# AUTOMATIC PARTITIONING FOR EFFICIENT COMBINATIONAL VERIFICATION

Rajarshi Mukherjee Jawahar Jain Koichiro Takayama
Masahiro Fujita

{rmukherj,jawahar,ktakayam,fujita}@fla.fujitsu.com
Fujitsu Laboratories of America
595 Lawrence Expressway
Sunnyvale, CA 94086

## Abstract

A majority of the state-of-the-art combinational verification techniques are based on the extraction and use of internal equivalences between two circuits. Verification can become difficult if the two circuits have none or very few internal correspondences. In this paper we investigate automatic circuit partitioning as a methodology to make otherwise intractable circuits relatively tractable to the verifier. We show that given any two circuits to be verified, finding the best partitions that minimize the verification runtime is NP-hard. Therefore, we propose efficient heuristics to utilize certain characteristics of typical circuit design styles to find good partitions for the circuits. A key difference between our approach and earlier approaches to circuit partitioning is that ours is fully automated and does not require any prior knowledge of the type of function being implemented by the circuit. Using circuit partitioning we are able to verify several hard industrial circuits that could not be verified otherwise.

## 1 Introduction

Automatic combinational verification or Boolean comparison (BC) is central to the successful design of any complex digital system. The BC problem can be stated as follows: *Given two Boolean networks, check whether the corresponding outputs of the two networks are equivalent for all possible input combinations.*

Ordered Binary Decision Diagrams (OBDDs) [4] are frequently employed as one of the primary tools for automating combinational verification [6, 9]. However, due to the OBDD blowup problem, an OBDD-based verification tool is not very robust. The efficiency of a combinational verifier can be enhanced by exploiting many relationships that exist among internal nodes of the two networks being verified [2, 3, 5, 7, 8, 12, 18, 19]. However, these tools often fail to verify circuits if the number of correspondences found is inadequate. In this paper we study circuit partitioning as a technique to simplify a given verification problem. Given two circuits to be verified, we show that the problem of finding partitions that minimize the verification time is NP-hard. Therefore, we study typical circuit design styles and propose efficient heuristics to automatically create "good" partitions that simplify the verification problem. The efficacy of our proposed methodology is demonstrated by the verification of several hard industrial circuits that could not be verified by the verifier alone.

## 2 Use of Partitioning in Combinational Verification

An efficient, robust and extensible framework for combinational verification based on *filter-theory* has been proposed in [19]. This methodology depends on the extraction and use of correspondences among internal nodes in the two circuits being verified. Therefore, its performance can deteriorate in cases where there are very few correspondences. Circuits with very few internal correspondences can be created in many circumstances. For instance, if a circuit undergoes rigorous optimization, many of the internal correspondences may be lost. Also, if two circuits are designed in different ways, (for example, an array multipler and a Booth multipler) internal correspondences can be very few in number. Since

the type of circuits encountered in industry is not known a-priori, it is very important that the BC tool is capable of handling circuits with few internal correspondences. In order to achieve this goal, a verification tool should have the following capabilities: **(1)** The tool should be able to analyze the two circuits being verified, estimate the number of internal correspondences among the nodes of the two circuits, and estimate the ease of verification using these correspondences, **(2)** If the number of internal correspondences is found to be inadequate, the tool must be able to transform the given circuits to create additional internal correspondences in order to simplify verification. In this work we propose several automatic heuristics to partition the given circuits in order to simplify verification.

Next, we define the notion of *circuit partitioning* used in this work.

**Definition 2.1 Circuit partitioning** *of a single-output circuit involves Shannon's expansion of the function realized by the primary output based on a set of k primary input variables. As a result,* $2^k$ *functions (circuits) are created. Each of these new circuits is called a partition of the original circuit. Verification of the original circuit can now be done by verifying each partition separately.*

Several forms of functional partitioning techniques such as those based on partitioning a circuit structure (circuit partitioning) [13, 15], and BDD partitioning [13, 14] have been developed. Circuit partitioning can help verification by : **(1)** Reducing the size of the circuits being verified. **(2)** Creating new conditional equivalences - For example, some gates $g_a, g_b$ may be only equivalent when variable $x_i = 0$. On the other hand, some gates $g_c, g_d$ may be equivalent only when $x_i = 1$. **(3)** Creating new conditional indirect implications (learnings) - For example, a learning $g_a \longrightarrow g_b$ may exist only when $x_i = 1$. **(4)** Allowing verification to be parallelized - Each partition can be verified on a separate machine.

**Definition 2.2** *Given two circuits* $C_1$ *and* $C_2$ *to be verified, a* **composite network** $C$ *is created by connecting their corresponding primary inputs and by feeding each pair of corresponding primary outputs to two-input XOR gates. Verification of* $C$ *consists of checking the satisfiability of the primary outputs of* $C$*. If all the primary outputs are unsatisfiable,* $C_1 \equiv C_2$*.*

Fig. 1 shows how a composite network is created from two given networks. Let $C$ be a given composite network and $T(C)$ be the time required to verify $C$ with a verification algorithm $T$.

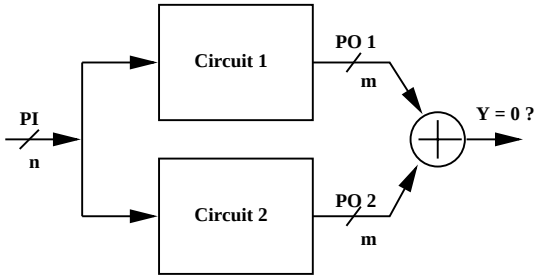


Figure 1: Composite Network

**Definition 2.3** *Let* $S = \{C_1, C_2, \ldots, C_k\}$ *be a set of k partitions of* $C$*.* $S$ *is called a* **complete set of partitions** *(CSP) if and only if the verification of all the circuits in* $S$ *implies the verification of* $C$*.*

Let $S = \{C_1, C_2, \ldots, C_k\}$ be a CSP of $C$. Let $\Sigma$ be the set of all possible CSPs of $C$. Let $T(S) = T(C_1) + T(C_2) + \ldots + T(C_k))$. Now, let us pose a problem $P$ as follows: Is there a CSP $S \in \Sigma \mid (T(S) < T(C))$?

**Theorem 2.1** *Problem P is NP-hard.*

**Proof Sketch:** Combinational verification is an NP-hard problem. Therefore, in order to solve $P$, we have to solve several NP-hard problems. □

Let us pose a second problem $Q$ as follows: Find a CSP $S_i \in \Sigma \mid \forall j, j \neq i, (T(S_i) \leq T(S_j))$.

**Theorem 2.2** *Problem Q is NP-hard.*

**Proof Sketch:** As in the previous theorem, we have to solve several NP-hard problems to solve $Q$. □

The two theorems prove that an exact solution of the partitioning problem is not possible. In this work, we design several efficient heuristics to create good partitions of given circuits. We take advantage of the fact that most large designs typically consist of smaller functional units that are combined by some control constructs like *if-else-if*, *if-then-else* and *case statements*. We can use the control variables to appropriately partition the circuit into smaller functional units which can be verified separately. Identification of these control constructs is relatively easy in HDL descriptions of designs. However, it is not an easy task in gate-level descriptions of designs. In the next section we present efficient heuristics to identify good partitions in given circuits so that the verification problem is simplified.

## 3 Efficient Partitioning Heuristics

During verification the partitioner is invoked at two different places: (1) *pre-verification partitioner* (PRPT) on primary outputs that need partitioning (2) *post-verification partitioner* (POPT) on primary outputs that are aborted by the verifier. The flow diagram for verification with partitioning is shown in Fig. 2.

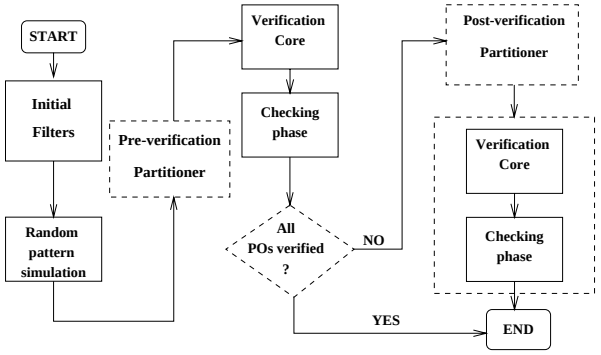


Figure 2: Verification with Partitioning

The pre-verification partitioner is invoked immediately after the collection of potential equivalent nodes with random pattern simulation (see [18] for details). Two nodes $g$ and $h$ are called *potentially equivalent* if they are assigned the same boolean value for all the random vectors applied during random pattern simulation. In other words, nodes $g$ and $h$ have a very good chance of being proved functionally equivalent by a BC tool. The pre-verification partitioner is targeted towards creating partitions that can minimize the number of nodes which do not belong to any list of potentially equivalent nodes. The post-verification partitioner is specifically targted to primary output pairs that have been aborted by the BC tool. It attempts to create partitions to facilitate the verification of these aborted outputs.

The following efficient heuristics have been designed to create good partitions:

1. **Sim:** In this heuristic Boolean 0 and 1 are assigned to each primary input of the circuit and the circuit is simulated. After each simulation, the following parameters are collected: (1) # of nodes set to a constant as a result of simulation (2) # of nodes that do not have any candidate for equivalence check that become constants (Nodes that do not belong to any list of potential equivalent nodes can cause problems for a BC tool. Hence, forcing them to boolean constants can potentially simplify verification.) (3) Disjointness of the partitions that this variable creates (Each variable can create two partitions. If a majority of nodes that occur in the first partition also occurs in the second partition, then the partitions are not disjoint. Disjoint partitions allow the BC tool to avoid repetition of work and thus make the verification more efficient.). The primary input for which each of these parameters is maximized is considered the best partitioning signal.

2. **Dfs:** In this heuristic a guided depth-first traversal is started from the primary output to be partitioned. The circuit is levelized. The fanin nodes that are closer to the primary inputs are traversed first. The primary inputs are arranged in the order they are seen. The primary input that is highest in the order is considered the best partitioning signal. It is important to note that most control signals would be reached early in *Dfs*.

3. **Rev-dfs:** In this heuristic, the order of the primary inputs is the reverse of their order in *Dfs*. The primary input topmost in the order is chosen to partition the output.

4. **Max-tfo:** The primary input that has the largest transitive-fanout (TFO) cone is chosen to partition the output. This heuristic tries to capture the effect a particular signal has on the functionality of the circuit by counting the number of gates it might affect.

5. **Mixed:** We run at most two partitioning passes on each primary output in both *PRPT* and *POPT*. In the first pass the partitioning signal is chosen using *Dfs*. In the second pass it is chosen using *Sim*. Therefore, the first pass tries to identify a control signal, while the second pass tries to identify a signal that has maximum control on the data-path functionality.

6. **Dfs-sim:** In this heuristic, we use *Dfs* in PRPT and *Sim* in POPT. This heuristic is another way to capture the effects of both *Dfs* and *Sim*, and thus combine the advantages of both those heuristics.

Note that in an automatic application of the partitioning heuristics, partitions can often be created in an *unbalanced* manner (number of partitions created is not a power of 2) as shown in Fig. 3. In the figure, two primary inputs $x1$ and $x2$ have been used to create three partitions $P1$, $P2$ and $P3$ of a given composite circuit.

| Ckt. | No-Part | Sim | Dfs | Rev-dfs | Max-tfo | **Mixed** | Dfs-sim |
|---|---|---|---|---|---|---|---|
| 1 | **abort** | 3348.39 | 762.18 | 794.48 | 852.98 | 2400.72 | 771.08 |
| 2 | **abort** | 4149.85 | **abort** | **abort** | **abort** | 1559.36 | **abort** |
| 3 | **abort** | **abort** | 1440.90 | 1190.35 | 1279.23 | 3170.14 | 1165.24 |
| 4 | **abort** | 6952.62 | 1282.00 | 2186.66 | 1462.39 | 2372.96 | 1336.05 |
| 5 | **abort** | **abort** | 600.14 | 659.47 | 819.21 | 2945.95 | 636.91 |
| 6 | **abort** | 1650.84 | 738.82 | 759.24 | 815.43 | 862.16 | 729.23 |

Table 1: Verification with partitioning

| Ckt. | Sim | Dfs | Rev-dfs | Max-tfo | Mixed | Dfs-sim |
|---|---|---|---|---|---|---|
| 1 | 7(0) | 4(0) | 4(0) | 4(0) | 4(0) | 4(0) |
| 2 | 7(0) | 6(4) | 5(2) | 5(2) | 4(0) | 5(2) |
| 3 | 8(2) | 4(0) | 4(0) | 4(0) | 4(0) | 4(0) |
| 4 | 7(0) | 4(0) | 7(0) | 4(0) | 4(0) | 4(0) |
| 5 | 8(2) | 4(0) | 4(0) | 4(0) | 4(0) | 4(0) |
| 6 | 4(0) | 4(0) | 4(0) | 4(0) | 4(0) | 4(0) |

Table 2: Number of Partitions Created and Aborted

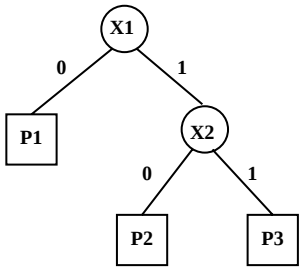


Figure 3: Creation of Partitions

## 4 Experimental Results

The partitioning algorithm has been implemented in C++ within ASSURE[1], which is the combinational verification tool of Fujitsu. All our experiments have been carried out on a Sun SPARC 20 with 512 MBytes of memory. The runtimes are reported in seconds. In Table 1 we present verification results (runtimes) on 6 difficult primary outputs of an industrial circuit. In these experiments one circuit was verified against another copy that had undergone synthesis and engineering change. Each circuit has 317 primary inputs and 232 primary outputs. The two circuits have 15242 and 14737 internal nodes respectively.

The entry "abort" indicates that the circuit could not be verified completely because memory or time limits were exceeded. From Table 1 it can be seen in the second column that ASSURE was unable to verify these circuits without partitioning. The subsequent columns tabulate the verification runtimes of ASSURE when the primary inputs to be partitioned on were selected using different strategies. Both *Dfs* and *Dfs-sim* have similar runtimes. But fewer partitions were aborted when *Dfs-sim* was used. *Mixed* combines the ability to identify both control signals as well as signals that have maximum effect on the data-path function. Using *Mixed* we could verify all the circuits. *Rev-dfs* and *Max-tfo* had similar performance. ASSURE did not perform well with *Sim*. In Table 2 we tabulate # partitions created (# partitions aborted).

[1]For details on ASSURE, please refer to [18, 19]

## 5 Conclusions

Verification methodologies based on extraction and use of internal correspondences can become inefficient if correspondences are few in number. In this paper we propose efficient heuristics which take advantage of known circuit design styles to automatically create good partitions for given circuits and thus transform several previously intractable verification problems into tractable ones. Results of verification (with partitioning) of several hard industrial circuits which could not be verified without partitioning, have been presented. Our future work includes further strengthening of the partitioning techniques, and application of partitioning to accurate error diagnosis.

## References


[1] J. Burch, "Using BDDs to verify multipliers", Int'l Workshop On Formal Methods in VLSI Design, Jan., 1991.

[2] C. L. Berman, L. H. Trevyllian, "Functional Comparison of Logic Designs for VLSI Circuits", ICCAD 1989.

[3] D. Brand, "Verification of Large Synthesized Designs", ICCAD 1993.

[4] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation", IEEE Transactions on Computers, vol. C-35, no. 8, Aug. 1986.

[5] E. Cerny, C. Mauras, "Tautology Checking Using Cross-Controllability and Cross- Observability Relations", ICCAD 1990.

[6] M. Fujita *et al.*, "Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagrams", ICCAD 1988.

[7] J. Jain *et al.*, "Advanced Verification Techniques Based on Learning", DAC 1995.

[8] W. Kunz, "HANNIBAL: An Efficient Tool for Logic Verification Based on Recursive Learning", ICCAD 1993.

[9] S. Malik *et al.*, "Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment", ICCAD 1988.

[10] R. Mukherjee *et al.*, "VERIFUL : VERIfication using FUnctional Learning", EDAC 1995.

[11] S. Reddy *et al.*, "Novel Verification Framework Combining Structural and OBDD Methods in a Synthesis Environment", DAC 1995.

[12] Y. Matsunaga, "An Efficient Equivalence Checker for Combinational Circuits", DAC 1996.

[13] J. Jain *et al.*, "Functional partitioning for verification and related problems", MIT VLSI Conference 1992.

[14] A. Narayan *et al.*, "Partitioned ROBDDs - A Compact, Canonical and Efficiently Manipulable Representation for Boolean Functions", ICCAD 1996.

[15] M. Fujita, "Verification of Arithmetic Circuits by comparing two similar Circuits", CAV 1996.

[16] R. Mukherjee *et al.*, "Efficient Combinational Verification Using Cuts and Overlapping BDDs," Int'l Workshop on Logic Synthesis 1997.

[17] A. Kuehlmann *et al.*, "Equivalence Checking Using Cuts and Heaps", DAC 1997.

[18] R. Mukherjee *et al.*, "An Efficient Filter-Based Approach For Combinational Verification", DATE 1999.

[19] R. Mukherjee *et al.*, "An Efficient Filter-Based Approach For Combinational Verification", to appear in Transactions on Computer-Aided Design, November 1999.