Analysis of Short-Circuit Conditions in Logic Circuits

João Afonso
INESC-ID, Portugal
jpa@algos.inesc-id.pt

José Monteiro
INESC-ID / IST, U Lisboa, Portugal
jcm@inesc-id.pt

***Abstract*—The motivation for this paper is the analysis of input conditions that cause a short-circuit in a logic circuit, that is, that create a direct path from the power supply to ground. We model the logic circuit as a graph where edges represent transistors which are either open or closed, function of the input conditions. From this graph we derive a Quantified Boolean Formula (QBF) problem whose solution identifies the existence of a valid input combination that creates a path in the graph between the pair of nodes that represent the power source and ground, without ever enumerating all input combinations. We build the QBF problem incrementally, minimizing the number of active nodes and hence of possible states. In the end, we obtain a relatively simple CNF expression, function only of the circuit inputs, that is handled by a generic SAT solver. We present results that demonstrate the practical applicability of our method on circuit instances that are intractable by alternative methods.**

I. Introduction

In this paper we propose a verification tool that identifies static input variable combinations that create a short-circuit in an logic circuit. A short-circuit represents an undesirable low resistance path between a power supply node V_{dd} and a ground node Gnd, inducing large currents that may ultimately destroy the device. Short-circuits in a logic circuit may be caused by a designer inadvertently connecting two outputs of a module, by non-standard designs assuming certain assumptions not always met, or by parasitic elements introduced by the fabrication process, identified in post-layout schematic netlists extracted from the layout.

Our approach models the logic circuit as a graph. Since we are considering a steady state analysis, we consider any resistors and inductors as wires and capacitors as open circuits. Hence, the graph represents only the transistors, which are treated as switches. The problem we solve is then to determine the input conditions that close the switches to create a path between V_{dd} and Gnd.

Enumerating paths in a graph is prohibitively expensive. Instead, we implicitly consider all possible paths in the graph by using local conditions for each transistor in the circuit. Then, we formulate the problem as a Quantified Boolean Formula (QBF [10]), looking for input conditions for which *all* state configurations present a transistor in short-circuit.

QBF solvers are still computationally expensive [8]. In the case at hand, all combinations of states of the internal nodes of the circuit need to be considered. We circumvent this tremendous complexity level by building our QBF problem incrementally and converting it into a SAT problem. We maintain only a subset of the conditions by removing from the model internal variables that have been completely processed. In the end, we obtain a CNF formula where the variables are simply the circuit inputs, which can be solved using a generic SAT solver. The alternative brute force approach to this problem would have be to simulate the circuit for all possible input combinations, which is not a viable option for most circuits.

The applicability of our method is actually much more general than analyzing short-circuits. For once, it is abstract enough to be used in any system modeled after the dynamical graphs we use in the paper. That also means it can be used to find the input conditions that activate electrical connections between any two arbitrary nodes of a circuit

The remainder of this document is organized as follows. Section II gives the context for this work, presenting background material and related work. The algorithm that we propose is described in detail in Sections III and IV. We provide results in Section V to demonstrate the effectiveness of our method on a set of benchmarks. Section VI presents the conclusions and discusses future work.

II. Problem Context

In this section we overview related work on circuit analysis in general, focusing on the sensitization of paths in the circuit in particular, and introduce basic concepts on SAT and QBF.

A. Short-Circuit Analysis

Electronics circuits are increasingly more complex and their development is only possible due to sophisticated tools for both design and test. Before fabrication, circuits need to be thoroughly validated, at many different levels: functionality, delay, power, signal integrity, design rules, etc [6]. In this work we focus on a tool that performs a thorough analysis of a logic circuit to determine if there exists any input value combination that turns on a set of transistors that establishes a direct path connection between the power supply and the ground.

In pure CMOS, gates/modules are designed so that with static inputs no path exists from the power supply to ground. Still, short-circuits can arise with a wrong interconnection of gates. This situation is illustrated in Figure 1. If, for example, the inputs of gate A activate its pull-up network and the inputs of gate B (which do not need to be disjunct from the inputs of A) activate its pull-down network, a direct path from *Vdd* to *Ground* is established.

Logic circuits do not always follow a standard CMOS design and in these cases the potential for a non-expected short-circuit situation increases. To illustrate this point, consider the

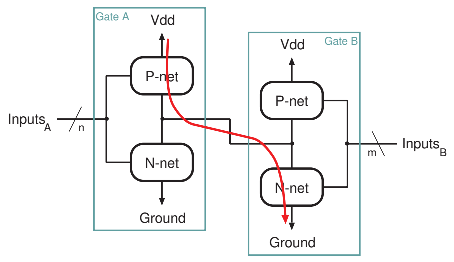


Fig. 1: Potential situation of a short-circuit using pure CMOS gates due to the interconnection of the outputs of two gates.

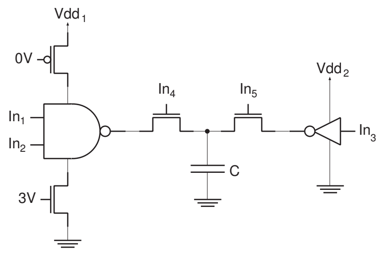


Fig. 2: Non-standard MOS circuit.

circuit depicted in Figure 2. If inputs In_4 and In_5 are not 1 at the same time, there is never a short-circuit condition. But if $In_4 = In_5 = 1$, this condition arises when the output of the two logic gates is different.

However, short-circuit analysis has recently become a pressing issue. The advanced FINFET process nodes have introduced a number of new design, place and route challenges that greatly lengthen the design cycle and consequently the expense of going to lower process nodes. Unintended short circuits due to unintentional hookup errors are already one of the major causes of initial chip failure [3]. At the new process nodes, the place & route of logic libraries can also introduce unintended electrical shorts in the diffusion, poly or metal layers. Checking and enforcing layout coloring rules during place and route should handle the situation where blocks are too close together. But shorts introduced in the other situations require a short-circuit check.

Since we are making a steady-state analysis, we consider circuits described at switch-level, where any resistors and inductors are simple connection lines, capacitors are open circuits and transistors are switches (switches corresponding to NMOS transistors are closed with the true value of the control signal at the gate of the transistor, and switches for PMOS transistors use the complement of this value). Figure 3 presents the switch-level equivalent to the circuit in Figure 2, where the logic gates have been expanded in terms of their internal structure.

Note that, at a more abstract level, we can consider the circuit in Figure 3 as a graph and the problem can be seen as determining a path in this graph whose edges have a weight of 0 or 1 depending on the input combination.

Simulation is impractical since, in order to guarantee that a circuit is short-circuit free, all input combinations need to be

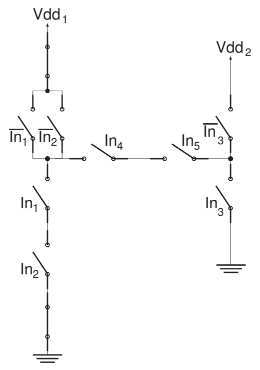


Fig. 3: Switch-level equivalent to the circuit in Figure 2.

evaluated, which is exponential in the number of inputs. The method we propose in this paper takes a different approach by solving the following question: is there a consistent state in the logic circuit that establishes a direct path between the power supply and the ground?

B. Related Work

To the best of our knowledge, the problem of identifying input combinations that cause a short-circuit path in a logic circuit has not been addressed before in the literature. Nor, despite of our best efforts, were we able to find references to similar problems, such as determining paths in a graph whose edges are defined by related logic functions.

We stress that we are addressing a completely different problem from the well-known path sensitization. Path sensitization is used in test pattern generation, timing analysis, and delay fault testing. For example, in stuck-at fault models [6] to compute input values that set a given node of the circuit to 0 or 1, and in timing verification to determine false paths in a circuit [7], namely to confirm that there is indeed an input combination that activates the critical path in the circuit.

Path sensitization consists in finding an input assignment that allows the value of one particular input signal to propagate to a given node in the circuit and thus defining this node's value. Therefore, the problem is defined on a directed graph representing the logic network and linear time algorithms have been presented [2]. Our work addresses the problem of a path defined at the electric circuit level, hence a different and undirected graph, and needs to search over all input combinations to guarantee that none activates a path between two specific nodes in the circuit.

C. SAT and QBF

The Boolean satisfiability problem (generally referred to as SAT) consists in determining if there exists a variable assignment that makes a Boolean logic expression evaluate to true. The logic expression is said to be satisfiable if such assignment exists, and unsatisfiable otherwise. For SAT problems, such expressions are typically expressed in conjunctive normal form (CNF). CNF consists of a conjunction of clauses, and each clause is a disjunction of literals, where a literal represents either a variable or its complement. An example of a logic expression in CNF with three clauses and four variables is:

$$(\overline{a} \vee b) \wedge (a \vee \overline{c} \vee \overline{d}) \wedge (\overline{b} \vee c)$$

An expression in CNF to be considered satisfiable must have all clauses satisfiable. It is easy to see that for this example is satisfiable since the assignment $a = b = c = d = F$ (all false) makes all clauses evaluate to T (true).

Although one of the first NP-complete problems, the continued advances in the algorithms have led to a significant increase in the capacity and efficiency of SAT solvers [4]. For this reason, SAT has seen its application in many domains, in particular, for software and hardware verification.

The Quantified Boolean formula problem (QBF) is a generalization of the SAT problem, through the usage of both existential and universal quantifiers applied to each variable. This expressiveness permits a much more compact encoding for many problems [1]. For the problem we are addressing in this paper, the following QBF problem models the question we are posing:

$$\exists_I \forall_V \ f(I, V) = T \tag{1}$$

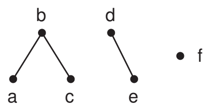
where I is a vector of input signals, V is a vector of over the remaining nodes and $f(I, V)$ is the logic relation that defines the consistent states between all variables in the circuit.

III. QBF Model

Our initial approach was to consider local conditions at each circuit node and device, codified in a fixed limited set of local binary variables, plus a local short-circuit detecting condition. We expected the total number of conditions and variables to grow linearly with the size of the circuit. An efficient SAT solver would then find all the input combinations that would cause a short-circuit.

This approach failed because a short-circuit is not a local phenomena. A short-circuit happens when there is an open low-resistive path through potentially all the circuit. To detect it locally, local conditions need to have the capacity to codify some structure of the circuit. We tried to follow this avenue, but to the best of our ingenuity, all solutions devised had variable local sets of binary conditions growing strongly with the number of nodes and devices in the circuit.

We decided to follow another approach. Consider the following graph, representing paths in a static circuit:



a to f represent nodes with associated binary values, subject to connection conditions: two connected nodes must assume the same logic value. A SAT solver will find all the valid combinations of node values respecting these conditions, by satisfying the relation $(a = b) \wedge (b = c) \wedge (d = e) = T$. Determining if a and b are connected through this formula is trivial given the equality $(a = b)$. If instead we consider a and c, we can deduce the expression $(a = c)$ from the transitivity of equality relations, which is equivalent to searching a path in the graph. In the case of a and d, we would have to test all the paths of graph before concluding that there is no connection.

Alternatively, we can verify that all solutions to the equation above always assume the same value for a and c, while nodes a and d may have different ones.[1]. Hence, connected (a,c) versus disconnected (a,d) nodes are distinguished by:

- $(a = b) \wedge (b = c) \wedge (d = e) \wedge (a \neq c)$ is unsatisfiable
- $(a = b) \wedge (b = c) \wedge (d = e) \wedge (a \neq d)$ is satisfiable

More generally, given a graph, let $\varphi_{a,b}$ be $(a = b)$ if there is an edge connecting nodes a and b in the graph, and $true$ otherwise. To determine if arbitrary nodes x and y are connected or disconnected, the following expression must evaluate to true or false respectively:

$$\forall_{V \in S}, [\,(x \neq y) \wedge \bigwedge_{(a,b)} \varphi_{a,b}(I, V)\,] = F \tag{2}$$

where S represents all possible states over the set of internal nodes (V represents a particular assignment over these nodes) and I represents any input vector. For static graphs, $\varphi_{a,b}(I, V)$ is fixed and depends only on the connected nodes (a,b). For dynamic graphs, the connections are dependent on the vector of external inputs I and even on the values of other graph nodes, represented by the vector V

This is a QBF (quantified Boolean Formula) and a standard way to solve it is by considering a disjunction of the expression over all the possible values V in S, *i.e.*, to replace the universal quantifier in the expression by a disjunction operator:

$$\bigvee_{V \in S} [(x \neq y) \wedge \bigwedge_{(a,b)} \varphi_{a,b}(I, V)] = F \tag{3}$$

We can complement this expression in order to transform it into a satisfiability problem:

$$\bigwedge_{V \in S} [(x = y) \vee \bigvee_{(a,b)} \neg\varphi_{a,b}(I, V)] = T \tag{4}$$

Expression 4 is a conjunction of disjunctions and if we map $(x = y)$ and $\neg\varphi_{a,b}(I, V)$ to single variables, the result is a proper CNF formula, ready to be fed directly to a general SAT solver, and find the I that solve the problem.

Digital circuits using P and N MOSFET devices can be modelled these dynamic graphs, each transistor behaving as a switch between two nodes, *Drain* (d) and *Source* (s), controlled by the value of a third node, *Gate* (g). When g turns the transistor on, s and d are connected, hence $\varphi_{s,d}$ must behave as $(s = d)$. When g turns the transistor off, there is no connection, and $\varphi_{s,d}$ must evaluate to true. This can be described as:

$$\varphi_{s,d}(I, V) = (g\,\text{off}) \vee (s = d) \tag{5}$$

[1]This example addresses the problem of finding connected components in a static graph for which efficient algorithms exist [5]. But our goal are dynamic graphs where the presence of edges is conditional on input values, where these methods do not apply.

Using (5) in formula (4) we obtain:

$$\bigwedge_{V}[(x = y) \vee \bigvee_{(a,b)} (g\,\text{on}) \wedge (a \neq b)] = T \qquad (6)$$

To turn this into a CNF formula, we require all the input nodes in I to be present only at the gate of the transistors[2]. In this case, literals x, y, a and b assume concrete values and we are able judge the equalities and inequalities for each V. The result is a proper CNF, keeping only the clauses that satisfy $(x \neq y)$, and formed only by the collection of control variables $(g\,\text{on})$ for which the conjugated $(a \neq b)$ condition in formula (6) was true. Furthermore, many disjunctions will end up including a control variable and its opposite, and may also be omitted from the CNF.

IV. Proposed Methodology

Algorithm 1 presents the pseudo-code for our solution to this problem. It works with a data space comprised by a matrix where each row will correspond to a disjunction, and there is one column for each internal node $v_i \in V$ of the circuit and an additional column collecting all the terms of the disjunction.

This method starts with a minimal circuit (unconnected V_{dd} and Gnd nodes), with the CNF for short-circuit condition set to false, and builds the model iteratively reading the switches one by one. The full set of disjunctions kept at each stage compose the CNF formula for the short-circuits of the circuit built so far. Each disjunction can be interpreted as a list of all the control signals that, if present with a particular state configuration of the nodes, forces "local" short-circuits, *i.e.*, a connection between nodes with different values. This allows a surprising interpretation for the conjugation of all these disjunctions: the inputs that forces a short-circuit are the ones for which all the possible internal states presents "local" short-circuits.

[2]If an input does not comply, it can be buffered using an additional pair of transistors.

Algorithm 1 Pseudo-code for the proposed algorithm.

```
1:  Initialization of the data space to a single clause
2:  for each switch $(g, s, d)$ read from the netlist do
3:      for each new internal node in the line do
4:          Add a new column mapped to the node
5:          Duplicate all rows
6:          Differentiate them with 1 and 0 in the new
            column
7:      end for
8:      for rows where $s$ and $d$ have different values do
9:          if control signal $g$ is an internal node then
10:             If its value is on, delete the row
11:         else if $\overline{g}$ is present at the clause column then
12:             Delete the row
13:         else
14:             Add $g$ to the clause column
15:         end if
16:     end for
17: end for
18: Simplify the collection of clauses produced
19: Feed them to the SAT solver
```

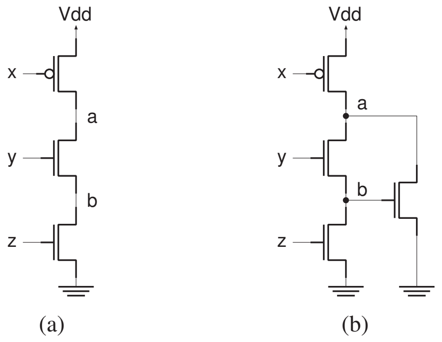


Fig. 4: Simple examples to illustrate the operation of the algorithm.

To clarify the workings of this algorithm we present a simple example. Consider the circuit in Figure 4a, with internal nodes a and b, controlled by input variables $\overline{x}$, y, and z, described by the following netlist:

$\overline{x}$	V_{dd}	a
y	a	b
z	b	Gnd

The model starts with (1,0,F):

V_{dd}	Gnd	Clauses
1	0	F

We read the first line describing a connection V_{dd} to a new node a, and apply the step for one new node (duplicate rows, add column and fill it):

V_{dd}	a	Gnd	Clauses
1	0	0	F
1	1	0	F

We then add the control condition on the rows where V_{dd} and a have different values, signifying a "local" short-circuit:

V_{dd}	a	Gnd	Clauses
1	0	0	$\overline{x}$
1	1	0	F

Next, we read the connection from a to b (new node):

V_{dd}	a	b	Gnd	Clauses
1	0	0	0	$\overline{x}$
1	1	0	0	y
1	0	1	0	$\overline{x} \vee y$
1	1	1	0	F

and from b to Gnd (no new nodes/columns):

V_{dd}	a	b	Gnd	Clauses
1	0	0	0	$\overline{x}$
1	1	0	0	y
1	0	1	0	$\overline{x} \vee y \vee z$
1	1	1	0	z

The conjunction of all these disjunctive clauses is $\overline{x} \wedge y \wedge (\overline{x} \vee y \vee z) \wedge z$ (or $\overline{x} \wedge y \wedge z$ after the simplification step), which is satisfied only for $x = 0$, $y = 1$ and $z = 1$. As expected by inspection of Fig. 4a, the circuit will be at short-circuit only when all control signals $\overline{x}$, y and z are on.

Consider now a circuit with an additional switch, as for example a connection between a and Gnd, controlled by node b, as depicted in Figure 4b. We will get:

V_{dd}	a	b	Gnd	Clauses
1	0	0	0	$\overline{x}$
1	1	0	0	y
1	0	1	0	$\overline{x} \vee y \vee z$
1	1	1	0	$z \vee T$

In the fourth row, the new switch is always on, short-circuiting different logic values at the terminals, hence the row can be deleted as instructed by line 10 of Algorithm 1. The final expression is $\overline{x} \wedge y \wedge (\overline{x} \vee y \vee z) = \overline{x} \wedge y$, satisfied with $x = 0$ and $y = 1$. The short-circuit will only depend on x and y as this implies a short-circuit irrespective of the value of z. This also illustrates how reading lines from the netlist may also decrement the number of rows.

This method is highly parallelizable since the processing of each row does not depend on the others. The last simplification (line 19 of Algorithm 1) is optional because the SAT solver is able to perform this step, and so, it is possible to distribute the rows by different machines only gathering the final result to feed to the SAT solver. This is not however enough to compensate the biggest objection to this method: the seemingly exponential growth of the number of rows with the number of internal nodes.

Each new node duplicates the rows, even if we later prune some or a lot of them. There is no sure good solution to this problem. However, it is possible to play intelligently to minimize this growth. If a node is not present again in the netlist, it will not be of consequence further, and the correspondent column deleted. After this removal, rows with the same node values can be merged, producing the conjunction of the respective clauses. This is possible because the action of the remaining switches will not depend on the deleted columns, and the operation performed are distributive in relation to conjunctions. We call this process "to close a node", and it is central to our method. The improved version is given in pseudo-code in Algorithm 2.

In this case, instead of simple clauses in each row, we may end up with more complex formulas. Still, these are never more costly to keep than the set of clauses which gave birth to them, because if that was the case, we would have kept them instead. As it is, the reduction is usually significant.

Returning to our first example of Figure 4a, after reading the connection of a to b, we may close node a because it is not used anymore. The result would be:

V_{dd}	b	Gnd	Clauses
1	0	0	$\overline{x}$
1	0	0	y
1	1	0	$\overline{x} \vee y$
1	1	0	F

which simplifies to:

V_{dd}	b	Gnd	Clauses
1	0	0	$\overline{x} \wedge y$
1	1	0	F

Algorithm 2 Pseudo-code for the optimized version of the algorithm with closing nodes.

```
1: Initialization of the data space to a single clause
2: for each switch $(g, s, d)$ read from the netlist do
3:     for each new internal node in the line do
4:         Add a new column mapped to the node
5:         Duplicate all rows, one with 1 the other with 0
           in the new column
6:     end for
7:     for rows where $s$ and $d$ have different values do
8:         if the control signal $g$ is an internal node then
9:             If its value is $on$, delete the row
10:        else
11:            Do a disjunction of the previous formula
               with the control signal
12:        end if
13:    end for
14:    for each internal node appearing for the last time do
15:        Merge same state rows conjugating the clauses
16:    end for
17: end for
18: Feed them to the SAT solver
```

Reading the last connection from b to Gnd, controlled by z:

V_{dd}	b	Gnd	Clauses
1	0	0	$\overline{x} \wedge y$
1	1	0	z

The final CNF will be $\overline{x} \wedge y \wedge z$.

The difference is that the total number of rows needed will be limited, not by 2 power to the number of nodes, but 2 to the number of nodes read minus closed nodes at each moment. The topology of the circuit and the right way to read the netlist are essential to minimize the resources needed.

V. Results

We present statistics on a range of transistor level circuits that we generate automatically. The algorithm to generate these circuits produces, repetitively until the desired number of devices has been achieved, a set of, on average, four transistors in series. Each of these sets starts at either the power supply node or any exiting internal node, and end at either ground or an internal node. The control signal of each transistor is either a primary input or an internal node.

Table I presents results for combinations of circuits with 15, 20, 25 and 30 primary inputs (#I), and 200 and 500 internal nodes (#V). For each of these cases we have generated 10 different circuits and the table gives the average (Ave) values measured for these 10 instances together with the worst case (Max). We report the execution time (CPU) in seconds, memory used (Mem) in MBytes, maximum size of the model in terms of number of states (# States) and size of the SAT problem after all the internal states have been quantified out in terms of the number of the CNF clauses (SAT Clauses).

First, note that the program never requires more than a few MBytes of memory to run, and that on average memory

TABLE I: Execution time, memory and problem size statistics for transistor-level circuits.

#I	#V	CPU (s)		Mem (MB)		# States (k)		# SAT clauses	
		Ave	Max	Ave	Max	Ave	Max	Ave	Max
15	200	0.09	0.36	0.2	0.4	2.9	4.0	11.8	23
	500	0.27	0.66	0.2	0.7	3.6	4.6	10.4	30
20	200	0.14	0.26	0.3	0.4	2.9	4.0	18.8	31
	500	0.81	2.26	0.4	0.9	4.0	5.1	15.3	50
25	200	0.29	0.78	0.4	0.7	2.2	4.0	34.4	115
	500	134	1182	17	83	25	196.6	2726	16384
30	200	3.32	15.54	2.0	9.0	2.0	6.9	432	1152
	500	3329	12873	113	256	14490	32756	20989	40960

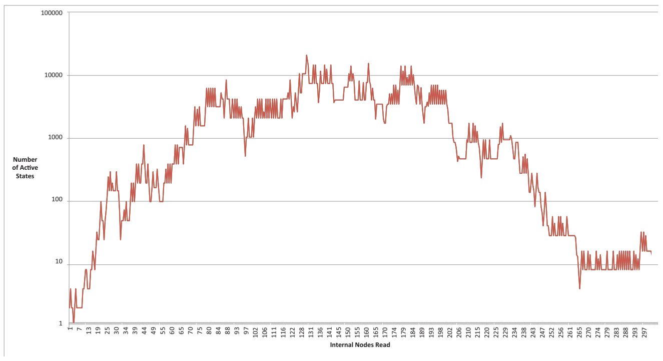


Fig. 5: Evolution of the number of states in the model as devices are read.

requirements are very low. Execution time grows rapidly with the number of internal nodes, and also with the number of primary inputs. It is no surprise that there is a high correlation between execution time and model size. Although we try to minimize the number of active nodes, these tend to increase with circuit size, exponentially increasing the number of internal states. Since there is a condition associated with each internal state that is a function of the primary inputs, the global complexity of the problem grows with this number also. This is to be expected and, unless the number of active internal nodes is kept under control by some clever ordering the devices are read, represents the limitation of the method. One final observation is related to the small size of the final SAT problem, function only of the input variables. We used CryptoMiniSat [9] to solve these problems, whose solution was obtained in less than one minute for all cases.

To illustrate the evolution of the number of internal states as the circuit devices are being read, Figure 5 depicts a representative case. We observe that the number of states starts to grow more or less slowly, reaches a peek and then starts reducing as the final devices close the remaining active nodes.

We believe this set of results demonstrates the applicability of the method on medium sized circuit instances, by effectively controlling the number of active states at any time of its execution. We are working on an optimized version of the tool, and experimenting with better input ordering algorithms, and expect not only to improve significantly execution times but also to extend the range of circuits it can handle.

VI. Conclusions and Future Work

In this paper we have addressed a novel problem, that of identifying input combinations that turn on a set of transistors in a logic circuit which establish a direct path between the power supply and the ground node. We proposed an elegant solution to this problem that avoids both the enumeration of paths in the circuit and of input combinations.

One critical aspect of the method is the sequence that the devices are processed, since this determines the number of active nodes, which in turn defines the number of states in the QBF model, the major limiting factor of the model size that can be handled. Further research needs to be carried out to investigate the best ordering algorithm for real circuits in order to guarantee that the number of states is kept under control.

We are extending the proposed approach to compute "short-circuit models" for generic circuit modules of arbitrary complexity. Then an hierarchical methodology can be applied where these models can be used for each instantiation of the modules, extending the reach of applicability of this work to circuit of arbitrary size.

Acknowledgments

We would like to thank Amelia Shen for bringing this problem to our attention and for the interesting discussions.

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

References

[1] M. Benedetti and H. Mangassarian. Qbf-based formal verification: Experience and perspectives. *Journal on Satisfiability, Boolean Modeling and Computation*, 5:133–191, 2008.

[2] J. Benkoski, E. V. Meersch, L. J. M. Claesen, and H. D. Man. Timing verification using statically sensitizable paths. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9(10):10723–10784, Oct 1990.

[3] S. Chen and K. Liu. Methods for layout verification for polysilicon cell edge structures in finfet standard cells, Sept. 18 2014. US Patent App. 13/840,789.

[4] N. Eén and N. Sörensson. *Theory and Applications of Satisfiability Testing: 6th International Conference, SAT 2003, Selected Revised Papers*, chapter An Extensible SAT-solver, pages 502–518. Springer Berlin Heidelberg, 2004.

[5] J. Hopcroft and R. Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, June 1973.

[6] L. Lavagno, G. Martin, and L. Scheffer. *Electronic Design Automation for Integrated Circuits Handbook - 2 Volume Set*. CRC Press, Inc., 2006.

[7] P. McGeer and R. Brayton. Efficient algorithms for computing the longest viable path in a combinational network. In *Proceedings of the 26th ACM/IEEE Design Automation Conference*, DAC '89, pages 561–567, New York, NY, USA, 1989. ACM.

[8] H. Samulowitz and R. Memisevic. Learning to solve qbf. In *Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 1*, AAAI'07, pages 255–260. AAAI Press, 2007.

[9] M. Soos. Cryptominisat a sat solver for cryptographic problems. `http://planete.inrialpes.fr/ soos/CryptoMiniSat2/`. Accessed: 2016-11-23.

[10] L. Zhang and S. Malik. Conflict driven learning in a quantified boolean satisfiability solver. In *Computer Aided Design, 2002. ICCAD 2002. IEEE/ACM International Conference on*, pages 442–449, Nov 2002.