Efficient (Non-)Reachability Analysis of Counterexamples

Rolf Drechsler[1] Wolfgang Günther[2] Burkhard Stubert[2]

[1]Institute of Computer Science
University of Bremen
28359 Bremen, Germany

[2]CL DAT TDM VM
Infineon Technologies
81730 Munich, Germany

Abstract

Formal verification of complete ASICs with up to several million gates can only be carried out, if the sequential problem is transformed to a combinational one. The circuits that have to be compared are modeled as Finite State Machines (FSMs). But the state encoding has to be (nearly) identical such that a matching of the states can be performed. Then combinational equivalence checking is carried out on the resulting design. If the designs are not equivalent, a counterexample consisting of values for some inputs and registers is returned. The remaining problem is to decide whether this counterexample is (sequentially) reachable, i.e. whether the state s specified in the counterexample is valid.

Since in general the problem of reachability analysis is not easier than sequential equivalence checking, we propose approximation schemes to efficiently show non-reachability of some target states. These schemes only focus on a small part of the sequential circuit and by this significantly reduce run time. Binary Decision Diagrams (BDDs) are used locally for these parts, and no global BDD for the state transition function is required. Experimental results on industrial designs show that using these methods, it is often possible to prove non-reachability of some states within seconds, while a complete construction of the BDD is infeasible.

1 Introduction

Formal verification of large designs is of growing interest, and it is used at various stages of the design flow. Especially *combinational* equivalence checking is successfully applied in industry [9]. If two designs with the same state encoding are not equivalent, commonly used tools return a counterexample, i.e. an assignment of values to the inputs of the designs for which the output values of the designs differ.

Due to the complexity of the problem, the general *sequential* equivalence checking problem is not practical for large designs [3, 14, 12]. However, the combinational approach can also be used for sequential designs as long as the state encoding is not modified: registers of the design are treated as additional (secondary) inputs and outputs, and only the combinational logic is compared. If the designs are combinationally equivalent, they are also sequentially equivalent. Otherwise a counterexample is returned. The problem that arises here is that the designs may be different in a state s which is not sequentially reachable. Since not all of the registers need to be specified, each assignment of values to the unspecified registers is a counterexample, resulting in a set of states S. Therefore, a reachability analysis for this set of states is required, i.e. it has to be shown whether there is at least one state in S that is reachable.

A similar problem arises in Bounded Model Checking (BMC). Assuming the initial state is arbitrary (i.e. it is not the reset state), a BMC solver may return a counterexample that starts in some state s, that is sequentially not reachable.

In contrast to the general problem of a complete reachability analysis where *all* reachable states have to be computed, in this case reachability has to be shown only for *some* states.

It is possible that a counterexample does not specify all inputs and registers. Arbitrary values can be assigned to unspecified registers, resulting in sets of states. Since it is enough that one of these states is reachable, counterexamples where only few registers are specified are more likely to be reachable. Furthermore, in some cases modifying a counterexample by setting some registers to “unspecified” is possible and the resulting counterexample is still valid, i.e. it shows differences of the two designs. Therefore, removing entries of a counterexample is possible as long as the counterexample is still valid. However, this may have an influence on its reachability, since the set of target states is increasing.

Several approaches for approximate reachability checking have been presented so far. Most of them are based on approximation techniques on BDD [15, 14, 5, 10]. In [11], an approach to speed up simulation-based verification has been presented.

In this paper, we present a simple algorithm to compress counterexamples originating from combinational equivalence checking. It is based on simulation and it can be performed very fast. Then a technique is described to prove the reachability or non-reachability of a counterexample. An approach based on random pattern simulation is used to prove reachability. This is done for two reasons. First, counterexamples that are easy to reach can be found quickly. Second, additional information which can be used later to speed up the BDD-based approach can be obtained without causing much overhead.

Using the simulation-based approach, it is only possible to prove reachability but not non-reachability. For non-reachability, an approach based on BDDs is proposed. Basically, a backward reachability analysis is carried out on the transitive fanin cone of the counterexample. Two over-approximation schemes are described that focus on proving non-reachability. The first one is based on ignoring parts of the counterexample that are easy to reach. Information gained from the simulation approach is used here. The second one is based on replacing registers in the fanin cone by input variables. This approach not only reduces the number of state variables that are necessary, but also reduces the size of the BDD for the state transition function. Especially for designs containing many registers, this approach is very powerful. This iterated technique targets very large circuits, since the method is fully automatic and is integrated in an industrial flow. Due to the size of the circuits considered it is important to have simple and fast core algorithms, like simulation, while symbolic techniques based on BDDs should be avoided, if possible.

Experimental results on large industrial designs are given, showing that the BDD-based approach is very fast in case the approximation techniques can be used.

The paper is structured as follows. First, some basic definitions are given. Compaction of counterexamples is described in Section 4. The simulation-based approach is presented in Section 5.1. In

Section 5.2, the symbolic approach and the approximation techniques are described. Finally, some experimental results are given.

2 Preliminaries

Definition 1 *A Finite State Machine (FSM) is a tuple $(I, O, S, s_0, \delta, \lambda)$, where I, O, and S are non-empty sets of input labels, output labels, and states, respectively. s_0 is an initial state. $\delta : S \times I \rightarrow S$ is the state transition function and $\lambda : S \times I \rightarrow O$ is the output function. When the FSM is in a current state $s \in S$ and receives an input $i \in I$, the next state is given by $\delta(s, i)$ and the output is given by $\lambda(s, i)$.*

A counterexample is given by an assignment of values to a subset of the input and state variables. This can be seen as an assignment of values to all input and state variables,

$$i_{ce} = (i_1, \ldots, i_n), s_{ce} = (s_1, \ldots, s_k),\ i_j, s_j \in \{0, 1, dc\},$$

using a non-Boolean don't care value dc. For this counterexample, some property is violated. For instance, in equivalence checking, for the two FSMs under consideration, it holds

$$\delta_1(s_{ce}, i_{ce}) \neq \delta_2(s_{ce}, i_{ce}) \text{ or } \lambda_1(s_{ce}, i_{ce}) \neq \lambda_2(s_{ce}, i_{ce})$$

for all possible extensions of the counterexample (i.e. replacing dc by some Boolean constant). In BMC, some property of a design is violated.

The problem of reachability analysis is to determine whether a sequence of input assignments, i_t, $t \in \mathbf{N}$, exists such that

$$\delta(\cdots \delta(\delta(s_0, i_0), i_1) \cdots) = s_{ce}$$

for the initial state s_0 of the FSM.

3 Problem Description and Basic Ideas

In the case of combinational equivalence checking we assume that all inputs, outputs and states are matched. Thus, the sequential problem is transferred to a combinational one. But the states are considered as primary inputs in the new model. This may result in invalid counterexamples, since states are used that are not reachable during normal operation of the sequential circuit.

To guarantee robust behavior of verification tools, it is important that these counterexamples can be detected and eliminated. In the following a step approach is used:

1. Compaction of counterexamples: To reduce the complexity of the proof process, the size of the counterexample is reduced.

2. Reachability analysis: Based on the reduced counterexample the reachability or the non-reachability of the counterexample is proven. For large designs a complete reachability analysis cannot be carried out. For this, approximation techniques are applied.

If a counterexample is proven to be non-reachable, constraints can be generated that are given to the equivalence checker in the next run.

4 Compaction of Counterexamples

In this section we focus on combinational equivalence checking and present a way to pre-processing to simplify the reachability check.

As long as a counterexample fulfills the property that two designs under consideration are different, i.e.

$$\delta_1(s_{ce}, i_{ce}) \neq \delta_2(s_{ce}, i_{ce}) \text{ or } \lambda_1(s_{ce}, i_{ce}) \neq \lambda_2(s_{ce}, i_{ce}),$$

some entries can be replaced by the value dc. A counterexample containing more dc entries for state variables is more likely to be reachable, since less state-bits require a specified value. Therefore, compaction of a counterexample in general is desired.

The following simple algorithm can be used to reduce the size of counterexamples: first, for each state variable s_j in s_{ce} not having value dc it is checked whether its value can be replaced by dc (i.e. whether it can be removed from the counterexample). In a second step, input variables are replaced by dc values. A three-valued simulator is used to check whether a counterexample is valid.

Another way to decide whether a counterexample is valid is to use provers (which are based on a combination of different algorithms, e.g. BDDs and SAT) to show that both output values of the designs are constant and that they are different.

More sophisticated methods can be used here, but the basic idea is to get reductions at very low cost. Our experiments have shown that the described simple technique is sufficient.

5 Reachability Analysis of Counterexamples

The reachability analysis is divided into two phases. First, a simulation-based approach is used to identify counterexamples that are reachable. During this phase, also some information about the "difficulty" of the registers is gained, i.e. how difficult it is to obtain the value specified in the counterexample for each register.

In a second phase, the main focus is to prove non-reachability. Two approximation techniques are used to avoid BDD blow-ups, using the information obtained by the previous phase.

5.1 Simulation-based Reachability Analysis

To decide reachability of a given counterexample, first an approach based on random pattern simulation is used. Initially, all registers have an unknown value, i.e. $s_0 = (dc, \ldots, dc)$. Then the successor state is computed using randomly chosen values for the inputs until the target state s_{ce} is reached or a given number of iterations is exceeded.

The motivation for this approach is the following: usually, after compaction counterexamples contain only few entries, and therefore this approach can detect reachable counterexamples very quickly. Furthermore, additional information can be gained during simulation, which can later be used to speed up the BDD-based approach which is described in the next section.

Note that using this approach, it is only possible to show reachability of a counterexample; if it is not reachable, no definite conclusion is possible. To prove non-reachability, more powerful methods have to be used.

```
**BackwardReachability**(Counterexample $ce$) {
    $newStates = targetStates = ce$;
    *while* ($targetStates \neq \mathbf{B}^k$) {
        $previousStates = \mathbf{getPreviousStates}(targetStates)$;
        $newStates = previousStates \setminus targetStates$;
        *if* ($newStates == \emptyset$) {
            return “Counterexample is not reachable”;
        }
        $targetStates = targetStates \cup newStates$;
    }
    return “Counterexample is reachable”;
}
```

Figure 1. Backward Reachability

5.2 BDD-based Reachability Analysis

Backward traversal is used to decide reachability. In each iteration the set of states from which the target states are reachable is computed and added to the set of target states. This computation is done symbolically: both the state transition function and sets of states are represented by BDDs. This method can determine both reachability and non-reachability of counterexamples.

Starting from the set of states given in the counterexample, a set of previous states is computed and added to the state set, as long as new states can be added. If the initial state is among the reached states, the target states are reachable. Otherwise, they are not. Since in practice the initial state may not exist or may be unknown, we use a slightly different criterium: the target states are reachable, if and only if all states have been reached. If the FSM has a reset state, i.e. a state that can be reached from any state of the FSM, and if the reset state is equal to the initial state, this criterium is equivalent to the above one.

In the following, the basic algorithm for backward reachability is described in more detail and it is discussed which problems may emerge. Then an approximation scheme that focuses on non-reachability is described.

5.2.1 Backward Reachability

For backward reachability, a standard iterative algorithm is used, see Figure 1.

Basically, there are two main approaches to compute the set of previous states. The first one is based on transition relations [2], the second one uses the state transition function directly [7]. A combination of both approaches has been presented in [13]. One drawback of the transition relation is that an additional set of state variables is necessary, slowing down dynamic reordering significantly. Furthermore, the transition relation is usually very large, and therefore sophisticated partitioning methods have to be used [16, 1, 6, 4, 14]. We use the approach based on transition functions. A sketch is given in Figure 2. After the computation, input variables have to be quantified existentially in the resulting set.

In practice, however, problems may arise due to two main problems:

```
**getPreviousStates**(BDD for state set $s$) {
    *if* ($s = \emptyset$ or $s = \mathbf{B}^k$) return $s$;

    $x =$ top variable of $s$;
    $t = \mathbf{getPreviousStates}(s_{x=1})$;
    $e = \mathbf{getPreviousStates}(s_{x=0})$;
    return $(x \cdot t + \overline{x} \cdot e)$;
}
```

Figure 2. Backward traversal based on state transition functions

A. the BDD for the state transition function may be too large, or

B. the BDD for the set of target states may be too large.

In the next two sections, over-approximation algorithms are described which try to overcome the problem of huge BDD sizes and which focus on proving non-reachability. They are both based on considering only a part of the sequential circuit, and trying to prove non-reachability within this part. This also means that the BDD for the state transition function and intermediate sets of states are much smaller.

5.2.2 Focus on Difficult Latches

Some of the registers may take the value required for the counterexample frequently, while some others match very seldom. This is the motivation to focus on the latter registers first: if these registers cannot meet their requirements, non-reachability is proven. On the other hand, it is necessary to do a full reachability analysis, if they *can* take the correct values.

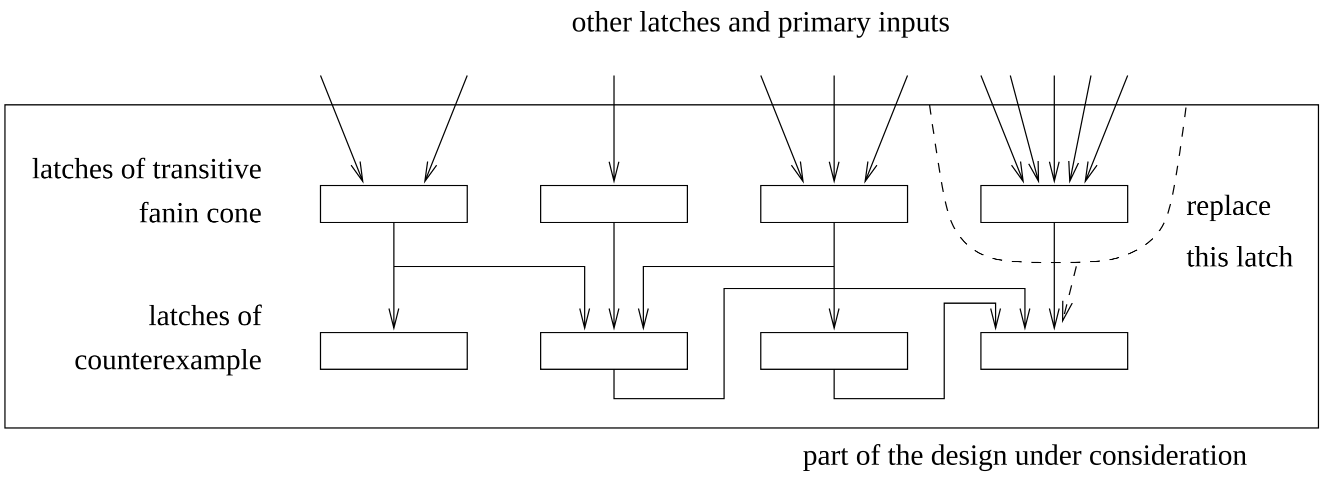
Therefore, in a first step some of the "difficult" registers are selected and all others are set to dc. Reachability is examined on this modified counterexample which is easier to handle. If non-reachability could not be proven, in a second step the reachability analysis is carried out on the full counterexample.

To determine how frequently a register has the value of the counterexample, the simulation-based approach of Section 5.1 is called before, and a counter for each register is updated each time the value of the register is equal to the value given in the counterexample.

5.2.3 Replacing Latches by Inputs

A second over-approximation scheme is based on replacing registers by inputs. If a register is replaced by an input, its value can arbitrarily be chosen, independent of the previous state of the FSM. Therefore, more states are reachable. Again, if the counterexample is not reachable in the over-approximation, it is definitely not reachable. Otherwise, no decision can be made. This approach is similar to the one proposed in [11].

To select registers for replacement, in a first step all registers in the transitive fanin cone of the counterexample are put into a priority queue, sorted by the number of variables of which the next state function depends on. Then iteratively the register with the largest number of variables in its support



other latches and primary inputs
latches of transitive fanin cone
replace this latch
latches of counterexample
part of the design under consideration

Figure 3. Replacing Latches by Inputs

is replaced by an input. Latches contained in the counterexample are never replaced by this approach, since this is already done in the previously described over-approximation scheme.

Even if all of the registers which are not contained in the counterexample are replaced, the resulting circuit may still be sequential, since registers of the counterexample may depend on other registers of the counterexample (see Figure 3). However, these loops are usually very short, and only few iterations are necessary for the reachability analysis.

Note that using the size of the fanin cone is only one possibility to select the registers. Many other criteria are also possible, like for instance to use a BFS run starting from the counterexample to order the registers.

5.3 Overall Algorithm

The algorithm starts with the random pattern simulation described in Section 5.1. If reachability could not be proven, the approach based on BDDs is used. First, the maximum degree of over-approximation is used, i.e. only very few of the registers contained in the counterexample are considered, and all other registers are replaced by input variables. Then successively the calculation gets more precise, until non-reachability is proven. In the BDD-based approach, reachability of a counterexample can only be proven in the last iteration, where no over-approximation techniques are used.

In the experiments, the following sequence of over-approximations showed good results:

1. Ignore $3/4$ of the counterexample, replace all other registers.
2. Ignore $3/4$, replace $3/4$ of the other registers.
3. Ignore $3/4$, replace $1/4$ of the other registers.
4. Ignore $3/4$, replace none of the other registers.
5. Use the full counterexample, replace all of the other registers.
6. Use the full counterexample, replace none of the other registers.

Table 1. Experimental results: Compression of Counterexamples

circuit	in	out	registers	gates	orig size	compressed
indust-1	285	27	2,335	27,305	25	20
indust-2	226	12	244	3,966	9	7
indust-3	48	1,359	1,395	17,097	67	67
indust-3	48	1,359	1,395	17,097	52	52
indust-3	48	1,359	1,395	17,097	67	67
indust-4	316	594	766	13,067	81	81
indust-4	316	594	766	13,067	115	115
indust-5	381	7	614	24,298	14	14
indust-5	381	7	614	24,298	13	13
indust-6	38	48	210	4,870	12	12
indust-6	38	48	210	4,870	28	20
indust-7	2843	4178	150,215	1,914,512	10	10

The factors $1/4$ and $3/4$ have been determined by experiments.

In many cases, non-reachability can already be shown within the first three steps. Then, it is not necessary to build the BDD for all registers of the counterexample, and therefore prove non-reachability also for large designs.

6 Experimental Results

In this section we describe experimental results that have been carried out on a 450 MHz *SUN Ultra 80*. We used a memory limit of 512 MBytes. All run times are given in CPU seconds. The algorithm has been implemented in C++ and has been integrated in the Infineon verification environment CVE [8, 9]. Several industrial designs containing errors have been used. The first 6 designs have been given on the register-transfer level, while the last one is a flat netlist.

The results showing the effectiveness of the overall algorithm are given in Table 1 and 2. In Columns 2 to 5, the number of primary inputs, outputs, registers, and gates are given, respectively. In Table 1, the size of the counterexample before and after compression is given in the next two columns. It can be seen that usually the counterexamples are very small compared to the total number of registers. Furthermore, in some cases the sizes can be further reduced by the simulation-based approach described in Section 4.

The reachability results without and with approximation techniques are given in Table 2. In many cases BDDs can be built for the whole transitive fanin cone of the counterexamples. In most cases the approximation techniques work very well, and a speed-up of several orders of magnitude can be observed. Even if the BDD for the state transition function cannot be built, non-reachability can be quickly decided.

Table 2. Experimental results: Reachability

				wo/ approx.		w/ approx.	
circuit	in	registers	gates	reach	time	reach	time
indust-1	285	2,335	27,305	BDD blowup		no	47.98
indust-2	226	244	3,966	yes	0.27	yes	0.27
indust-3	48	1,395	17,097	yes	26.08	yes	56.12
indust-3	48	1,395	17,097	yes	22.58	yes	26.22
indust-3	48	1,395	17,097	yes	23.64	yes	100
indust-4	316	766	13,067	no	2349	no	3.03
indust-4	316	766	13,067	no	2303	no	3.41
indust-5	381	614	24,298	no	1181	no	982
indust-5	381	614	24,298	no	710	no	634
indust-6	38	210	4,870	no	23.07	no	2.48
indust-6	38	210	4,870	no	90	no	16.41
indust-7	2843	150,215	1,914,512	BDD blowup		no	0.77

7 Conclusions

We presented a framework to prove non-reachability of counterexamples originating from combinational equivalence checking and bounded model checking. In a first phase, the focus is to prove reachability using simulation. During this phase, also some statistical information is gathered. In a second phase, a local BDD-based backward reachability analysis is carried out. Approximation techniques use the statistical information of the previous phase to prove non-reachability of the counterexample.

The main focus of this approach is to avoid some of the false negatives reported by formal verification tools. This is one of the drawbacks of combinational equivalence checking or bounded model checking. By proving that a counterexample is sequentially unreachable, the usability and reliability of formal verification tools can be significantly improved.

References

[1] J.R. Burch, E.M. Clarke, and D.E. Long. Representing circuits more efficiently in symbolic model checking. In *Design Automation Conf.*, pages 403–407, 1991.

[2] J.R. Burch, E.M. Clarke, K.L. McMillan, and D.L. Dill. Sequential circuit verification using symbolic model checking. In *Design Automation Conf.*, pages 46–51, 1990.

[3] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[4] G. Cabodi, P. Camurati, L. Lavagno, and S. Quer. Disjunctive partitioning and partial iterative squaring: An effective approach for symbolic traversal of large circuits. In *Design Automation Conf.*, pages 728–733, 1997.

[5] G. Cabodi, P. Camurati, and S. Quer. Improving symbolic reachability analysis by means of activity profiles. *IEEE Trans. on CAD*, 19(9):1065–1075, 2000.

[6] H. Cho, G.D. Hachtel, E. Macii, B. Plessier, and F. Somenzi. Algorithms for approximate FSM traversal. In *Design Automation Conf.*, pages 25–30, 1993.

[7] O. Coudert and J.C. Madre. A unified framework for the formal verification of sequential circuits. In *Int'l Conf. on CAD*, pages 126–129, 1990.

[8] R. Drechsler. GateComp: equivalence checking in CVE. In *ITG/GI/GMM-Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen"*, 2001.

[9] R. Drechsler and S. Höreth. Gatecomp: Equivalence checking of digital circuits in an industrial environment. In *Int'l Workshop on Boolean Problems*, pages 195–200, 2002.

[10] A. Hett, C. Scholl, and B. Becker. Distance driven finite state machine traversal. In *Design Automation Conf.*, pages 39–42, 2000.

[11] P.-H. Ho, T. Shiple, K. Harer, J. Kukula, R. Damiano, V. Bertacco, J. Taylor, and J. Long. Smart simulation using collaborative formal and simulation engines. In *Int'l Conf. on CAD*, pages 120–126, 2000.

[12] J.-H.R. Jiang and R.K. Brayton. On the verification of sequential equivalence. *IEEE Trans. on CAD*, 22(6):686–697, 2003.

[13] I.-H. Moon, J. Kukula, K. Ravi, and F. Somenzi. To split or to conjoin: The question in image computation. In *Design Automation Conf.*, pages 23–28, 2000.

[14] A. Narayan, A. Isles, J. Jain, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. Reachability analysis using Partitioned-ROBDDs. In *Int'l Conf. on CAD*, pages 388–393, 1997.

[15] K. Ravi and F. Somenzi. High-density reachability analysis. In *Int'l Conf. on CAD*, pages 154–158, 1995.

[16] H. Touati, H. Savoj, B. Lin, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. Implicit enumeration of finite state machines using BDDs. In *Int'l Conf. on CAD*, pages 130–133, 1990.