Preprint from Proceedings of GI/ITG/GMM-Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen", Erlangen, Germany, March 2007

Counterexample Generation for Incomplete Designs

Tobias Nopper Christoph Scholl

{nopper, scholl}@informatik.uni-freiburg.de
Institute of Computer Science, Albert-Ludwigs-University Freiburg, Germany

Counterexample generation is a crucial task for error diagnosis and debugging of sequential circuits. The simpler a counterexample is – i.e. more general and fewer assigned input values – the better it can be understood by humans.

We will use the concept of *Black Boxes* – parts of the design with unknown behavior – to mask out components for counterexample computation. By doing this, the resulting counterexample will argue about a reduced number of components in the system in order to facilitate the task of understanding and correcting the error effect.

1 Introduction

Given a sequential circuit and properties in some temporal logic like CTL or LTL, model checking is a method for verifying these properties [1, 2]. In the early nineties, by introducing *symbolic* model checking, Burch et al. substantially extended the class of systems which can be verified [3, 4]. In (unbounded) symbolic model checking, binary decision diagrams (BDDs) [5] or and-inverter graphs (AIGs) [19] are used both for state set representation and for state traversal. In the last few years SAT based techniques like Bounded Model Checking (BMC) [6, 7] have also attracted much interest. BMC applied to certain properties (safety properties or, more generally, LTL formulas) 'unfolds' the transition relation for k steps and checks whether there is a run of length k which does not fulfill the specified property. If such a run does not exist, then k is increased and BMC is used again. However, for *proving* properties using BMC, a suitable upper bound on k is needed. In the case of safety properties, e.g., the procedure can be stopped, when k equals the *diameter* of the system, i.e. the maximum length of all shortest paths between states in the system. In this case, BMC ends up with a proof of the property.

In this paper we address an important feature of *unbounded symbolic* model checkers, the generation of counterexamples for cases when the specified property does not hold. If the property does not hold, counterexamples are an important means for the designer, since they explain the error by showing a run of the system violating the property and may guide the designer to a correction of the error in the system. In that way the strength of model checking as a verification technique largely relies on the counterexamples produced for failing properties.

Standard methods for counterexample generation implemented in model checkers like SMV [3, 4] produce a sequence of assignments to all input signals defining an erroneous run of the system. Thus, a consideration of all parts of the system is needed in order to understand and reproduce the error effect in the current design. In this paper we consider a

method for constructing counterexamples arguing about a reduced number of components of the system in order to facilitate the task of understanding and correcting the error effect.

The goals of our work are most closely related to approaches in the Bounded Model Checking (BMC) context which try to improve a counterexample produced by a SAT solver in BMC (see [8]). Starting from a counterexample which specifies all signals at all times prior to the error, [8] tries to remove ('lift') assignments to input signals without losing the property that the remaining assignments imply a violation of the property.

In contrast, our approach to the improvement of counterexamples in symbolic model checking is based on Black Box model checking methods [9] where components of a system are removed and replaced by so-called Black Boxes. If Black Box model checking is able to prove that the property is violated *independently from the implementation of the Black Boxes*, then we know that those parts of the design which were replaced by Black Boxes are not important for explaining the error effect and we have successfully reduced the number of components to be considered when analyzing the error. In our current approach Black Boxing is performed interactively based on conjectures of the user about which parts of the design could potentially be irrelevant for the error. Black Boxing may be performed step by step until further replacements of components by Black Boxes would lead to the situation that Black Box model checking is not able to prove a violation of the property independently from the Black Box implementation.

Based on existing methods for Black Box model checking [9], we had to develop methods for counterexample generation in this context. In Section 3 however we show that a straightforward generalization of counterexample generation from classical symbolic model checking to Black Box model checking will not meet our demands. The reason for this lies in the fact that counterexamples generated by the straightforward method may still argue about the signals at the interface of the Black Boxes and different implementations of the Black Boxes may lead to different counterexamples. Hence we had to develop more sophisticated methods which produce 'uniform' counterexamples that do not depend on the behavior implemented by the Black Boxes.

In contrast to previous approaches dealing with unknown initial states [10, 11], we have to cope with different possible behaviours of the system due to different Black Box implementations since the "uniform counterexample" is defined to be a counterexample *for all* possible Black Box implementations.

Of course, our novel methods for counterexample generation can also be used for other application scenarios of partial designs with Black Boxes, e.g. in applications where Black Boxes represent parts of the design which are not yet present in early phases of the design process [9].

The paper is structured as follows: Section 2 reviews the main ideas of Black Box model checking which is a necessary ingredient for our counterexample optimization. Then, Sect. 3 introduces our approach generating 'uniform', minimized counterexamples and presents some interesting theoretical results wrt. the length of uniform counterexamples. Experimental results demonstrating counterexample minimization for a VLIW processor design are presented in Sect. 4 and, finally, Sect. 5 summarizes the paper and gives directions for further research.

2 Preliminaries

2.1 Incomplete Designs

In this section we briefly review symbolic representations of incomplete designs.

Unknown parts of the design can be combined into so-called 'Black Boxes' (see Fig. 1 for a small sequential example with one Black Box).

For simulating the combinational part of a sequential circuit defining the transition function δ wrt. some input vector we can make use of the ternary $(0, 1, X)$-logic [12, 13]: We assign a value X to each output of the Black Box (since the Black Box outputs are unknown) and we perform a conventional $(0, 1, X)$-simulation [14]. If the value of some primary output is X, we do not know the value due to the unknown behavior of the Black Boxes.

For a symbolic representation of the incomplete circuit we model the additional value X by a new variable Z as in [15, 13]. For each output g_i of the incomplete design with primary input variables $x_1, \ldots, x_n$ we obtain a symbolic representation of g_i by using a slightly modified version of symbolic simulation with

$$g_i|^{x_1=\epsilon_1}_{\ldots}{}_{x_n=\epsilon_n} = \begin{cases} 1, \text{ if the (0,1,}X\text{)-simulation with input } (\epsilon_1, \ldots, \epsilon_n) \text{ produces } 1 \\ 0, \text{ if the (0,1,}X\text{)-simulation with input } (\epsilon_1, \ldots, \epsilon_n) \text{ produces } 0 \\ Z, \text{ if the (0,1,}X\text{)-simulation with input } (\epsilon_1, \ldots, \epsilon_n) \text{ produces } X \end{cases}$$

This modified version of symbolic simulation is called *symbolic (0,1,X)-simulation*.

Since $(0, 1, X)$-simulation cannot distinguish between unknown values at different Black Box outputs, some information is lost in symbolic $(0, 1, X)$-simulation. This problem can be solved at the cost of additional variables: Instead of using the same variable Z for all Black Box outputs, symbolic Z_i-simulation introduces a new variable Z_i for each Black Box output and performs a (conventional) symbolic simulation.

A flexible representation of incomplete circuits which allows some Black Box outputs to be represented as in symbolic $(0, 1, X)$-simulation and some Black Box outputs as in symbolic Z_i-simulation has been presented in [16]: For each output of the Black Boxes, which is to be handled as in symbolic $(0, 1, X)$-simulation, we use the variable Z to model the Black Box output, while for each output of the Black Boxes, which is to be handled by symbolic Z_i-simulation we use a new Z_i variable. This *flexible symbolic simulation* now considers the latter Black Box outputs as additional inputs and then performs a symbolic $(0, 1, X)$-simulation.

Symbolic representations of the circuit outputs g_i are obtained with primary input variables $x_1, \ldots, x_n$, (Z_i-simulated) Black Box outputs $Z_1, \ldots, Z_m$ and the Z-variable as inputs:

$$g_i|_{\substack{x_1=\epsilon_1,\ldots,x_n=\epsilon_n \\ Z_1=\eta_1,\ldots,Z_m=\eta_m}} = \begin{cases} 1, \text{ if the (0,1,}X\text{)-simulation with input } (\epsilon_1, \ldots, \epsilon_n, \eta_1, \ldots, \eta_m) \text{ produces } 1 \\ 0, \text{ if the (0,1,}X\text{)-simulation with input } (\epsilon_1, \ldots, \epsilon_n, \eta_1, \ldots, \eta_m) \text{ produces } 0 \\ Z, \text{ if the (0,1,}X\text{)-simulation with input } (\epsilon_1, \ldots, \epsilon_n, \eta_1, \ldots, \eta_m) \text{ produces } X \end{cases}$$

2.2 Fixed and Possible Transitions

For incomplete designs, we have to consider Black Boxes in the functional block defining the transition function δ. For this reason, two types of transitions were defined for symbolic model checking of incomplete designs [9]: These are

- transitions which exist independently from the replacement of the Black Boxes, i.e. for all possible replacements of the Black Boxes (called 'fixed transitions') and
- transitions which may or may not exist in a complete version of the design – depending on the implementation for the Black Boxes (called 'possible transitions').

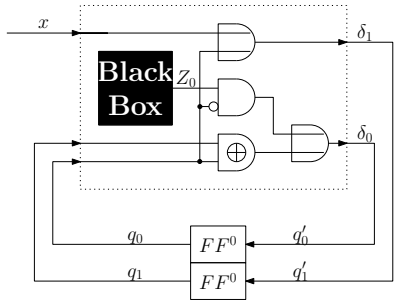

Figure 1: An exemplary sequential circuit with one Black Box. Initially, $q_1 = q_0 = 0$.

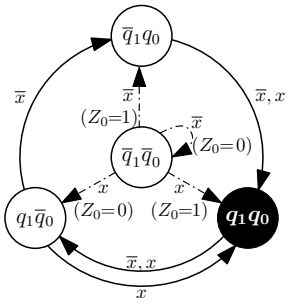

Figure 2: Automaton of the circuit in Fig. 1 with fixed transitions (solid arrows) and possible transitions (both solid and dashed arrows). $\overline{q}_1\overline{q}$ is the initial state, the safety property $AG(\neg q_0 \vee \neg q_1)$ is violated in state $q_1 q_0$.

Let there be a number of Black Boxes with outputs modeled by Z and some other Black Boxes with outputs modeled by Z_i's. Thus, we introduce new variables Z and $\vec{Z} = (Z_1, Z_2, \dots)$. The symbolic simulation described above now provides a symbolic representation of the transition functions $\delta_j(\vec{q}, \vec{x}, Z, \vec{Z})$.

If $\delta_i|_{\vec{q}=\vec{q}^{\,\text{fix}},\, \vec{x}=\vec{x}^{\,\text{fix}}} = 1$ for some state $(\vec{q}^{\,\text{fix}}, \vec{x}^{\,\text{fix}}) \in \mathbb{B}^{|\vec{q}|\times|\vec{x}|}$, then we know that δ_i is definitively 1 in this state independent of the replacement of the Black Boxes. If $\delta_i|_{\vec{q}=\vec{q}^{\,\text{fix}},\, \vec{x}=\vec{x}^{\,\text{fix}}} = 0$, then the output δ_i is definitively 0 in this state independent of the replacement of the Black Boxes.
Based on this, $(\vec{q}, \vec{x}, \vec{q}\,')$ is put into the set of fixed transitions R_A if for all i: $\delta_i(\vec{q}, \vec{x})$ definitely equals the value of q_i' for all Black Box substitutions. This leads to the symbolic representation

$$\chi_{R_A}(\vec{q}, \vec{x}, \vec{q}') = \forall\vec{Z}\Big(\prod\nolimits_{i=0}^{|\vec{q}|-1} \forall Z\big(\delta_i(\vec{q}, \vec{x}, Z, \vec{Z}) \equiv q_i'\big)\Big).$$

On the other side, $(\vec{q}, \vec{x}, \vec{q}')$ is put into the set of possible transitions R_E if for all i: $\delta_i(\vec{q}, \vec{x})$ is not definitely disparate to the value of q_i'. This leads to the symbolic representation

$$\chi_{R_E}(\vec{q}, \vec{x}, \vec{q}') = \exists\vec{Z}\Big(\prod\nolimits_{i=0}^{|\vec{q}|-1} \exists Z\big(\delta_i(\vec{q}, \vec{x}, Z, \vec{Z}) \equiv q_i'\big)\Big).$$

Based on fixed and possible transitions, [9] defines a symbolic model checking procedure that is able to falsify so-called realizability problems and to prove validity problems for designs with Black Boxes. A property is called realizable, if there is a replacement of the Black Boxes by some implementation such that the overall design fulfills the property and a property is called valid, if it holds independently of the replacement of the Black Boxes. More details on Black Box model checking can be found in [9]. To understand this paper, the existence of fixed and possible transitions is the most important concept.

2.3 Example

Figure 1 shows the running example for this paper, an incomplete design with one Black Box and a single Z_i-modeled output. Flexible symbolic simulation (see Sect. 2.1) of this circuit returns

$$\begin{aligned} \delta_0(q_1, q_0, x, Z_0) &= \overline{q}_0 \cdot Z_0 + \overline{q}_0 \cdot q_1 + q_0 \cdot \overline{q}_1 \\ \delta_1(q_1, q_0, x, Z_0) &= x + q_0 \end{aligned}$$

For the remainder of this paper, we abbreviate the state names for this example as follows: We use $\overline{q}_1\overline{q}_0$ for $(q_1 = 0, q_0 = 0)$, $\overline{q}_1 q_0$ for $(q_1 = 0, q_0 = 1)$, $q_1\overline{q}_0$ for $(q_1 = 1, q_0 = 0)$, and $q_1 q_0$ for $(q_1 = 1, q_0 = 1)$. Likewise, we use $\overline{x}$ for $x = 0$ and x for $x = 1$.

Based on δ, the fixed transition relation is

$$R_A = \big\{(\overline{q}_1 q_0, \overline{x}, q_1 q_0), (\overline{q}_1 q_0, x, q_1 q_0), (q_1 \overline{q}_0, \overline{x}, \overline{q}_1 q_0), (q_1 \overline{q}_0, x, q_1 q_0), (q_1 q_0, \overline{x}, q_1 \overline{q}_0), (q_1 q_0, x, q_1 \overline{q}_0)\big\}$$

and the possible transition relation is

$$\begin{aligned} R_E = \big\{&(\overline{q}_1 \overline{q}_0, \overline{x}, \overline{q}_1 \overline{q}_0), (\overline{q}_1 \overline{q}_0, \overline{x}, \overline{q}_1 q_0), (\overline{q}_1 \overline{q}_0, x, q_1 \overline{q}_0), (\overline{q}_1 \overline{q}_0, x, q_1 q_0), (\overline{q}_1 q_0, \overline{x}, q_1 q_0), \\ &(\overline{q}_1 q_0, x, q_1 q_0), (q_1 \overline{q}_0, \overline{x}, \overline{q}_1 q_0), (q_1 \overline{q}_0, x, q_1 q_0), (q_1 q_0, \overline{x}, q_1 \overline{q}_0), (q_1 q_0, x, q_1 \overline{q}_0)\big\} \end{aligned}$$

Figure 2 illustrates the fixed transitions (solid arrows) and the possible transitions (both dashed and solid arrows).

3 Counterexample Generation for Incomplete Designs

In this section, we present a method to compute counterexamples for a given incomplete design and a safety property, i.e. a property that has to be satisfied in every state reachable from the initial state.

In the case that symbolic model checking for an incomplete design returns that a safety property is not realizable, it is proven that for every Black Box substitution, there is an input sequence leading to a state violating the safety property (hence a counterexample).

Here, our goal is to find a counterexample that works for all Black Box replacements. We now discuss three approaches to this.

3.1 Approaches

As a first approach to generate counterexamples for incomplete designs, one could consider to adapt the standard method for computing counterexamples [17], meaning that the Black Box outputs could – like inputs – hold arbitrary values in every state. Yet, this allows the counterexample computation to make assumptions about the Black Box output values. For the example in Fig. 2 with initial state $\overline{q}_0 \overline{q}_1$ and erroneous state $q_1 q_0$, the input sequence $x = 1$ would be a counterexample for all Black Box replacements that produce output $Z_0 = 1$. However, this means that the counterexample may not work for other replacements, e.g. for our example, all replacements that produce output $Z_0 = 0$ in the initial state. Thus, this approach does not meet our goal to find a counterexample which does not depend on the behavior of the Black Box.

Alternatively, it is possible to consider *fixed* transitions in R_A only. By that, the input sequence forces the design into a specific sequence of states, independently from the Black Box substitutions. Thus, if such an input sequence is found, it will work for every Black Box replacement. Yet, in many cases, it is not possible to find such a counterexample, since the erroneous states are not reachable by fixed transitions only. For our example in Fig. 2, there is no fixed successor for the initial state $\overline{q}_0 \overline{q}_1$ (see R_A for this example as given in Sect. 2.3), and thus no such counterexample can be generated here.

The following approach is not limited to fixed transitions, but still meets our goal:

A *uniform* counterexample for an incomplete design is a sequence of input values that force the design from a specific initial state into an erroneous state violating the safety property, *independently of which possible transition is used*, i.e. independently of the Black Boxes substitution. Both the intermediate states and the final erroneous state may vary depending on the substitutions of the Black Boxes, yet the computed uniform counterexample works for all Black Box substitutions.

For our example in Fig. 2, the input sequence $\tilde{x} = (0, 1, 0, 1)$ is a *uniform* counterexample that works for all replacements of the Black Box, since it works for all three possible

paths $(\overline{q}_1\overline{q}_0) \xrightarrow{\overline{x}} (\overline{q}_1 q_0) \xrightarrow{x} (q_1 q_0) \xrightarrow{\overline{x}} (q_1\overline{q}_0) \xrightarrow{x} (q_1 q_0)$, $(\overline{q}_1\overline{q}_0) \xrightarrow{\overline{x}} (\overline{q}_1\overline{q}_0) \xrightarrow{x} (q_1 q_0) \xrightarrow{\overline{x}} (q_1\overline{q}_0) \xrightarrow{x}$ $(q_1 q_0)$ and $(\overline{q}_1\overline{q}_0) \xrightarrow{\overline{x}} (\overline{q}_1\overline{q}_0) \xrightarrow{x} (q_1\overline{q}_0) \xrightarrow{\overline{x}} (\overline{q}_1 q_0) \xrightarrow{x} (q_1 q_0)$.

3.2 Computation of Uniform Counterexamples

The main idea of uniform counterexample computation is as follows: Starting from the set of states directly violating the safety property, we iteratively compute all the predecessors of this set for which there is an input sequence that leads into an erroneous state, regardless of which possible transitions are taken. These input sequences are built in the iteration process and are stored together with the states.

To simplify matters, we will first illustrate our method in an explicit way and show how to perform symbolic computation later. Let Q be the set of states and X be the set of possible input values. In our case, $Q = \mathbb{B}^{|\vec{q}|}$ and $X = \mathbb{B}^{|\vec{x}|}$.

The safety property φ has the form $\varphi = AG(\psi)$, whereas ψ only consists of state variables, negation, conjunction and disjunction[1]. The set of erroneous states is defined as $\{q \in Q | q \not\models \psi\}$.

The following invariant is used to prove the correctness of our approach: For every $C_i \subseteq Q \times X^i$: If $(q, \tilde{x}) \in C_i$, then starting with state q, the input sequence $\tilde{x} = (x_1, \ldots, x_i)$ will lead into an erroneous state, *independently of the substitution of the Black Boxes.*

$i = 0$

Initially, the length of the considered input sequences is $i = 0$. We define

$$C_0 := \big\{ \big(q, ()\big) \;\big|\; q \in Q,\ q \not\models \psi \big\}.$$

Obviously, C_0 satisfies the invariant defined above.

$i \rightarrow i + 1$

Based on C_i, C_{i+1} can be computed: Considering a state $q \in Q$ and an input $x \in X$, if there is an input sequence $\tilde{x} \in X^i$ so that for every possible successor $q' \in Q$ with $(q, x, q') \in R_E$: $(q', \tilde{x}) \in C_i$, then starting from q, the input sequence $(x, \tilde{x})$ with length $i + 1$ will lead into an erroneous state, independently of the substitution of the Black Boxes:

$$C_{i+1} := \Big\{ \big(q, (x, \tilde{x})\big) \;\Big|\; q \in Q, x \in X, \tilde{x} \in X^i;\ \forall q' \in Q\colon \Big(\big((q, x, q') \in R_E\big) \rightarrow \big((q', \tilde{x}) \in C_i\big) \Big) \Big\}$$

The argumentation above directly implies that the invariant is satisfied.

If there is a sequence starting at an initial state in some C_i, a uniform counterexample with length i has been found.

The computation of uniform counterexamples is applied after the Black Box model checker produced the result that the safety property is violated for every possible Black Box substitution. That means that there is a (possibly non-uniform) counterexample of length d (where d is the number of steps that were necessary to reach the fixed point in the model checking's AG computation). However, it may be possible that there is no uniform counterexample of length d. Due to that, it is reasonable to look also for uniform counterexamples that are slightly larger than d. Here, we chose a maximum length of $c \cdot d$ with $c = 4$ in our implementation. We are not interested in uniform counterexamples much larger than d, since it is our goal to *simplify* counterexamples by our Black Box techniques. This also handles the case of instances for which there is no uniform counterexample at all.

[1]For ease of explanation, we assume that ψ contains only state variables. However note that we generalized our method to the case that ψ may contain arbitrary signals of the design.

Altogether, this leads to the following algorithm:

$i := 0$;
$C_0 := \{(q, ()) \mid q \in Q,\ q \not\models \psi\}$;
while $\big((\nexists q \in I, \tilde{x} \in X^i \colon (q, \tilde{x}) \in C_i) \wedge (i \leq c \cdot d)\big)$ {
  $C_{i+1} := \Big\{(q, (x, \tilde{x})) \;\Big|\; q \in Q, x \in X, \tilde{x} \in X^i;\ \forall q' \in Q \colon \Big(\big((q, x, q') \in R_E\big) \rightarrow \big((q', \tilde{x}) \in C_i\big)\Big)\Big\}$
  i:=i+1;
}

The argumentation given above proves that the algorithm is correct and – if possible – always computes a uniform counterexample with minimum length.

3.3 Example

We again use our example given in Figures 1 and 2 together with the safety property $\varphi = AG(\neg q_0 \vee \neg q_1)$. The possible transition relation R_E for this example is given in Section 2.3.

Model checking for incomplete designs proves that φ is not satisfied for any substitution of the Black Box. For uniform counterexample computation, we evaluate C_i as defined above for increasing counterexample length i. For $i = 0$, C_0 contains all states directly violating the safety property $\psi = \neg q_0 \vee \neg q_1$ plus an empty input sequence:

$$C_0 = \big\{\big(q_1 q_0, ()\big)\big\}$$

In the next step, we iteratively increase the length of the input sequence:

$$C_1 = \big\{\big(q_1\overline{q}_0, (x)\big), \big(\overline{q}_1 q_0, (\overline{x})\big), \big(\overline{q}_1 q_0, (x)\big)\big\}$$

Note that e.g. $\big(\overline{q}_1\overline{q}_0, (x)\big) \notin C_1$ due to $\big(\overline{q}_1\overline{q}_0, x, q_1\overline{q}_0\big) \in R_E$ yet $\big(q_1\overline{q}_0, ()\big) \notin C_0$. We continue:

$$\begin{aligned} C_2 = \big\{&\big(q_1\overline{q}_0, (\overline{x}, \overline{x})\big), \big(q_1\overline{q}_0, (\overline{x}, x)\big), \big(q_1 q_0, (\overline{x}, x)\big), \big(q_1 q_0, (x, x)\big)\big\} \\ C_3 = \big\{&\big(\overline{\mathbf{q_1}}\overline{\mathbf{q_0}}, (\mathbf{x}, \overline{\mathbf{x}}, \mathbf{x})\big), \big(q_1\overline{q}_0, (x, \overline{x}, x)\big), \big(q_1\overline{q}_0, (x, x, x)\big), \big(\overline{q}_1 q_0, (\overline{x}, \overline{x}, x)\big), \\ &\big(\overline{q}_1 q_0, (\overline{x}, x, x)\big), \big(\overline{q}_1 q_0, (x, \overline{x}, x)\big), \big(\overline{q}_1 q_0, (x, x, x)\big), \big(q_1 q_0, \overline{x}, \overline{x}, \overline{x})\big), \\ &\big(q_1 q_0, (\overline{x}, \overline{x}, x)\big), \big(q_1 q_0, (x, \overline{x}, \overline{x})\big), \big(q_1 q_0, (x, \overline{x}, x)\big)\big\} \end{aligned}$$

Since there is an input sequence for the initial state $\overline{q}_1\overline{q}_0$ in C_3, we have found the uniform counterexample $(1, 0, 1)$ with minimum length $i = 3$.

3.4 Symbolic Computation

For symbolic computation, we use the symbolic transition relation $\chi_{R_E}(\vec{q}, \vec{x}, \vec{q}')$ as defined in Sect. 2.2. The uniform counterexamples can then be symbolically computed as follows:

$i := 0$;
$\chi_{C_0}(\vec{q}) := \chi_{Sat(\neg\psi)}(\vec{q})$;
while $\big(\chi_{C_i} \cdot \chi_I = \text{false}) \wedge (i \leq c \cdot d)\big)$ {
  $\chi_{C_{i+1}}(\vec{q}, \vec{x}_{i+1}, \ldots, \vec{x}_1) := \forall\vec{q}'\Big(\big(\chi_{R_E}|_{\vec{x}\leftarrow\vec{x}_{i+1}}\big)(\vec{q}, \vec{x}_{i+1}, \vec{q}') \rightarrow \big(\chi_{C_i}|_{\vec{q}\leftarrow\vec{q}'}\big)(\vec{q}', \vec{x}_i, \ldots, \vec{x}_1)\Big)$;
  $i := i + 1$;
}

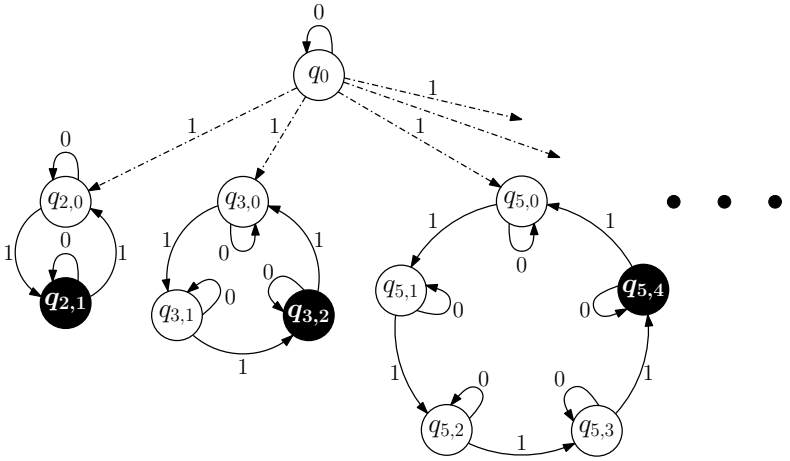

Figure 3: Exemplary automaton for uniform counterexample length consideration. Fixed transitions are indicated by solid arrows, possible transitions are indicated by dashed and solid arrows. q_0 is the initial state and for every prime number p_i, q_{p_i,p_i-1} is an erroneous state.

3.5 Length of Uniform Counterexamples

Theorem 1 justifies our decision to abort the search for a uniform counterexample whenever its length exceeds its non-uniform counterpart by a constant factor.

Theorem 1. *There are incomplete circuits with n states where the length of a uniform counterexample is in $\Omega(2^{\sqrt{n}})$.*

According to Theorem 1, there are examples where the size of the shortest uniform counterexample is much larger than the size of the shortest non-uniform counterexample. Whereas this is an interesting theoretical result, this does not mean that such examples will be encountered in practical applications.

Proof. Let $p = (2, 3, 5, 7, \dots)$ be the sequence of prime numbers.

Consider the automaton shown in Figure 3. Starting from the initial state q_0 and for input $x = 1$, there are m successors $(q_{p_1,0}, q_{p_1,0}, \dots, q_{p_m,0})$ in different circles. It is easy to see that the number of states is $n = 1 + \sum_{i=1}^{m} p_i$.

In every circle i, the input $x = 1$ leads from $q_{p_i,j}$ to the next state in the circle $q_{p_i,j+1 (\mathrm{mod}\ p_i)}$, while $x = 0$ always leads back to the same state. For each circle, q_{p_i,p_i-1} is the only state violating the safety property.

Due to this construction, it can be seen that the shortest uniform counterexample has length $l = \prod_{i=1}^{m} p_i$ and always sets the input to 1.

The primorial $\prod_{i=1}^{m} p_i$ is known to be in $\Omega(2^{(p_m)})$. Together with $n = \sum_{i=1}^{m} p_i + 1 \leq \sum_{j=1}^{p_m} j = \frac{p_m^2 - p_m}{2} \leq p_m^2$, one can see that for this circuit with n states, the shortest uniform counterexample has length $\Omega(2^{\sqrt{n}})$ □

The result of Theorem 1 is also interesting in the context of counterexample simplification in Bounded Model Checking [8]: It essentially says that there are examples where the number of unfoldings in BMC has to be exponentially enlarged before it is possible to "simplify" the counterexample by "lifting" all output signals of Black Boxes in all time frames (lifting means removing assignments to signals without losing the property that the remaining assignments imply a violation of the safety property).

From a theoretical point of view, it is also interesting to note that the length of the shortest uniform counterexample is bounded by 2^n (where n is the number of states). This can be seen by a power set construction similar to the well-known transformation of non-deterministic finite automata to deterministic finite automata.

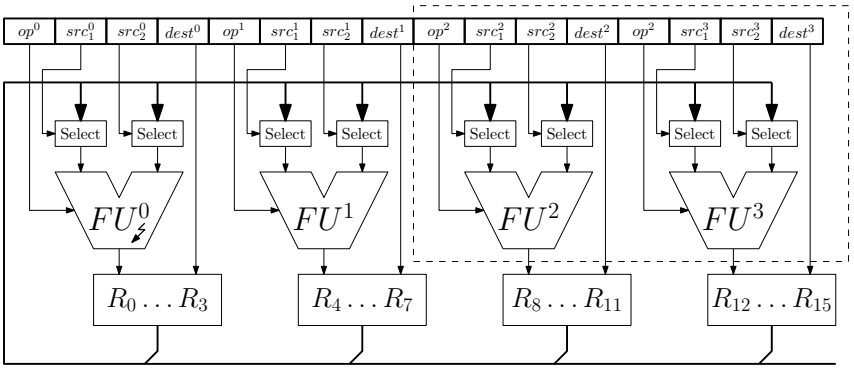

Figure 4: A simple VLIW ALU with an error in the XOR operation in FU^0.

4 Experimental Results

The symbolic model checker for incomplete designs that was used in [9] was based on the BDD package CUDD [18] and used relational preimage computation. For our experiments, we used an improved version in which boolean formulas are represented by And-Inverter-Graphs [19] and that used functional preimage computation [20, 21].

For our experiments, we used a simple VLIW ALU with four functional units as illustrated in Fig. 4; the VLIW instruction was considered the input of the design. Each functional unit $FU^0 \ldots FU^3$ was able to read from every register, while writing was limited to a local part of the registers.

FU^0 realized the four logical instructions AND, OR, XOR and NOP, whereas the XOR operation was faulty and computed an OR instead (note that XOR and OR only differ when both inputs are 1). FU^1 implemented a "Load Immediate" operation that wrote a value given in the operation code into a specified register plus a NOP operation. FU^2 and FU^3 both realized the arithmetic operations ADD, SUB, MUL and NOP. All registers were assumed to be initialized by 0.

We checked a property stating that if the last operation that FU^0 had executed was "$R_2 := R_0 \oplus R_1$"[2], then the current value of R_2 would be the current value of R_0 XOR the current value of R_1. Due to the erroneous XOR implementation, the property failed and it was possible to compute a counterexample for the complete design, which included a value for each primary input, thus a complete VLIW instruction in every step of the counterexample.

We then replaced FU^3 and FU^4 together with their instruction inputs by a Black Box (comp. the dashed box in Fig. 4). Note that for this example, the Black Box lies inside the cone of influence for the considered property. Our method for uniform counterexample computation was able to find a uniform counterexample which included only the instructions for FU^0 and FU^1:

	FU^0						FU^1					
Step	R_0	R_1	R_2	R_3	Instruction	Effect	R_4	R_5	R_6	R_7	Instruction	Effect
0	0	0	0	0	`NOP`	no operation	0	0	0	0	`LOADI R7 1`	$R_7 := 1$
1	0	0	0	0	`OR R1 R15 R7`	$R_1 := R_{15} \vee R_7$	0	0	0	1	`NOP`	no operation
2	0	1	0	0	`OR R0 R15 R7`	$R_0 := R_{15} \vee R_7$	0	0	0	1	`NOP`	no operation
3	1	1	0	0	`XOR R2 R0 R1`	$R_2 := R_0 \oplus R_1$	0	0	0	1	`NOP`	no operation
4	1	1	**1**	0	–	–	0	0	0	1	–	–

Our method was able to provide this counterexample for a word width of 32 bits in less than 3 hours of CPU time on a Dual Opteron running at 2 GHz and with 4 GB of RAM.

Note that the second concept discussed in Sect. 3.1, which considered fixed transitions only, would not have been able to generate a counterexample, since no erroneous state is reachable by only fixed transitions.

[2] the last operation was always cached so it would be available in the next state

It is interesting to see that FU^1 may not be replaced by a Black Box in order to further simplify the counterexample, since intuitively FU^1 is needed to introduce constants 1 into the register file, which is a precondition for making the error in FU^0 observable.

5 Conclusion and Future Work

We introduced a method to compute counterexamples for incomplete designs. This method can be used to compute more comprehensible counterexamples arguing about a reduced number of components in the system.

At the moment we are working on further improvements concerning the properties that can be processed. While *EF*-formulas are dual to *AG*-formulas, the method can be extended to *AF*- and *AU*-formulas resp. *EG*- and *EU* by loop detection.

Currently, the selection about which parts of the design are replaced by Black Boxes and which are not is made by the user. It is our goal to develop a method that automatically determines the Black Box positions in order to receive simplified counterexamples without user interaction.

References

[1] A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[2] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite–state concurrent systems using temporal logic specifications. *ACM Trans. on Programming Languages and Systems*, 8(2):244–263, 1986.

[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] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publisher, 1993.

[5] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[6] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Constuction and Analysis of Systems*, volume 1579 of *LNCS*. Springer Verlag, 1999.

[7] A. Biere, A. Cimatti, E.M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using SAT procedures instead of BDDs. In *Design Automation Conf.*, 1999.

[8] Kavita Ravi and Fabio Somenzi. Minimal assignments for bounded model checking. In *TACAS*, pages 31–45, 2004.

[9] T. Nopper and C. Scholl. Approximate symbolic model checking for incomplete designs. In *Formal Methods in Computer-Aided Design*, pages 290–305, Nov 2004.

[10] C. Pixley. A Theory and Implementation of Sequential Hardware Equivalence. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 11(12):1469–1494, December 1992.

[11] Thomas Kropf and Hans-Joachim Wunderlich. A common approach to test generation and hardware verification based on temporal logic. In *International Test Conference*, pages 57–66, 1991.

[12] A. Jain, V. Boppana, R. Mukherjee, J. Jain, M. Fujita, and M. Hsiao. Testing, verification, and diagnosis in the presence of unknowns. In *VLSI Test Symp.*, pages 263–269, 2000.

[13] C. Scholl and B. Becker. Checking equivalence for partial implementations. In *Design Automation Conf.*, pages 238–243, 2001.

[14] M. Abramovici, M.A. Breuer, and A.D. Friedman. *Digital Systems Testing and Testable Design*. Computer Science Press, 1990.

[15] C. Scholl and B. Becker. Checking equivalence for partial implementations. Technical Report 145, Albert-Ludwigs-University, Freiburg, October 2000.

[16] T. Nopper and C. Scholl. Flexible Modeling of Unknowns in Model Checking of Incomplete Designs. In *GI/ITG/GMM-Workshop "Modellierung und Verifikation"*, 2005.

[17] O. Coudert, C. Berthet, and J. C. Madre. Verification of synchronous sequential machines based on symbolic execution. In *Proceedings of the international workshop on Automatic verification methods for finite state systems*, pages 365–373, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[18] F. Somenzi. *CUDD: CU Decision Diagram Package Release 2.3.1*. University of Colorado at Boulder, 2001.

[19] F. Pigorsch, C. Scholl, and S. Disch. Advanced Unbounded Model Checking Based on AIGs, BDD Sweeping and Quantifier Scheduling. In *Proceedings of the Conference on Formal Methods in Computer Aided Design (FMCAD)*. IEEE Computer Society Press, November 2006.

[20] Thomas Filkorn. Functional extension of symbolic model checking. In *CAV '91: Proceedings of the 3rd International Workshop on Computer Aided Verification*, pages 225–232, London, UK, 1992. Springer-Verlag.

[21] P.F. Williams, A. Biere, E.M. Clarke, and A. Gupta. Combining decision diagrams and SAT procedures for efficient symbolic model checking. In *Computer Aided Verification*, volume 1855 of *LNCS*, pages 124–138. Springer Verlag, 2000.