Circuit Partitioning for SAT-based Combinational Circuit Verification
A Case Study

Marc Herbstritt
Thomas Kmieciak
Bernd Becker

Institute of Computer Science
Albert-Ludwigs-University
Georges-Köhler-Allee 51
79110 Freiburg im Breisgau, Germany

Report 206, July 2004

Circuit Partitioning for SAT-based Combinational Circuit Verification — A Case Study

Marc Herbstritt Thomas Kmieciak Bernd Becker
Institute of Computer Science, Albert–Ludwigs–University,
D 79110 Freiburg im Breisgau, Germany
email: {herbstri,kmieciak,becker}@informatik.uni-freiburg.de

July 26 2004

Abstract

Hardware verification is nowadays one of the most time-consuming tasks during chip design. In the last few years SAT-based methods have become a core technology in hardware design, especially for the verification of combinational parts of the circuits. Verifying the equivalence of some specification and a corresponding implementation is typically done by building a so-called miter. In practice this miter is often build for each pair of primary outputs or for all primary outputs at once. In this work we have a closer look on partitioning the primary outpouts of the circuit and how structural partitionings can speed-up the verification process when SAT-based methods are used.

1 Introduction

When verifying a circuit, i.e. checking the equivalence between some specification and a corresponding implementation, it is a common task to prove that some primary output of the specification is functionally equivalent to the corresponding primary output of the implementation. BDDs and SAT have become core technologies to tackle this problem. In this work our focus is on SAT based methods. There, to prove equivalence, typically a miter is build. Often this is done using two approaches:

1. The Single-Output approach (SOG) [1, 20, 12, 22]: The circuit is verified by inspecting each pair of corresponding primary outputs separately.

2. The All-Outputs approach (AOG) [15, 16, 17, 13, 22]: The circuit is verified by building a miter for all primary output pairs altogether.

Clearly, not all verification methodologies fit directly into this scheme, e.g. [4, 2, 6], although they rather belong to the AOG approach than to the SOG approach. They aim to solve easy-to-verify primary outputs first, but it is also possible that they get caught in areas belonging to hard-to-verify primary outputs.

In this paper a new approach is presented that partitions the primary outputs of a circuit into several groups, and then checks equivalence by checking equivalence of each of the groups. We report on speedups of 276% on the average compared to traditional techniques.

The paper is structured as follows. Section 2 gives preliminaries. Then, in Section 3 we will motivate the ideas presented in this work. In 4 we present structural partitioning methods and an equivalence checking algorithm that is based on partitioning. Section 5 presents experimental results. Finally, in Section 6 the paper is concluded.

2 Preliminaries

This paper is concerned with equivalence checking of combinational circuits. A combinational circuit C represents a boolean function $f^C : \mathbf{B}^n \rightarrow \mathbf{B}^m$, i.e. C has n primary inputs and m primary outputs. Checking the equivalence of two boolean functions f^S and f^I that are given as combinational circuits C_S and C_I, respectively, is a common task during chip design [11]. A popular method for equivalence checking is to build a so-called miter [4] where the primary outputs of C_S and C_I are structurally combined using an `XOR`-gate (see Figure 1). Now, if any of these `XOR`-gates is satisfiable, i.e. there exists an assignment to the primary inputs such that a boolean '1' is generated at the `XOR`-gate output, then C_S and C_I are not equivalent. If no such satisfying assignment exists the circuits C_S and C_I are equivalent.
Using this miter construction, it is now possible to apply a satisfiability engine that has to prove satisfiability (resp. unsatisfiability). This can be e.g. an ATPG-tool that tests the miter output for a stuck-at-0 fault [3, 14], or a SAT-solver that works on a CNF derived from the miter circuit [18, 17].

3 Single-Output approach versus All-Outputs approach

The motivation for our work which is described in detail in Section 4 is based upon a discussion about the pros and cons of the SOG approach versus the AOG approach. Figure 1 depicts both approaches.

SOG: The most important advantage of the SOG approach is that it can result in partial verification results. Let's have a look at the verification methodology used in [17] where the AOG approach is used. In [17] a CNF formula is generated that is unsatisfiable iff the implementation meets its specification. But what if the underlying SAT solver is not able to solve the SAT instance because of resource constraints such as time or memory limit? Then we do not know anything about the equivalence status of the individual primary outputs. In contrast, the verification methods used in [1, 20, 12] are able to prove equivalence of primary output pairs separately although there may exist primary outputs where the applied method fails. Then at least the solved primary outputs can be discarded from consecutive verification steps.

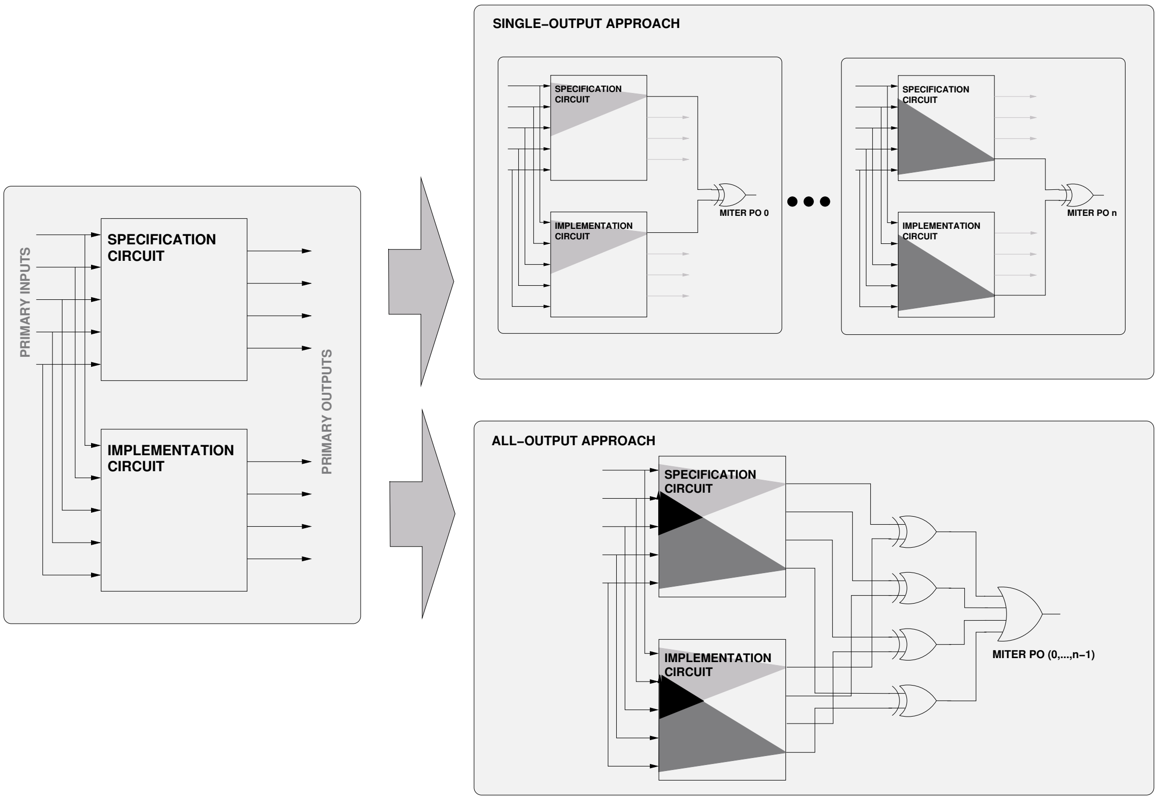


Figure 1: Two circuit verification scenarios: Single-Output grouping (SOG) versus All-Outputs grouping (AOG).

AOG: Since all primary outputs are handled at once this approach directly forces the applied verification method to reuse computations on components that are shared among the primary outputs. Therefore some computations on such a shared component must often be done only once compared to the application of SOG where the computation must be done for each primary output separately (e.g. the verification method used in [1]).

4 Structural Circuit Partionining

In [9, 7] two heuristics were presented for partitioning the set of primary outputs of a combinational circuit based on structural information. These partitioning heuristics were proved useful in the construction of Word-Level Decision Diagrams [8, 10] which are an extension of the popular Binary Decision Diagram [5]. In this work we will use these partitioning heuristics for generating multiple verification tasks when verifying combinational circuits. In the following we will review the heuristics from [9] and will provide a formal analysis of their behaviour.

4.1 Grouping heuristics WOG and BOG

The heuristics rely on interdependencies between primary outputs with respect to the number of primary inputs that they share. These interdependencies are captured in the *Output-Correspondence-Matrix (OCM)*.

Definition 1 (Output-Correspondence-Matrix (OCM)) *Let C be a combinational circuit implementing the function $f^C : \mathbf{B}^n \to \mathbf{B}^m$, i.e. C has n primary inputs and m primary outputs. Let f_k^C be the projection of f^C to its kth output.* $\texttt{OCM}(C)$ *is a $(m \times m)$-matrix. Let $supp(i)$ (resp. $supp(j)$) be the set of primary inputs on which the function f_i^C (resp. f_j^C) depends. Then,*

$$\texttt{OCM}[i][j] := |supp(i) \cap supp(j)|,$$

i.e. the number of primary inputs that the primary outputs i and j have in common.

Note that for a given combinational circuit C the set of primary inputs on which some primary output depends can be computed by a structural analysis of C.
Now that we have $\texttt{OCM}(C)$ we can define the heuristics from [9]. Both heuristics compute a disjoint partitioning of the set of primary outputs. For a given circuit C this partitioning is called a *permissible grouping* of C.

Definition 2 (Permissible Grouping) *Let C be a combinational circuit implementing the function $f^C : \mathbf{B}^n \to \mathbf{B}^m$, $PO(C)$ be the set of primary outputs of C, and $\#PO := |PO(C)|$. A* permissible grouping *$G(C)$ of C is defined as follows:*

1. *Let $PO^* = \bigcup_{i=1}^{\#PO}(PO(C))^i$. Then, $G(C) := (g_1, \ldots, g_t)$ where $\forall i, 1 \le i \le t, g_i \in PO^*$. I.e. $G(C)$ is a ordered set of ordered subsets of $PO(C)$.*
2. *$t \le m$, i.e. there are not more elements in the grouping than the number of primary outputs of C.*
3. *It holds: $\bigcup_{i=1}^{m} g_i = PO(C)$, i.e. all primary outputs are covered by the grouping.*
4. *$\forall i, j \in \{1, ..., t\}, i \neq j : g_i \cap g_j = \emptyset$, i.e. the group elements are pairwise disjoint.*[1]

Algorithm 1 gives pseudocode for computing a permissible grouping. The computation of the group elements is done along the diagonal of the OCM. $\texttt{OCM}[i][i]$ gives the number of primary inputs on which primary output i depends. The heuristics are greedy in the sense that a group element g_i is initialized by selecting the primary output k which has the largest support (see line 10 in Algorithm 1). This ouput is called the *leader* of g_i. Then, as long as primary outputs are not assigned to a group, these outputs are analyzed to check wether they share all primary inputs with the leader of g_i, or with all other elements already included in the group element g_i, respectively. If no outputs exist that satisfy the sharing constraints, the group element g_i is closed and a new group element $g_{(i+1)}$ is established.
Hence, we briefly characterize these two heuristics as follows.

[1]Please note that '∩' is the standard set operation and does not take the ordering into account. This is done throughout the paper with every standard set operator as long as the ordering is not of interest.

Definition 3 (Word-oriented output grouping heuristic (WOG)) *Include a primary output k into the group element g_i, if its support $supp(k)$ is a subset of the support of the leader of g_i.*

Definition 4 (Bit-oriented output grouping heuristic (BOG)) *Include a primary output k into the group element g_i, if its support $supp(k)$ is a subset of the support of each element $q \in g_i$.*

Pseudocode for WOG and BOG is given in Algorithms 2 and 3. Note that one of both algorithms is called in line 13 of Algorithm 1. The criteria of WOG and BOG (cf. line 6 in Algorithms 2 and 3) can be easily checked using the OCM.

Example 1 *Let's have a look at Figure 2 where a sample circuit is depicted. Assume an ordering of the primary outputs according to the size of their supports, e.g. $f_1 < f_2 < f_3 < f_4$.[2] The supports of the primary outputs are:*

$$\begin{aligned}
supp(f_1) &= \{x_1, x_2, x_3, x_4\} \\
supp(f_2) &= \{x_1, x_2\} \\
supp(f_3) &= \{x_3, x_4\} \\
supp(f_4) &= \{x_3, x_4, x_5\}
\end{aligned}$$

Consequently, the OCM looks like:

	f_1	f_2	f_3	f_4
f_1	4	2	2	2
f_2	2	2	0	0
f_3	2	0	2	2
f_4	2	0	2	3

WOG: *It holds $supp(f_2) \subseteq supp(f_1)$ and $supp(f_3) \subseteq supp(f_1)$. Therefore the first group element is (f_1, f_2, f_3). It only remains f_4, because $supp(f_4) \not\subseteq supp(f_1)$. Thus, WOG generates the grouping $((f_1, f_2, f_3), (f_4))$.*

BOG: *In contrast to WOG, since $supp(f_3) \not\subseteq supp(f_2)$ the first group element of BOG is (f_1, f_2). The next group element will be established using f_4 as the leader, because $|supp(f_4)| > |supp(f_3)|$. Since $supp(f_3) \subseteq supp(f_4)$ the second group element of BOG is (f_4, f_3). Finally, BOG generates the grouping $((f_1, f_2), (f4, f_3))$.*

In [9] it was found that WOG tends to generate smaller groupings than BOG, i.e. WOG puts more primary outputs into the group elements whereas BOG tends to produce larger groupings. That is the reason why WOG is called word-oriented, and BOG is called bit-oriented.
As can be seen from the example, from a structural point of view, WOG and BOG generate different groupings. At first glance, the grouping generated by BOG seems to be a refinement of the grouping generated by WOG. But as the example shows this is not the case, since for the second group of WOG, namely (f_4), it is not possible to extract a subset of the grouping of BOG such that the union of the elements of this subset is equal to (f_4).

[2]In the implementation whenever some primary outputs have the same size of their supports, the ordering is determined from the circuit specification.

Algorithm 1 Permissible Grouping

```
Input: C, a combinational circuit
Output: (g_1, ..., g_t), a permissible grouping of C
PO ←GET-PRIMARY-OUTPUTS(C)
OCM ←COMPUTE-OCM(C)
G(C) ←∅
g ←∅
while PO ≠ ∅ do
  if g = ∅ then
    g ←GET-LARGEST-OUTPUT(OCM,PO)
    PO ←PO \ g
  else
    while some primary output p ∈PO meets criterion according to WOG or BOG do
      g ←g ∪ p
      PO ←PO \ p
    end while
    G(C) ←G(C) ∪ g
    g ←∅
  end if
end while
return G(C)
```

Algorithm 2 Word-oriented Grouping (WOG)

```
Input: PO, a set of primary outputs not assigned to a group element
Input: (g, leader(g)), a group element and its leader
Output: p, a primary output
for all p∈ PO in decreasing order wrt. the size of SUPPORT(p) do
  if SUPPORT(p) ⊆ SUPPORT(leader(g)) then
    return p
  end if
end for
return no primary output satisfies WOG criterion
```

Algorithm 3 Bit-oriented Grouping (BOG)

```
Input: PO, a set of primary outputs not assigned to a group element
Input: g, a group element
Output: p, a primary output
for all p∈ PO in decreasing order wrt. the size of SUPPORT(p) do
  if for all primary outputs q∈ g it holds: SUPPORT(p) ⊆ SUPPORT(q) then
    return p
  end if
end for
return no primary output satisfies BOG criterion
```

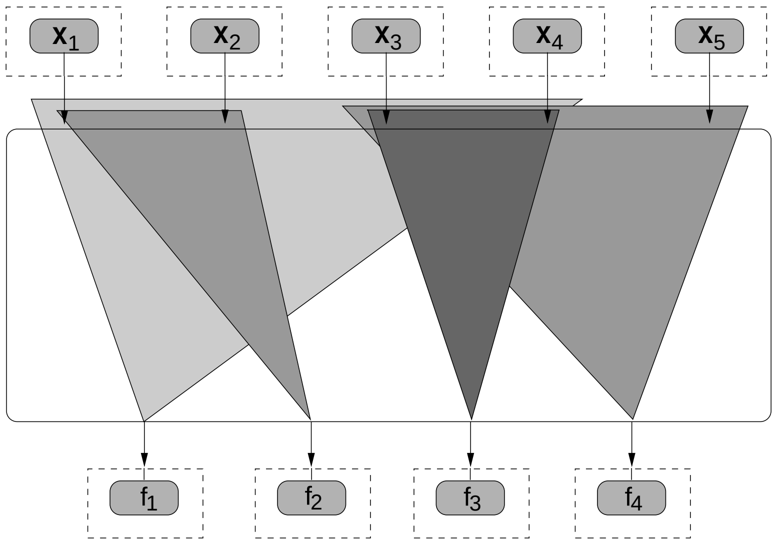


Figure 2: Sample circuit for demonstrating grouping heuristics WOG and BOG.

4.2 Formal analysis of WOG and BOG

We will now give a formal analysis of WOG and BOG which shows that BOG always does not generate groupings with less elements than the grouping generate by WOG.
For shorter writing, we use the following notations. g_i^{W} (g_i^{B}) denotes the ith group element from the grouping generated by WOG (BOG). Also $\#W$ ($\#B$) denotes the number of group elements generated by WOG (BOG). The leader of a group element g will be denoted $l(g)$.

Lemma 4.1 *Let C be a circuit and $PO(C)$ be the set of primary outputs of C. Let $WOG(C)$ and $BOG(C)$ be the groupings computed by WOG and BOG, respectively. Let $t = \min(\#B, \#W)$.*
It holds

$$\forall i, 1 \leq i \leq t : \bigcup_{j=1}^{i} g_j^{\mathrm{W}} \supseteq \bigcup_{j=1}^{i} g_j^{B},$$

i.e. the union of the group elements generated by BOG at stage i is covered by the union of the group elements generated by WOG at stage i.

Proof: We prove the lemma by induction on i.

[beginning step: $i = 1$ **]** Both WOG and BOG establish their first group element using the same primary output, i.e. it holds

$$l(g_1^{\mathrm{W}}) = l(g_1^{\mathrm{B}}). \tag{1}$$

Now assume there's a primary output p in g_1^{B} which is not contained in g_1^{W}, i.e. $(p \in g_1^{\mathrm{B}}) \wedge (p \notin g_1^{\mathrm{W}})$.

This primary output p was included in g_1^{B}, in particular because $supp(p) \subseteq supp(l(g_1^{\mathrm{B}}))$, due to Definition 4 of BOG.

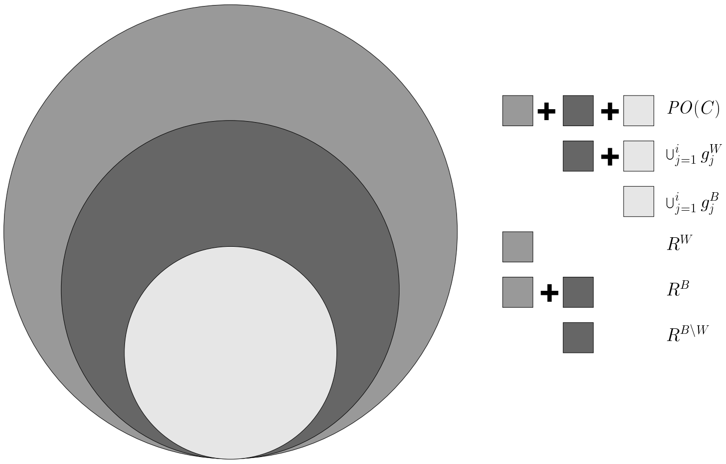


Figure 3: "Grouping" landscape used in the proof of Lemma 4.1.

But then p must also be included in g_1^{W} according the Definition 3 of WOG, because when using (1) it holds $supp(p) \subseteq supp(l(g_1^{\text{W}})))$ which is a contradiction to the assumption $p \notin g_1^{\text{W}}$.

Therefore it holds $g_1^{\text{W}} \supseteq g_1^{\text{B}}$ which concludes the beginning step.

[induction step: $i \to (i+1)$] Using the inductive assumption we have

$$\bigcup_{j=1}^{i} g_j^{\text{W}} \supseteq \bigcup_{j=1}^{i} g_j^{\text{B}}. \tag{2}$$

We now define the rest of the primary outputs R^{W} (R^{B}) which must be analyzed by WOG (BOG):

$$R^{\text{W}} := PO(C) \setminus \left(\bigcup_{j=1}^{i} g_j^{\text{W}} \right) \tag{3}$$

$$R^{\text{B}} := PO(C) \setminus \left(\bigcup_{j=1}^{i} g_j^{\text{B}} \right) \tag{4}$$

Using (2) it holds

$$R^{\text{W}} \subseteq R^{\text{B}}. \tag{5}$$

We differentiate between two cases: $R^{\text{W}} = R^{\text{B}}$ and $R^{\text{W}} \subset R^{\text{B}}$.

[case: $R^{\text{W}} = R^{\text{B}}$] Here, WOG as well as BOG will establish new groups g_{i+1}^{W} and g_{i+1}^{B} with the same leader, i.e. $l(g_{i+1}^{\text{W}}) = l(g_{i+1}^{\text{B}})$. Using the same argumentation as for the beginning step $i = 1$, it holds $g_{i+1}^{\text{W}} \supseteq g_{i+1}^{\text{B}}$, concluding the proof for this case.

[case: $R^{\mathrm{W}} \subset R^{\mathrm{B}}$] Let

$$R^{\mathrm{B}\setminus\mathrm{W}} := R^{\mathrm{B}} \setminus R^{\mathrm{W}}. \tag{6}$$

Note that

$$R^{\mathrm{B}\setminus\mathrm{W}} \subseteq \bigcup_{j=1}^{i} g_j^{\mathrm{W}}. \tag{7}$$

See Figure 3 for the visualization of the sets used in this proof. Let g_{i+1}^{B} be the group generated by BOG at stage $(i+1)$ with $g_{i+1}^{\mathrm{B}} = (r_1^{\mathrm{B}}, \ldots, r_s^{\mathrm{B}})$. Again, we have two cases: $l(g_{i+1}^{\mathrm{B}}) \in R^{\mathrm{B}\setminus\mathrm{W}}$ and $l(g_{i+1}^{\mathrm{B}}) \notin R^{\mathrm{B}\setminus\mathrm{W}}$.

[case: $l(g_{i+1}^{\mathrm{B}}) \in R^{\mathrm{B}\setminus\mathrm{W}}$] Using (7) we have that

$$l(g_{i+1}^{\mathrm{B}}) \in \bigcup_{j=1}^{i} g_j^{\mathrm{W}}. \tag{8}$$

Therefore there exists a group element g_k^{W},$1 \leq k \leq i$, s.t.

$$l(g_{i+1}^{\mathrm{B}}) \in g_k^{\mathrm{W}}. \tag{9}$$

Then, for each element r_l^{B},$1 \leq l \leq s$, of g_{i+1}^{B} it holds

$$supp(r_l^{\mathrm{B}}) \subseteq supp(l(g_k^{\mathrm{W}})), \tag{10}$$

due to Definition 3 of WOG.
Hence, it holds that either $r_l^{\mathrm{B}} \in g_k^{\mathrm{W}}$ or there exists an "earlier" group element $g_{k'}^{\mathrm{W}}$, $1 \leq k' < k$, s.t. $r_l^{\mathrm{B}} \in g_{k'}^{\mathrm{W}}$. Either way, it holds

$$\forall l, 1 \leq l \leq s : r_l^{\mathrm{B}} \in \bigcup_{j=1}^{i} g_j^{\mathrm{W}}. \tag{11}$$

This concludes the proof for this case.

[case: $l(g_{i+1}^{\mathrm{B}}) \notin R^{\mathrm{B}\setminus\mathrm{W}}$] Since $l(g_{i+1}^{\mathrm{B}}) \notin R^{\mathrm{B}\setminus\mathrm{W}}$, it must hold

$$l(g_{i+1}^{\mathrm{B}}) \in R^{W}. \tag{12}$$

Using the same argumentation as in the beginning step $i = 1$, it holds $g_{i+1}^{\mathrm{W}} \supseteq g_{i+1}^{\mathrm{B}}$, finishing the proof.

□

Using Lemma 4.1 it is easy to prove that BOG does not generate less group elements than WOG.

Lemma 4.2 *Let $BOG(C)$ and $WOG(C)$ be the groupings generated by BOG and WOG, respectively, for a circuit C. It holds:*

$$\#B \geq \#W.$$

Proof: Assume $\#B < \#W$. Then $t = min(\#B, \#W) = \#B$. It holds

$$\bigcup_{j=1}^{t} g_j^{\mathrm{B}} = PO(C). \tag{13}$$

Lemma 4.1 yields

$$\bigcup_{j=1}^{t} g_j^{\mathrm{B}} \subseteq \bigcup_{j=1}^{t} g_j^{\mathrm{W}} \tag{14}$$

Using (13) we have

$$PO(C) \subseteq \bigcup_{j=1}^{t} g_j^{\mathrm{W}}, \tag{15}$$

which means that

$$\bigcup_{j=1}^{t} g_j^{\mathrm{W}} = PO(C). \tag{16}$$

But then, $\#W = t = \#B$ which contradicts the assumption $\#B < \#W$. $\square$

Within the context of partitioning the set of primary outputs, the two approaches AOG and SOG (as described in Section 3) can be redefined as some special grouping heuristics. The SOG approach corresponds to a grouping heuristic where each primary output is put into a group element of its own. On the other hand, the AOG approach mimics a grouping heuristic where all primary outputs are put into one single group element.
Therefore we now have a spectrum of four different grouping heuristics with AOG and SOG as the "extreme" cases, and WOG and BOG are positioned inbetween.
To capture this taxonomy formally we introduce the granularity of a grouping heuristic.

Definition 5 (Granularity) *The* granularity *of a grouping heuristic X wrt. a (combinational) circuit C is the ratio between the number of group elements computed by X and the total number of primary outputs of C. Note that for sequential circuits flip-flop inputs will become primary outputs since we only consider the combinational part of the circuit. More precisely, the granularity $G(X, C)$ is defined as*

$$G(X, C) = \frac{\#X}{\#PO(C) + \#FF(C)},$$

where $\#X$ denotes the number of groups computed by a grouping heuristic X.

For SOG it holds $G(\text{SOG}, C) = 1$, and for AOG it holds $G(\text{AOG}, C) = 1/(\#PO(C) + \#FF(C))$. Based on this definition and using Lemma 4.2 we can now formally give a taxonomy wrt. the grouping heuristics analyzed in this paper.

Lemma 4.3 (Taxonomy of grouping heuristics) *For the groupings heuristics AOG, SOG, WOG and BOG and for some circuit C the following holds:*

$$1 \geq G(SOG, C) \geq G(BOG, C) \geq G(WOG, C) \geq G(AOG, C) = \frac{1}{\#PO(C) + \#FF(C)}.$$

Proof: The lemma directly follows by Definition 5 and Lemma 4.2. □

4.3 Partitioning-based equivalence checking

We now propose a partitioning-based equivalence checking procedure. In Algorithm 4 pseudocode is given.

Algorithm 4 Partitioning-based Equivalence Checking

1: **Input**: C_S, the specification circuit
2: **Input**: C_I, the implementation circuit
3: **Output**: Equivalence status
4:
5: $(g_1, \ldots, g_t) \leftarrow$PERMISSIBLE-GROUPING($C_S$)
6: *solved-outputs* $\leftarrow \emptyset$
7: *unresolved-outputs* $\leftarrow \emptyset$
8: **for all** m 1 to t **do**
9:     $C_M \leftarrow$CONSTRUCT-MITER(C_S,C_I,g_m)
10:     *res* $\leftarrow$CHECK-SATISFIABILITY(C_M)
11:     **if** *res* = `SATISFIABLE` **then**
12:         **return** C_S and C_I are not equivalent and differ on some primary output from g_m
13:     **end if**
14:     **if** *res* = `UNRESOLVED` **then**
15:         *unresolved-outputs* $\leftarrow$*unresolved* $\cup\ g_m$
16:     **end if**
17: **end for**
18: **if** *unresolved-outputs* = $\emptyset$ **then**
19:     **return** C_S and C_I are equivalent
20: **else**
21:     **return** primary outputs *solved-outputs* are proved to be equal, primary outputs *unresolved-outputs* could not be proved
22: **end if**

After a permissible grouping is computed (cf. line 5), for each element in the grouping a miter is constructed with respect to the primary outputs of this element. Then, the satisfiability of the miter is checked. At this stage of the algorithm we are free to select any admissible satisfiability engine whereby we have applied `zChaff` [19]. If the miter is satisfiable, i.e. there exists an assignment to the primary inputs such that a boolean '1' is computed at the miter output, we know that the circuits under verification are not equivalent. If for all group elements the

Benchmark	#PI	#PO	#FF	WOG	BOG	G(WOG,-)	G(BOG,-)
C1355	41	32	0	1	1	0.031	0.031
C1908	33	25	0	1	2	0.040	0.080
C2670	233	140	0	88	107	0.628	0.764
C3540	50	22	0	1	6	0.045	0.272
C499	41	32	0	1	1	0.031	0.031
C5315	178	123	0	60	67	0.478	0.544
C7552	207	108	0	9	50	0.083	0.462
mm30a	33	30	90	30	30	0.250	0.250
s13207.1	62	147	638	287	370	0.356	0.459
s15850.1	77	149	534	334	411	0.489	0.601
s38417	28	106	1636	603	906	0.346	0.520
s38584.1	38	304	1426	857	971	0.495	0.561
s9234.1	36	39	211	96	134	0.384	0.536

Figure 4: Results of grouping heuristics WOG and BOG applied to some instances of the ISCAS85 and LGSynth91 benchmark suites.

corresponding miter is unsatisfiable, then both circuits are equivalent. When for some group elements the satisfiability engine fails to prove satisfiability (resp. unsatisfiability) because of resource constraints, these primary outputs can be passed in a consecutive step to more sophisticated verification methods.
In the following section we will report on experimental results for the grouping heuristics WOG and BOG and their impact on SAT-based equivalence checking.

5 Experimental Results

At first we will have a look on partitioning the primary outputs according to WOG and BOG. Then we present experiments on verifying the ISCAS85 and LGSynth91 benchmarks against their synthesized versions [21] using Algorithm 4.

5.1 Partitioning using WOG or BOG

In Table 4 results are presented for partitioning the number of primary outputs according to WOG and BOG. Column 1 denotes the analyzed benchmark. In column 2, 3 and 4 the number of primary inputs, primary outputs and flip-flops are given, respectively. The number of groups computed by WOG and BOG are given in column 5 and 6. Finally, column 7 and 8 denote the granularity of WOG and BOG. The upper part relates to benchmarks from the ISCAS85 suite and the lower part to benchmarks from the LGSynth91 suite.
Especially for the sequential benchmarks from LGSynth91 it can be seen that WOG and BOG really behave different compared to AOG and SOG.

To get an impression of the sizes of the group elements produced by WOG and BOG in Figure 5 the distributions are depicted for the benchmark s38417. In general, about 60% - 90% percent of the group elements consist only of one element (see Table 6 for details). These are the parts which are in common with SOG. But it is interesting to see that the granularity relation between WOG and BOG does not transfer to the number of group elements of size 1, e.g. for C5315 WOG results in 71.1% but BOG results only in 49.3%. Despite the taxonomy given in Lemma 4.3 often WOG is closer to SOG than BOG with respect to the group elements of size 1. A possible explanation for this behaviour is the fact that the WOG criterion is more relaxed and thus WOG is more greedy wrt. incorporating primary outputs into a group element. Particularly in the beginning the group elements are larger compared to BOG, incorporating primary outputs that will — contrary to expectations — appear in small group elements in the grouping generated by BOG. E.g. see again Example 1 where WOG generates one group element of size 1, namely (f_4), but BOG does not generate any.

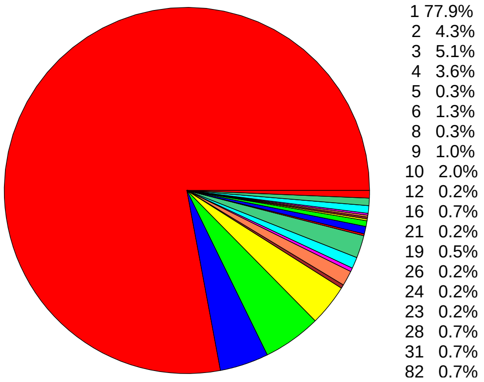
5.2 Equivalence checking of ISCAS85 and LGSynth91 benchmarks

We have conducted experiments for checking the efficiency of Algorithm 4. Table 7 gives the CPU runtimes.[3] To get a meaningful set of benchmarks we synthesized the original benchmarks from ISCAS85 and the combinational parts of the LGSynth91 benchmarks using `SIS` [21] with different scripts: `red-removal`, `nand-nor-genlib`, `script-rugged`, and `msu-genlib`. Afterwards, we verified these synthesized versions against the original circuit.
The CPU runtimes for AOG (SOG,WOG,BOG) can be retained from column 3 (4,5,6). The last row gives the total time for each of the analyzed grouping heuristics. Clearly, our approach, using WOG and BOG, outperforms AOG and SOG significantly. The total runtimes of WOG compared to AOG result in a speedup of 276% on the average. On individual instances a speedup by a factor of about 21 is achieved (see s38417 when BOG is compared to AOG). Indeed, there are benchmarks were our approach does not that well (e.g. C5315), but in general WOG and BOG are near the minimum of AOG and SOG, and sometimes even better (see s38417.1 and s38584).

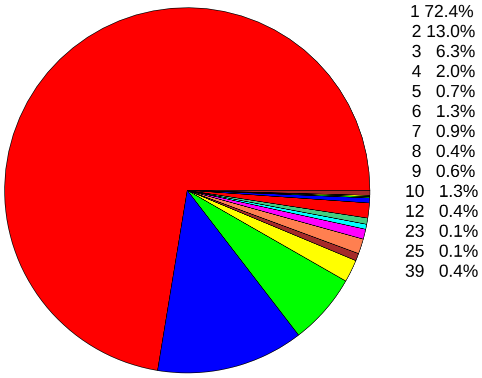
6 Conclusions

In this paper we evaluated a new approach for SAT-based combinational circuit verification. Our approach is based on a structural partitioning of the primary outputs of the circuit. Using this method we achieved a speedup of 276% on the average compared to traditional approaches. Future work is focused on extending the exploitation of structural information, e.g. by using semi-canonical data structures for boolean function representation.

[3]The experiments for the ISCAS85 C-benchmarks were performed on an AMD Athlon XP1700+. The remaining benchmarks were analyzed using an Intel Xeon 2Mhz machine. Both machines were restricted to 1000sec time limit and 512MB memory limit for each instance.



(a) WOG



(b) BOG

Figure 5: Distribution of group element sizes of WOG and BOG applied to s38417 from LGSynth91.

Benchmark	WOG	BOG
C1355	0.0	0.0
C1908	0.0	0.0
C2670	90.9	86.9
C3540	0.0	50.0
C499	0.0	0.0
C5315	71.7	49.3
C7552	22.2	86.0
mm30a	96.7	0.0
s13207.1	76.0	66.8
s15850.1	76.0	62.5
s38417	77.9	72.4
s38584.1	73.3	62.7
s9234.1	75.0	59.0

Figure 6: Percentage of group elements of size 1 for the benchmarks from Table 4.

Benchmark	SIS-script	AOG	SOG	WOG	BOG
C1335	msu-genlib	1.89	54.51	1.85	1.85
	nand-nor-genlib	1.91	60.74	1.91	1.91
	red-removal	5.46	47.27	5.45	5.45
	script-rugged	1.88	33.93	1.88	1.88
C1908	msu-genlib	2.71	75.78	2.70	13.47
	nand-nor-genlib	1.80	11.89	1.80	2.78
	red-removal	1.52	13.52	1.53	3.73
	script-rugged	1.05	8.50	1.07	2.01
C2670	msu-genlib	4.43	4.73	1.33	3.70
	nand-nor-genlib	4.43	2.31	1.03	2.10
	red-removal	0.99	1.88	1.07	4.48
	script-rugged	3.12	1.44	1.83	3.65
C3540	msu-genlib	67.02	697.11	67.80	202.43
	nand-nor-genlib	95.47	497.61	94.28	197.71
	red-removal	57.94	393.31	57.75	265.64
	script-rugged	86.01	555.55	86.27	429.22
C499	msu-genlib	1.97	5.50	2.01	2.01
	nand-nor-genlib	0.75	31.33	0.76	0.76
	red-removal	0.26	23.07	0.27	0.27
	script-rugged	1.58	30.93	1.59	1.59
C5315	msu-genlib	295.60	243.62	275.89	185.62
	nand-nor-genlib	35.26	485.08	203.52	314.36
	red-removal	56.65	282.84	233.31	227.96
	script-rugged	23.90	145.29	87.71	95.35
C7552	msu-genlib	250.61	9.50	41.91	47.50
	nand-nor-genlib	143.29	11.36	36.25	47.00
	red-removal	123.67	8.43	33.09	39.04
	script-rugged	526.21	10.48	39.74	25.73
mm30a	msu-genlib	2.45	18.51	17.48	14.51
	nand-nor-genlib	4.56	19.04	15.16	15.99
	red-removal	4.02	16.94	15.27	16.23
	script-rugged	1.97	8.36	8.07	8.22
s13207.1	msu-genlib	31.09	1.93	2.72	1.26
	nand-nor-genlib	31.20	1.63	3.90	1.39
	red-removal	30.81	1.14	2.09	0.93
	script-rugged	24.49	3.11	3.61	2.00
s15850.1	msu-genlib	40.29	8.04	5.31	5.04
	nand-nor-genlib	45.61	10.43	5.61	6.76
	red-removal	46.47	7.52	4.78	5.43
	script-rugged	33.92	9.23	4.37	5.51
s38417	msu-genlib	540.01	83.13	36.42	46.91
	nand-nor-genlib	489.38	81.35	35.79	43.93
	red-removal	515.33	76.24	34.54	33.32
	script-rugged	660.67	55.85	36.32	31.39
s38584	msu-genlib	338.33	1.51	1.38	0.86
	nand-nor-genlib	381.74	1.43	1.97	1.09
	red-removal	379.03	1.40	1.56	0.80
	script-rugged	314.48	2.53	1.68	0.76
s5378	msu-genlib	2.99	0.10	0.16	0.16
	nand-nor-genlib	3.19	0.08	0.15	0.16
	red-removal	2.90	0.09	0.16	0.15
	script-rugged	2.90	0.06	0.16	0.13
s9234	msu-genlib	7.70	0.68	1.03	0.80
	nand-nor-genlib	8.61	0.66	1.01	0.82
	red-removal	6.82	0.47	0.74	0.68
	script-rugged	5.77	0.75	0.73	0.62
	Total time	5754.11	4159.72	1527.77	2375.05

Figure 7: CPU runtimes for the verification of the benchmarks from Table 4 against synthesized versions.

References

[1] Gunar Andersson, Per Bjesse, Byron Cook, and Ziyad Hanna. A Proof Engine Approach to Solving Combinational Design Automation Problems. In *Design Automation Conf.*, pages 725–730, June 2002.

[2] C.L. Berman and L.H. Trevillyan. Functional Comparison of Logic Designs for VLSI Circuits. In *Int'l Conf. on CAD*, pages 456–459, 1989.

[3] A. Biere and W. Kunz. SAT and ATPG: Boolean Engines for Formal Hardware Verification. In *Int'l Conf. on CAD*, 2002.

[4] D. Brand. Verification of large synthesized designs. In *Int'l Conf. on CAD*, pages 534–537, 1993.

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

[6] J.R. Burch and V. Singhal. Tight Integration of Combinational Verification Methods. In *Int'l Conf. on CAD*, pages 570–576, 1998.

[7] R. Drechsler. *Formal Verification of Circuits*. Kluwer Academic Publishers, 2000.

[8] R. Drechsler and B. Becker. An Overview on Decision Diagrams. *IEE Proc. Computers and Digital Techniques*, 144(3):187–193, 1997.

[9] Rolf Drechsler, Marc Herbstritt, and Bernd Becker. Grouping heuristics for word-level decision diagrams. In *Int'l Symp. Circ. and Systems (ISCAS)*, pages 411–415, Orlando, Florida, June 1999.

[10] S. Höreth and R. Drechsler. Formal verification of word-level specifications. In *Design, Automation and Test in Europe*, pages 52–58, 1999.

[11] J.Jain, A. Narayan, M. Fujita, and A. Sangiovanni-Vincentelli. Formal verification of combinational circuits. In *VLSI Design Conf.*, pages 218–225, 1997.

[12] A. Kuehlmann, V. Paruthi, F. Krohm, and M.K. M.K. Ganai. Robust Boolean Reasoning for Equivalence Checking and Functional Property Verification. *IEEE Trans. on CAD*, 2002.

[13] W. Kunz, D.K. Pradhan, and S.M. Reddy. A Novel Framework for Logic Verification in a Synthesis Environment. *IEEE Transactions on Computer Aided Design*, 15(1):20–32, Jan. 1996.

[14] T. Larrabee. Test pattern generation using boolean satisfiability. *IEEE Trans. on CAD*, 11:4–15, 1992.

[15] Feng Lu, Li-C. Wang, K.-T. (Tim) Cheng, John Moondanos, and Ziyad Hanna. A Signal Correlation Guided ATPG Solver And Its Application For Solving Difficult Industrial Cases. In *Design Automation Conf.*, pages 436–441, 2003.

[16] Feng Lu, Li-C. Wang, Kwang-Ting Cheng, and Ric C-Y Huang. A Circuit SAT Solver With Signal Correlation Guided Learning. In *Design, Automation, and Test in Europe (DATE '03)*, pages 892–897, Munich, Germany, March 2003.

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

[18] J.P. Marques-Silva and K.A. Sakallah. Boolean satisfiability in electronic design automation. In *Design Automation Conf.*, pages 675–680, 2000.

[19] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engeneering an efficient SAT solver. In *Design Automation Conf.*, 2001.

[20] Sherief Reda and Ashraf Salem. Combinational Equivalence Checking using Boolean Satisfiability and Binary Decision Diagrams. In *Proceedings of Design, Automation and Test in Europe Conference (DATE)*, 2001.

[21] E. Sentovich, K. Singh, L. Lavagno, Ch. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical report, University of Berkeley, 1992.

[22] Paul Stephan, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. Combinational Test Generation Using Satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(9):1167–1176, September 1996.