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 concerning 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 [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. 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 require 2 – 3 weeks for performing the soft-error masking analysis for the LEON-3 processor netlist 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 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 specifies a target system reliability level (e.g., mean-time-to-failure as the device error rate) with a probability of

1There 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].

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

3Synthesized 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

4For 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.ietc.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 path 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.

**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.

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.

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

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

<table>
<thead>
<tr>
<th>Step</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step Calculation</td>
<td>0.5</td>
<td>0.75</td>
<td>0.875</td>
<td>0.9375</td>
<td>0.9688</td>
<td>0.9844</td>
<td>0.9922</td>
<td>0.9961</td>
<td>0.9981</td>
<td>0.9990</td>
</tr>
</tbody>
</table>

**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 (0.096%). 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 combinational circuit is given as a set of \( G = (\mathcal{G}, \mathcal{L}) \), where \( \mathcal{G} = \{ g_1, g_2, \ldots, g_n \} \) is the set of logic gates connected with links given in the set \( \mathcal{L} = \{ l_1, l_2, \ldots, l_m \} \). We consider logic gates with multiple inputs and single output, i.e., gate \( g \in \mathcal{G} \) has a set of input signals \( \mathcal{I}_g = \{ I_{g_1}, I_{g_2}, \ldots, I_{g_n} \} \) 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 \( g \in \mathcal{G} \), we define \( \mathcal{X}_g = \{ x_{g_1}, x_{g_2}, \ldots, x_{g_n} \} \), s.t. \( x_{g_i} \in \{ 0, 1 \} \), as a set of input value combinations and \( \mathcal{Y}_g = \{ y_o \} \) as the potential output value combination, s.t. \( y_o \in \{ 0, 1 \} \).

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 \( I_{g_i}(g) \in \mathcal{G} \), the input signal probabilities are denoted as \( P_{\text{in}}(I_{g_i}(g)) \) and \( P_{\text{in}}(I_{g_i}(g)) \) for a signal value of 0 and 1, respectively. Similarly, signal probabilities for the gate output \( O_g \) are denoted as \( P_{\text{out}}(O_g) \) and \( P_{\text{out}}(O_g) \). For a gate \( g \) and a given input value combination \( \mathcal{X}_g \), the output signal probability of \( O_g \) can be estimated as:

\[
P_{\text{out}}(O_g) = \prod_{i=0}^{n} P_{\text{in}}(x_{g_i}(g)) \quad \text{s.t.} \quad x_{g_i} \in \mathcal{X}_g, y_o \in \mathcal{Y}_g \tag{1}
\]

For \( k \in [0, n^2 - 1] \) different input combinations of \( \mathcal{X}_g \), the output \( O_g \) can be generated as an output combination \( \mathcal{Y}_g = \{ y_{g_1}, y_{g_2}, \ldots, y_{g_k} \} \).

Using these signal probabilities, we derive the individual soft error masking probabilities \( p_{\text{mask}}(g) \) for each gate \( g \in \mathcal{G} \). In the following, we explain the procedure, how masking probability for an AND gate is derived:

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 \( P_{\text{in}}(I_1) \) and \( P_{\text{in}}(I_2) \). And the input signal probability for signal 1 can be denoted as \( P_{\text{in}}(I_1) \) and \( P_{\text{in}}(I_2) \). 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. \( \text{Err}_{I_1} \) and \( \text{Err}_{I_2} \) 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 \( O_g \) 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 of eight cases. Therefore, the native (i.e., average-case) output error probability of a 2-input AND gate is given as \( p_{\text{mask}}(O_g) = 0.5 \). Depending upon the input signal probabilities, the final output error probability for a 2-input AND gate is represented as:

\[
p_e(O_g) = 0.5 \times (P_{\text{in}}(I_1) \times P_{\text{in}}(I_2) + P_{\text{in}}(I_1) \times P_{\text{in}}(I_2)) + P_{\text{in}}(I_1) \times P_{\text{in}}(I_2) \tag{2}
\]

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 probability of individual gates, we derive the maximum number of steps \( S_{\text{MAX}} \) 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 \( p_{\text{mask}} \). 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 \( p_{\text{mask}} = 0.05 \).

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:

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

Therefore, the probability of an error being masked in \( N \) steps in the combinational circuit is derived as follows:

\[
Pr[\text{Mask in S Steps}] = \sum_{i=0}^{N-1} p_M \times (1 - p_M)^i = 1 - (1 - p_M)^N \tag{4}
\]

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.

\[
p_E = E[\text{Error Propagates}] = \frac{\text{Output reuse}}{\text{Total outputs used}} = \frac{n_I}{2 \times n_O} \tag{5}
\]

\( n_I \) 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.

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

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:

\[
E[\text{Steps till Mask}] = \sum_{s=0}^{N-1} s \times \left(1 - \frac{P_M}{PE}\right)^s = S1N \left(1 - \frac{P_M}{PE}\right)
\]  (7)

The function \(S1N\) is given as:

\[
S1N(a) = \sum_{s=0}^{N-1} s \times a^s = a \sum_{s=0}^{N-1} s \times a^{s-1} + 1 = \frac{a}{1-a^2} + 1\]  (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:

\[
Var[\text{Steps until masking}] = \sum_{s=0}^{N-1} s^2 \times E[\text{Errors at } s \text{ Steps}]
\]  (9)

\[
= \sum_{s=0}^{N-1} s^2 \times \left(1 - \frac{P_M}{PE}\right)^s = S2N \left(1 - \frac{P_M}{PE}\right)
\]

The function \(S2N\) is given as:

\[
S2N(a) = \sum_{s=0}^{N-1} s^2 \times a^s = \frac{a}{(a-1)^2} + \frac{2a}{1-a^2}\]  (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 \(p_{R} = 1 - P_{M}.\) \(\sigma\) is the standard deviation which can be derived from the variance and \(K\) is a constant.

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

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

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

\[
S_{MAX} \geq \sqrt{\frac{S2N\left(1 - \frac{P_M}{PE}\right)}{p_R}} + S1N \left(1 - \frac{P_M}{PE}\right)
\]  (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 \(p_R\) 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 \(p_{R} = 0.001\), 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

\[6\text{The complete derivation is omitted due to space limitations}\]

\[7\text{The complete derivation is omitted due to space limitations}\]

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

\[\text{Algorithm 1 Masking probability calculation}\]

\[\text{Input: Circuit netlist } C = (\mathcal{G}, \mathcal{E}) \text{ with a set of gates as nodes } (\mathcal{G}) \text{ and connections as edges } (\mathcal{E}), \text{ a set of inputs of all gates } (I_g, \forall g \in \mathcal{G}) \text{ set } I \text{ of inputs, signal probability of all gates } (p_S = [P_{o0}(I), P_{o1}(I)], \forall g \in \mathcal{G}), \text{ maximum step size computed in Section III-B } (S_{MAX}).\]

\[\text{Output: set } \mathcal{P} \text{ of paths with corresponding masking probabilities } P_M(P), \forall P \in \mathcal{P}.\]

1: for \(g \in \mathcal{G}\) do \(\triangleright\) calc. masking probability for all gates
2: \(\triangleright\) set \(\mathcal{P} = \{\}\)
3: \(\triangleright\) set \(\mathcal{P} = \{\}\)
4: \(\triangleright\) set \(\mathcal{P} = \{\}\)
5: \(\triangleright\) set \(\mathcal{P} = \{\}\)
6: \(\triangleright\) set \(\mathcal{P} = \{\}\)
7: while \(\text{true}\) do \(\triangleright\) loop until all output nodes are processed
8: \(\triangleright\) set \(\mathcal{P} = \{\}\)
9: \(\triangleright\) set \(\mathcal{P} = \{\}\)
10: for \(i = 0; i < \text{top.Lactive}\) do \(\triangleright\) loop for all input edges of top
11: \(\triangleright\) set \(\mathcal{P} = \{\}\)
12: \(\triangleright\) set \(\mathcal{P} = \{\}\)
13: \(\triangleright\) set \(\mathcal{P} = \{\}\)
14: \(\triangleright\) set \(\mathcal{P} = \{\}\)
15: \(\triangleright\) set \(\mathcal{P} = \{\}\)
16: \(\triangleright\) set \(\mathcal{P} = \{\}\)
17: while \(\text{true}\) do \(\triangleright\) loop for all input edges of top
18: \(\triangleright\) set \(\mathcal{P} = \{\}\)
19: \(\triangleright\) set \(\mathcal{P} = \{\}\)
20: \(\triangleright\) set \(\mathcal{P} = \{\}\)
21: \(\triangleright\) set \(\mathcal{P} = \{\}\)
22: \(\triangleright\) set \(\mathcal{P} = \{\}\)
23: \(\triangleright\) set \(\mathcal{P} = \{\}\)
24: \(\triangleright\) set \(\mathcal{P} = \{\}\)
25: \(\triangleright\) set \(\mathcal{P} = \{\}\)
26: \(\triangleright\) set \(\mathcal{P} = \{\}\)
27: \(\triangleright\) set \(\mathcal{P} = \{\}\)
28: \(\triangleright\) set \(\mathcal{P} = \{\}\)
29: \(\triangleright\) set \(\mathcal{P} = \{\}\)
30: \(\triangleright\) set \(\mathcal{P} = \{\}\)
31: \(\triangleright\) set \(\mathcal{P} = \{\}\)
32: \(\triangleright\) set \(\mathcal{P} = \{\}\)
33: \(\triangleright\) set \(\mathcal{P} = \{\}\)
34: \(\triangleright\) set \(\mathcal{P} = \{\}\)
35: \(\triangleright\) set \(\mathcal{P} = \{\}\)
36: \(\triangleright\) set \(\mathcal{P} = \{\}\)
37: \(\triangleright\) set \(\mathcal{P} = \{\}\)
38: \(\triangleright\) set \(\mathcal{P} = \{\}\)

2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)
TABLE II: Summary of Synthesized Circuits

<table>
<thead>
<tr>
<th>Circuit</th>
<th># of gates</th>
<th># of paths</th>
<th># of nodes</th>
<th># of edges</th>
<th># of gate types</th>
</tr>
</thead>
<tbody>
<tr>
<td>quadsub</td>
<td>98</td>
<td>$3.6 \times 10^4$</td>
<td>108</td>
<td>236</td>
<td>6</td>
</tr>
<tr>
<td>sav</td>
<td>58</td>
<td>$1.6 \times 10^4$</td>
<td>194</td>
<td>234</td>
<td>8</td>
</tr>
<tr>
<td>collapse</td>
<td>77</td>
<td>$1.6 \times 10^4$</td>
<td>157</td>
<td>284</td>
<td>8</td>
</tr>
<tr>
<td>transform</td>
<td>514</td>
<td>$6.5 \times 10^3$</td>
<td>594</td>
<td>1330</td>
<td>18</td>
</tr>
<tr>
<td>pointfilter</td>
<td>883</td>
<td>$1.2 \times 10^3$</td>
<td>963</td>
<td>2321</td>
<td>48</td>
</tr>
<tr>
<td>bfs4</td>
<td>333</td>
<td>$4.4 \times 10^3$</td>
<td>413</td>
<td>960</td>
<td>23</td>
</tr>
<tr>
<td>clip5</td>
<td>791</td>
<td>$2.3 \times 10^3$</td>
<td>871</td>
<td>2069</td>
<td>29</td>
</tr>
<tr>
<td>c7552</td>
<td>3512</td>
<td>$6.8 \times 10^2$</td>
<td>3827</td>
<td>6046</td>
<td>14</td>
</tr>
<tr>
<td>Leon-3</td>
<td>4205</td>
<td>$&gt; 10^3$</td>
<td>4544</td>
<td>12826</td>
<td>124</td>
</tr>
</tbody>
</table>

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 hereby 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 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:

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

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 BDS algorithm with the step mechanism will be degraded to a full BDS 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 BDS 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
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 lfs84 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