Flexible Modeling of Unknowns in Model Checking for Incomplete Designs

Tobias Nopper Christoph Scholl

{nopper, scholl}@informatik.uni-freiburg.de
Albert-Ludwigs-Universität Freiburg, Institut für angewandte Wissenschaften,
D-79110 Freiburg im Breisgau, Germany

We consider the problem of checking whether an incomplete design (i.e. a design containing so-called Black Boxes) can still be extended to a complete design satisfying a given property or whether the property is satisfied for all possible extensions.

In this paper we extend a method based on a series of approximate, yet sound algorithms to prove or disprove CTL properties for incomplete designs [1]. Each algorithm in the previous approach was able to model the effect of the unknown information at the outputs of the Black Boxes at a different level of accuracy (leading to different requirements for computational resources). In this paper we present a novel approach which is able to use different methods for modeling unknowns at the outputs of different Black Boxes within a single model checking run. This permits us to handle less relevant Black Boxes (in terms of the CTL formula) with larger approximation and thus faster, whereas we do not lose important information when the possible effect of more relevant Black Boxes is modeled by more exact methods.

Finally we give a series of experimental results demonstrating the effectiveness and feasibility of the new method.

1 Introduction

Deciding the question whether a circuit implementation fulfills its specification is an essential problem in computer-aided design of VLSI circuits. Growing interest in universities and industry has led to new results and significant advances concerning topics like property checking, state space traversal and combinational equivalence checking.

For proving properties of sequential circuits, Clarke, Emerson, and Sistla presented model checking for the temporal logic CTL [2]. Burch, Clarke, and McMillan et al. improved the technique by using symbolic methods based on binary decision diagrams [3] for both state set representation and state traversal in [4, 5].

In this paper we will consider how to perform model checking of *incomplete* circuits, i.e. circuits which contain unknown parts. These unknown parts are combined into so-called Black Boxes. In doing so, we will approach two potentially interesting questions, whether it is still possible to replace the Black Boxes by circuit implementations, so that a given model checking property is satisfied ('realizability') and whether the property is satisfied for any possible replacement ('validity').

There are three major benefits symbolic model checking for incomplete circuits can provide: First, instead of forcing the verification runs to the end of the design process where the design is completed, it rather allows model checking in early stages of design, where parts may not yet be finished, so that errors can be detected earlier. Second, complex parts of a design can be replaced by Black Boxes, simplifying the design, while many properties of the design still can be proven, yet in shorter

time. Third, the location of design errors in circuits not satisfying a model checking property can be narrowed down by iteratively masking potentially erroneous parts of the circuit.

Some well-known model checking tools like SMV [5] (resp. NuSMV [6]) and VIS [7] provide the definition of non-deterministic signals (see [8, 9, 10]), and – at first sight – signals coming from unknown areas can be handled as non-deterministic signals. However, it was already shown in [1] that modeling by non-deterministic signals is not capable of answering the questions of realizability ('is there a replacement of the Black Boxes so that the overall implementation satisfies a given property?') or validity ('is a given property satisfied for all Black Box replacements?'). This approach is even not able to provide approximate solutions for realizability or validity (assuming arbitrary CTL formulas).

The approach of [1] for solving the problem is based on the following concept considering two sets of states: The first set is an overapproximation of the set of states satisfying the given CTL formula for at least one substitution of the Black Boxes and the second set is an underapproximation of the set of states satisfying the formula for all Black Box substitutions. Given these two sets, is is possible to prove validity (the CTL formula is satisfied for all Black Boxes substitutions) or to disprove realizability (the CTL formula is satisfied for no Black Box substitution). Based on this concept a series of approximate, yet sound algorithms to process incomplete designs with increasing quality and computational resources was presented.

Whereas [1] presents a series of approximation algorithms where each algorithm models the effect of the unknown information at the outputs of the Black Boxes in a uniform way, this paper presents a novel approach which is able to use different methods for modeling unknowns at the outputs of different Black Boxes within a single model checking run. This permits us to handle less relevant Black Boxes (in terms of the CTL formula) with larger approximation and thus faster, whereas we do not lose important information when the possible effect of more relevant Black Boxes is modeled by more exact methods. The advantage of our novel method becomes apparent on the basis of the following scenario:

Consider an incomplete design with multiple Black Boxes, in which the output provided by some Black Boxes has a large influence on the result of the model checking run, while the output of others has only a smaller influence.

Furthermore, assume that the methods from [1] having a small level of accuracy are not capable of providing a conclusive answer to the model checking problem due to the 'Black Boxes with large influence'. In this case *all* Black Boxes have to be handled with high accuracy by the approach in [1], leading to a high demand for memory and runtime. However, this accuracy might not be necessary for all Black Boxes. For this reason, we present a new method that allows us to specify the method (and thus the level of approximation and the computational resources) *separately for each Black Box*. This method provides a much higher flexibility in modeling unknowns and leads to much smaller needs for computational resources compared to a uniform modeling (see Sect. 5).

The paper is structured as follows: After giving a brief review of symbolic model checking for incomplete designs in Sect. 2, we will discuss our new method gaining the accuracy advantages of the more exact methods without losing the speed advantages of the less exact methods in Sect. 3. Finally we give a series of experimental results demonstrating the effectiveness and feasibility of the presented methods in Sect. 4 and we conclude the paper in Sect. 5.

2 Preliminaries

2.1 Incomplete Designs

Representing Incomplete Designs: If parts of a circuit are not yet known or cut off, we have to handle *incomplete designs*. In this section we briefly review symbolic representations of incomplete designs.

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

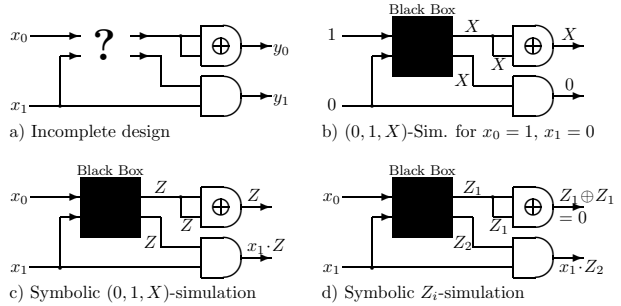

Figure 1: An incomplete design

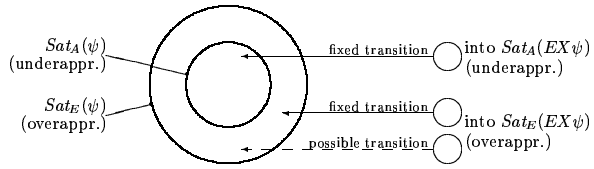

Figure 2: Evaluation of $Sat_A(EX\psi)$ and $Sat_E(EX\psi)$

Consider the combinational part of a sequential circuit (Mealy automaton) defining the transition function δ and the output function λ. For simulating this combinational circuit wrt. some input vector we can make use of the ternary $(0,1,X)$-logic [11, 12]: 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 [13] (see Fig. 1b). 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 [14, 12]. For each output g_i of the incomplete design with primary input variables $x_1, \ldots, x_n$ we obtain a BDD 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*, see Fig. 1c for an example.

Since $(0,1,X)$-simulation can not 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, we introduce a new variable Z_i for each Black Box output and perform a (conventional) symbolic simulation. This approach was called symbolic Z_i-simulation in [12]. Fig. 1d shows an example for symbolic Z_i-simulation. (Note that the first output can now be shown to be constant 0.)

Please note that in contrast to [12], we will consider Black Boxes that can be replaced not only by combinational, but also by sequential circuits, so that for two states in a computation path that generate the same Black Box input, the Black Box may answer with different outputs.

Realizability and Validity: Given an incomplete design with Black Boxes and a CTL formula, the questions we will consider in the following are:

1. Is there a replacement of the Black Boxes in the incomplete design, so that the resulting circuit satisfies a given CTL formula φ? If this is true, then the property φ is called *realizable* for the incomplete design. The corresponding decision problem is called *realizability* problem.

2. Is a CTL formula φ satisfied for all possible replacements of the Black Boxes? If this is the case, then φ is *valid* for the incomplete design; the corresponding decision problem is denoted as *validity* problem.

2.2 Approximate Symbolic Model Checking for Incomplete Designs

Symbolic Z- and Z_i-Model Checking: We here briefly review approximate symbolic model checking for incomplete designs [1, 15], which is based on well-known symbolic model checking for complete designs [4] for properties specified in CTL (Computation Tree Logic).

For a given incomplete design and a CTL formula φ, approximate symbolic model checking for an incomplete designs considers two sets of states: $Sat_A(\varphi)$, an underapproximation of the set of states in which φ is satisfied for all Black Box substitutions and $Sat_E(\varphi)$, an overapproximation of the set of states in which φ is satisfied for at least one Black Box substitution.

Given these two sets, it is potentially possible to prove validity and to disprove realizability: If all initial states lie within $Sat_A(\varphi)$, then for all initial states and all Black Box substitutions, φ is satisfied and thus φ is valid. If there is one initial state lying outside $Sat_E(\varphi)$, then there is a initial state that does not satisfy φ for any Black Box substitution and thus φ is not realizable.

The approximations of $Sat_E(\varphi)$ and $Sat_A(\varphi)$ can be computed based on an approximate transition relation and on approximate output functions for the corresponding Mealy automaton. In incomplete designs we have Black Boxes in the functional block defining the transition function δ and the output function λ (see Fig. 3). For this reason there are transitions which exist independently from the replacement of the Black Boxes, i.e. for all possible replacements of the Black Boxes (we call them '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 (we call them 'possible transitions').

Due to that, we work with two types of approximations of the transition relation: An underapproximation χ_{R_A} only contains fixed transitions and an overapproximation χ_{R_E} contains at least all possible transitions (of course, this includes all fixed transitions).

In the same manner we approximate the sets of states in which the output value y_i of λ_i is true: An underapproximation $Sat_A(y_i)$ contains only states in which y_i is true independently from the replacements of the Black Boxes and an overapproximation $Sat_E(y_i)$ contains at least all states in which y_i may be true for some replacement of the Black Boxes.

Based on χ_{R_A}, χ_{R_E}, $Sat_A(y_i)$ and $Sat_E(y_i)$, it is possible to define rules how arbitrary CTL formulas can be recursively evaluated. As an exemplary case we here show how to evaluate $Sat_A(EX\psi)$ and $Sat_E(EX\psi)$:

Given $Sat_A(\psi)$, the set of states which definitely satisfy ψ for all Black Box replacements, we include into $Sat_A(EX\psi)$ all states with a fixed transition to a state in $Sat_A(\psi)$. Likewise, we include all the states into $Sat_E(EX\psi)$ which have a possible transition to a state in $Sat_E(\psi)$. Fig. 2 illustrates the sets.

$Sat_A(\neg\psi)$, the set of states surely satisfying $\neg\psi$, can be computed by using the inverse set of $Sat_E(\psi)$ and vice versa. $Sat_A(\varphi_1 \vee \varphi_2)$ is build by the union of $Sat_A(\varphi_1)$ and $Sat_A(\varphi_2)$, analogously for $Sat_E(\varphi_1 \vee \varphi_2)$. Finally, $\varphi = EG\psi$ and $\varphi = E\psi_1 U \psi_2$ can be evaluated by their standard fixed point iterations (see [4], e.g.) based on the evaluation of EX defined above (two separate fixed point iterations for Sat_A and Sat_E). Since every CTL property can be expressed using the operands discussed so far, the model checking method is complete.

All sets described above are symbolically represented by using BDDs for their characteristic functions which allows us to perform the set operations by using boolean functions. No explicit representation is necessary at any point.

For symbolic Z-model checking, the transition function δ and the output function λ are determined by symbolic $(0, 1, X)$-simulation. Based on the transition function δ, an underapproximation χ_{R_A} of the transition relation is computed by

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

and an overapproximation χ_{R_E} of the transition relation is computed by

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

$Sat_A(y_i)$ and $Sat_E(y_i)$ are computed based on λ_i: $\chi_{Sat_A(y_i)}(\vec{q}, \vec{x}) = \forall Z \big(\lambda_i(\vec{q}, \vec{x}, Z) \big)$ and $\chi_{Sat_E(y_i)}(\vec{q}, \vec{x}) = \exists Z \big(\lambda_i(\vec{q}, \vec{x}, Z) \big)$.

For symbolic Z_i-model checking, δ and λ are determined by using symbolic Z_i-simulation. The approximations χ_{R_A}, χ_{R_E}, $Sat_A(y_i)$, and $Sat_E(y_i)$ are computed in an analogous manner with every appearance of Z in the formulas above replaced by $\vec{Z}$.

Since symbolic Z_i-simulation is more accurate than symbolic $(0, 1, X)$-simulation, symbolic Z_i-model checking is more accurate than symbolic Z-model checking as well. However, this gain of accuracy has to be paid by an increased need of variables for the Black Box outputs, affecting the time and computation resources.

Symbolic Output Consistent Z_i-Model Checking: A further improvement on the accuracy of the two approximated sets considered in symbolic Z_i-model checking can be gained by including the Z_i-variables assigned to Black Box outputs into the state space.

In this way, the values of the transition function δ and the output function λ are uniquely determined by the current state and it is no longer necessary to distinguish between possible and fixed transitions and states possibly resp. surely satisfying an atomic CTL formula y_i. However, this fact is paid by larger state spaces, which typically leads to an increase of computational resources.

3 Building an Approximate Symbolic Model Checking Method for Incomplete Designs with Flexible Handling of Unknowns

Symbolic Z-model checking, symbolic Z_i-model checking and symbolic output consistent Z_i-model checking can be seen as a sequence of methods with increasing accuracy, but increasing need of time and computational resources, too.

All these methods are only able to handle every Black Box in an incomplete design with the same accuracy, so it is possible that one has to use a method with good accuracy in order to determine a conclusive result, although this accuracy is disproportionate for some Black Boxes that only have a low influence to the result of the model checking run.

Due to this, we will now describe how symbolic Z-, Z_i- and output consistent Z_i-model checking can be combined into one single method. This combined method is able to handle each Black Box with different accuracy and thus different complexity. For that, we will start with a combined Z- and Z_i-model checking method and will then add output consistent Z_i-model checking.

3.1 Combining Symbolic Z- and Z_i-Model Checking

For a combined Z- and Z_i-model checking method, we first have to explore a combined method performing symbolic $(0, 1, X)$- and Z_i-simulation in parallel.

Combining Symbolic $(0, 1, X)$- and Z_i-Simulation. Symbolic $(0, 1, X)$-simulation [12] is based on the well-known $(0, 1, X)$-simulation [11] and introduces a new Z variable, which is used to model the unknown value X of the Black Box outputs. As described in Sect. 2.1, for each output g_i of the incomplete design with primary input variables $x_1, \ldots, x_n$, a BDD representation of g_i is obtained by using a slightly modified version of symbolic simulation with:

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

Please note that the only difference to conventional symbolic simulation is that for NOT gates, Z has to be replaced by $\overline{Z}$; both AND and OR gates can be handled as usual.

In contrast to symbolic Z-simulation, symbolic Z_i-simulation [12] introduces a new Z_i variable for each Black Box output and then performs symbolic simulation as if the Black Box outputs were inputs.

We can now construct a combined symbolic $(0, 1, X)$- and Z_i-simulation: For the Black Boxes, which are to be handled by symbolic $(0, 1, X)$-simulation, we assign Z to model the Black Box outputs, while for each output of the Black Boxes, which are to be handled by Z_i-model checking, we use a new Z_i variable. The combined simulation now considers the latter Black Box outputs as additional inputs

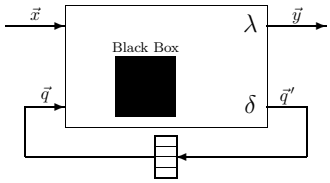

Figure 3: Mealy automaton with Black Box

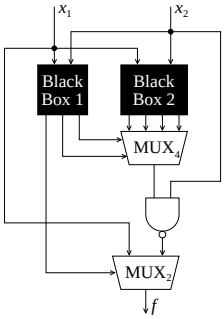

Figure 4: An exemplary incomplete circuit

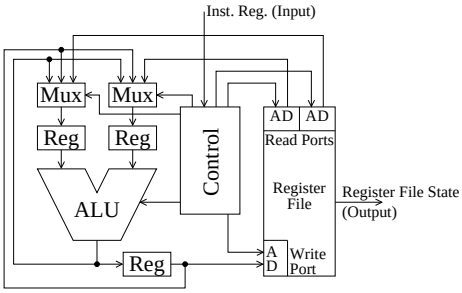

Figure 5: Pipelined ALU

and then performs symbolic $(0, 1, X)$-simulation (always replacing Z by $\overline{Z}$ when processing NOT gates).

We receive BDD representations of the circuit outputs g_i with inputs $x_1, \ldots, x_n$ and (Z_i-simulated) Black Box outputs $Z_1, \ldots, Z_n$:

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

Example Figure 4 shows an exemplary circuit: If this circuit is simulated by using symbolic $(0, 1, X)$-simulation, meaning that Z is assigned to the outputs of both Black Box 1 and Black Box 2, a total number of 3 variables are needed (x_1, x_2, Z) and the resulting function for the output is $f_Z = Z$.

If the circuit is simulated by using symbolic Z_i-simulation instead, meaning that for each output of Black Box 1 and Black Box 2 a new Z_i variable is used, 9 variables are needed $(x_1, x_2, Z_1, \ldots, Z_7)$, and the function for the output is $f_{Z_i} = \overline{Z}_1 x_1 + Z_1 \cdot \big(\overline{x}_2 + \neg(\overline{Z}_2\overline{Z}_3 Z_4 + Z_2\overline{Z}_3 Z_5 + \overline{Z}_2 Z_3 Z_6 + Z_2 Z_3 Z_7)\big)$.

When using a combined method, assigning Z to all outputs of Black Box 2 but different Z_i's to the outputs of Black Box 1, we end up using 6 variables $(x_1, x_2, Z, Z_1, Z_2, Z_3)$ and receive the function $f_{\text{comb}} = \overline{Z}_1 x_1 + Z_1 \cdot (\overline{x}_2 + Z)$.

So, the combined method generates an output function that is obviously less complicated than the result of symbolic Z_i-simulation, yet contains more information than the result of symbolic $(0, 1, X)$-simulation. To give an example, for $x_1 = 1$ and $x_2 = 0$, the output can be shown to be 1 using the combined method, while it is not possible to gain this information from symbolic $(0, 1, X)$-simulation.

Note that in general, the combined method is at most as exact as symbolic Z_i-simulation, but at least as exact as symbolic $(0, 1, X)$-simulation.

Using Combined Symbolic $(0, 1, X)$- and Z_i-Simulation for Combining Symbolic Z- and Z_i-Model Checking. In this chapter, we will discuss how to combine symbolic Z- and Z_i-model checking. This combined method has a lower need of BDD variables compared to pure Z_i-model checking, since for those Black Box outputs which are to be handled by Z-model checking, one single Z variable can be used instead of an extra Z_i variable for each single Black Box output.

Here we adopt the concept of [1] which considers 'possible' and 'fixed' transitions together with the sets of states for each atomic property y_i that 'possibly' or 'surely' satisfy this property. Based on this concept, it is possible to compute an overapproximation of the set of states satisfying a given CTL property φ for at least one Black Box substitution ($Sat_E(\varphi)$) and the underapproximation of the set of states satisfying the property for all Black Box substitutions ($Sat_A(\varphi)$).

Given these two sets, it is possible to provide sound statements whether the property is satisfied for all Black Box substitutions ('the property is valid') or for none ('the property is not realizable') can be provided: If all initial states lie within $Sat_A(\varphi)$, then for all start states and all Black Box substitutions, φ is satisfied and thus φ is valid. If there is one initial stale lying outside $Sat_E(\varphi)$,

then there is an initial state that does not satisfy φ for any Black Box substitution and thus φ is not realizable.

This concept can be applied for the combined method as well: For an incomplete circuit, let there be a number of Black Boxes that are to be handled by symbolic Z-model checking and some other Black Boxes that are to be handled by symbolic Z_i-model checking.

We then apply combined symbolic $(0,1,X)$- and Z_i-simulation for computing the transition function δ and the output function λ. Thus, we introduce new variables Z and $\vec{Z}_l = (Z_{l,1}, Z_{l,2}, \dots)$. Combined symbolic $(0,1,X)$- and Z_i-simulation now provides symbolic representations of the output $\lambda_i(\vec{q}, \vec{x}, Z, \vec{Z}_l)$ and transition function $\delta_j(\vec{q}, \vec{x}, Z, \vec{Z}_l)$.

As for symbolic Z- and Z_i-model checking, a state *surely* satisfies an atomic property y_i, if it is satisfied for *all* possible assignments to Z and $\vec{Z}_l$ and a state *possibly* satisfies an atomic property y_i, if it is satisfied for *at least one* possible assignment to Z and $\vec{Z}_l$. Likewise, there is a *fixed* transition between two states if the transition exists for *all* possible assignments to Z and $\vec{Z}_l$ and there is a *possible* transition between two states if the transition exists for *at least one* possible assignment to Z and $\vec{Z}_l$. Based on this definition of fixed and possible states and transitions, a combined method of symbolic Z- and Z_i-model checking can be built:

If $\lambda_i|_{\vec{q}=\vec{q}^{\,\text{fix}},\, \vec{x}=\vec{x}^{\,\text{fix}}} = 1$ for some state $(\vec{q}^{\,\text{fix}}, \vec{x}^{\,\text{fix}}) \in \mathbb{B}^{|\vec{q}|\times|\vec{x}|}$, then we know that λ_i is 1 in this state independently from the replacement of the Black Boxes, so we include $(\vec{q}^{\,\text{fix}}, \vec{x}^{\,\text{fix}})$ into $Sat_A(y_i)$ and $Sat_E(y_i)$. If $\lambda_i|_{\vec{q}=\vec{q}^{\,\text{fix}},\, \vec{x}=\vec{x}^{\,\text{fix}}} = 0$, then the output λ_i is 0 in this state independently from the replacement of the Black Boxes, so we include $(\vec{q}^{\,\text{fix}}, \vec{x}^{\,\text{fix}})$ neither into $Sat_A(y_i)$ nor $Sat_E(y_i)$. In any other case, the value of y_i is unknown in this state and thus we include $(\vec{q}^{\,\text{fix}}, \vec{x}^{\,\text{fix}})$ into $Sat_E(y_i)$, but not into $Sat_A(y_i)$. This leads to the following symbolic representations

$$\chi_{Sat_A(y_i)}(\vec{q}, \vec{x}) = \forall Z \forall \vec{Z}_l \big(\lambda_i(\vec{q}, \vec{x}, Z, \vec{Z}_l)\big) \quad \text{and} \quad \chi_{Sat_E(y_i)}(\vec{q}, \vec{x}) = \exists Z \exists \vec{Z}_l \big(\lambda_i(\vec{q}, \vec{x}, Z, \vec{Z}_l)\big).$$

An analogous argumentation leads to fixed transitions and possible transitions of χ_R, since the outputs of the transition functions may be definitely 1 or 0 (independently from the Black Boxes) or they may be unknown: For χ_{R_A}, representing only fixed transitions we obtain

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

and for χ_{R_E} representing at least all possible transitions we obtain

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

The rules computing $\chi_{Sat_A(\varphi)}$ and $\chi_{Sat_E(\varphi)}$ for the remaining operators EX, $\neg$, EG and EU can be adopted from symbolic Z- (resp. Z_i-) model checking (see Sect. 2), since they are based on $\chi_{Sat_A(y_i)}$, $\chi_{Sat_E(y_i)}$, χ_{R_A} and χ_{R_A} and do not consider any Black Box outputs.

As for symbolic Z- and Z_i-model checking, the result of the recursive calculation can then be evaluated as follows:[1]

$$\forall \vec{x} \big(\chi_{Sat_A(\varphi)}|_{\vec{q}=\vec{q}^{\,0}}\big) = 1 \implies \varphi \text{ is valid}$$
$$\forall \vec{x} \big(\chi_{Sat_E(\varphi)}|_{\vec{q}=\vec{q}^{\,0}}\big) = 0 \implies \varphi \text{ is not realizable}$$

3.2 Adding Symbolic Output Consistent Z_i-Model Checking

A further improvement on the accuracy of the two approximated sets considered in symbolic Z- or Z_i-model checking can be gained by including the Z_i-variables assigned to Black Box outputs into the state space.

Consider the CTL formula $EF(y \wedge \neg y)$ for a design in which a Black Box output is directly connected to the primary output y. It can be seen that the combined method given in the last section is neither able to prove validity nor to falsify realizability for the given incomplete design and the given formula.

[1] Let $\vec{q}^{\,0}$ be the initial state of the circuit.

However, it is clear that there will be no time during the computation when y is both true and false. This problem can be solved if we include Z_i-variables assigned to Black Box outputs into the states of the Kripke structure. In this way the Black Box output values $\vec{Z}$ are constant within each single state and therefore in our example y will have a fixed value for each state.

Yet, it is not necessary to include *all* Z_i's into the state space. Let $\vec{Z}_o$ be the Z_i-simulated Black Box outputs that are included into the state space and let $\vec{Z}_l$ be the Z_i-simulated Black Box outputs that are not included. Then the values of $\vec{Z}_o$ are constant within each single state, while the values of $\vec{Z}_l$ are arbitrary as they were in symbolic Z_i-model checking.

Both the output function $\lambda(\vec{q}, \vec{x}, Z, \vec{Z}_l, \vec{Z}_o)$ and the transition function $\delta(\vec{q}, \vec{x}, Z, \vec{Z}_l, \vec{Z}_o)$ can be computed by using the combined symbolic $(0, 1, X)$- and Z_i-simulation, whereas for simulation it is not necessary to distinguish between $\vec{Z}_l$ and $\vec{Z}_o$.

We now describe how to compute the sets of states surely or possibly satisfying the atomic CTL formula y_i: If $\lambda_i|_{\vec{q}=\vec{q}^{\,\text{fix}},\, \vec{x}=\vec{x}^{\,\text{fix}},\, \vec{Z}_o=\vec{Z}_o^{\,\text{fix}}} = 1$ for some state $(\vec{q}^{\,\text{fix}}, \vec{x}^{\,\text{fix}}, \vec{Z}_o^{\,\text{fix}}) \in \mathbb{B}^{|\vec{q}|\times|\vec{x}|\times|\vec{Z}_o|}$, then we know that λ_i is 1 in this state independently from the replacement of the Black Boxes, so we include $(\vec{q}^{\,\text{fix}}, \vec{x}^{\,\text{fix}}, \vec{Z}_o^{\,\text{fix}})$ into $Sat_A(y_i)$ and $Sat_E(y_i)$. If $\lambda_i|_{\vec{q}=\vec{q}^{\,\text{fix}},\, \vec{x}=\vec{x}^{\,\text{fix}},\, \vec{Z}_o=\vec{Z}_o^{\,\text{fix}}} = 0$, then the output λ_i is 0 in this state independently from the replacement of the Black Boxes, so we include $(\vec{q}^{\,\text{fix}}, \vec{x}^{\,\text{fix}}, \vec{Z}_o^{\,\text{fix}})$ neither into $Sat_A(y_i)$ nor $Sat_E(y_i)$. In any other case, the value of y_i is unknown in this state and thus we include $(\vec{q}^{\,\text{fix}}, \vec{x}^{\,\text{fix}}, \vec{Z}_o^{\,\text{fix}})$ into $Sat_E(y_i)$, but not into $Sat_A(y_i)$.

Based on this, the set of states surely resp. possibly satisfying the atomic property y_i can now be calculated as follows:

$$\chi_{Sat_A(y_i)}(\vec{q}, \vec{x}, \vec{Z}_o) = \forall Z \forall \vec{Z}_l \big(\lambda_i(\vec{q}, \vec{x}, Z, \vec{Z}_l, \vec{Z}_o)\big), \quad \chi_{Sat_E(y_i)}(\vec{q}, \vec{x}, \vec{Z}_o) = \exists Z \exists \vec{Z}_l \big(\lambda_i(\vec{q}, \vec{x}, Z, \vec{Z}_l, \vec{Z}_o)\big).$$

Likewise, we define fixed transitions (symbolically represented by χ_{R_A}) and possible transitions (symbolically represented by χ_{R_E}):

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

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

Based on χ_{R_A} and χ_{R_E}, we define $Sat_A(EX\psi)$ and $Sat_E(EX\psi)$ as follows: For $Sat_A(EX\psi)$, we include all states $(\vec{q}, \vec{x}, \vec{Z}_o)$, for which there exist $\vec{q}'$ and $\vec{x}'$, so that for all Black Box output values $\vec{Z}_o'$: $(\vec{q}', \vec{x}', \vec{Z}_o')$ is a sure successor of $(\vec{q}, \vec{x}, \vec{Z}_o)$ and $(\vec{q}', \vec{x}', \vec{Z}_o')$ surely satisfies ψ.

$$\chi_{Sat_A(EX\psi)}(\vec{q}, \vec{x}, \vec{Z}_o) = \exists \vec{q}' \exists \vec{x}' \Big(\chi_{R_A}(\vec{q}, \vec{x}, \vec{Z}_o, \vec{q}') \cdot \forall \vec{Z}_o' \big(\chi_{Sat_A(\psi)}|^{\vec{q}\leftarrow\vec{q}'}_{\vec{x}\leftarrow\vec{x}',\ \vec{Z}_o\leftarrow\vec{Z}_o'}\big)(\vec{q}', \vec{x}', \vec{Z}_o')\Big)$$

For $Sat_E(EX\psi)$, we include all states $(\vec{q}, \vec{x}, \vec{Z}_o)$, for which there exist $\vec{q}'$ and $\vec{x}'$, so that for at least one Black Box output value $\vec{Z}_o'$: $(\vec{q}', \vec{x}', \vec{Z}_o')$ is a possible successor of $(\vec{q}, \vec{x}, \vec{Z}_o)$ and $(\vec{q}', \vec{x}', \vec{Z}_o')$ possibly satisfies ψ.

$$\chi_{Sat_E(EX\psi)}(\vec{q}, \vec{x}, \vec{Z}_o) = \exists \vec{q}' \exists \vec{x}' \Big(\chi_{R_E}(\vec{q}, \vec{x}, \vec{Z}_o, \vec{q}') \cdot \exists \vec{Z}_o' \big(\chi_{Sat_E(\psi)}|^{\vec{q}\leftarrow\vec{q}'}_{\vec{x}\leftarrow\vec{x}',\ \vec{Z}_o\leftarrow\vec{Z}_o'}\big)(\vec{q}', \vec{x}', \vec{Z}_o')\Big)$$

The calculation of all remaining CTL operators $\neg$, EG and EU can be adopted from combined symbolic Z- and Z_i-model checking. The result of the recursive evaluation can be evaluated as follows:[2]

$$\forall \vec{x} \forall \vec{Z}_o \big(\chi_{Sat_A(\varphi)}|_{\vec{q}=\vec{q}^0}\big) = 1 \implies \varphi \text{ is valid}$$

$$\forall \vec{x} \exists \vec{Z}_o \big(\chi_{Sat_E(\varphi)}|_{\vec{q}=\vec{q}^0}\big) = 0 \implies \varphi \text{ is not realizable}$$

When comparing this combined method to (pure) symbolic output consistent Z_i-model checking, it is obvious that there is a smaller set of states to be considered, and thus a smaller set of transitions, which can lead to a less complex model checking run without necessarily losing all of the improved accuracy of symbolic output consistent Z_i-model checking.

Note that we do not need to perform two separate model checking runs to compute $Sat_E(\varphi)$ and $Sat_A(\varphi)$. By using an additional encoding variable e and defining $\chi_R = \overline{e} \cdot \chi_{R_A} + e \cdot \chi_{R_E}$, we can easily combine the two computations of $\chi_{Sat_A(\varphi)}$ and $\chi_{Sat_E(\varphi)}$ into one computation for $\chi_{Sat(\varphi)} = \overline{e} \cdot \chi_{Sat_A(\varphi)} + e \cdot \chi_{Sat_E(\varphi)}$.

[2]Let $\vec{q}^0$ be the initial state of the circuit.

4 Experimental Results

To demonstrate the feasibility and effectiveness of the presented method we modified the prototype model checker called MIND (Model Checker for Incomplete Designs) introduced in [1, 15] so it is now capable of performing combined symbolic Z-, Z_i- and output consistent Z_i-model checking. MIND is based on the BDD package CUDD 2.40 [16] and uses 'Lazy Group Sifting' [17], a reordering technique particularly suited for model checking, and partitioned transition functions [18]. All experiments were performed on an Dual Opteron 2GHz with 4GB RAM and with a limited runtime of 86.400 seconds.

For our experiments we used a class of simple synchronous pipelined ALUs (see Fig. 5) similar to the ones presented in [4]. In contrast to [4], the complete version of our pipelined ALU contains a combinational multiplier. Since combinational multipliers show exponential size regarding to their width if represented by BDDs [3], symbolic model checking for the complete design can only be performed up to a moderate bit width of the ALU.

In the following we consider a series of incomplete pipelined ALUs with 256 registers in the register file and varying word width. Both the adder and the multiplier of the pipelined ALU are substituted by Black Boxes, and all registers in the register file except the first four are masked out as well.

In our experiments, we checked the CTL formula $\varphi = AG\big(\text{"}\mathtt{R_2} := \mathtt{R_0} \oplus \mathtt{R_1}\text{"} \rightarrow \big((AX)^2\mathbf{R}_0 \oplus (AX)^2\mathbf{R}_1 \equiv (AX)^3\mathbf{R}_2\big)\big)$ which corresponds to formula (1) in [4]. It says that whenever the instruction $\mathtt{R_2} := \mathtt{R_0} \oplus \mathtt{R_1}$ is given at the inputs, the values in $\mathbf{R}_2$ three clock cycles in the future will be identical to the exclusive-or of $\mathbf{R}_0$ and $\mathbf{R}_1$ in the state two clock cycles in the future ($\mathbf{R}_0$, $\mathbf{R}_1$ and $\mathbf{R}_2$ are the respective first, second and third register in the register file). This property is true for our incomplete design, independently of the implementation of the adder function, the multiplier function and the registers masked out. Due to that, φ is satisfied for every possible Black Box replacement in the incomplete pipelined ALUs, thus valid.

First, we used symbolic output consistent Z_i-model checking for all Black Boxes in the circuit. We then reduced the accuracy for the Black Boxes in the register file by using symbolic Z_i-model checking resp. symbolic Z-model checking, while still using symbolic output consistent Z_i-model checking for the Black Boxes replacing the adder and the multiplier. Except for the timeouts, we always were able to prove the validity of φ. If symbolic Z- or Z_i-model checking was used for the Black Boxes replacing the adder and the multiplier, nothing could be proven.

In Tab. 1 we give the results for the incomplete pipelined ALUs with varying word width tested with φ, using symbolic Z-, Z_i- resp. output consistent Z_i-model checking for the Black Boxes in the register file and output consistent Z_i-model checking for the Black Boxes replacing the adder and the multiplier. For each word width and each method used for the Black Boxes in the register file, the table shows the number of BDD variables ('BDD vars'), the peak memory usage, the peak number of BDD nodes and the overall time in CPU seconds.

If symbolic output consistent Z_i-model checking is used for all Black Boxes, a complex transition relation has to be build between the states that contain the Z_i variables of all Black Boxes in the register file. On account of this, it is only possible to prove validity for a word width up to 8 bit before exceeding the time limit. Note that this method corresponds to *pure* symbolic output consistent Z_i-model checking as presented in [1].

If symbolic Z_i-model checking is used for the Black Boxes in the register file instead, only the Z_i's of the multiplier's and the adder's Black Box are included into the state space. Due to that, we have to deal with a smaller state space and a less complex transition relation, which leads to the result that we are able to prove validity for all instances within the time limit.

In the case that symbolic Z-model checking is used for the Black Boxes in the register file, there is a significant decrease of the number of necessary BDD variables, since all the Black Boxes in the register file now share one single Z variable. On account of this, there is a further speedup compared to the last experiment.

Taken together, the results show that by a flexible modeling of unknowns we are able to provide sound and useful results, yet at shorter time and with fewer memory consumption compared to methods which only allow to handle every Black Box in a design the same way.

word width	Black Boxes in Register File modeled by: symb. o.c.Z-model checking: BDD vars	memory used	BDD nodes	time	symb. Z_i-model checking: BDD vars	memory used	BDD nodes	time	symb. Z-model checking: BDD vars	memory used	BDD nodes	time
2	605	197923828	8499065	20606.16	605	12688820	93783	78.81	101	8104996	111386	4.42
4	1141	494039764	25627103	78505.29	1141	27136068	127272	453.89	133	17063700	140038	10.51
6	1677	387220180	17562606	64164.95	1677	37372260	92581	179.92	165	37980292	466098	85.06
8	2213	509172420	19683830	77214.36	2213	44797860	103384	157.24	197	28477652	153328	20.40
12	timeout				3285	61407956	97342	191.22	261	32558180	142978	31.67
16	timeout				4357	76706740	159200	241.49	325	40574868	156514	55.35
24	timeout				6501	105234068	192840	811.76	453	47584628	164318	119.68
32	timeout				8645	145393908	249088	1518.85	581	48671524	204879	197.85
48	timeout				12933	226031012	418484	15315.37	837	55666644	287263	538.13
64	timeout				17221	304953364	578314	37783.82	1093	59316948	566758	2031.79

Table 1: Incomplete pipelined ALU with 256 registers: Proving the validity of $\varphi = AG\big(''\mathtt{R}_2 := \mathtt{R}_0 \oplus \mathtt{R}_1'' \rightarrow \big((AX)^2\mathbf{R}_0 \oplus (AX)^2\mathbf{R}_1 \equiv (AX)^3\mathbf{R}_2\big)\big)$ using different methods to model the Black Boxes in the register file

5 Conclusions and Future Work

We introduced a method that combines symbolic Z-, Z_i- and output consistent Z_i-model checking into one single algorithm in such a way that different Black Boxes can be handled with differing methods (trading off accuracy and computation resources) in one single model checking run.

This allows us to handle less relevant (in terms of the CTL formula) Black Boxes with larger approximation and thus faster, without necessarily losing important information only the more exact methods can provide.

The combined method is able to provide sound results for falsifying realizability and for proving validity of incomplete designs (even if the Black Boxes lie inside the cone of influence for the considered CTL formula). Experimental results using our prototype implementation MIND proved that the need for computational resources (memory and time) of the combined method could be substantially decreased in comparison to one single complex and time-consuming method while still gaining the same result.

At the moment we are working on further improvements concerning the accuracy of our approximate symbolic model checking methods. Starting from a concept for exact symbolic model checking of incomplete designs (containing several Black Boxes with bounded memory) we develop appropriate approximations trading off accuracy and computational resources. Interestingly, it is possible to add these concepts to the combined method as well.

References

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

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

[4] 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.

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

[6] A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new Symbolic Model Verifier. In N. Halbwachs and D. Peled, editors, *Proceedings Eleventh Conference on Computer-Aided Verification (CAV'99)*, number 1633 in Lecture Notes in Computer Science, pages 495–499, Trento, Italy, July 1999. Springer.

[7] The VIS Group. VIS: A system for verification and synthesis. In *Computer Aided Verification*, volume 1102 of *LNCS*, pages 428–432. Springer Verlag, 1996.

[8] K.L. McMillan. *The SMV system - for SMV version 2.5.4*. Carnegie Mellon University, Nov. 2000.

[9] K. L. McMillan. *The SMV language*. Cadence Berkeley Labs, Cadence Berkeley Labs.

[10] T. Villa, G. Swamy, and T. Shiple. *VIS User's Manual*. Electronics Research Laboratory, University of Colorado at Boulder.

[11] 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.

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

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

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

[15] T. Nopper and C. Scholl. Symbolic model checking for incomplete designs. Technical report, Albert-Ludwigs-University, Freiburg, March 2004.

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

[17] H. Higuchi and F. Somenzi. Lazy group sifting for efficient symbolic state traversal of FSMs. In *Int'l Conf. on CAD*, pages 45–49, 1999.

[18] R. Hojati, S.C. Krishnan, and R.K. Brayton. Early quantification and partitioned transition relations. In *Int'l Conf. on Comp. Design*, pages 12–19, 1996.