithm is more efficient when the circuit complexity is high. For a small and simple circuit, for instance, the *quadsub* circuit, for step value larger than 8, the masking calculation time and total execution time barely changes. Note, that the value difference can be considered as deviation that is caused by operating system or IO operation. As the complexity of the circuit grows, the step mechanism becomes more profitable, which saves a significant amount of time. The differences between total execution time and masking calculation time are mainly due to path extraction and other IO operations.

Maximum Masking Probability: The maximum masking probability until the output is plotted in Fig. 7(d). It is visible that the masking probability does not have a strong correlation with the complexity of the circuit, since the masking probability of one gate is only related to the type of the gate and the input signal probability. For example, as discussed above, the *pointfilter* circuit has the highest complexity among all circuits. However, the *lfbs4* and *c7552* circuit reach a very high output masking probability even at step 5, which is even higher than *pointfilter*. The reason for this phenomenon is that there are many gates which have a high masking capability. Those gates reached a high output masking probability, which increases the total output masking probabilities. For small and simple circuits, for example *quadsub*, the maximum output masking probability is limited by its average length of paths.

V. Conclusion

In this paper, a fast Accuracy-Configurable Soft Error Masking analysis (ACSEM) technique is presented. For a user-defined accuracy level, we theoretically derive the maximum number of steps that need to be traversed in the circuit graph, such that the error masking probability requirements corresponding to the accuracy level are met. It thereby significantly curtails the analysis time, especially for large-sized circuits. Additionally, an algorithm for extracting, parsing and analyzing the circuit netlist is proposed. We evaluate our technique for various circuits including complex hardware accelerators, ISCAS‘89, and a full LEON3 processor. Our approach achieves analysis speed of orders of magnitude, thus enabling fast reliability analysis for complex circuits under varying workloads that may contribute towards avoiding over-design and shortening the time-to-market.

Acknowledgments

This work is supported in parts by the German Research Foundation (DFG) as part of the priority program ”Dependable Embedded Systems” (SPP 1500 - spp1500.itec.kit.edu).

References

[1] Robert C. Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Transactions on Device and Materials Reliability*, 5(3):305–316, Sept 2005.

[2] Jörg Henkel, Lars Bauer, Nikil Dutt, Puneet Gupta, Sani Nassif, Muhammad Shafique, Mehdi Tahoori, and Norbert Wehn. Reliable on-chip systems in the nano-era: Lessons learnt and future trends. In *DAC*, 2013.

[3] S.S. Mukherjee, J. Emer, and S.K. Reinhardt. The soft error problem: an architectural perspective. In *International Symposium on High-Performance Computer Architecture (HPCA)*, pages 243–247, 2005.

[4] N. J. Wang, J. Quek, T. M. Rafacz, and S. J. Patel. Characterizing the effects of transient faults on a high-performance processor pipeline. In *Intl. Conference on Dependable Systems and Networks*, DSN, 2004.

[5] S. Rehman, M. Shafique, F. Kriebel, and J. Henkel. Reliable software for unreliable hardware: embedded code generation aiming at reliability. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 237–246, 2011.

[6] L. Entrena, M. Garcia-Valderas, R. Fernandez-Cardenal, A Lindoso, M. Portela, and C. Lopez-Ongil. Soft error sensitivity evaluation of microprocessors by multilevel emulation-based fault injection. *Computers, IEEE Transactions on*, 61(3):313–322, March 2012.

[7] Xiaodong Li, S.V. Adve, P. Bose, and J.A Rivers. Softarch: an architecture-level tool for modeling and analyzing soft errors. In *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, pages 496–505, June 2005.

[8] Feng Wang and Yuan Xie. Soft error rate analysis for combinational logic using an accurate electrical masking model. *IEEE Transactions on Dependable and Secure Computing*, 8(1):137–146, 2011.

[9] Liang Chen, Mojtaba Ebrahimi, and Mehdi Baradaran Tahoori. Cep: Correlated error propagation for hierarchical soft error analysis. *J. Electronic Testing*, 29(2):143–158, 2013.

[10] T. Takata and Y. Matsunaga. A robust algorithm for pessimistic analysis of logic masking effects in combinational circuits. In *IEEE International On-Line Testing Symposium (IOLTS)*, pages 246–251, 2011.

[11] N. Miskov-Zivanov and D. Marculescu. Formal modeling and reasoning for reliability analysis. In *Design Automation Conference*, DAC, pages 531–536, 2010.

[12] R. Rajaraman, Jungsub Kim, Narayanan Vijaykrishnan, Yuan Xie, and Mary Jane Irwin. Seat-la: A soft error analysis tool for combinational logic. In *VLSI Design*, pages 499–502. IEEE Computer Society, 2006.

[13] J.-C. Baraza, J. Gracia, S. Blanc, D. Gil, and P.-J. Gil. Enhancement of fault injection techniques based on the modification of vhdl code. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 16(6):693–706, June 2008.

[14] R.R. Rao, K. Chopra, D. Blaauw, and D Sylvester. An efficient static algorithm for computing the soft error rates of combinational circuits. In *Design, Automation and Test in Europe, 2006. DATE ’06. Proceedings*, volume 1, pages 1–6, March 2006.

[15] Bin Zhang, Wei-Shen Wang, and M. Orshansky. Faser: fast analysis of soft error susceptibility for cell-based designs. In *Quality Electronic Design, 2006. ISQED ’06. 7th International Symposium on*, pages 6 pp.–760, March 2006.

[16] M. Ebrahimi, Liang Chen, H. Asadi, and M.B. Tahoori. Class: Combined logic and architectural soft error sensitivity analysis. In *Design Automation Conference (ASP-DAC), 2013 18th Asia and South Pacific*, pages 601–607, Jan 2013.

[17] Bin Zhang and Michael Orshansky. Symbolic simulation of the propagation and filtering of transient faulty pulses. In *Workshop on Silicon Errors in Logic-System Effects*, 2005.

[18] Natasa Miskov-Zivanov and Diana Marculescu. Circuit reliability analysis using symbolic techniques. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 25(12):2638–2649, 2006.