On the Impact of Structural Circuit Partitioning on SAT-based Combinational Circuit Verification

(Presentation at IEEE MTV 2004)

Marc Herbstritt
(joint work with Thomas Kmieciak and Bernd Becker)
http://ira.informatik.uni-freiburg.de
herbstri@informatik.uni-freiburg.de

Institute of Computer Science, Albert-Ludwigs-University
Freiburg im Breisgau, Germany
Overview

- Introduction
- Traditional techniques: SOG and AOG
- Discussion: SOG vs. AOG
- Idea for our approach based on output partitioning
- Two partitioning heuristics
- Partitioning-based verification using SAT
- Experimental results
- Conclusions
Traditional Circuit Verification

verification of "golden" specification
Traditional Circuit Verification

verification of "golden" specification against an implementation
Traditional Circuit Verification

2 verification techniques: Single-Outputs and All-Outputs
Traditional Circuit Verification

Single-Output Grouping (SOG):
Traditional Circuit Verification

Single-Output Grouping (SOG): Verify each output separately

Traditional Circuit Verification

Single-Output Grouping (SOG): Verify each output separately
All-Outputs Grouping (AOG):
Traditional Circuit Verification

Single-Output Grouping (SOG): Verify *each* output *separately*

All-Outputs Grouping (AOG): Verify *all* outputs *at once*
Discussion: SOG vs. AOG

Partial Verification:

*Report the equivalence status of those outputs for which the underlying verification method is able to check equivalence.*

![Diagram](image)
Discussion: SOG vs. AOG

Shared Components:

*Reuse computations made on components that are used by several outputs.*

Learn: \((f=0) \Rightarrow (b=0)\)
Discussion: SOG vs. AOG

SOG
- Pros: partial verification
- Cons: no use of shared components

AOG
- Pros: use of shared components
- Cons: no partial verification
Discussion: SOG vs. AOG

SOG

- **pros**: partial verification

AOG
Discussion: SOG vs. AOG

SOG

*pros*: partial verification

*cons*: no use of shared components

AOG
Discussion: SOG vs. AOG

SOG

- *pros*: partial verification
- *cons*: no use of shared components

AOG

- *pros*: use of shared components
Discussion: SOG vs. AOG

SOG

- **pros:** partial verification
- **cons:** no use of shared components

AOG

- **pros:** use of shared components
- **cons:** no partial verification
Our approach

We need partitionings of the primary outputs s.t.
- partial verification is possible
- computations on shared components are exploited
Our approach

We need partitionings of the primary outputs s.t.
- partial verification is possible
- computations on shared components are exploited

In this work, we ...

- ... use two already known output partitioning heuristics: WOG and BOG
- ... formally analyze WOG and BOG
- ... propose a SAT-based verification algorithm based on partitionings
WOG and BOG

WOG and BOG both compute ordered groupings

\[ ((p_{1,1}, p_{1,2}, \ldots, p_{1,k_1}), \ldots, (p_{n,1}, p_{n,2}, \ldots, p_{n,k_n})) \]

where

- \( p_{i,j} \) is primary output
- all \( p_{i,j} \) are pairwise disjoint
- \( n \) is the size of the grouping
- \( p_{i,1} \) is called the leader of group element \( i \)
WOG and BOG both compute ordered groupings

\[
\left( (p_{1,1}, p_{1,2}, \ldots, p_{1,k_1}), \ldots, (p_{n,1}, p_{n,2}, \ldots, p_{n,k_n}) \right)
\]

WOG and BOG differ in the support relation of the outputs in a group element \((p_{i,1}, p_{i,2}, \ldots, p_{i,k_i})\)
WOG and BOG

- WOG and BOG both compute ordered groupings

\[ ((p_{1,1}, p_{1,2}, \ldots, p_{1,k_1}), \ldots, (p_{n,1}, p_{n,2}, \ldots, p_{n,k_n})) \]

- WOG and BOG differ in the support relation of the outputs in a group element \( (p_{i,1}, p_{i,2}, \ldots, p_{i,k_i}) \)

- WOG: \( \forall j, 1 \leq j \leq k_i : \text{supp}(p_{i,j}) \subseteq \text{supp}(p_{i,1}) \)
  (word-oriented, support relation is in terms of the leader)
WOG and BOG both compute ordered groupings

\[((p_1,1, p_1,2, \ldots, p_1,k_1), \ldots, (p_n,1,p_n,2,\ldots,p_n,k_n))\]

WOG and BOG differ in the support relation of the outputs in a group element \((p_i,1, p_i,2, \ldots, p_i,k_i)\)

**WOG:** \(\forall j, \ 1 \leq j \leq k_i : \text{supp}(p_{i,j}) \subseteq \text{supp}(p_{i,1})\)

(word-oriented, support relation is in terms of the leader)

**BOG:** \(\forall j, \ 1 \leq j \leq k_i : \text{supp}(p_{i,j}) \subseteq \text{supp}(p_{i,j-1})\)

(bit-oriented, support relation is in terms of all previously included outputs)
How WOG works

ordering of outputs: $f_1 < f_4 < f_2 < f_3$
initialise first group element: \( f_1 \)
How WOG works

do not add $f_4$ to $(f_1)$ because $\text{supp}(f_4) \not\subseteq \text{supp}(f_1)$
How WOG works

add $f_2$ to $(f_1)$ because $\text{supp}(f_2) \subseteq \text{supp}(f_1)$: $(f_1, f_2)$
How WOG works

add \( f_3 \) to \((f_1, f_2)\) because \( supp(f_3) \subseteq supp(f_1) \): \((f_1, f_2, f_3)\)
How WOG works

it remains $f_4$ and WOG computes: $(f_1, f_2, f_3), (f_4)$
How BOG works

How BOG works

initialise first group element: \((f_1)\)
How BOG works

do not add $f_4$ to $(f_1)$ because $\text{supp}(f_4) \not\subseteq \text{supp}(f_1)$
How BOG works

add $f_2$ to $(f_1)$ because $\text{supp}(f_2) \subseteq \text{supp}(f_1)$: $(f_1, f_2)$
How BOG works

do not add \( f_3 \) to \((f_1, f_2)\) because \( \text{supp}(f_3) \not\subseteq \text{supp}(f_2) \)
How BOG works

initialise next group element with $f_4: ((f_1, f_2), (f_4))$
How BOG works

it remains \( f_3 \) and BOG computes: \( ((f_1, f_2), (f_4, f_3)) \)
Notes on WOG and BOG

- WOG: \(((f_1, f_2, f_3), (f_4))\)
- BOG: \(((f_1, f_2), (f_4, f_3))\)

Computation of output relations by structural circuit analysis

\(\rightarrow\) Output-Correspondence Matrix

BOG is not a refinement of WOG

E.g. a subset of BOG for which the union results in \((f_4)\)
**Lemma**: Let $\#W$ ($\#B$) be the number of group elements of the grouping generated by WOG (BOG). Then it holds

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

**Proof**: Since BOG is not a refinement of WOG, the proof is more technical than expected.

*Hint*: Check which outputs are already handled at stage $i$ of the algorithms.
Verification using output partitioning

1. Input: $C_S$: specification circuit, $C_I$: implementation circuit
2. Output: Equivalence status
Verification using output partitioning

1. Input: \( C_S \): specification circuit, \( C_I \): implementation circuit
2. Output: Equivalence status
3. \( (g_1, \ldots, g_t) \leftarrow \text{WOG-OR-BOG-GROUPING}(C_S) \)
Verification using output partitioning

1. Input: $C_S$: specification circuit, $C_I$: implementation circuit
2. Output: Equivalence status
3. $(g_1, \ldots, g_t) \leftarrow \text{WOG-OR-BOG-GROUPING}(C_S)$
4. solved-outputs $\leftarrow \emptyset$; unresolved-outputs $\leftarrow \emptyset$
5. forall $m = 1$ to $t$ do
6. $C_M \leftarrow \text{CONSTRUCT-GROUP-ELEMENT-MITER}(C_S, C_I, g_m)$
Verification using output partitioning

1. Input: $C_S$: specification circuit, $C_I$: implementation circuit
2. Output: Equivalence status
3. $(g_1, \ldots, g_t) \leftarrow \text{WOG-OR-BOG-GROUPING}(C_S)$
4. solved-outputs $\leftarrow \emptyset$; unresolved-outputs $\leftarrow \emptyset$
5. forall $m = 1$ to $t$ do
6. $C_M \leftarrow \text{CONSTRUCT-GROUP-ELEMENT-MITER}(C_S, C_I, g_m)$
7. $res \leftarrow \text{CHECK-SATISFIABILITY-OF-MITER}(C_M)$
Verification using output partitioning

1. Input: $C_S$: specification circuit, $C_I$: implementation circuit
2. Output: Equivalence status
3. $(g_1, \ldots, g_t) \leftarrow \text{WOG-OR-BOG-GROUPING}(C_S)$
4. $\text{solved-outputs} \leftarrow \emptyset$; $\text{unresolved-outputs} \leftarrow \emptyset$
5. $\text{forall } m = 1 \text{ to } t \text{ do}$
   6. $C_M \leftarrow \text{CONSTRUCT-GROUP-ELEMENT-MITER}(C_S, C_I, g_m)$
   7. $\text{res} \leftarrow \text{CHECK-SATISFIABILITY-OF-MITER}(C_M)$
   8. $\text{if } \text{res} = \text{SATISFIABLE}$
      9. $\text{return } C_S \text{ and } C_I \text{ differ on some primary output from } g_m$
   10. $\text{endif}$
Verification using output partitioning

1. **Input:** \( C_S \): specification circuit, \( C_I \): implementation circuit
2. **Output:** Equivalence status
3. \((g_1, \ldots, g_t) \leftarrow \text{WOG-OR-BOG-GROUPING}(C_S)\)
4. solved-outputs \( \leftarrow \emptyset \); unresolved-outputs \( \leftarrow \emptyset \)
5. **forall** \( m = 1 \) to \( t \) **do**
6. \( C_M \leftarrow \text{CONSTRUCT-GROUP-ELEMENT-MITER}(C_S, C_I, g_m)\)
7. \( \text{res} \leftarrow \text{CHECK-SATISFIABILITY-OF-MITER}(C_M)\)
8. **if** \( \text{res} = \text{SATISFIABLE} \)
9. **return** \( C_S \) and \( C_I \) differ on some primary output from \( g_m \)
10. **endif**
11. **if** \( \text{res} = \text{UNRESOLVED} \)
12. unresolved-outputs \( \leftarrow \text{unresolved-outputs} \cup g_m \)
13. **else**
14. solved-outputs \( \leftarrow \text{solved-outputs} \cup g_m \)
15. **endif**
16. **endfor**
Verifying using output partitioning

1. Input: $C_S$: specification circuit, $C_I$: implementation circuit
2. Output: Equivalence status
3. $(g_1, \ldots, g_t) \leftarrow \text{WOG-OR-BOG-GROUPING}(C_S)$
4. solved-outputs $\leftarrow \emptyset$; unresolved-outputs $\leftarrow \emptyset$
5. forall $m = 1$ to $t$ do
6. $C_M \leftarrow \text{CONSTRUCT-GROUP-ELEMENT-MITER}(C_S, C_I, g_m)$
7. res $\leftarrow \text{CHECK-SATISFIABILITY-OF-MITER}(C_M)$
8. if res = SATISFIABLE
9. return $C_S$ and $C_I$ differ on some primary output from $g_m$
10. endif
11. if res = UNRESOLVED
12. unresolved-outputs $\leftarrow$ unresolved-outputs $\cup$ $g_m$
13. else
14. solved-outputs $\leftarrow$ solved-outputs $\cup$ $g_m$
15. endif
16. endfor
17. return solved-outputs are equivalent, unresolved-outputs could not be proved
Comparison

**Comparison**

- **Single-Output Grouping (SOG)**
- **Bit-Oriented Grouping (BOG)**
- **Word-Oriented Grouping (WOG)**
- **All-Outputs Grouping (AOG)**
Comparison

WORD-ORIENTED GROUPING (WOG)

BIT-ORIENTED GROUPING (BOG)

SINGLE-OUTPUT GROUPING (SOG)

Comparison

### Comparison of Grouping Techniques

- **SINGLE-OUTPUT GROUPING (SOG)**
- **BIT-ORIENTED GROUPING (BOG)**
- **WORD-ORIENTED GROUPING (WOG)**
- **ALL-OUTPUTS GROUPING (AOG)**

PO 0, PO 1, PO 2, PO 3

PO 0, 1, 2, 3

MITER PO (0,1,2,3)

Comparison

SINGLE-OUTPUT GROUPING (SOG)

BIT-ORIENTED GROUPING (BOG)

WORD-ORIENTED GROUPING (WOG)

ALL-OUTPUTS GROUPING (AOG)
Comparison

SINGLE-OUTPUT GROUPING (SOG)

BIT-ORIENTED GROUPING (BOG)

WORD-ORIENTED GROUPING (WOG)

ALL-OUTPUTS GROUPING (AOG)
Partitioning results

Granularity := \frac{\text{Number of group elements}}{\text{Number of primary outputs}}

![Graph showing Granularity vs. Circuit IDs](image_url)
Partitioning results

Granularity := \frac{\text{Number of group elements}}{\text{Number of primary outputs}}
Partitioning results

Granularity := \frac{\text{Number of group elements}}{\text{Number of primary outputs}}
Verification results (script-rugged)

![Graph showing CPU runtime for various benchmarks (AOG, SOG, BOG, WOG)].

Verification results (script-rugged)
Verification results (script-rugged)
Verification results (script-rugged)
Verification results (script-rugged)
Verification results (script-rugged)
Conclusions

- Partitioning the outputs can have large impact
- Heuristics BOG and WOG are building a robust compromise between traditional techniques like AOG or SOG
- WOG compared to AOG: speedup of 276% on the average
- BOG compared to SOG: speedup of 75% on the average
Conclusions

Partitioning the outputs can have large impact

Heuristics BOG and WOG are building a robust compromise between traditional techniques like AOG or SOG

WOG compared to AOG: speedup of 276% on the average

BOG compared to SOG: speedup of 75% on the average

Future work:

- Analyze in detail why SAT benefits from WOG/BOG
- Use of semi-canonical data-structures
- Partitioning heuristics incorporating more structure
- Development of SAT-oriented heuristics