IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 31, 2022, accepted April 23, 2022, date of publication May 12, 2022, date of current version May 19, 2022.

*Digital Object Identifier 10.1109/ACCESS.2022.3174564*

# Fast Error Propagation Probability Estimates by Answer Set Programming and Approximate Model Counting

**ANSELM BREITENREITER[1], MARKO ANDJELKOVIĆ[1], OLIVER SCHRAPE[1], AND MILOŠ KRSTIĆ[1,2]**

[1]IHP—Leibniz-Institut für innovative Mikroelektronik, 15236 Frankfurt (Oder), Germany
[2]Department of Design and Test Methodology, University of Potsdam, 14469 Potsdam, Germany

Corresponding author: Anselm Breitenreiter (breitenreiter@ihp-microelectronics.com)

This work was supported by the German Federal Ministry of Education and Research under Project 16ME0122K-16ME0140+16ME0465 (Scale4Edge).

**ABSTRACT** We present a method employing Answer Set Programming in combination with Approximate Model Counting for fast and accurate calculation of error propagation probabilities in digital circuits. By an efficient problem encoding, we achieve an input data format similar to a Verilog netlist so that extensive preprocessing is avoided. By a tight interconnection of our application with the underlying solver, we avoid iterating over fault sites and reduce calls to the solver. Several circuits were analyzed with varying numbers of considered cycles and different degrees of approximation. Our experiments show, that the runtime can be reduced by approximation by a factor of 91, whereas the error compared to the exact result is below 1 %.

**INDEX TERMS** Answer set programming, approximate model counting, error propagation, radhard design, reliability analysis, selective fault tolerance, single event upsets.

## I. INTRODUCTION

Microelectronic systems have pervaded every aspect of our life and although many of them are safety critical, resource usage is often strictly limited. Satellites for instance have a very tight power budget, because the energy consumption directly translates into the size of the solar panels. In terrestrial applications, autonomous driving tremendously increases the performance requirements for on-board signal processing, whereas the energy consumption directly affects the range of electric cars. In medical products like wearables or implantables, both energy and area usage are limited. In situations like these, protection of all circuit elements against soft errors is often not possible. For this reason, the design needs to be thoroughly analyzed to identify the elements, which have the highest contribution to the overall soft error rate (SER) and where protection provides most effective results. Fault tolerance, where the system is only protected against a subset of all possible faults, is commonly called *selective fault tolerance*. The challenge of such an approach is to efficiently select circuit elements for protection.

The associate editor coordinating the review of this manuscript and approving it for publication was Qi Zhou.

Fault sites i.e. the circuit elements where a fault occurs can be characterized by their error propagation probability (EPP). This is the probability, that a fault propagates and manifests as an error at the primary outputs. There are electrical, temporal and logic masking effects, which could influence the fault propagation. Electrical and temporal masking effects are mainly determined by technology and circuit parameters. The proposed method is implemented on the gate-level netlist and the fault model is ''single event upsets (SEUs) in flip-flops''. SEUs are bit-flips caused by high-energy particles. In this context we consider logic masking effects as the most relevant for this approach.

There are three main classes of methods for the calculation of EPP based on logic masking effects: methods based on fault-injection simulation, analytic methods and methods based on solving combinatorial search problems.

Fault-injection simulation [1]–[4] might be the most obvious approach. A high number of simulations is required to cover all fault sites, circuit states and input sequences.

There are analytic methods, which propagate and accumulate signal probabilities along the gate-level netlist [5]–[7]. They rely on procedural algorithms, which comprise extensive preparation of input data, repetitive traversals of data

structures and computationally intensive arithmetic operations. They are only applicable to circuits with a few hundred flip-flops. Some methods rely on structural netlist properties only, to reduce computational effort but provide insufficient accuracy [8], [9].

A different class of methods assesses EPP by solving combinatorial search problems. Propositional satisfiability (SAT) and satisfiability modulo theory (SMT) solvers proofed to be efficient recently [10], [11]. Answer set programming (ASP) is another approach to modeling and solving of search problems. SAT is limited to classic propositional formulas in canonical normal form (CNF), whereas ASP offers a declarative constraint logic programming interface. On the one hand, this offers more compact problem modeling and easier integration of application-specific knowledge. On the other hand, structural information about the problem is preserved and can support the solving process, whereas in SAT, these information is lost in flattened CNF.

The availability of efficient ASP solvers, that provide competitive performance with respect to state-of-the-art SAT solutions [12], motivates research towards the exploration of ASP for EPP calculation. In this paper, we present an EPP estimation method based on ASP and evaluate its competitiveness compared to existing methods. Furthermore, we investigate the potential of features exclusive to ASP with respect to fault susceptibility analysis. Methods based on combinatorial search require counting of solutions for the calculation of EPP. Because the search space grows exponentially with circuit size and number of considered cycles, approximation is inevitable for the analysis of large designs. We use ASP in combination with approximate model counting to enable the application of our method to real-world circuits. To the best of the author's belief, this is the only work performing reliability analysis of digital circuits with ASP.

After all, our research contribution is characterized by the following key points:

- Demonstration of Answer Set Programming in combination with Approximate Model Counting as a suitable method for EPP assessment.
- Temporal behavior is encoded directly to avoid sequential unrolling.
- Reconvergent paths are considered by design of the method and do not need to be taken into account explicitly.
- Similarity of input data format to a Verilog netlist reduces preprocessing to a minimum.
- Proposal of a design flow for selective hardening of flip-flops based on the presented EPP assessment method.

In section II we put the proposed approach into context with related work and introduce to ASP and approximate model counting. In section III we describe how we employ it for the estimation of error propagation probabilities. Experimental as well as methodological results are presented in section IV. Afterwards, we discuss limitations in section V, before we conclude in section VI.

## II. BACKGROUND AND RELATED WORK

Let us have a closer look at the different approaches for EPP assessment and their characteristics. We would like to classify some of the most prominent methods first, to discuss their characteristics and limitations but also show recent improvements.

Simulation based analysis of error propagation is implemented in the following way. A test case from functional verification is executed and at a point in time an SEU is injected by flipping the value of a flip-flop. The simulation is continued then and the values of the primary outputs are permanently compared to the values of a fault-free simulation run. If the outputs differ, the fault is considered propagating. Usually, only one SEU is injected per simulation run. The fault coverage is determined by the coverage of the test cases (stimulus), the number of simulated fault sites and the number of time points where a fault was injected. A fault injection campaign is called exhaustive, when every possible fault was simulated, which requires a very high number of simulations even for medium sized circuits. For almost any practical use this is not possible, since the simulation needs to be evaluated also for all different states of the system where a fault could appear. For non-exhaustive fault injection campaigns, faults can be either equally distributed in place and time with a certain density, or chosen randomly. However, a high number of simulations is required to achieve sufficient accuracy [1]–[4].

The class of analytical methods does not require simulation and is based on the analysis of signal probabilities instead. FIGURE 1 shows an example for the propagation of signal and error probabilities in a small combinational circuit. Outgoing from the analysis of combinational circuits [13], various progress has been made by expressing the arithmetics as matrix operations [14], extending the analysis to sequential circuits [5], taking signal correlations into consideration [6] or even considering multiple faults [7]. However, due to sequential properties and signal correlations of reconvergent paths [15], the computational effort grows exponentially with circuit size and number of considered cycles. Additionally, preprocessing is required to parse the netlist and transform it to suitable data structures e.g. matrices.

The third class of methods is based on combinatorial search and constraint programming. The idea is to express a problem in some generic problem domain and to solve it with a universal solver. Most recent solvers rely on conflict-driven clause learning (CDCL) [16]–[19], which originates from the Davis-Putnam-Logeman-Loveland (DPLL) algorithm [20]. A well-known example is propositional satisfiability (SAT), where a satisfying assignment of a Boolean formula is searched. CDCL based solvers have made significant progress during the last two decades. The reduction of problems to a SAT problem and the solution with SAT solvers has widespread applications in academia and industry, especially in electronic design automation (EDA) tools.

The applications can be separated in applications where a limited number of satisfying assignments is required and

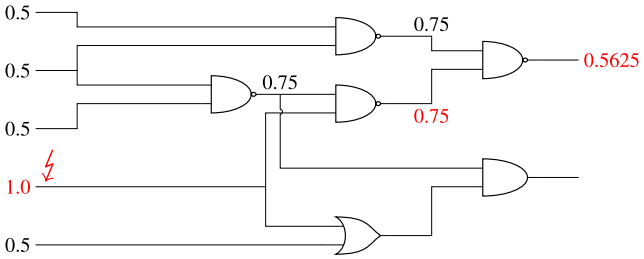



**FIGURE 1. Combinational propagation of signal probabilities. The red flash marks the fault site. Signal probabilities are typeset in black, error probabilities in red.**

applications where all satisfying assignments are required. When only the number of solutions is required, we speak of counting problems, which are associated with the complexity class #P [21]. The counting problem associated with SAT is called #SAT. EPP assessment is such an application where counting is required. The solutions of complex counting problems often can only be approximated [22], [23]. The approximation method, which was used for the presented work, is described in section II-B. How to use a SAT solver for EPP assessment is shown in [10].

Satisfiability Modulo Theories (SMT) is a generalization of SAT, extending it by background theories as the theories of integers, real numbers or data structures as lists and arrays. This allows more complex formulas. Application of SMT for EPP calculation is demonstrated in [11]. However, the input language of SAT and SMT solvers is limited to formulas in canonical normal form.

### A. ANSWER SET PROGRAMMING AND APPROXIMATE MODEL COUNTING

Answer set programming follows an approach similar to SAT, but it comes with a more powerful input language. On the one hand this often leads to a more efficient problem encoding due to more direct modeling. On the other hand more information about the structure of the problem is preserved and therefore available to the solving process. There are efficient implementations of ASP solvers, which were even able to win SAT competitions [12]. Additionally, it is important to evaluate possible advantages of ASP in applications where SAT solvers have proven to be successful. Furthermore, ASP eases the inclusion of background information in the solving process, which often enables advanced application scenarios and solving of related problems.

ASP was introduced by Gelfond and Lifschitz in 1988 [24], combining the syntax of logic programs with stable model semantics. It is a declarative approach to problem solving. In contrast to procedural programming languages, problem solving and problem encoding are separated. A problem is encoded by a set of clauses, which are processed by a solving engine to find solutions which fulfill them, so called answer sets or stable models. The programmer, solving problems with ASP, can focus on the description of the problem rather than its solution.

A program is implemented by three types of clauses [25]:

$$\begin{aligned} \textit{Facts}: &\quad A_0. \\ \textit{Rules}: &\quad A_0 :- L_1, \ldots, L_n. \\ \textit{IntegrityConstraints}: &\quad :- L_1, \ldots, L_n. \end{aligned}$$

Facts define expressions that hold unconditionally. Rules can be read as an implication. $L_1$ to $L_n$ imply $A_0$, so if $L_1$ to $L_n$ are true, $A_0$ must be true, but $A_0$ can be true with $L_1$ to $L_n$ being false. The third type of clauses are integrity constraints which are used to filter the answer sets. The solution will contain only answer sets where not all of the literals $L_1$ to $L_n$ hold. The solving process can be divided into grounding and solving. User-input clauses usually contain variables. Grounding transforms them into a set of clauses, which is free of variables and can be processed by a solver based on CDCL [16]–[19]. The encoding of a problem class and a specific problem instance can be separated. A problem class is usually described by a set of rules and integrity constraints. The problem can then be instantiated by a number of facts. For the solution of counting problems the solver is called iteratively, providing one stable model after the other, until all are found and the solver returns UNSAT (unsatisfiable).

The adaptation of CDCL, which originates from the domain of SAT solving, to ASP is called conflict-driven nogood learning (CDNL) [25]. No to overly extend the length of this article and not to obstruct readability, we sketch the functioning of the algorithm only briefly. The original DPLL algorithm is based on guessing and backtracking on encounter of an invalid solution. This creates a lot overhead due to wrong guesses and neglects a lot of information which could be acquired from wrong guesses. The idea of CDCL/CDNL is to learn from wrong guesses. Considering ASP, the learning can happen based on so called *nogoods* [25]. A nogood is a set of signed literals which cannot be contained in any satisfying assignment. Information can be acquired from a nogood, if all but one literal are contained in an assignment. The nogood is then called unit-resulting and the complement of the violating literal is added to the assignment. If a nogood is completely contained in an assignment, a conflict has occured. Instead of backtracking, the algorithm performs a backjump to the stage where the violating literal has been added. The algorithm operates in an iterative manner, until the assignment is complete or it turned out, that there is no satisfying assignment.

### B. APPROXIMATE MODEL COUNTING

If the search space and especially the number of solutions is too large to enumerate them, the counting problem can only be approximated. The approximation method is based on the idea of dividing the search space into subspaces of approximately the same size to reduce the number of models to be counted. According to [22], [23], this can be achieved by adding random parity constraints, i.e XOR constraints. An important property of XOR constraints is their pairwise (even 3-wise) independence. This means, that if we know

the elimination result of a given XOR constraint, we cannot derive any information about the elimination of satisfying assignments of any other randomly chosen XOR constraint. Any added XOR constraint eliminates a satisfying assignment with probability 1/2. A proof based on linear algebra can be found in [22].

The general complexity of linear programming under stable model semantics can be derived by simulating a nondeterministic Turing machine [26]. The complexity of approximate solutions to the associated counting problems using pairwise independent hash functions was analyzed by Stockmeyer in 1983 [27].

#### 1) ILLUSTRATIVE EXAMPLE FOR SAMPLING FROM THE SOLUTION SPACE WITH XOR CONSTRAINTS

Here, the idea should be illustrated in a less formal way by an example. A propositional formula is given by

$$y = ab + ac + bd + cd. \tag{1}$$

Three arbitrary XOR constraints are introduced as

$$c_1 = a \oplus b, \quad c_2 = a \oplus c, \; c_3 = a \oplus d. \tag{2}$$

A truth table for all 16 possible assignments of $a$, $b$, $c$ and $d$ is shown in TABLE 1. It can be seen that any XOR constraint is satisfied by eight assignments, the conjunction of any two XOR constraints is satisfied by four assignments and the conjunction of all three XOR constraints $c_1 \cdot c_2 \cdot c_3$ is satisfied by two assignments. $y$ is satisfied by 9 out of 16 assignments. The conjunction $y \cdot c_1 \cdot c_2 \cdot c_3$ is satisfied by one out of two assignments. This shows how the search space as well as the solution space are approximately split in half by every XOR constraint, resulting in a solution space approximately divided by eight using three XOR constraints.

#### 2) CLASSES OF APPROXIMATE MODEL COUNTERS

There are three categories of algorithms for approximate model counting [28]. So called $(\epsilon, \delta)$ counters return an approximate count $x$ which lies in the interval $[x(1+\epsilon)^{-1}, x(1+\epsilon)]$ with probability $1-\delta$. $\epsilon$ is called tolerance, $\delta$ is called confidence. Then there are bounding counters, which do not provide any guarantees on tolerance and only return an lower or upper limit which holds with probability $1-\delta$. Formal proofs for tolerance and confidence are mainly based on Markov's inequality, Chebyshev's inequality and the Chernoff bound [29]. The last category is called guarenteeless counters, which do not provide any guarantees at all.

#### 3) SCOPE OF THIS ARTICLE

A formal comparison of different approximate counting algorithms is not a core component of our work. The presented investigations are more of a practical nature and focus on the general suitability of ASP and approximate model counting for EPP assessment. For this reason, we investigated the accuracy of our approach empirically as described in detail in section IV.

**TABLE 1. Example of XOR constraints applied to a truth table. Satisfaction of XOR constraints illustrated by green highlighting, the higher the color intensity, the more XOR constraints are satisfied. This shows how the search space as well as the solution space are approximately split in half by every XOR constraint.**

| $c_1$ | $c_2$ | $c_3$ | $a$ | $b$ | $c$ | $d$ | $y$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |

## III. PROPOSED METHOD

In the following section we present a method to calculate approximate EPP values of flip-flops in digital circuits with the help of ASP and approximate model counting. The proposed EPP estimation method is supposed to be the main component of a design flow for selective fault tolerance as shown in FIGURE 2.

At first we would like to give an overview of the design flow including input and output parameters of our implementation. We continue with the definition of EPP calculation based on model counting. Then, we show how to encode the behavior of digital circuits and fault injection in ASP and how to generate random XOR constraints afterwards. Finally, we present our algorithm to iteratively call the solver, extract information about error propagation and accumulate EPP values.

We propose a design flow for radiation-hardened digital circuits, which extends the standard digital design flow between the synthesis step and physical implementation. It buils upon a fault susceptibility analysis of the post-synthesis netlis. The analysis yields a dictionary of flip-flops with their corresponding EPP values. The n-most critical flip-flops in terms of fault susceptibility can be selected for hardening then. The hardening is applied by replacing the unhardened flip-flops in the netlist with suitable radiation-hardened variants. Details with respect to the hardening of a digital standard cell library can be found in our recent publication [30].

The main input parameters of our implementation are the gate-level netlist of the circuit under investigation, the number of considered cycles and the number of applied XOR constraints, which determines the degree of approximation. The main output parameter is a dictionary containing the EPP values per flip-flop using flip-flop instance names as keys. Then there are technical input parameters (e.g. timeout),

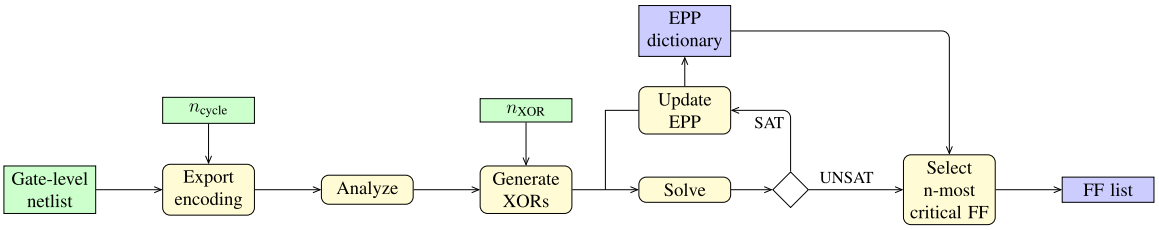



**FIGURE 2.** Activity diagram illustrating the proposed flow for selective fault-tolerance by ASP and Approximate Model Counting.

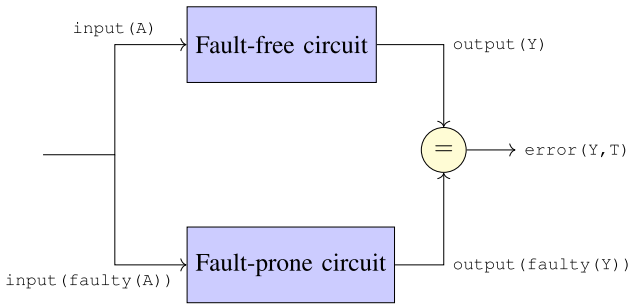



**FIGURE 3.** Conceptual approach of duplication, fault injection and comparison for modeling the analysis problem in ASP.

which control the execution and technical output parameters (e.g. runtime), which are used for the evaluation of the method. Furthermore, our implementation outputs circuit parameters, as the number of flip-flops in the circuit, which are also used for evaluation and comparison. All input and output parameters are listed in TABLE 2.

The definition of error propagation probability (EPP) used here is the following: the EPP is the probability that a single event upset is not logically masked, so that it propagates and manifests as an error at the primary outputs. Logic masking depends on the circuit state and the stimulus, i.e. the signal assignments, while the circuit is in a faulty state. The EPP can be calculated as the ratio of the number of signal assignments where the fault propagates to the number of all possible signal assignments:

$$\text{EPP} = \frac{\#\text{fault propagating assignments}}{\#\text{all assignments}} = \frac{n_{\text{cnt}}}{n_{\text{asgn}}} \tag{3}$$

A signal assignment consists out of an initial circuit state, i.e. the values stored in the flip-flops, and an input sequence with respect to all primary inputs and all considered clock cycles. The number of all possible signal assignments can be calculated as

$$n_{\text{asgn}} = 2^{n_{\text{input}} \cdot n_{\text{cycle}} + n_{\text{ff}}}, \tag{4}$$

with $n_{\text{input}}$ being the number of primary inputs, $n_{\text{ff}}$ the number of registers and $n_{\text{cycle}}$ the number of considered clock cycles. The identification of fault propagating assignments relies on circuit duplication, fault injection and comparison. The principle is illustrated in FIGURE 3.

Throughout this work we use the Potsdam Answer Set Solving Collection (Potassco) with clingo [31] as a monolithic application combining grounder and solver (cf. section II-A). All ASP encodings are presented in the precise input language of clingo [32]. The proposed method is based on a similar encoding as the one we have used for some preliminary investigations in [33] and which is shown in Listing 1. Now, we would like to go through the five main parts of the encoding, describing the circuit behavior, duplicating the circuit, injecting a fault, detecting the error and spanning the search space.

We start, encoding the circuit elements by describing the behavior of combinational gates and flip-flops in code section Ⓐ. An OR gate is described in line 1. An atom

**TABLE 2.** Description of input and output parameters of an analysis run with the proposed implementation.

| | |
|---|---|
| **Input Parameter** | |
| $C$ | Circuit under investigation. |
| $n_{\text{cycle}}$ | Number of considered cycles. |
| $n_{\text{XOR}}$ | Number of parity constraints. |
| $a$ | Approach for solving the parity constraints. |
| $q$ | XOR constraint density. |
| $n_{\text{iter}}$ | Number of iterations, each with different random parity constraints. |
| $n_{\text{limit}}$ | Stop solving when nlimit models are found. This is used to investigate runtime behavior when calculating all models is not feasible. |
| $t_{\text{timeout}}$ | Timeout in seconds to guarantee termination of analysis suite. |
| **Output Parameter** | |
| $n_{\text{model}}$ | Number of stable models found by the solver. This can be either all models, if no parity constraints are applied, or the number of models in the subspace. |
| $n_{\text{cnt}}$ | Approximate model count. Number of models multiplied by $2^{n_{\text{xor}}}$. |
| $n_{\text{SAT}}$ | Number of iterations that were satisfiable (c.f. $n$iter). |
| $n_{\text{ff}}$ | Number of flip-flops in the circuit. |
| $n_{\text{input}}$ | Number of primary inputs of the circuit. |
| $n_{\text{asgn}}$ | Number of all possible signal assignments $(2^{n_{\text{input}} \cdot n_{\text{cycle}} + n_{\text{ff}}})$. |
| $n_{\text{output}}$ | Number of primary outputs of the circuit. |
| $n_{\text{gate}}$ | Number of combinational gates in the circuit. |
| $t_{\text{total}}$ | Total runtime. |
| $t_{\text{mod}}$ | Time per model. |
| $b_{\text{timeout}}$ | Boolean indicating whether the timeout has occurred. |
| $n_{\text{conflct}}$ | Number of conflicts according to CDCL algorithm. |
| $D_{\text{ff}}$ | Dictionary of EPP per flip-flop using flip-flop instance name as key. |

`gate(G,F,Q,A0,A1)` is introduced, with gate name `G`, function `F` (here "or"), output signal `Q` and input signals `A0` and `A1`. The atom `high(Q,T)`, modeling the value of `Q` at time `T`, holds, if `high(Q,T)` holds for at least one of the input signals at the same time `T`, where `T` is measured in cycles. AND gate and inverter (INV) are modeled analogously. Line 4 describes a flip-flop, where the sequential behavior is modeled by deriving the atom `high(Q,T)` for time `T` from the input value `high(A0,T-1)` at previous cycle `T-1`. We can directly model the sequential behavior, due to ASP's support of variables and the introduction of a temporal variable, and avoid sequential unrolling as in related approaches [10], [11].

Code section Ⓑ defines a duplicate of the circuit under investigation to inject faults in it. Throughout the code, the suffix `c` (correct) indicates an element of the fault-free circuit and the suffix `f` (faulty) an element in the circuit prone to faults. For instance, line 6 can be understood in a way, that for every signal in the fault-free circuit `sigc` with name `X`, there is a (possibly) faulty signal `sigf` with name `faulty(X)` in the circuit instance where faults are injected.

The next paragraph of code, section Ⓒ, models fault injection for inputs and flip-flops. Line 11 states, that the input signal of the faulty circuit `faulty(X)` is high, if the input signal of the correct circuit `X` is not high and a fault is injected in it. The other opportunity for the signal in the faulty circuit to be high is, that no fault is injected and the correct signal is high (line 12).

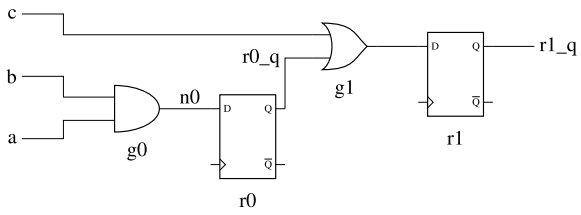
Code section Ⓓ defines the comparison of the correct and the faulty circuit. A difference between the values of the output signals of the correct and faulty circuit is regarded as an error. A difference between the values of internal signals is denoted as fault. The signal values are different, if exactly one of them is high.

Finally, we span the search space in code section Ⓕ. So called *aggregates* are used to form values from groups of selected items. Using comparison operators, conditions over these items can be expressed [25]. Aggregates are another language feature of ASP which is superior to SAT. Line by line the following dimensions are described: all possible initial circuit states, all possible input sequences, a single fault in a flip-flop at cycle 0, no faults at the inputs. Instead of injecting faults in one register after the other as in [10], [11], we search for all solutions where an upset is present in any single register at cycle zero:

```
{inject(X,0) : register(X)} = 1.
```

This significantly increases computational efficiency, because we require less calls to the solver and the solution density, which is the ratio between satisfying solutions and the size of the search space, is higher.

Given this problem encoding, a circuit can be instantiated for EPP analysis as shown in FIGURE 4. The proposed analysis is performed on the gate-level netlist obtained from logic synthesis. The circuit description suitable for our




(a) example circuit schematic

```
1  sigc(n0).
2  sigc(n1).
3  sigc(r0_q).
4  sigc(r1_q).
5  gatec(g0, and, n0, a, b).
6  gatec(g1, or, n1, r0_q, c).
7  regc(r0,r0_q,n0).
8  regc(r1,r1_q,n1).
9  input(a;b;c).
10 output(r1_q).
```

(b) corresponding circuit instantiation following the proposed encoding

**FIGURE 4. A simple circuit example with schematic and corresponding enconding as a demonstration of problem instantiation.**

ASP encoding is exported by a set of Tcl commands directly from the synthesis tool.

Passing the presented encoding and a circuit instantiation to clingo, would return answer sets where the error propagates, as well as answer sets where the error does not propagate. In order to calculate EPP as described before, we are interested in the number of answer sets where the fault propagates. By applying the following integrity constraint, we filter for solutions where the error propagates:

```
:{-} not error(_,_).
```

Here is an exemplary answer set for the circuit example from FIGURE 4 considering three cycles:

```
high(a,2) high(b,2)
high(faulty(r1_q),0)
inject(r1,0) high(faulty(b),2)
high(faulty(a),2) high(n0,2)
high(faulty(n0),2) error(r1_q,0)
```

This answer set describes the case, where a fault is injected in flip-flop r1 at cycle 0. Because r1 is an output register connected to the primary output r1_q, the fault propagates immediately, so that there is an error at r1_q at cycle 0.

Standalone clingo with its command-line interface could be used to generate an exact count, iteratively calling the solver until all answer sets, where the fault propagates, are found. In order to generate and solve random XOR constraints for approximate counting and to have more control about the execution, we implemented an application in Python and directly access clingo's Application Programming Interface (API). clingo on its own does not support parity constraints, but it offers a theory propagator interface to add solution capabilities for background theories [34]. There is a prototype application called *xorro* [35], which implements several approaches for the solution of parity constraints. Our application reuses some of the code from xorro,

```
high(O,T) :- gate(G,F,O,I0,I1), sig(O), sig(I0), sig(I1), F = or,
    {high(I0,T);high(I1,T)} > 0, cycles(C), T=0..C.
high(O,T) :- gate(G,F,O,I0,I1), sig(O), sig(I0), sig(I1), F = and,
    {high(I0,T);high(I1,T)} = 2, cycles(C), T=0..C.
high(O,T) :- gate(G,F,O,I0), sig(O), sig(I0), F = inv, not high(I0,T), cycles(C),
    T=0..C.
high(O,T)  :- reg(R,O,I), sig(O), sig(I), high(I,T-1), cycles(C), T=1..C.

sigf(faulty(X)) :- sigc(X).
gatef(faulty(G),F,faulty(O),faulty(I0),faulty(I1)) :- gatec(G,F,O,I0,I1).
regf(faulty(R),faulty(O),faulty(I)) :- regc(R,O,I).
gatef(faulty(G),F,faulty(O),faulty(I0)) :- gatec(G,F,O,I0).

high(faulty(X),T) :- not high(X,T), inject(X,T), input(X).
high(faulty(X),T) :- high(X,T), not inject(X,T), input(X).
high(faulty(X),T) :- not high(X,T), inject(R,T), regc(R,X,_), sigc(X).
high(faulty(X),T) :- high(X,T), not inject(R,T), regc(R,X,_), sigc(X).

error(Y,T) :- {high(Y,T);high(faulty(Y),T)} = 1, output(Y), cycles(C), T=0..C.
fault(Y,T) :- {high(Y,T);high(faulty(Y),T)} = 1, sigc(Y), cycles(C), T=0..C.
fault(T)   :- fault(_,T).

{high(S,0)    : reg_out(S)}.
{high(I,T)    : input(I)} :- cycles(C), T=0..C.
{inject(X,0) : register(X), not disable(X)} = 1.
{inject(X,0) : input(X)}    = 0.
```

A logic behavior
B circuit duplication
C fault injection
D error detection
E search space definition

**Listing 1.** **Problem encoding describing digital circuits elements, circuit duplication, fault injection, error detection and spanning of search space. Auxiliary clauses are omitted.**

but the main algorithm is adapted towards the presented application of EPP assessment as described in the following.

The algorithm is outlined in Algorithm 1. After read-in of all input parameters including the circuit instance, we use an initial grounding and solving to extract the number of inputs and the number of flip-flops in order to calculate the number of all possible assignments. Then we generate the random XOR constraints and add them to the problem.

The implementation of xorro is not specific to any application. For this reason, the constraint generation algorithm included in xorro generates constraints from all symbolic atoms of the problem. Adding XOR constraints generated this way can easily cause the problem to become unsatisfiable because of correlations between atoms. Due to the pairwise independence of XOR constraints (c.f. section II-B), we can also generate the constraints from just a subset of all possible constraints of a certain length. Our implementation generates XOR constraints based on signal names from the fault-free circuit and combines them with the atom `high(Q,T)`, where `Q` is the name of the signal and `T` is a random cycle. By this approach we reduce correlations between the individual terms of a constraint and the problem is less likely to become unsatisfiable.

After the random XOR constraints were added, we need to ground the problem again, before we enter our main loop. In this loop, we call the solver and if it returns an answer set, we increment the corresponding dictionary entry of the flip-flop where the fault was injected by $2^{n_{\text{XOR}}}/n_{\text{asgn}}$. If the call to the solver does not return another answer set but instead UNSAT, this means that all answer sets in this subspace (defined by the random XOR constraints) are found and the analysis is completed.

**Algorithm 1** Iterative Solving and Incremental EPP Calculation

```
ground()
M ← solve()
(n_ff, n_input) ← extractNetlistProperties(M)
n_asgn ← 2^(n_input·n_cycle+n_ff)
generateRandomXORs(n_XOR)
ground()
while true do
    M ← solve()
    if M ≠ ∅ then
        D_ff(ffName(M)) += 2^n_xor / n_asgn
    else
        return
    end if
end while
```

## IV. EXPERIMENTAL EVALUATION AND RESULTS

We evaluated the presented method with respect to its suitability for EPP estimation in order to select flip-flops for protection in terms of criticality of fault propagation. We performed experiments with several ISCAS89 and ITC99 benchmark circuits. The circuits are described in TABLE 3.

There is a general challenge inherent to the evaluation of the proposed method: Due to the nature of the problem an exact solution can only be found for small circuits and a small number of considered cycles. Even for a small circuit like s27 and for only 10 cycles, the search space size is about $4.5 \cdot 10^{15}$ signal assignments.

For more complex problems we add parity constraints for approximation. As described in section II-B, adding parity

**TABLE 3. Characteristics of analyzed circuits.**

| Circuit | PIs | POs | FFs | Gates |
|---|---|---|---|---|
| s27 | 4 | 1 | 8 | 12 |
| s444 | 3 | 6 | 21 | 137 |
| s5378 | 35 | 49 | 179 | 1350 |
| s9234 | 39 | 39 | 211 | 1772 |
| s15850 | 77 | 150 | 534 | 3436 |
| b01 | 2 | 2 | 5 | 47 |
| b02 | 1 | 1 | 4 | 28 |
| b03 | 4 | 4 | 30 | 150 |
| b06 | 2 | 6 | 9 | 55 |
| b09 | 1 | 1 | 28 | 165 |
| b10 | 11 | 6 | 17 | 176 |

constraints divides the search space into smaller subspaces. Every set of parity constraints defines a certain subspace: the more constraints, the smaller the subspace. However, the subspace must contain a representative number of models to yield a valid approximation. If the subspace is empty, no solution can be found and the problem instance becomes unsatisfiable. Even if the subspace is not empty, but the number of constraints is very large compared to the number of models to be found, the CDCL algorithm encounters more conflicts, thus it becomes harder to find a model.

As a consequence of these two issues, our test cases used for evaluation cannot be too complex, if we want to use an exact count as a reference. Otherwise the exact counting would take too long. But the number of parity constraints which can be added to speed up the computation is limited, because otherwise the problem instance becomes unsatisfiable. We cannot fully leverage the potential of our approximation method on small problems.

This issue is addressed by evaluating the method in three steps. First, we investigate runtime and accuracy for small problems with no approximation or a low level of approximation. Then, we artificially increase problem complexity by considering a number of cycles larger than the sequential depths. When increasing the number of considered cycles, the EPP is constant as long as the influence of sequential feedback paths is small and the number of considered cycles is larger than the sequential depth. In the end, we apply our method to larger circuits, where we cannot calculate the accuracy, but where we can investigate large scale runtime behavior and identify bottlenecks and limitations of the proposed method.

Most of the following discussion is led with the circuit s27 as a reference. The circuit is very small so that we are able to calculate an exact solution fast, but it is complex enough to yield a non-trivial solution. For investigating larger problems we initially use s27 with an increased number of considered cycles, which increases the search space but the EPPs stay the same.

On the one hand our experimental results are technical, consisting out of performance parameters of the analysis. On the other hand we provide methodological results evaluating the presented approach with respect to the targeted application. Let us start with technical results by investigating

**TABLE 4. Exemplary error propagation probability results per flip-flop for circuit s27 for 10 considered cycles with 35 parity constraints representing a rough estimation.**

| FF name | EPP |
|---|---|
| dff_out_g17_q_reg_reg | 1.010498 |
| dff_2_q_reg_reg | 0.061279 |
| dff_0_q_reg_reg | 0.169006 |
| dff_in_g3_q_reg_reg | 0.046432 |
| dff_in_g1_q_reg_reg | 0.061279 |
| dff_1_q_reg_reg | 0.106491 |
| dff_in_g0_q_reg_reg | 0.162262 |
| dff_in_g2_q_reg_reg | 0.055389 |

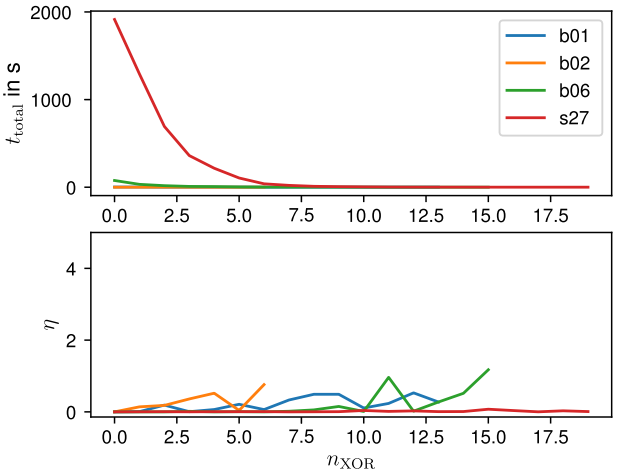



**FIGURE 5. Runtime and accuracy of analysis of several small circuits with exact counting and different approximations. The number of considered cycles is 3.**

runtime and accuracy for different degrees of approximation with the exact count as a reference as shown in FIGURE 5.

The relative error is calculated as

$$\eta = \frac{\left| n_{\text{model}}|_{n_{\text{XOR}}=0} - n_{\text{model}} \right|}{n_{\text{model}}|_{n_{\text{XOR}}=0}} \tag{5}$$

The exact counting takes 1915 s. Adding parity constraints leads to an exponential decay of runtime. Adding for instance 7 parity constraints reduces the runtime to 21 s, which is a factor of 91. The error is below 1 %.

Progressing towards larger search spaces, we analyze s27 considering 3 to 5 cycles. The results are shown in FIGURE 6. It can be seen, that, by approximating, the runtime for all cases can be reduced to a fraction compared to the exact counting whereas the error stays small. A timeout of 3600 s was set, which explains the result for small numbers of parity constraints when considering 5 cycles. When applying 15 parity constraints the runtime considering 5 cycles is only 41 s whereas the error is still below 1 %.

In order to compare our approach to [11], we analyzed a selection of ITC99 benchmark circuits (b01-b10). The runtime of these experiments is depicted in FIGURE 7. The timeout was set to 4000 s. The timeout occurred for b10 for all executions and for b06 for test cases with less than 15 parity constraints. For all other cases the runtime could be kept

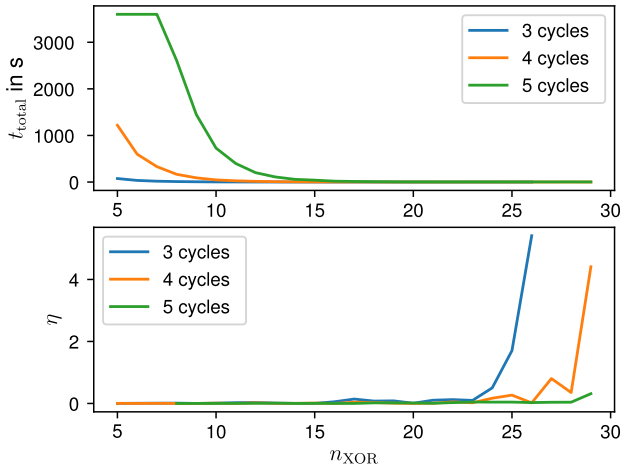



**FIGURE 6.** Runtime and accuracy of analysis with increased number of cycles and rougher approximations for circuit s27 with 3 to 5 considered cycles.

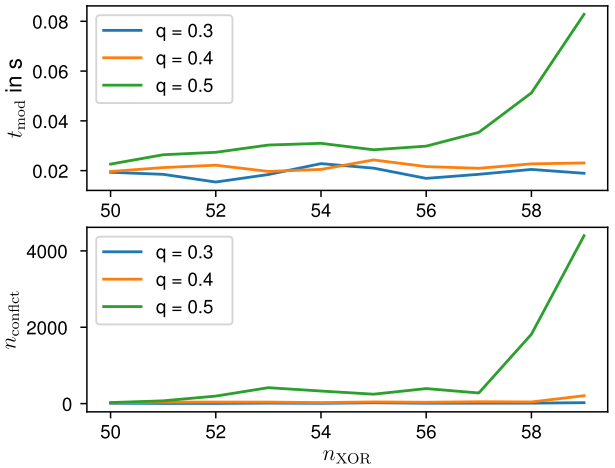



**FIGURE 8.** Number of conflicts and time per model in dependence of the number of XOR constraints for circuit s27. 20 cycles were considered and constraints of different densities $q$ were applied.

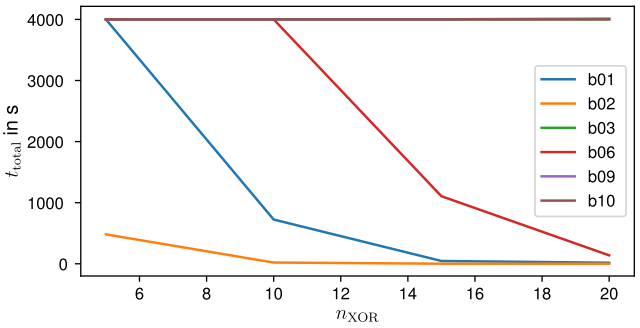



**FIGURE 7.** Runtime in dependence of number of XOR constraints for the same selection of ITC99 circuits as in [11]. Timeout was set to 4000 s.

below the runtime results of [11] choosing an appropriate number of parity constraints. The SER estimates are not comparable directly, because [11] only calculates the errors at the outputs of the final sequential stage after sequential unrolling, whereas our approach detects all errors at the primary outputs independent of sequential depth.

During our work we observed timeout occurrences for more complex circuits and a larger number of considered cycles when increasing the number of parity constraints. From an analytical point of view, solutions should still exist in the corresponding subspaces. It turned out, that the timeout occurred even before the first model was found. For this reason, we started to investigate the time per model. To guarantee termination of the solving process, the number of requested models was limited, so that we could investigate the time per model $t_{\text{mod}}$ individually. The results of these investigations are shown in FIGURE 8 for circuit s27 and in FIGURE 9 for the more complex circuit s5378. It can be seen, that there is a sudden increase of $t_{\text{mod}}$, when the number of parity constraints increases. We discovered, that this is caused by an increased number of conflicts (c.f. CDCL algorithm). As can be seen, also the density $q$ of the XOR constraints plays an important role for the computational efficiency. The number of conflicts can be decreased by lowering the density of the XOR constraints, but this could also reduce the accuracy of the approximation [23]. The general reason for this phenomenon is, that the solution of parity constraints is not inherently supported by the CDCL solver, because it operates with clauses in canonical normal form (CNF) [35]–[38]. Parity constraints can be transformed efficiently into CNF by Gauss-Jordan elimination (GJE). A deep integration of GJE into CDCL solvers and further optimizations have been shown as a consecutive success in [37]–[39] and corresponding improvements are part of our ongoing work. In summary it can be seen, that the degree of approximation is limited in its current implementation, but that the proposed method can still be applied to complex circuits. The reasons for these limitations have been identified and approaches to address them were referenced [37]–[39]. Limitations are discussed in greater detail in section V.

After all, the results show a competitive performance of the proposed method for the estimation of EPP values of flip-flops in digital circuits. As stated in the beginning, the application scenario we have in mind is selective fault tolerance. A given circuit should be hardened against soft errors, e.g. radiation induced SEU, with limited resources in terms of power and area consumption. The goal would be to protect the n-most critical flip-flops according to available resources. For this purpose, an exact value of EPPs is not required. It is rather important that the order of flip-flops, sorted by their EPP, is maintained. By the distribution of the values in TABLE 4, one can see that these order is given. An EPP value larger than one is an effect of approximation. It means that the respective flip-flop is over represented in the subspace chosen for model counting. Due to the freely adjustable degree of approximation, the analysis can be adapted to the specific accuracy requirements and available computational resources, so that this method is also applicable when precise information is required. A worst case runtime can be estimated as

$$t_{\text{total}} = t_{\text{mod}} \cdot n_{\text{asgn}} / 2^{n_{\text{XOR}}}, \tag{6}$$

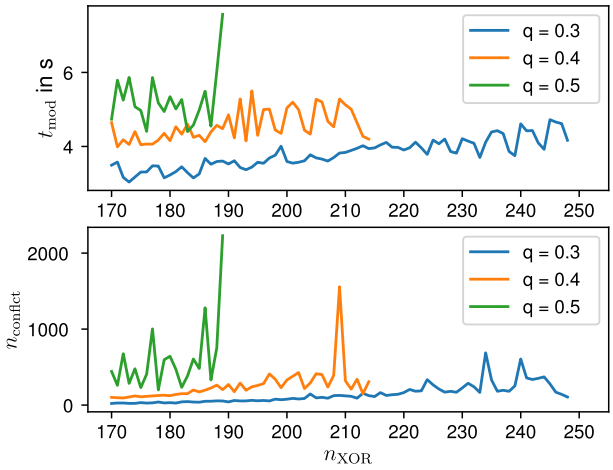



**FIGURE 9.** **Number of conflicts and time per model in dependence of the number of XOR constraints. Circuit s5378 was chosen as an example for large-scale runtime behavior. 20 cycles were considered and constraints of different densities $q$ were applied.**

with $t_{\mathrm{mod}}$ being the average time per model and $n_{\mathrm{asgn}}$ being an upper limit for $n_{\mathrm{model}}$, assuming that every assignment leads to fault propagation. The number of required XOR constraints can then be calculated as

$$n_{\mathrm{XOR}} = \log_2\left(\frac{t_{\mathrm{mod}}}{t_{\mathrm{limit}}} \cdot n_{\mathrm{asgn}}\right), \tag{7}$$

with $t_{\mathrm{limit}}$ being the amount of time dedicated to the analysis. $t_{\mathrm{mod}}$ is usually in the order of several seconds and can be considered constant for a certain density $q$ over a wide range of degrees of approximation (cf. FIGURE 8). A small test run prior to the actual analysis can confirm the assumption for $t_{\mathrm{mod}}$.

In addition to a good performance, we especially consider the following aspects as important advantages. Due to the fact, that the input data format of the circuit instance is similar to a Verilog netlist, no preprocessing is required. We export a suitable description directly from the synthesis tool. Automatic transformation from a Verilog netlist based on a couple of string operations is also possible. Prior to most formal methods is an extensive preprocessing of the Verilog netlist to transform it into the required data structures as graphs or matrices. In many methods certain circuit aspects require special treatment, as for instance unrolling of sequential feedback paths [10], [11] or approximation of signal correlations [40]. In our approach sequential behavior is modeled directly, so that no sequential unrolling is required. Signal correlations are implicitly taken into account as they are reflected in the set of all possible signal assignments.

In spite of the compact and comprehensive problem encoding, our approach is extensible to cover neighboring problems or to yield valuable side information. Two examples are provided as an illustration. The first example addresses the modification of the error definition. For instance, if errors at the primary outputs are not a concern, but instead the analysis should identify flip-flops, where a fault causes the circuit to

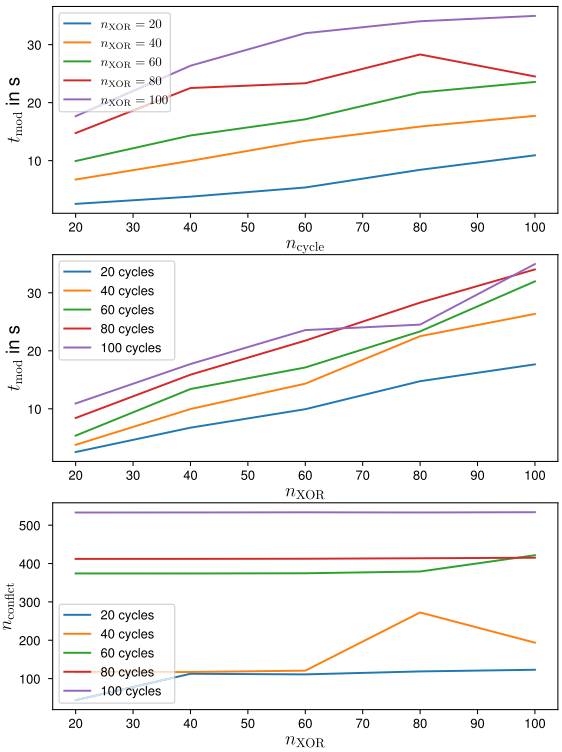



**FIGURE 10.** **Runtime behavior of the analysis of medium-sized ISCAS89 circuit s15850.**

be still in faulty state `C` cycles after the fault occurred, the following integrity constraint could be applied:

```
:{-} not fault(C),cycles(C).
```

The second example shows how to narrow down the search space and speed up the analysis by including functional information from circuit application level. We could limit the analysis to valid input sequences. For instance, if it is known, that two circuit inputs `inputa` and `inputb` cannot be high at the same time for two consecutive cycles, we could use the following rule and integrity constraint:

```
invalid :- high(inputa,T),
    high(inputb,T), high(inputa,T-1),
    high(inputb,T-1), cycles(C), T=1..C.
:{-} not invalid.
```

Of course, these two examples can only give a glance on the flexibility of the presented implementation.

## V. LIMITATIONS

The analysis of output sensitization of digital circuits has been proven to be NP-complete [41]. Consequently, EPP assessment is member of the complexity class #P, which is the class of counting problems associated with problems in NP [21]. By approximation we can reduce computational complexity, but the problem is still NP-hard [27]. Still, we have shown an exponential decrease of computational effort by increasing the degree of approximation determined by the number of XOR constraints. This can be seen in

Figures 5 to 7. Additionally, we have shown at the end of section IV, how to reduce the search space by including application-level information or modifying the error definition.

We have also verified our current implementation for medium-sized circuits like s15850 (cf. TABLE 3) and corresponding runtime metrics are shown in FIGURE 10. It can be seen, that the time per model $t_{\text{mod}}$ is linearly dependent on the number of considered cycles $n_{\text{cycle}}$ and the number of parity constraints $n_{\text{XOR}}$, while the number of conflicts $n_{\text{conflct}}$ is nearly constant. Nevertheless, the time per model and the number of error propagating signal assignments are too high to complete the analysis in a reasonable amount of time. The degree of approximation needs to increased further.

However, currently the maximum degree of approximation is still limited by increasing number of CDCL conflicts due to a high number of XOR constraints. The reason is an unoptimized implementation for solving of parity constraints. Several improvements, for instance an integration of Gauss-Jordan elimination, which were successfully applied in another context [37], [38], are ongoing and part of our future work. Furthermore, adaptations of the solver as well as further modifications of the generation of XOR constraints with respect to this specific application are currently under investigation.

## VI. CONCLUSION

We presented a method for fast and accurate EPP estimation based on ASP and Approximate Model Counting and we demonstrated how to use it for the selection of flip-flops for SEU protection. Modeling the problem with ASP has proven to be a compact and efficient approach. In combination with approximate model counting based on random parity constraints, i.e. XOR constraints, the method is applicable to complex problems and shows a competitive performance.

A challenge was identified in choosing an appropriate density of the XOR constraints, on the one hand to avoid, that the problem becomes unsatisfiable due to too short XORs and on the other hand that the computational effort of solving the parity constraints is not too high due to higher length of constraints. Additionally, a deep integration of Gauss-Jordan elimination into our implementation is ongoing work.

Furthermore, we are currently investigating performance improvements by including further information from application level of the circuit to narrow down the search space. The approach has shown a high flexibility, so that it should be evaluated also for different fault models and related problems.

## ACKNOWLEDGMENT

The authors thank the Potassco Team, especially Torsten Schaub and Flavio Everardo, for their support.

## REFERENCES

[1] 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," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 6, pp. 693–706, Jun. 2008, doi: 10.1109/TVLSI.2008.2000254.

[2] C.-K. Chang, S. Lym, N. Kelly, M. B. Sullivan, and M. Erez, "Hamartia: A fast and accurate error injection framework," in *Proc. 48th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshops (DSN-W)*, Jun. 2018, pp. 101–108, doi: 10.1109/DSN-W.2018.00046.

[3] L. Entrena, M. García-Valderas, R. Fernández-Cardenal, A. Lindoso, M. P. García, and C. López-Ongil, "Soft error sensitivity evaluation of microprocessors by multilevel emulation-based fault injection," *IEEE Trans. Comput.*, vol. 61, no. 3, pp. 313–322, Mar. 2012, doi: 10.1109/TC.2010.262.

[4] K. Wu, H. Pahlevanzadeh, P. Liu, and Q. Yu, "A new fault injection method for evaluation of combining SEU and SET effects on circuit reliability," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Jun. 2014, pp. 602–605, doi: 10.1109/ISCAS.2014.6865207.

[5] H. Asadi and M. B. Tahoori, "Soft error modeling and protection for sequential elements," in *Proc. 20th IEEE Int. Symp. Defect Fault Tolerance VLSI Syst. (DFT)*, Oct. 2005, pp. 463–471, doi: 10.1109/DFTVS.2005.61.

[6] L. Chen, M. Ebrahimi, and M. B. Tahoori, "CEP: Correlated error propagation for hierarchical soft error analysis," *J. Electron. Test.*, vol. 29, no. 2, pp. 143–158, Apr. 2013, doi: 10.1007/s10836-013-5365-0.

[7] M. Fazeli, S. N. Ahmadian, S. G. Miremadi, H. Asadi, and M. B. Tahoori, "Soft error rate estimation of digital circuits in the presence of multiple event transients (METs)," in *Proc. Design, Autom. Test Eur.*, Mar. 2011, pp. 1–6, doi: 10.1109/DATE.2011.5763020.

[8] I. Wali, B. Deveautour, A. Virazel, A. Bosio, P. Girard, and M. S. Reorda, "A low-cost reliability vs. cost trade-off methodology to selectively harden logic circuits," *J. Electron. Test.*, vol. 33, no. 1, pp. 25–36, Feb. 2017, doi: 10.1007/s10836-017-5640-6.

[9] A. Breitenreiter, S. Weidling, O. Schrape, S. Zeidler, P. Reviriego, and M. Krstic, "Selective fault tolerance by counting gates with controlling value," in *Proc. IEEE 25th Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jul. 2019, pp. 15–20, doi: 10.1109/IOLTS.2019.8854380.

[10] S. Shazli and M. Tahoori, "Using Boolean satisfiability for computing soft error rates in early design stages," *Microelectron. Rel.*, vol. 50, no. 1, pp. 149–159, Jan. 2010, doi: 10.1016/j.microrel.2009.08.006.

[11] G. Kazma, G. B. Hamad, O. Ait Mohamed, and Y. Savaria, "Analysis of SEU propagation in sequential circuits at RTL using satisfiability modulo theories," in *Proc. 15th IEEE Int. New Circuits Syst. Conf. (NEWCAS)*, Jun. 2017, pp. 237–240, doi: 10.1109/NEWCAS.2017.8010149.

[12] M. Järvisalo, D. Le Berre, O. Roussel, and L. Simon, "The international SAT solver competitions," *AI Mag.*, vol. 33, no. 1, pp. 89–92, Mar. 2012. Accessed: Mar. 31, 2022. [Online]. Available: https://ojs.aaai.org/index.php/aimagazine/article/view/2395, doi: 10.1609/aimag.v33i1.2395.

[13] G. Asadi and M. B. Tahoori, "An accurate SER estimation method based on propagation probability [soft error rate]," in *Proc. Design, Automat. Test Eur.*, vol. 1, Mar. 2005, pp. 306–307, doi: 10.1109/DATE.2005.49.

[14] S. Krishnaswamy, G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Probabilistic transfer matrices in symbolic reliability analysis of logic circuits," *ACM Trans. Design Autom. Electron. Syst.*, vol. 13, no. 1, pp. 8:1–8:35, Jan. 2008. Accessed: Dec. 11, 2019, [Online]. Available: http://doi.acm.org/10.1145/1297666.1297674, doi: 10.1145/1297666.1297674.

[15] S. Ercolani, M. Favalli, M. Damiani, P. Olivo, and B. Ricco, "Estimate of signal probability in combinational logic networks," in *Proc. 1st Eur. Test Conf.*, Apr. 1989, pp. 132–138, doi: 10.1109/ETC.1989.36234.

[16] R. J. Bayardo and R. C. Schrag, "Using CSP look-back techniques to solve real-world SAT instances," in *Proc. 14th Nat. Conf. Artif. Intell., 9th Conf. Innov. Appl. Artif. Intell. (AAAI/IAAI)*. Providence, RI, USA: AAAI Press, 1997, pp. 203–208.

[17] J. Marques-Silva and K. Sakallah, "GRASP: A search algorithm for propositional satisfiability," *IEEE Trans. Comput.*, vol. 48, no. 5, pp. 506–521, May 1999, doi: 10.1109/12.769433.

[18] L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik, "Efficient conflict driven learning in a Boolean satisfiability solver," in *IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD) IEEE/ACM Dig. Tech. Papers*, Nov. 2001, pp. 279–285, doi: 10.1109/ICCAD.2001.968634.

[19] J. Marques-Silva, I. Lynce, and S. Malik, "Conflict-driven clause learning SAT solvers," in *Handbook of Satisfiability*. Amsterdam, The Netherlands: IOS Press, 2009, pp. 131–153. Accessed: Jul. 10, 2021. [Online]. Available: https://ebooks.iospress.nl/doi/10.3233/978-1-58603-929-5-131, doi: 10.3233/978-1-58603-929-5-131.

[20] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Commun. ACM*, vol. 5, no. 7, pp. 394–397, Jul. 1962, doi: 10.1145/368273.368557.

[21] L. G. Valiant, "The complexity of computing the permanent," *Theor. Comput. Sci.*, vol. 8, no. 2, pp. 189–201, 1979. Accessed: Feb. 23, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0304397579900446, doi: 10.1016/0304-3975(79)90044-6.

[22] C. P. Gomes, A. Sabharwal, and B. Selman, "Near-uniform sampling of combinatorial spaces using XOR constraints," in *Proc. 19th Int. Conf. Neural Inf. Process. Syst. (NIPS)*. Cambridge, MA, USA: MIT Press, Dec. 2006, pp. 481–488.

[23] C. P. Gomes, A. Sabharwal, and B. Selman, "Model counting: A new strategy for obtaining good bounds," in *Proc. 21st Nat. Conf. Artif. Intell. (AAAI)*, vol. 1. Boston, MA, USA: AAAI Press, Jul. 2006, pp. 54–61.

[24] M. Gelfond and V. Lifschitz, "The stable model semantics for logic programming," in *Proc. Int. Log. Program. Conf. Symp.*, R. Kowalski and B. Kenneth, Eds. Cambridge, MA, USA: MIT Press, 1988, pp. 1070–1080. [Online]. Available: http://www.cs.utexas.edu/users/ai-lab?gel88

[25] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, "Answer set solving in practice," *Synth. Lectures Artif. Intell. Mach. Learn.*, vol. 6, no. 3, pp. 1–238, Dec. 2012. Accessed: Dec. 10, 2021. [Online]. Available: http://www.morganclaypool.com/doi/abs/10.2200/S00457ED1V01Y201211AIM019, doi: 10.2200/S00457ED1V01Y201211AIM019.

[26] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov, "Complexity and expressive power of logic programming," in *Proc. 12th Annu. IEEE Conf. Comput. Complex.*, Jun. 1997, pp. 82–101, doi: 10.1109/CCC.1997.612304.

[27] L. Stockmeyer, "The complexity of approximate counting," in *Proc. 15th Annu. ACM Symp. Theory Comput. (STOC)*. New York, NY, USA: Association for Computing Machinery, 1983, pp. 118–126. Accessed: Feb. 18, 2022, doi: 10.1145/800061.808740.

[28] S. Chakraborty, K. S. Meel, and M. Y. Vardi, "A scalable approximate model counter," Jul. 2013, *arXiv:1306.5726*. Accessed: Apr. 27, 2021.

[29] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 1995. Accessed: Nov. 10, 2021. [Online]. Available: https://www.cambridge.org/core/books/randomized-algorithms/6A3E5CD760B0DDBA3794A100EE2843E8, doi: 10.1017/CBO9780511814075.

[30] O. Schrape, M. Andjelkovic, A. Breitenreiter, S. Zeidler, A. Balashov, and M. Krstic, "Design and evaluation of radiation-hardened standard cell flip-flops," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 11, pp. 4796–4809, Nov. 2021. Accessed: Mar. 23, 2022. Available: https://ieeexplore.ieee.org/document/9531952/, doi: 10.1109/TCSI.2021.3109080.

[31] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, "Multi-shot ASP solving with clingo," *Theory Pract. Log. Program.*, vol. 19, no. 1, pp. 27–82, Jan. 2019. Accessed: Jun. 30, 2020. [Online]. Available: https://www.cambridge.org/core/product/identifier/S1471068418000054/type/journal_article, doi: 10.1017/S1471068418000054.

[32] M. Gebser, A. Harrison, R. Kaminski, V. Lifschitz, and T. Schaub, "Abstract gringo," *Theory Pract. Log. Program.*, vol. 15, nos. 4–5, pp. 449–463, Jul. 2015. Accessed: Dec. 12, 2021, doi: 10.1017/S1471068415000150.

[33] A. Breitenreiter, O. Schrape, M. Andjelkovic, and M. Krstic, "Reliability analysis in less than 200 lines of code," presented at the IEEE 12th Latin Amer. Symp. Circuits Syst. (LASCAS), Arequipa, Peru, Feb. 2021.

[34] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and P. Wanko, "Theory solving made easy with clingo 5," in *Proc. ICLP Tech. Commun. (OASICS)*, vol. 52. Wadern, Germany: Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik, pp. 2:1–2:15.

[35] F. Everardo, T. Janhunen, R. Kaminski, and T. Schaub, "The return of *xorro*," in *Logic Programming and Nonmonotonic Reasoning* (Lecture Notes in Computer Science), M. Balduccini, Y. Lierler, and S. Woltran, Eds. Cham, Switzerland: Springer, 2019, pp. 284–297, doi: 10.1007/978-3-030-20528-7_21.

[36] F. Everardo, M. Hecher, and A. Shukla, "An approximate model counter for ASP," in *Proc. 18th Int. Workshop Non-Monotonic Reasoning (NMR)*, 2020, pp. 208–216.

[37] M. Soos and K. S. Meel, "BIRD: Engineering an efficient CNF-XOR SAT solver and its applications to approximate model counting," in *Proc. AAAI Conf. Artif. Intell.*, Jul. 2019, vol. 33, no. 1, pp. 1592–1599. Accessed: Jul. 17, 2019. [Online]. Available: Available: https://ojs.aaai.org/index.php/AAAI/article/view/3974, doi: 10.1609/aaai.v33i01.33011592.

[38] M. Soos, S. Gocht, and K. S. Meel, "Tinted, detached, and lazy CNF-XOR solving and its applications to counting and sampling," in *Computer Aided Verification* (Lecture Notes in Computer Science), S. K. Lahiri and C. Wang, Eds. Cham, Switzerland: Springer, pp. 463–484.

[39] S. Chakraborty, K. S. Meel, and M. Y. Vardi, "Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls," in *Proc. 25th Int. Joint Conf. Artif. Intell. (IJCAI)*. New York, NY, USA: AAAI Press, Jul. 2016, pp. 3569–3576.

[40] L. Chen, M. Ebrahimi, and M. B. Tahoori, "Formal quantification of the register vulnerabilities to soft error in RTL control paths," *J. Electron. Test.*, vol. 31, no. 2, pp. 193–206, Apr. 2015, doi: 10.1007/s10836-015-5519-3.

[41] F. Najm and I. Hajj, "The complexity of fault detection in MOS VLSI circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 9, no. 9, pp. 995–1001, Sep. 1990. Accessed: Mar. 22, 2022. [Online]. Available: http://ieeexplore.ieee.org/document/59075/, doi: 10.1109/43.59075.

**ANSELM BREITENREITER** received the M.Sc. degree in computer engineering from Technical University Berlin, Germany, in 2015. After his graduation, he joined IHP, Frankfurt (Oder), Germany, where he is currently a member of the Prof. Dr. Milos Krstic' Department of System Architectures and Fault Tolerant Computing. His research interests include methods for fault susceptibility analysis and partial fault tolerance and their integration into the digital design flow.

**MARKO ANDJELKOVIĆ** received the Dipl.-Ing. degree in electronics from the Faculty of Electronic Engineering, University of Nis, Serbia, in 2008. Since 2010, he has been a Scientific Researcher at the Faculty of Electronic Engineering in Nis, where he was working on characterization of custom-made dosimeters, evaluation of dosimetric properties of commercial semiconductor components, and design of readout electronics for experimental evaluation of various types of dosimeters. Since 2016, he has been with IHP, where he is currently employed as a Research Associate with the System Architectures Department. His research interests include characterization and modeling of radiation-induced effects in digital circuits and rad-hard design.

**OLIVER SCHRAPE** received the Diploma degree in computer science from the Humboldt University of Berlin, Germany, in 2008. Since 2007, he has been with the Department of System Architectures, IHP. His area of work and research interests include high-speed digital design, design automation for differential logic applications, and design methodologies and techniques for fault-tolerant and radiation-hardness-by-design applications.

**MILOŠ KRSTIĆ** received the Dr.-Ing. degree in electronics from the Brandenburg University of Technology, Cottbus, Germany, in 2006. Since 2001, he has been with IHP, Frankfurt (Oder), Germany, where he currently leads the Department of System Architectures. Since 2016, he has also been a Professor of design and test methodology at the University of Potsdam. For the last few years, his work was mainly focused on fault tolerant architectures and design methodologies for digital systems integration. He has been managing many international and national research and development projects at IHP (GALAXY, EMPHASE, IC-NAO, ENROL, RTU-ASIC, SEPHY, DIFFERENT, VHiSSi, and RESCUE). He is also leading and coordinating space activities at IHP. He has published more than 200 journals and conference papers. He has registered nine patents.