Preprint from Proceedings of ITG/GI/GMM-Workshop “Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen”, Meissen, Germany, February 2001, pp. 31-43

Checking Equivalence for Partial Implementations

Christoph Scholl Bernd Becker
Institute of Computer Science
Albert–Ludwigs–University
D 79110 Freiburg im Breisgau, Germany
email: <name>@informatik.uni-freiburg.de

Abstract

We consider the problem of checking whether a partial implementation can (still) be extended to a complete design which is equivalent to a given full specification.

Several algorithms trading off accuracy and computational resources are presented: Starting with a simple 0,1,X-based simulation, which allows approximate solutions, but is not able to find all errors in the partial implementation, we consider more and more exact methods finally covering all errors detectable in the partial implementation. The exact algorithm reports no error if and only if the current partial implementation conforms to the specification, i.e. it can be extended to a full implementation which is equivalent to the specification.

We give a series of experimental results demonstrating the effectiveness and feasibility of the methods presented.

1 Introduction

Verification, i.e. the check whether a circuit implementation fulfills its specification, is a crucial task in VLSI CAD. Growing interest in universities and industry has lead to new results and significant advances concerning topics like property checking, state space traversal and combinational equivalence checking [7, 9, 18, 14].

For the purpose of this paper combinational equivalence checking is of particular interest. Here, the task is to check whether the Boolean functions corresponding to the specification and the implementation are the same. Besides functional validation by the application of test patterns, mainly two approaches are used to perform the equivalence check: One possibility is to translate implementation and specification into one Boolean formula which is satisfiable if and only if implementation and specification realize the same Boolean function [23, 16, 11]. As an alternative implementation and specification can be transformed into a canonical form such that the equivalence check reduces to a check whether the canonical representations of implementation and specification are the same. BDDs[4] and Word-level Decision Diagrams like *BMDs[6], HDDs[8] or K*BMDs[10] are popular choices for such canonical forms.

In this paper we address the problem of *Black Box Equivalence Checking*, which occurs when the specification is known, but only parts of the implementation are finished or known. (For an example see Figures 1(a), 1(b).) *Black Box Equivalence Checking* enables the use of verification techniques in early stages of the design. Design errors can be already detected when only a partial implementation is at hand – e.g. due to a distribution of the implementation task to several groups of designers. Parts of the implementation, which are not yet finished, are combined into *Black Boxes*. If the implementation differs from the specification *for all possible substitutions of the Black Boxes*, a design error is found in the current partial implementation, i.e. to detect an error in the current partial implementation it is necessary to find an assignment of zeros and ones to the primary inputs, which produces erroneous values at the outputs *independently from the final implementation of the Black Boxes*.

Another application of *Black Box Equivalence Checking* is the abstraction from "difficult parts" of an implementation, which would cause a large peak size in memory consumption during the construction of a canonical form for the implementation. These "difficult parts" of the design can be put into a Black Box and Black Box Equivalence Checking is performed. An exact statement about the correctness of the full implementation is not possible, but it is still possible to find errors in the partial implementation given to the Black Box Equivalence Checker.

Black Box Equivalence Checking can also be used to verify assumptions concerning the location of errors in implementations, which do not fulfill their specifications: If there is some assumption on the location of errors (produced by an automatic error diagnosis tool or found by hand), then these regions of the design are cut off and put into Black Boxes. If Black Box Equivalence Checking gives the information that no error can be found in the design containing Black Boxes, we can conclude that the assumptions on the error location were correct, otherwise we know that there must be errors also in other regions of the design.

The present paper deals with algorithms for equivalence checking of partial implementations under the assumption that a combinational circuit is given as specification and also all implementations and Black Boxes are of combinational nature. First methods to handle this problem have been proposed in [13, 12]. While these papers provide algorithms to find errors, it is not clear which errors and how many of the potential errors are detected. In this paper we present a thorough analysis of the problem leading to several algorithms to attack the Black Box Equivalence Checking problem. For the time being, our algorithms rely on symbolic simulation [5] by using BDDs. An implementation using SAT-engines [17] to solve the corresponding Boolean formula seems feasible, but is not the focus of the current paper. Our algorithms need different amounts of resources (space and time) and differ from their accurateness: They range from a simple algorithm using symbolic simulation for an approximation of the solution to an exact solution of the problem. Thereby the methods given in [13, 12] are classified too. Approximate solutions are not able to find all errors in the partial implementation, but they are correct in the sense that they do not report an error if there is still a possibility to implement the Black Boxes leading to a correct overall implementation. However, if we solve the Black Box Equivalence Checking approximatively, the information, that no error can be found, can be due to the approximative character of the approach and does not necessarily imply that there is an implementation of the Black Boxes leading to a correct overall implementation. E.g. when Black Box Equivalence Checking is used to verify assumptions on the location of design errors, it can not be guaranteed that the information, that no error can be found, implies that the error location is confined to the Black Boxes (since this information can be due the approximative character of the approach). We performed several experiments, which showed that improving the accuracy of the algorithms indeed leads to a significant improvement of the error detection capabilities (paid with an increase of computational resources).

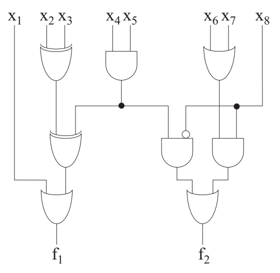
The paper is structured as follows: In Section 2 we give some preliminaries. Several algorithms for the Black Box Equivalence Checking problem are presented in Section 3. In Section 4 the different approaches are compared for numerous partial implementations of benchmark circuits. The paper ends with some concluding remarks and directions for further research in Section 5. For shortness of the paper no formal proofs are given. They can be found in [21].

2 Preliminaries

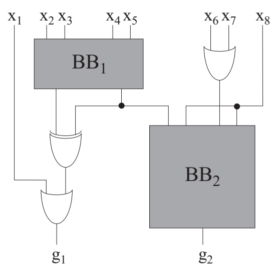
Let $f : \{0,1\}^n \rightarrow \{0,1\}$ be a Boolean function with n inputs. As usual, for a constant $b \in \{0,1\}$ and an input variable $x_i (1 \leq i \leq n)$ $f|_{x_i=b}(x_1,\dots,x_n) = f(x_1,\dots,x_{i-1},b,x_{i+1},\dots,x_n)$ denotes the *cofactor* of f with respect to $x_i = b$. Instead of $f|_{x_i=0}$ and $f|_{x_i=1}$ we also write $f_{\overline{x}_i}$ and f_{x_i}, respectively.

The *cofactor* of f with respect to a set of variables and constants is defined inductively:

$$f|_{x_{i_1}=b_1,\dots,x_{i_r}=b_r} = (f|_{x_{i_1}=b_1,\dots,x_{i_{r-1}}=b_{r-1}})|_{x_{i_r}=b_r}.$$



(a) Specifying circuit



(b) Partial implementation with two Black Boxes

Figure 1: Specification and partial implementation.

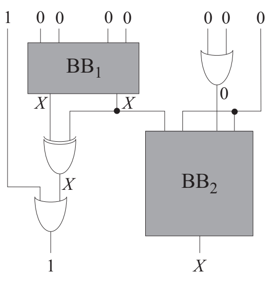
A cofactor $f|_{x_1=b_1,\ldots,x_n=b_n}$ with respect to all input variables is called a *complete* cofactor.
Boolean functions can be represented by BDDs [15, 2, 19]. In the restricted form of ROBDDs they even provide canonical representations for Boolean functions and they allow efficient manipulations [4]. ROBDDs can be used to check equivalence of Boolean functions by a simple check for equality. Since we work only with ROBDDs in the following we briefly call them BDDs.
Given a circuit representation of a Boolean function, a BDD for this Boolean function can be computed by *symbolic simulation* [5]. At the beginning of the symbolic simulation each input of the circuit is associated with a unique BDD variable. Then the BDD representations of the functions computed by the gates of the circuit are computed in topological order starting with the inputs. The BDD for the function of a gate can be computed using BDD operations [4, 3], when the BDDs for the functions of all its predecessor gates are already computed.

3 Equivalence checking and partial implementations

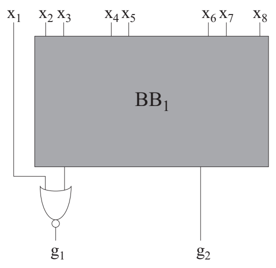
In this section we provide several algorithms to handle the Black Box Equivalence Checking. We start with a simple symbolic simulation with respect to the 0,1,X logic (Sec. 3.1). Then we successively increase the exactness (and the complexity) of the algorithm leading to a *local check* (Sec. 3.2.1), an *output exact check* (Sec. 3.2.2) and an *input exact check* (Sec. 3.2.3).
In particular, in Section 3.2.3 we give for the first time an *exact* criterion to decide for a given partial implementation and a specification whether the partial implementation is correct or not. Unlike previous approaches [13, 12] we can guarantee that there is really an extension of the partial implementation to a correct complete implementation, if the criterion of Section 3.2.3 reports no error (and of course, vice versa, there is no extension of the partial implementation to a complete implementation, if it does report an error).
As a running example for the demonstration of our algorithms we use the specification given in Figure 1(a). Figure 1(b) shows a partial implementation containing two Black Boxes. Clearly, after a suitable implementation of the two Black Boxes the final implementation fulfills its specification.

3.1 Symbolic Z–simulation

A first algorithm for checking partial implementations is based on the usual 0,1,X simulation, which is well-known in the area of testing [1].
To evaluate a partial implementation for some input vector a new symbol X different from 0 and 1 is introduced. The value X means an “unknown” value due to the unknown functionality of the Black Boxes. To simulate a partial implementation with n primary inputs for an input vector $(\epsilon_1,\ldots,\epsilon_n) \in \{0,1\}^n$ we assign the unknown value X to all outputs of the Black Boxes. If all values for the inputs of a gate are in $\{0,1\}$, then the output of the gate is computed according to the gate function as usual. If some inputs of a gate are set to X, the output is equal to X if and only if there are two different replacements of the X values at the inputs by 0’s and 1’s, which lead to different outputs of the gate.



(a) Evaluation wrt. input vector $(1, 0, 0, 0, 0, 0, 0, 0)$



(b) Partial implementation with a detectable error

Figure 2: 0,1,X simulation for partial implementations

As an example Figure 2(a) shows the evaluation of the partial implementation of Figure 1(b) with respect to input vector $(1, 0, 0, 0, 0, 0, 0, 0)$. Note that the first output of the partial implementation is 1 independently from the functionality of the Black Boxes.

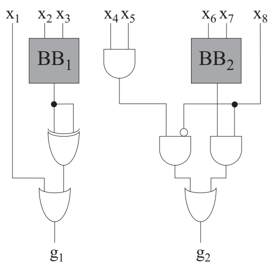
We can take advantage of this simulation using 0, 1 and X to detect errors in partial implementations. If the evaluation of the partial implementation results in a value 0 (1) for some output, this means that the output value is 0 (1) independently from the functionality of the Black Boxes. If on the other hand the specification produces 1 (0) for the same input vector, then we have found an error in the partial implementation.

Figure 2(b) shows such a situation: When we compare the partial implementation to the specification of Figure 1(a) applying input vector $(1, 0, 0, 0, 0, 0, 0, 0)$, we see that the first output of the (partial) implementation is 0 whereas it is 1 for the specification. Generalizing the usual notion of a distinguishing vector for designs without Black Boxes to designs containing Black Boxes we can say that $(1, 0, 0, 0, 0, 0, 0, 0)$ is a *distinguishing vector* for the specification in Figure 1(a) and the partial implementation in Figure 2(b). Of course, only vectors which produce 0 or 1 (not X) at outputs of the partial implementation can play a role as distinguishing vectors.

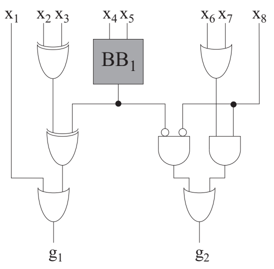
Since we do not want to simulate specification and implementation for all 2^n input vectors one after the other to find distinguishing vectors, we make use of "symbolic simulation". This leads to our first method for Black Box equivalence checking, which is similar to [13]. In contrast to [13] we do not use a two-bit-encoding of 0, 1 and X leading to a duplication of the signals of the circuit, but we use an additional variable Z to model the new value X. For an output i of a partial implementation our symbolic simulation computes a BDD for a function g_i with $n + 1$ inputs $x_1, \ldots x_n, Z$ and

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

To compute BDDs for the functions g_i by symbolic simulation the inputs of the circuit are associated with unique BDD variables as in a conventional symbolic simulation. All output signals of Black Boxes are associated with the new variable Z. Now BDDs for the functions computed by the gates of the circuit are built in topological order treating the Black Box outputs (associated with variable Z) as inputs of the circuit. The gates of the circuit can be processed in a manner similar to a conventional symbolic simulation. Since all types of gates can be expressed using two-input and_2 gates, two-input or_2 gates and inv gates, we can assume w.l.o.g. that the gates have types and_2, or_2 or inv. When we process an and_2 (or_2) gate, we combine the BDDs for the two predecessor functions by a BDD AND (OR) operation as in the conventional symbolic simulation. For an inv gate we perform a NOT operation on the BDD of the predecessor function, now followed by a $compose$ operation (see e.g. [4]) which composes $\overline{Z}$ for Z (written as $g|_{Z \leftarrow \overline{Z}}$ for a composition of $\overline{Z}$ for Z in g).



(a) Partial implementation with error detectable by Z_i–simulation, but not by Z–simulation



(b) Motivation for output exact check

Figure 3: Motivation for local check and output exact check.

It is easy to see that this procedure leads to BDD representations fulfilling property (1).
We call the procedure described above "symbolic Z-simulation". After we have obtained functions g_i for all outputs of the partial implementation by symbolic Z-simulation and functions f_i for all outputs of the specification by a conventional symbolic simulation, the check whether there is a distinguishing vector between specification and implementation is based on the following lemma, whose correctness follows from the definitions and basic boolean manipulations.

Lemma 3.1 (Z-simulation) *There is no input vector* $(\epsilon_1, \ldots, \epsilon_n)$ *with* $g_i|_{x_1=\epsilon_1,\ldots,x_n=\epsilon_n} = 1$ *and* $f_i(\epsilon_1, \ldots, \epsilon_n) = 0$ *iff*

$$(g_i|_{Z=0} \rightarrow f_i) = 1$$

and there is no input vector $(\epsilon_1, \ldots, \epsilon_n)$ *with* $g_i|_{x_1=\epsilon_1,\ldots,x_n=\epsilon_n} = 0$ *and* $f_i(\epsilon_1, \ldots, \epsilon_n) = 1$ *iff*

$$(\overline{g_i}|_{Z=1} \rightarrow \overline{f_i}) = 1.$$

3.2 Symbolic Z_i–simulation

A disadvantage of symbolic Z–simulation lies in the fact that not all errors, which are present in a partial implementation, can be found by the procedure described above.
Figure 3(a) shows an example for such a situation. The partial implementation of Figure 3(a) does not fulfill the specification of Figure 1(a), i.e. there is no implementation for the Black Boxes which leads to a correct overall implementation. However the approach of the previous section always computes X at the output of the $exor_2$ gate, since both inputs of the $exor_2$ gate are X. Therefore the first primary output is X, if $x_1 = 0$, and 1, if $x_1 = 1$. Since the first output of the specification is 1 as well, if $x_1 = 1$, no error can be detected at the first output. Moreover it is easy to see that the partial implementation of the second output is correct (replace BB_2 by an or_2 gate). So the method of the previous section cannot detect an error in the partial implementation.
If we have a closer look at the partial implementation, we can see, that the first output does not depend on the output of Black Box BB_1: The output of the $exor_2$ gate whose inputs are connected to the output of BB_1 is 0 *independently from the output of* BB_1. Therefore input vector $(0, 0, 0, 1, 1, 0, 0, 0)$ leads to output $(0, 1)$ which is different from the output $(1, 1)$ of the specification.
The reason why the error could not be detected by Z–simulation lies in the fact that the values X at the inputs of the $exor_2$ gate are not accurately processed: The simple X–propagation does not take into account that the X–information comes from the same output of the Black Box BB_1 (resulting in value 0 at the output).
To consider the origin of X–informations it is not enough to introduce one variable Z for all Black Box outputs. Instead of that we introduce *different* variables Z_i for each Black Box output i. After the inputs of the circuit have been associated with unique BDD variables a

(conventional) symbolic simulation is performed. The result for primary output j of the circuit is a function g_j which depends on the primary input variables $x_1, \ldots, x_n$ and the l variables $Z_1, \ldots, Z_l$ for the l outputs of Black Boxes.

Example 3.1 *Symbolic simulation with output variables Z_1 for BB_1 and Z_2 for BB_2 in Figure 3(a) results in $g_1 = x_1$ for output 1 and $g_2 = \overline{x_8} \cdot (x_4 \cdot x_5) + x_8 \cdot Z_2$ for output 2.*

3.2.1 Local check

As in the case of Z–simulation we consider *complete* cofactors of implementation and specification with respect to all primary input variables[1]. If a complete cofactor of some output function of the partial implementation is 0 (1), this means that the output value is 0 (1) independently from the functionality of the Black Boxes. If for the same output function of the specification this cofactor is 1 (0), then we have found an error in the partial implementation. The only difference of this method compared to Z–simulation is the fact that the effect of the unknown values at the outputs of Black Boxes is evaluated more accurately.
A check whether there is such a distinguishing vector for an output j of partial implementation and specification can be done according to the following lemma. Again, the correctness of the lemma follows from definitions and basic boolean manipulations. The check according to the lemma is called “local check”, since the check is done for each output separately.

Lemma 3.2 (local check) *Let g_j be the function of output j after symbolic Z_i–simulation for a partial implementation with primary inputs $x_1, \ldots, x_n$ and l outputs of Black Boxes with corresponding variables $Z_1, \ldots, Z_l$. Let f_j be output j of a specification with primary inputs $x_1, \ldots, x_n$.*
There is no input vector $(\epsilon_1, \ldots, \epsilon_n)$ with $g_j|_{x_1=\epsilon_1,\ldots,x_n=\epsilon_n} = 1$ and $f_j(\epsilon_1, \ldots, \epsilon_n) = 0$ iff

$$((\forall Z_1 \ldots \forall Z_l \; g_j) \rightarrow f_j) = 1$$

and there is no input vector $(\epsilon_1, \ldots, \epsilon_n)$ with $g_j|_{x_1=\epsilon_1,\ldots,x_n=\epsilon_n} = 0$ and $f_j(\epsilon_1, \ldots, \epsilon_n) = 1$ iff

$$((\forall Z_1 \ldots \forall Z_l \; \overline{g_j}) \rightarrow \overline{f_j}) = 1.$$

3.2.2 Output exact check

The local check of the previous section based on Z_i–simulation is more exact than Z–simulation (see Fig. 3(a)). However implications between different outputs are not taken into account. We obtain an even more accurate check, if we use a more “global” viewpoint.
This is illustrated by Figure 3(b). For the first output the only possibility to fulfill the specification of Figure 1(a) is to replace BB_1 by the function $x_4 \cdot x_5$. However for the second output the only possibility to fulfill the specification is to replace BB_1 by $\overline{x_4 \cdot x_5}$. This implies that the partial implementation of Figure 3(b) is incorrect. In spite of that, the error can not be detected by the “local check” of the previous section, since it is done for each output separately.
To detect errors of this type we have to compute “local conditions” for each output, which guarantee correctness for the single outputs, and then, we have to combine the local conditions to check, if these local conditions can be fulfilled at the same time for all outputs.
The local conditions are computed based on the following considerations: To obtain a correct implementation, for each primary output j and each assignment $(\epsilon_1, \ldots, \epsilon_n)$ to the primary inputs an assignment $(\delta_1, \ldots, \delta_l)$ to the Black Box outputs has to be chosen such that

$$g_j(\epsilon_1, \ldots, \epsilon_n, \delta_1, \ldots, \delta_l) \quad \text{and} \quad f_j(\epsilon_1, \ldots, \epsilon_n)$$

[1]Here we call a cofactor complete, if it is a cofactor with respect to all *primary* input variables.

are identical, i.e. such that

$$[(g_j)|_{x_1=\epsilon_1,\ldots,x_n=\epsilon_n} \equiv (f_j)|_{x_1=\epsilon_1,\ldots,x_n=\epsilon_n}](\delta_1,\ldots,\delta_l) = 1.$$

The relation between input assignments $(\epsilon_1,\ldots,\epsilon_n)$ and assignments $(\delta_1,\ldots,\delta_l)$ to the outputs of Black Boxes, which are necessary to fulfill the specification, can be expressed by the characteristic function

$$\begin{aligned} &cond_j(\epsilon_1,\ldots,\epsilon_n,\delta_1,\ldots,\delta_l) = \\ &= \bigvee_{(\epsilon_1,\ldots,\epsilon_n)\in\{0,1\}^n} x_1^{\epsilon_1}\cdot\ldots\cdot x_n^{\epsilon_n}\cdot[(g_j)|_{x_1=\epsilon_1,\ldots,x_n=\epsilon_n} \equiv (f_j)|_{x_1=\epsilon_1,\ldots,x_n=\epsilon_n}]. \end{aligned}$$

The characteristic function $cond_j$ equals 1 for argument $(\epsilon_1,\ldots,\epsilon_n,\ \delta_1,\ldots,\delta_l)$, if and only if $g_j(\epsilon_1,\ldots,\epsilon_n,\delta_1,\ \ldots,\delta_l)$ and $f_j(\epsilon_1,\ldots,\epsilon_n)$ are identical, i.e. if and only if an assignment of $(\delta_1,\ldots,\delta_l)$ to the Black Box outputs for input $(\epsilon_1,\ldots,\epsilon_n)$ leads to a correct output of the implementation for this input $(\epsilon_1,\ldots,\epsilon_n)$. It holds that

$$cond_j = (g_j \equiv f_j)$$

and $cond_j$ can be computed using one BDD operation for g_j and f_j.
For a correct partial implementation all conditions $cond_1,\ldots,cond_m$ have to be true. If there is an input assignment $(\epsilon_1,\ldots,\epsilon_n)$ such that for all assignments $(\delta_1,\ldots,\delta_l)$ to the Black Box outputs at least one condition $cond_j$ is false, then it is clear that the partial implementation can not be used to obtain a correct final implementation. This leads us to a new, more accurate check, which we call "output exact".

Lemma 3.3 (output exact check) *If*

$$\exists x_1\ldots\exists x_n\forall Z_1\ldots\forall Z_l\bigvee_{j=1}^{m}\overline{cond_j} = 1$$

then the partial implementation does not fulfill its specification.

Example 3.2 *Consider the partial implementation of Figure 3(b) and the specification of Figure 1(a). For input* $(0,0,0,1,1,0,0,0)$*, output 1 of the implementation equals output 1 of the specification only if the Black Box output is* 1 *for this input. This implies* $cond_1(0,0,0,1,1,0,0,0,\delta_1) = 1$ *only if* $\delta_1 = 1$*. However, for input* $(0,0,0,1,\ 1,0,0,0)$ *output 2 of the implementation equals output 2 only if the Black Box output is* 0 *for this input. This implies* $cond_2(0,0,0,1,1,\ 0,0,0,\delta_1) = 1$ *only if* $\delta_1 = 0$*. There is no assignment to* δ_1*, such that both* $cond_1(0,0,0,1,1,0,0,0,\delta_1)$ *and* $cond_2(0,0,0,1,1,0,\ 0,0,\delta_1)$ *are equal to* 1 *and therefore the "output exact" check reports an error.*

Note that our "output exact" check reports an error in exactly the same cases as the check of [12]. However it is computed in a different way and does not need a representation of the overall circuit as a Boolean relation.
It is also easy to see that there is no error in the partial implementation (i.e. we can replace the Black Boxes to obtain a correct final implementation), if our check reports no error and *we are allowed to use all primary inputs as inputs of the Black Boxes.* The check reports no error iff

$$\forall x_1\ldots\forall x_n\exists Z_1\ldots\exists Z_l\bigwedge_{j=1}^{m} cond_j = 1,$$

i.e. iff for each assignment $(\epsilon_1,\ldots,\epsilon_n)$ to the primary inputs there exists an assignment $(\delta_1,\ldots,\ \delta_l)$ to the Black Box outputs, such the conditions $cond_j$ for all outputs j are true, which means that $g_j(\epsilon_1,\ldots,\epsilon_n,\delta_1,\ldots,\delta_l)$ and $f_j(\epsilon_1,\ldots,\epsilon_n)$ are identical for all $1 \le j \le m$. Thus we can choose these values $\delta_1,\ldots,\delta_l$ to define the function values for Black Box outputs $1,\ldots,l$ under input $(\epsilon_1,\ldots,\epsilon_n)$.

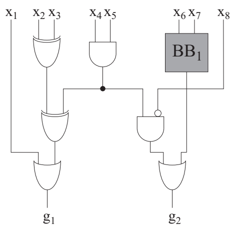


Figure 4: Motivation for input exact check

3.2.3 Input exact check

The output exact check is able to find all errors which are already present in the partial implementation only if we assume that all primary inputs are also inputs of the Black Boxes. But this is not a realistic assumption. If we have fixed sets of input signals for the Black Boxes (which may be different from all primary inputs), it is possible that the output exact check does not find all errors.

Figure 4 shows such a case. It shows a partial implementation (for the specification of Figure 1(a)) with one Black Box BB_1. If the Black Box is replaced by $x_8 \cdot (x_6 + x_7)$ implementation and specification are equivalent. However the inputs of the Black Box are only x_6 and x_7.

- Output 2 of the specification is 0 for $x_4 = x_5 = 0$, $x_6 = x_7 = 1$ and $x_8 = 0$. Therefore BB_1 has to be 0 under this input, since output 2 of the implementation is 1, if the output of BB_1 is 1.
- Output 2 of the specification is 1 for $x_4 = x_5 = 0$, $x_6 = x_7 = 1$ and $x_8 = 1$. Therefore BB_1 has to be 1 under this input, since output 2 of the implementation would be 0 otherwise.

We can conclude that there is no correct implementation for BB_1 which does not depend on input x_8 and so the partial implementation is incorrect.

Now we have to define a check which also reflects this problem. For this check we use (among others) the condition

$$cond = \bigwedge_{j=1}^{m} cond_j$$

of the section before. $cond$ can be interpreted as the characteristic function of a Boolean relation between assignments $(\epsilon_1, \ldots, \epsilon_n)$ to the primary input variables and assignments $(\delta_1, \ldots, \delta_l)$ to the outputs of the Black Boxes:

$$cond(\epsilon_1, \ldots, \epsilon_n, \delta_1, \ldots, \delta_l) = 1$$

if and only if $(\delta_1, \ldots, \delta_l)$ is a “legal assignment” to the outputs of the Black Boxes for primary input vector $(\epsilon_1, \ldots, \epsilon_n)$, i.e. if and only if all output values of the partial implementation with $(\epsilon_1, \ldots, \epsilon_n)$ assigned to the primary inputs and $(\delta_1, \ldots, \delta_l)$ assigned to the Black Box outputs are identical to the corresponding output values of the specification for assignment $(\epsilon_1, \ldots, \epsilon_n)$ to the primary inputs.

Now we have to take into account that the inputs of Black Boxes can be internal signals of the partial implementation and not all primary inputs are connected to the Black Box inputs. In the following we assume that we have b Black Boxes BB_1 to BB_b which can have several outputs and inputs. The input signals of Black Box BB_j are connected to variables $i_{j,1}, \ldots, i_{j,l_j}$ and the output signals are connected to variables $o_{j,1}, \ldots, o_{j,p_j}$ $(\bigcup_{j=1}^{b} \{o_{j,1}, \ldots, o_{j,p_j}\} = \{Z_1, \ldots, Z_l\})$. To simplify the notations we abbreviate $i_{j,1}, \ldots, i_{j,l_j}$ by I_j, $o_{j,1}, \ldots, o_{j,p_j}$ by O_j and the primary input variables $x_1, \ldots, x_n$ by X. Moreover $\forall I_j$ means $\forall i_{j,1} \ldots \forall i_{j,l_j}$ and $\forall O_j$ means $\forall o_{j,1} \ldots \forall o_{j,p_j}$ (accordingly for $\exists$).

We assume that the Black Boxes BB_1 to BB_b are topologically ordered, i.e. BB_1 is the first Black Box in topological order, BB_b the last Black Box. Consider the Boolean functions which compute the assignments of the Black Box inputs. For Black Box BB_j there are l_j such functions $h_1^j, \ldots, h_{l_j}^j$. Because of the topological order of the Black Boxes, $h_1^j, \ldots, h_{l_j}^j$ can depend (at most) on primary input variables X and the output variables $O_1, \ldots, O_{j-1}$ of $BB_1, \ldots, BB_{j-1}$. The characteristic function of the Boolean relation for $h_1^j, \ldots, h_{l_j}^j$ is computed by

$$H_j(X, O_1, \ldots, O_{j-1}, I_j) = \bigwedge_{k=1}^{l_j} (i_{j,k} \equiv h_k^j(X, O_1, \ldots, O_{j-1})).$$

Based on $cond(X, O_1, \ldots, O_b)$, which is a Boolean relation between primary input assignments and output assignments of Black Boxes, we compute the characteristic function of a Boolean relation $cond'(I_1, \ldots, I_b, O_1, \ldots, O_b)$ between input assignments of Black Boxes and output assignments of Black Boxes. $cond'$ is defined as

$$\begin{aligned} & cond'(I_1, \ldots, I_b, O_1, \ldots, O_b) = \\ = \ & \forall X \left(\overline{H_1(X, I_1)} + \ldots + \overline{H_b(X, O_1, \ldots, O_{b-1}, I_b)} + cond(X, O_1, \ldots, O_b) \right). \end{aligned}$$

$cond'$ computes 1 for an assignment $(\iota_1, \ldots, \iota_b, \omega_1, \ldots, \omega_b)$ to the Black Box inputs and outputs iff for all assignments ξ to the primary inputs

- ξ and $(\iota_1, \ldots, \iota_b, \omega_1, \ldots, \omega_b)$ lead to a signal assignment, which is not consistent with the circuit of the partial implementation (this is checked by the part $\overline{H_1(X, I_1)} + \ldots + \overline{H_b(X, O_1, \ldots, O_{b-1}, I_b)}$ of the formula above)
  or
- $(\omega_1, \ldots, \omega_b)$ is a "legal output" of the Black Boxes under input ξ, i.e. ξ and $(\omega_1, \ldots, \omega_b)$ result in correct values at the primary outputs of the partial implementation
  (this is checked by the part $cond(X, O_1, \ldots, O_b)$ of the formula above).

I.e. for $cond'(\iota_1, \ldots, \iota_b, \omega_1, \ldots, \omega_b)$ to be 1, $(\omega_1, \ldots, \omega_b)$ has to be a "legal output" of the Black Boxes under input ξ, whenever ξ and $(\iota_1, \ldots, \iota_b, \omega_1, \ldots, \omega_b)$ lead to a signal assignment, which is consistent with the circuit of the partial implementation.
It can be shown that there is a replacement of the Black Boxes $BB_1, \ldots, BB_b$ by totally specified Boolean functions with input variables $I_1, \ldots, I_b$, respectively, leading to a correct overall implementation if and only if there is a appropriate decomposition of $cond'$ into b Boolean relations:

Theorem 3.1 (input exact check) *Let $f_1, \ldots, f_m$ be Boolean functions with input variables $x_1, \ldots, x_n$, which is used as a specification for a partial implementation with input variables $x_1, \ldots, x_n$ and b Black Boxes $BB_1, \ldots, BB_b$. The input variables of BB_j are I_j, the output variables O_j, the characteristic function $cond'(I_1, \ldots, I_b, O_1, \ldots, O_b)$ is defined as shown above. Then there is a replacement of $BB_1, \ldots, BB_b$ by completely specified Boolean functions with input variables $I_1, \ldots, I_b$, respectively, leading to a correct overall implementation if and only if $cond'$ can be decomposed into $\chi_j(I_j, O_j)$, such that*

$$\forall I_j \exists O_j \chi_j(I_j, O_j) = 1 \text{ and} \tag{2}$$

$$cond' \geq \bigwedge_{j=1}^{b} \chi_j. \tag{3}$$

circuit	in	out	#nodes	detected errors					#nodes implementation			peak during check				run time				
			spec.	$r.p.$	Z	$loc.$	oe	ie	Z	$loc., oe$	ie	Z	$loc.$	oe	ie	$r.p.$	Z	$loc.$	oe	ie
alu4	14	8	389	90%	95%	95%	96%	96%	458	455	490	86	88	96	159	1.17	0.06	0.06	0.06	0.06
apex7	49	37	314	92%	97%	97%	98%	98%	256	258	263	38	41	132	132	0.41	0.08	0.08	0.08	0.08
C17	5	2	8	84%	88%	88%	88%	96%	6	6	8	5	6	6	7	0.02	0.01	0.01	0.01	0.01
C432	36	7	1211	50%	62%	65%	68%	80%	796	3705	3725	123	257	5779	38411	3.22	0.13	1.77	0.42	0.99
C499	41	32	25866	26%	59%	59%	69%	80%	4377	12700	12672	487	496	28562	39142	6.29	4.46	5.54	7.19	7.76
C880	60	26	4870	78%	87%	91%	92%	92%	2956	5600	5553	247	658	105919	116561	3.84	0.75	1.62	37.94	49.16
comp	32	3	137	27%	63%	65%	67%	90%	82	90	111	39	41	105	124	1.57	0.04	0.04	0.04	0.04
term1	34	10	81	92%	95%	95%	95%	95%	97	97	108	31	32	34	69	1.44	0.07	0.07	0.07	0.07
average				63%	81%	82%	84%	91%												

Table 1: 10% of the gates included in one Black Box

The proof of Theorem 3.1 is technically complicated and can be found in [21].

Theorem 3.1 gives us a necessary and sufficient condition for the correctness of the partial implementation.

However, we can show using a non–trivial reduction from 3SAT that for a number $b \geq 2$ of Black Boxes the check of Theorem 3.1 is NP-complete, even if the characteristic function for $cond'$ in Theorem 3.1 is given as a function table, which is already exponential in the number of inputs and outputs of the Black Boxes. For this reason we use for practical application a modified check which is

- exact for $b = 1$ (one Black Box) and
- an approximation for $b \geq 2$ (more than one Black Box).

Our new check, which reflects that the inputs of the Black Boxes are not necessarily equal to all primary input signals, reports *no error*, if

$$\forall I_1 \exists O_1 \forall I_2 \exists O_2 \ldots \forall I_b \exists O_b \; cond' = 1 \tag{4}$$

The following theorem holds

Theorem 3.2 *The check of equation (4) is exact (in the sense that in finds all errors in the partial implementation), if $b = 1$, i.e. if there is only one Black Box in the partial implementation.*

Proof: The proof follows directly from the fact, that for $b = 1$ the checks of equation (4) and of Theorem 3.1 are the same, if we choose $\chi_i := cond'$ (the truth of part (3) is then trivial). □

In the general case, when more than one Black Box is present, the check of equation 4 is not exact, i.e. it is not equivalent to the check of Theorem 3.1, but we can formally prove that it is at least as good as our best check so far (see Section 3.2.2). In Section 4 we present experiments to demonstrate that it is *really better* also for examples with several Black Boxes.

4 Experimental results

To evaluate the different equivalence checks for partial implementations we implemented the described procedures using $CUDD$ 2.3.0 [22] as the underlying BDD package. Dynamic reordering [20] was activated during all experiments. The experiments were performed on a PentiumIII PC with 550 MHz, 1 GB memory, running Linux 6.3.

For our experiments we generated partial implementations from benchmark circuits: For each benchmark circuit a certain fraction of the gates was included in Black Boxes. In a first experiment we included 10% of the gates in one Black Box (with several outputs). All reported results are an average on 5 different random selections of Black Boxes.

Then we inserted errors into the partial implementations: We randomly selected a gate, which did not belong to a Black Box, and inserted an error. The error type was also selected randomly between several choices: We added/removed an inverter for an input or output signal of the gate, changed the type of the gate (and_2 to or_2 or or_2 to and_2) or removed an input line from

circuit	in	out	#nodes	detected errors					#nodes implementation			peak during check				run time				
			spec.	$r.p.$	Z	$loc.$	oe	ie	Z	$loc., oe$	ie	Z	$loc.$	oe	ie	$r.p.$	Z	$loc.$	oe	ie
alu4	14	8	389	75%	81%	81%	84%	90%	393	68005	52619	71	84	821289	800302	2.26	0.04	462.34	341.04	129.35
apex7	49	37	314	92%	97%	97%	98%	98%	284	282	287	38	42	128	127	0.41	0.06	0.06	0.06	0.06
C17	5	2	8	59%	88%	88%	88%	98%	6	6	8	4	6	8	8	0.07	0.01	0.01	0.01	0.01
C432	36	7	1211	47%	60%	65%	70%	80%	579	7186	7034	84	390	12094	603833	3.03	0.40	3.67	0.94	29.27
C499	41	32	25866	13%	38%	40%	62%	66%	3125	14495	14530	298	330	39552	1224345	6.39	2.99	3.61	4.92	129.98
C880	60	26	4870	78%	87%	91%	92%	92%	2941	5557	5527	246	661	108383	121482	3.84	0.47	0.95	25.14	32.08
comp	32	3	137	29%	64%	66%	68%	88%	79	91	109	36	38	95	135	1.51	0.03	0.03	0.03	0.04
term1	34	10	81	92%	96%	96%	96%	96%	99	102	106	30	32	34	44	1.39	0.04	0.05	0.05	0.05
average				61%	76%	78%	82%	89%												

Table 2: 40% of the gates included in one Black Box

circuit	in	out	#nodes	detected errors					#nodes implementation			peak during check				run time				
			spec.	$r.p.$	Z	$loc.$	oe	ie	Z	$loc., oe$	ie	Z	$loc.$	oe	ie	$r.p.$	Z	$loc.$	oe	ie
alu4	14	8	389	50%	92%	92%	94%	94%	346	372	548	83	85	103	419	4.93	0.08	0.08	0.08	0.10
apex7	49	37	314	88%	96%	96%	98%	98%	235	232	249	28	37	220	720	0.53	0.10	0.09	0.11	0.12
C17	5	2	8	84%	88%	88%	88%	96%	6	6	7	5	6	6	7	0.03	0.02	0.02	0.02	0.02
C432	36	7	1211	34%	54%	66%	72%	87%	417	5675	6065	104	463	6577	28471	4.47	0.18	1.39	0.61	1.40
C499	41	32	25866	20%	44%	46%	58%	75%	1858	8443	9246	199	207	190168	534687	6.75	3.23	4.40	41.82	68.52
C880	60	26	4870	61%	75%	80%	82%	88%	1276	3851	4055	207	444	1249899	1521876	6.16	1.22	1.10	1140.11	1369.16
comp	32	3	137	10%	43%	54%	57%	83%	46	89	125	29	35	170	209	2.10	0.04	0.04	0.04	0.05
term1	34	10	81	74%	87%	88%	88%	92%	139	144	184	33	43	241	291037	2.76	0.04	0.15	0.15	8.48
average				53%	72%	76%	80%	89%												

Table 3: 10% of the gates included in five Black Boxes

an and or or gate. Then we applied our check to detect errors in the partial implementation. Note that an error is reported only if there is no implementation for the Black Boxes such that the resulting circuit fulfills its specification. (The original benchmark circuit is used as the specification.) Each experiment was repeated for 100 error insertions.

In Table 1 we give the results for the first experiment, when 10% of the gates were included in one Black Box. In column 1 the name of the benchmark is given, in columns 2 and 3 the number of inputs and outputs of the benchmark are given. Column 4 shows the number of BDD nodes needed to represent the specifying benchmark circuit. In columns 5–9 the error detection ratio for 100 error insertions (per black box selection) using different equivalence checks is reported. For comparison Column 5 ("$r.p.$") shows the result of a $0, 1, X$-based non symbolic simulation with 5000 random patterns. Column 6 ("Z") shows the error detection ratio for symbolic Z–simulation, column 7 ("$loc.$") for symbolic Z_i–simulation with local equivalence check (see Section 3.2.1), column 8 ("oe") for symbolic Z_i–simulation with the "output exact" check of Section 3.2.2 and column 9 ("ie") for symbolic Z_i–simulation with the "input exact" check of Section 3.2.3. Note that in this experiment the check of Section 3.2.3 is exact, since there is only one Black Box; i.e., in all cases, when this check does not report any error, there really exists an implementation, which can compensate the error insertion. The following columns indicate the resources needed to achieve the results. Columns 10–12 give the numbers of BDD nodes which are needed to represent the implementation. Columns 13–16 show the maximum number of additional BDD nodes, which are needed the perform the four different checks which are based on symbolic simulation. And finally, columns 17–21 show the run times in CPU seconds for the random pattern simulation and the four symbolic checks, respectively.

Note that the error detection ratios for symbolic Z–simulation are equal to the error detection ratios of approach [13]. Although our implementation differs (using Z–simulation instead of signal duplication and conventional symbolic simulation), errors are reported in the same cases. Similarly, the error detection ratios for the output exact check (column "oe") are the same as in [12], although the implementation is different (our implementation does not need a representation of the overall circuit as a Boolean relation, e.g.).

As a first result we can notice that *the* $0, 1, X$*-based simulation with 5000 random patterns can not compete with the symbolic methods*. The detection ratios are considerably smaller than for symbolic Z–simulation (see columns 5 and 6) while the run times are larger (columns 17 and 18). For the other methods we can really observe an *improved error detection accuracy from method to method* (columns 6–9): With the exception of *term1*, which obviously is easy for Black Box Equivalence Checking, all other examples profit from a more sophisticated check in the sense that more, sometimes significantly more errors are detected. In particular, we observe, that *the application of the input exact check leads to a considerable improvement compared to the output exact check in many cases (see e.g. comp, C499).* The average numbers given in the last line of the tables underline our observations made before[2].

The experiments also show that the resources needed to perform a check increase with its accuracy. Especially for the output and input exact check the improved accuracy has to be paid by an increased memory consumption and by larger run times. However memory consumption and run times remain in a reasonable range. The equivalence check needs at most a few seconds in the worst case.

In a second experiment we varied our method to generate partial implementations: the number of gates to be included in a Black Box was changed from 10% to 40%. The results are given in Table 2. We can observe that the improvement in accuracy for the more exact checks compared to the less exact checks is slightly larger in this scenario. E.g. for *C499* the input exact check beats Z–simulation in 28% of the cases compared to 21% in the first experiment. Memory consumption and run times stay about in the same range.

In a third experiment we varied the generation of partial implementations of our first experiment to obtain 5 different Black Boxes instead of one. Results are given in Table 3. Memory consumption and run times are about in the same range compared to the first two experiments with exception of circuit *C880* where time and memory consumption for output and input exact checks increase (about 22 minutes for the input exact check)[3]. However the comparison of error detection ratios shows an interesting result: *Although the input exact check in this case is not exact, the advantage of the input exact check compared to the other checks in this case is even larger* (compare e.g. the line giving the average values in the tables). This obviously demonstrates the power of our heuristics.

Taken together, the high number of error detections for all symbolic checks (even for simple Z–simulation) demonstrates the validity of the concept of checking partial implementations already at a stage of the design process where a significant portion of the design has still to be performed.

5 Conclusions and future work

Experimental results showed that improving the accuracy of the algorithms for Black Box Equivalence Checking indeed leads to a significant improvement of the error detection capabilities. We have defined a series of different algorithms with increasing accuracy and increasing consumption of computational resources. This suggests to use these algorithms as a series of more and more exact methods to detect errors in partial implementations: first use $0, 1, X$–based simulation with only a few random patterns, then symbolic Z–simulation, Z_i–simulation with local check, with output exact check and finally with input exact check.

In the future we plan to compare our BDD based implementation of the different checks to a version using SAT–engines. Another interesting question is how the methods can be extended to verify also sequential circuits containing Black Boxes.

References

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

[2] S.B. Akers. Binary decision diagrams. *IEEE Trans. on Comp.*, 27:509–516, 1978.

[2]Since in this case of one Black Box the input exact check is exact, an average of 91% detected errors means, that for the remaining 9% of the cases our circuit modification described above did not really insert an error into the partial implementation, i.e. an implementation for the Black Box can be found, such that the overall implementation fulfills its specification.

[3]This is due to a peak memory consumption during quantification in 3 out of 5 different random selections of Black Boxes.

[3] K.S. Brace, R.L. Rudell, and R.E. Bryant. Efficient implementation of a BDD package. In *Design Automation Conf.*, pages 40–45, 1990.

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

[5] R.E. Bryant. Symbolic Boolean manipulation with ordered binary decision diagrams. *ACM, Comp. Surveys*, 24:293–318, 1992.

[6] R.E. Bryant and Y.-A. Chen. Verification of arithmetic functions with binary moment diagrams. In *Design Automation Conf.*, pages 535–541, 1995.

[7] 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.

[8] E.M. Clarke, M. Fujita, and X. Zhao. Hybrid decision diagrams - overcoming the limitations of MTBDDs and BMDs. In *Int'l Conf. on CAD*, pages 159–163, 1995.

[9] O. Coudert, C. Berthet, and J.C. Madre. Verification of sequential machines based on symbolic execution. In *Automatic Verification Methods for Finite State Systems, LNCS 407*, pages 365–373, 1989.

[10] R. Drechsler, B. Becker, and S. Ruppertz. K*BMDs: A new data structure for verification. In *European Design & Test Conf.*, pages 2–8, 1996.

[11] E. Goldberg, M. Prasad, and R. Brayton. Using SAT for combinational equivalence checking. In *Int'l Workshop on Logic Synth.*, pages 185–191, 2000.

[12] W. Günther, N. Drechsler, R. Drechsler, and B. Becker. Verification of designs containing black boxes. In *EUROMICRO*, pages 100–105, 2000.

[13] 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.

[14] A. Kuehlmann and F. Krohm. Equivalence checking using cuts and heaps. In *Design Automation Conf.*, pages 263–268, 1997.

[15] C.Y. Lee. Representation of switching circuits by binary decision diagrams. *Bell System Technical Jour.*, 38:985–999, 1959.

[16] J. Marquez-Silva and T. Glass. Combinational equivalence checking using satisfiability and recursive learning. In *Design, Automation and Test in Europe*, pages 145–149, 1999.

[17] J. Marquez-Silva and K. Sakallah. GRASP – a new search algorithm for satisfiability. In *Int'l Conf. on CAD*, pages 220–227, 1996.

[18] 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.

[19] B.M.E. Moret. Decision trees and diagrams. In *Computing Surveys*, volume 14, pages 593–623, 1982.

[20] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Int'l Conf. on CAD*, pages 42–47, 1993.

[21] C. Scholl and B. Becker. Checking equivalence for partial implementations. Technical report, Albert-Ludwigs-University, Freiburg, October 2000. URL:http://ira.informatik.uni-freiburg.de/papers/Year_2000/SB_2000b.ps.gz.

[22] F. Somenzi. *CUDD: CU Decision Diagram Package Release 2.3.0.* University of Colorado at Boulder, 1998.

[23] P. Tafertshofer, A. Ganz, and M. Henftling. A SAT-based implication engine for efficient ATPG, equivalence checking, and optimization of netlists. In *Int'l Conf. on CAD*, pages 648 – 655, 1997.