AN EFFICIENT FILTER-BASED APPROACH FOR COMBINATIONAL VERIFICATION

Rajarshi Mukherjee[1] Jawahar Jain[1] Koichiro Takayama[1]
Masahiro Fujita[1] Jacob A. Abraham[2] Donald S. Fussell[2]

[1]Fujitsu Laboratories of America
595 Lawrence Expressway
Sunnyvale, CA 94086
USA

[2]Computer Engineering Research Center
University of Texas at Austin
Austin, TX 78712
USA

Abstract

We have developed a filter-based framework where several fundamentally different techniques can be combined to provide fully automated and efficient heuristic solutions to verification and possibly other NP-complete problems. Such an integrated methodology is far more robust and efficient than any single existing technique on a wide variety of circuits. Our methodology has been applied to verify the ISCAS 85 benchmark circuits and efficient verification results have been presented on a large set of industrial circuits which could not be verified using several published techniques and commercial verification tools available to us.

1 Introduction

The problem of automatic combinational verification or Boolean comparison (BC) can be stated as follows: *Given two Boolean netlists, check whether the corresponding outputs of the two circuits are equivalent for all possible input combinations.* The existing methods for combinational verification can be broadly classified into two categories: **(1)** Based on building and comparing the BDDs of entire networks **(2)** Based on the extraction and use of internal correspondences using a combination of structural and functional techniques [1, 2, 5, 8, 9, 10, 11, 12, 13, 14].

1.1 Deficiencies in current BC approaches

Most current methods for combinational verification are based on a single "core" technique such as OBDDs, ATPG, learning etc. We refer to any verification technique **A** as a *core* technique if given enough time, it can verify a particular circuit without the help of another verification technique. We call two core techniques **A** and **B** *mutually orthogonal* if there are many circuits where **A** is far more superior than **B** and vice-versa. Thus, exhaustive simulation, OBDDs, functional partitioning, resynthesis/learning based methods, *BMDs [4], or the use of ATPG in the framework of [2] are core techniques which are orthogonal. For example, ATPG can efficiently verify erroneous multipliers, whereas *BMDs are very efficient when the design is correct. Also, by the *orthogonality* of two techniques we do not imply that there are no circuits on which both techniques are equally effective. Due to the NP-hard nature of the verification problem it is expected that a single core technique will not perform well on a wide range of circuits. In addition, the nature of any given verification problem is usually not known a-priori. This uncertainty is further aggravated in internal correspondence-based verification techniques where the entire verification problem is broken down into many other verification problems. Thus, a verification program that uses only a single core technique, or a very small number of such techniques (especially if improperly combined), cannot be expected to be very robust.

2 Characteristics of an Efficient BC Scheme

An efficient verification methodology should have the following characteristics: **(1)** it should have good performance on a wide variety of circuits; **(2)** it must be robust in memory usage; **(3)** it must be modular and extensible. To achieve these objectives we propose a *filter*-based combinational verification methodology. The **filter approach** is a combination of communicating techniques where each technique calculates (*filters out*) the information it is most suited for, alters the circuit accordingly, and passes (sieves) its results to the subsequent techniques (*filters*). Typically, easier cases of verification are handled first with fast, low-cost filters, followed by more complex and expensive filters that have a higher time and space complexity. In this paper we propose an extremely efficient verification procedure by systematically integrating various orthogonal verification techniques.

The *key contributions* of this paper are as follows: **(1)** We have developed a filter configuration for very efficient Boolean comparison **(2)** We have intuitively explained the need for such a configuration and the reason for its efficiency and robustness **(3)** Several specialized techniques have been developed to further enhance the performance of the verifier and make very difficult verification problems more tractable.

We demonstrate the efficiency of our BC frame-

work on a large set of industrial designs as well as on the ISCAS 85 benchmark circuits. Many of the industrial circuits could not be verified by several published techniques and commercial BC tools available to us.

2.1 Designing an Efficient BC Methodology

In most current verification methodologies a given verification problem is broken down into several *verification instances*[1] Since the nature and the probability distribution of these instances cannot be known ahead of time, it is impossible for any single verification engine like BDDs or ATPG to perform well on all such instances. In this paper we develop a robust framework for BC which overcomes the drawback that typical single-engine-based techniques would have.

Given a set of core techniques and a set of verification instances, we first make the following important observations:

(1) Each core technique has an **easy-domain** (a set of verification instances which it can verify efficiently with very low space and time overhead), a **moderately-hard domain** (a set of verification instances on which it is not too efficient) and a **hard-domain** (a set of verification instances which are very hard or intractable for the technique).

(2) Given a verification instance and a core technique, we can check (by setting parameters) if the instance is in the **easy-domain** of the technique. For example, if a verification instance can be solved by an ATPG-based tool with a very small backtrack limit (say, 5), then the instance is in the **easy-domain** of ATPG.

(3) For real life BC problems we do not know the nature of the verification instances a-priori. Therefore, to be robust on a wide variety of circuits, a BC tool should not depend on a single core technique.

(4) Therefore, an efficient BC tool should be configured as a set of *communicating orthogonal* verification techniques (filters) arranged in a certain order. A verification instance can trickle down through successive filters starting at the topmost filter. A set of filters can also use partial results of each other to solve a verification instance more efficiently. For example, BDDs can be used to prune a large portion of the search space for an ATPG-based filter, thus making the latter more effective. Such interaction of two or more filters could result in a verification instance being verified more efficiently than by any filter alone. The goal of the filter configuration is to minimize the time and space resources used by each technique and to try to verify each instance using the least expensive technique available. Each technique uses three criteria to pass a verification instance to a subsequent technique: **(a)** runtime bound, **(b)** memory usage bound and, **(c)** based on information extracted from the network.

3 Details of a Filter-Based Verifier

The proposed combinational verifier consists of a set of filters arranged one after another. Given a pair of circuits N_1, N_2 to be verified, the miter circuit (N_C) created from N_1, N_2 is first subjected to fast techniques with very low memory overhead followed by a series of increasingly powerful methods that require more time and memory. The entire verifier proceeds by verifying pairs of internal nodes in N_C and merging the equivalent pairs. The flow diagram of the filter approach is shown in Fig. 2. Note, an important advantage of the filter configuration is that the early verification of easier instances can make the subsequent verification of harder instances easier. Fig. 1 shows how finding easy equivalences can either remove the need to solve some harder problems (such as $H_1 \oplus H_{1'}$) or modify the fault propagation/excitation problem for some gates (such as $H_2 \oplus H_{2'}$) so that ATPG becomes easier. Table 2 (compare column 8 with column 2) quantifies how finding easy equivalences earlier can dramatically reduce the total verification time.

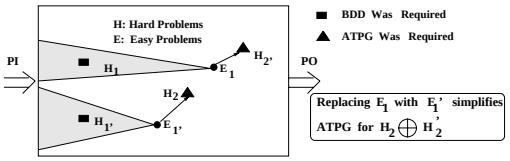

Figure 1: *Simplifying effect of early filters*

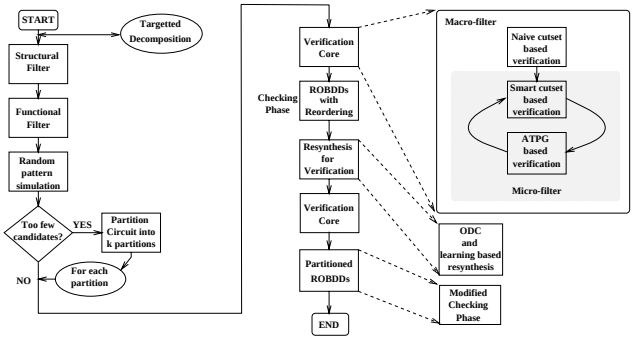

Figure 2: *Flow Diagram of Verification: Each Filter Passes a Modifed Circuit to the Next Filter*

3.1 Filter 1: Targetted Decomposition (TD)

During multilevel optimization of circuits many internal correspondences between the original network and its optimized version could disappear. We carry out a fast re-synthesis of the given network (N_C) to create a new network (N_{C_1}) where many internal correspondences can be uncovered to help verification. Complex gates with more than 4 inputs and simple gates with more than 10 inputs are decomposed into AND and OR trees respectively.

3.2 Filters For Easier Verification Instances

In this section we describe the two low-cost filters that are sequenced at the beginning of the combinational verifier in order to verify the easier instances of verification.

[1] We refer to a pair of nodes g and h, whose functional equivalence has to be proved, as a *verification instance.*

3.2.1 Filter 2: Subgraph Isomorphism-based Structural Filter (SIF)

Since synthesis tools often make only local modifications to circuit structures, the structurally isomorphic parts of the two networks can be identified (creating a new network N_{C_2}) using a fast method based on subgraph isomorphism recognition and can be merged.

3.2.2 Filter 3: OBDD Hashing-based Functional Filter (BHF)

This filter is based on the techniques proposed in [9, 13]. It consists of building BDDs of internal nodes of the circuit in terms of different cutsets and hashing the nodes based on their BDDs. Nodes that hash to the same location in the hash table are merged (creating a new circuit N_{C_3}). This filter can verify most easy instances with a very low computational cost.

3.3 Auxilliary Filters

Before entering the filters which verify harder instances of verification we attempt to simplify the existing verification problem using two filters which collect information about potentially equivalent nodes in N_{C_3} and decide to partition the circuit based on this information in order to create more equivalences and further simplify the BC problem.

3.3.1 Filter 4: Random Pattern Simulation

In this phase we carry out regression-based random pattern simulation to determine candidate nodes which may be functionally equivalent. Many inequivalent primary outputs can be identified and removed to produce an altered network N_{C_4}.

3.3.2 Filter 5: Circuit Partitioning

If the number of candidates for functional equivalence are very few compared to the total number of gates in N_{C_4}, or if the structural distance between the candidates nearest to the primary outputs and the primary output nodes is very large, a set of reduced circuits (functions) can be created by partitioning the 2^n Boolean space of the original circuit (function) having n primary inputs by assigning Boolean constants to a small number of primary inputs. In order to verify N_1 and N_2, we verify if each partition of N_{C_4} can be reduced to a Boolean 0. Circuit partitioning also helps verification by creating **(1)** new **conditional equivalences** (e.g. two gates g_a, g_b may be equivalent only when $x_i = 0$) **(2)** new **conditional learnings**.

4 Verification in The Hard Domain

If for a given node pair (n_1, n_2) we have still not been able to resolve whether $n_1 \equiv n_2$, then we are faced with one of the following two scenarios: **(1)** $n_1 \equiv n_2$: Typically, BDDs are best suited to prove this relation. If ATPG is used, we would have to prove the redundancy of a s-a-0 fault at the output of a miter as in [2]. But this could be computationally expensive if the smallest search space required to prove the redundancy is very large. **(2)** $n_1 \not\equiv n_2$: ATPG is best suited to derive an assignment to the primary inputs which differentiates n_1 and n_2. This is because building the canonical BDD for all the differentiating vectors is needlessly expensive. However, we have encountered a small number of cases where ATPG is faster than OBDDs in proving $n_1 \equiv n_2$. Due to a rigorous regression-based simulation, we expect a majority of the candidate pairs (n_1, n_2) to be actually functionally equivalent. Therefore, in order to minimize the time spent for each verification instance, the initial filters during this phase of verification are BDD-based. These are followed by an ATPG-based approach.

4.1 Filter 6: Macro-Filter

The core (VC) of the verification program consists of three distinct filters that are tightly bound by an efficient time-out (to limit the time resources used) and space-out mechanism (to limit the memory used). Two of these three techniques are combined in a loop to create a highly efficient internal filter. We refer to the top-level filter as the **macro-filter** (Fig. 2) and the internal filter as the **micro-filter**. Pairs of nodes are picked up from the candidate lists created by random pattern simulation and VC is applied to verify if they are functionally equivalent. All equvalent nodes are merged.

The three filters constituting VC are: **(1) Naive cutset**-based verification (NCV) **(2) Smart cutset**-based verification (SCV) **(3) ATPG**-based verification (AV). NCV and SCV are variants of a BDD-based filter. Each technique is associated with its own **time-out limit**. The first two methods are BDD-based and are associated with their own **space-out limits**. Variable reordering is invoked only for large BDDs. The ATPG-based method is controlled by setting a limit on the number of backtracks. It is aborted if a preset number of backtracks is exceeded. The smart-cutset-based technique and ATPG-based technique are arranged in a loop (**micro-filter**). Each successive entry into this loop causes these two techniques to be invoked with a higher time-out, space-out, and backtrack limit, and hence with higher strength. This enables the micro-filter to try and choose the exact technique (BDD or ATPG) and allocate the right amount of computational resources for each difficult verification instance. Thus, given a pair of nodes to be verified, we give each of the two methods (the BDD-based method and the ATPG-based method) a fair chance to solve the problem before spending more computational resources on the other method. Our experiments have shown that this technique is very efficient, especially on difficult circuits. The micro-filter can be further enhanced by using the partial information captured by BDDs to reduce the ATPG search space and by using the information found by ATPG to reduce the BDD size.

If after the first pass through SCV and AV, we could not answer if $n_1 \equiv n_2$, then SCV and AV are repeated (within the micro-filter) with higher time-out, space-out and backtrack limits. The time complexity of VC depends on the space and time limits set on the OBDD and ATPG routines and is fully controllable.

4.2 Filter 7: ROBDDs with Reordering

This phase (checking phase) is focussed on pairs of corresponding primary outputs not yet proved equivalent. To prove the equivalence of two primary outputs F_1 and F_2, we first successively compose and reorder the OBDDs of $F_1 \oplus F_2$ in terms of cutsets of internal equivalent points [8, 11] till we reach the primary inputs, or the function is reduced to a 0. Since this phase

Table 1: Comparing our method with other published techniques (all methods run on SUN Sparc-20)

Ckt.	**Filter** (in SIS Environ.)	[11] (A) (in SIS Environ.)	[11] (B) (Original Code) (Non-SIS Env.)	[12] (Original Code) (Non-SIS Env.)	OBDDs (CUDD)
c432 vs. c432nr	**0.40**	0.88	0.73	0.49	1.8
c499 vs. c499nr	**0.37**	1.05	0.84	1.14	46.2
c1355 vs. c1355nr	**0.95**	4.55	1.67	3.5	155.8
c1908 vs. c1908nr	**2.13**	5.08	4.32	5.76	8.1
c2670 vs. c2670nr	**3.38**	7.73	2.90	48	8.3
c3540 vs. c3540nr	**12.65**	22.03	15.14	365	36.5
c5315 vs. c5315nr	**8.32**	12.85	10.04	417	5.9
c6288 vs. c6288nr	**7.20**	44.72	11.85	24.87	unable
c7552 vs. c7552nr	**20.78**	45.73	17.56	1911	32.8
RC2 vs. RC3	**15.43**	11.57	10.80	unable	unable
RC2 vs. RC2.opt	**1.75**	5.18	2.22	unable	unable
RC2 vs. RC2.high.opt	**38.45**	unable	unable	unable	unable
fms vs. fmst	**123.35**	unable	unable	unable	unable
ut vs. utnr	**48.50**	unable	unable	unable	unable
Mp.6288 vs. Mp.6288diff	**58.72**	unable	unable	unable	unable
msw vs. msw.new1	**52.88**	unable	unable	unable	unable
msw vs. msw.new2	**51.35**	unable	unable	unable	unable
nin vs. nin.new.1	**44.18**	unable	unable	unable	unable
ut vs. ut.new	**1073.10**	unable	unable	unable	unable

NOTES: **(1)** All runtimes in seconds. **(2) unable** $\Rightarrow$ (Space $>$ 512 MB) OR (Time $>$ 100K seconds) **(3) For [11] (B), and [12] their original source code used**; **(4)** Note that [11] (A) (our prototype of the algorithm of [11]) has somewhat slower run times than the original program of [11] (referred as [11] (B)) that we have obtained from its author. Upon our enquiries with the author of [11], as well as upon examination of his *source code*, we found out that this is due to our use of the SIS environment and of a substantially slower BDD package.

could face a memory explosion, we terminate the BDD construction if a large preset space-out/time-out limit is exceeded. This enhances the efficiency of the verifier.

4.3 Verifying Very Hard Instances

In this phase the checking phase is re-invoked separately on each hard output pair in the presence of BDD partitioning [7]. Each partition is maintained under a separate BDD manager and is independently reordered to reduce the possibility of a memory explosion. Selective re-synthesis of the circuit is also carried out to create more equivalences.

5 Experimental Results

The proposed algorithm has been implemented in C within the SIS environment, using the CMU BDD package with dynamic reordering, and run on a Sun SPARC 20 with 512 MBytes RAM. Our test circuits include the combinational parts of various designs from Fujitsu, such as data transfer buffers, data transfer controllers, hard-wired models for logic/fault simulation, crossbar switch controllers, and the switching unit of a parallel machine. We have also successfully verified numerous difficult circuits provided by other industrial organizations, including EDA vendors. The sizes of the circuits verified ranged up to 100K gates.

We first present a comparison of our methodology with some of the existing techniques. Then we present the results of several experiments in order to prove that our organization of the various filters is by far the best among many other options.

Explanation of Table 1: Here we compare our methodology with several published techniques whose programs we could obtain and thus run on the same machine. Also we have implemented a prototype of the algorithm in [11] (one of the core techniques incorporated in our filter framework) in the SIS environment. Our methodology is, almost always, much faster than the other techniques. Due to our highly effective filter approach we are able to verify many circuits that [11] is unable to verify. When the algorithm in [11] fails, it is mostly because of a BDD blow-up while trying to verify a pair of internal candidates. Since we incorporate efficient orthogonal techniques for addressing such a case, our method is robust over a much wider range of circuits. Functional learning-based techniques [8, 12] failed to verify many industrial circuits and were inferior to the proposed approach on the ISCAS85 circuits due to the high computational cost of extracting implications. Since extracting implications using recursive learning (a patented technique, unavailable to us) and functional learning have similar complexity, we believe a recursive learning-based verifier [10] would also fail on these circuits. Our own version of the approach in [2] using our own ATPG tool also failed on the industrial circuits.

Explanation of Table 2: Table 2 establishes that our proposed filter configuration has the best average performance among several possible alternative filter configurations. Although several of the other configurations, shown in the table, outperform our proposed method on some circuits, they have much worse average performance. The results show that our proposed configuration has the best overall performance. From Table 2 we can draw the following additional conclusions. We see that methods (1) and (5) have comparable performance. Note that (1) and (5) are very similar except that in the latter case, while testing the equivalence of two nodes n_1, n_2 using ATPG techniques, we

Table 2: Runtime comparison for different filter configurations

Ckt.	**(1)**	(2)	(3)	(4)	(5)	(6)	(7)
c432 vs. c432nr	**0.40**	0.47	0.50	0.43	0.38	0.48	0.90
c432 vs. c432.opt.1	**10.45**	17.58	23.82	10.30	10.60	11.80	2.65
c432 vs. c432.opt.2	**13.78**	12.53	15.85	13.53	13.60	15.0	13.75
c499 vs c499nr	**0.37**	0.37	0.37	0.38	0.35	0.32	1.13
c1355 vs. c1355nr	**0.95**	0.90	0.88	0.87	0.92	0.72	4.47
c1908 vs. c1908nr	**2.13**	1.95	1.85	2.22	2.12	2.08	4.83
c2670 vs. c2670nr	**3.38**	5.45	5.55	3.37	3.47	2.30	7.68
c3540 vs. c3540nr	**12.65**	27.53	27.57	69.20	12.23	11.10	22.95
c5315 vs. c5315nr	**8.32**	35.10	35.13	8.18	8.15	6.0	14.52
c6288 vs. c6288nr	**7.20**	7.07	7.22	7.05	7.07	5.63	56.05
c7552 vs. c7552nr	**20.78**	45.48	45.52	20.58	25.80	19.28	41.37
RC2 vs. RC3	**15.43**	26.20	25.05	13.75	15.62	14.52	15.50
RC2 vs. RC2.opt	**1.75**	1.60	1.65	1.70	1.63	1.47	11.05
RC2 vs. RC2.high.opt	**38.45**	87.45	86.35	34.05	43.07	146.68	17.00
fms vs. fmst	**123.35**	263.70	268.68	599.68	123.23	145.65	244.30
ut vs. utnr	**48.50**	47.80	48.08	42.70	47.90	50.83	1418.32
Mp.6288 vs. Mp.6288diff	**58.72**	1543.80	1543.43	254.45	56.32	53.97	78.93
msw vs. msw.new1	**52.88**	182.15	183.65	555.32	52.37	69.70	81.10
msw vs. msw.new2	**51.35**	188.55	194.73	360.28	51.47	65.97	175.13
nin vs. nin.new.1	**44.18**	47.18	46.10	43.65	47.30	40.72	1036.08
ut.new	**1073.10**	5380.70	5262.40	1100.03	1167.32	1037.23	1824.73
TOTAL	**1588.18**	7928.54	8203.03	2713.87	1738.79	1752.28	6534.87

(1) proposed algorithm; (2) micro-filter sequence: ATPG, naive-cut, smart-cut, ATPG, smart-cut; (3) micro-filter sequence: no loop: ATPG, naive-cut, smart-cut; (4) micro-filter sequence: no loop: naive-cut, smart-cut, ATPG; (5) No fault tested by ATPG under ODC; (6) subgraph isomorphism disabled; (7) both subgraph isomorphism and BDD-hash-based filters turned off.

only check if $n_1 \oplus n_2 = 0$ rather than whether the difference in the functions of the two nodes is observable at the primary outputs. We find that on most optimized circuits, method (5) is slower than (1). This shows that equivalences derived under ODC are helpful in the verification of highly optimized circuits. Method (2) investigates the performance of the micro-filter if ATPG precedes the other core techniques. The results show that this is very inefficient. This is because after a rigorous regression-based simulation and pruning of candidate lists, it is expected that most of the candidate pairs would be functionally equivalent. Therefore, in a majority of the internal verification cases, the ATPG tool has to prove that the corresponding fault is redundant, which may be a difficult task if the search space is large. BDDs, on the other hand, can be used for efficient compaction of the search space and can be more effective in proving the equivalence of internal nodes. Experiments (3) and (4) show that an internal loop in the micro-filter is highly useful. The results show that both these methods are very inefficient. This is because, in the absence of the internal loop, there is no way to identify the easy-domain of a given instance of verification. Thus, since each core technique in the micro-filter can be invoked only once, it has to be invoked with a fairly high timeout and space-out limit. Therefore, much higher time and space resources could be required for many easy verification instances. Experiments (6), (7) have been carried out to evaluate the importance of SIF, SIF-with-BHF. The results are slower than those of experiment (1), proving that they are an important part of the entire framework.

5.1 Simplifying Effect of Earlier Filters

Verification of easier instances by the earlier filters can modify the circuit so that the verification of harder instances by the subsequent filters can become disproportionately easier. For example, in verifying msw vs. msw.new.1, we find that initial filters (SIF, and BHF) consume a total of 4.6 seconds in processing/modifying the circuit. Later, when ATPG is invoked on the modified circuit, in 32 seconds it identifies 20 particular pairs of gates as equivalent. On the other hand, without the initial filters, ATPG took 48 seconds on the same 20 verification instances. Similarly, the BDD time in the NCV, and SCV filter increased from 9 to 22 seconds, and also larger BDDs were required. In the circuit *nin*, the earlier filters reduce verification time from 1036 seconds to 44 seconds. In c3540, with easy equivalences identified by initial filters, space-out was invoked 10 times. This increased to 34 times without the modifying effect of such filters, which can remove (or, modify) large cones in a circuit, and thereby effectively make many hard verification problems easier.

5.2 An Illustration of the Economy Inherent in the Filter Process

Table 3: Comparison of Time spent in Different Filters

Phase	# runs	eq/inv.	ineq.	time	TPI
str-filter	1	11	-	0.07	0.006
BDD-hash	1	13	-	0.17	0.013
(Rand-sim.)	1	-	-	0.13	-
BDD-based-1*	19	11	0	7.12	0.37
ATPG-1	2	0	1	1.55	0.77
BDD-based-2**	1	1	0	4.37	4.37
ATPG-2	-	-	-	-	-

***** Naive-Cut + Smart-Cut: first iteration; ****** Smart-cut: Second iteration.

Table 3 presents detailed results for verifying *c432* against its heavily optimized version. The results show that our arrangement of the filters ensures that the

time per instance[2] (**TPI**) for each filter in the verifier is usually less than the TPI of its subsequent filters. This implies that the filter framework is usually successful in verifying each verification instance using a filter with lowest computational cost. An instance is passed to a more expensive subsequent filter only when it is recognized that the instance is too hard for the present filter to handle. This ensures that the cost of the overall verification is minimized. Both NCV and SCV being variants of a filter based on (BDD + internal Equivalent Points), we have added their times and represented it by a filter called *BDD-based-1*. The first (second) pass of ATPG filter is called *ATPG-1* (ATPG-2), the second pass of SCV filter as *BDD-based-2*. The number of verification instances found to be equivalent and inequivalent by each filter are shown in the third and fourth column respectively. It is seen that a majority of the equivalences are recognized by the initial low cost filters SIF and BHF. A majority of the remaining equivalences and inequivalences are caught in the phase *BDD-based-1*. There is one instance which does not fall in the easy domains of either SCV or AV but the next run of SCV with a higher strength automatically caught it.

5.3 Verification In the Very Hard Domain: Use of Partitioning

In all the circuits presented in Table 1 and Table 2 partitioning was not necessary. We discuss next the verification of two difficult circuits (not shown in Tables 1 and 2) on which automatic partitioning had to be invoked. **All published techniques available to us and several commercial verifiers failed to verify these circuits**.

Case 1: OP1 vs. OP2: In this circuit we found that the frontier of candidate nodes (found by simulation) that is nearest to the primary outputs is actually on an average 15 structural levels away from the primary outputs. Thus, both BDDs and ATPG proved ineffective. Partitioning on the input space was applied (7 partitions were created), which reduced the verification time from 23,000 seconds to only 1336 seconds!

Case 2: Two artificial circuits were created to verify multipliers in [6]. A function $f(x, y)$ is a multiplier if the following relations are satisfied: $f(x, 0) = 0$ and $f(x, y+1) = f(x, y) + x$. The first condition is easy to check. To verify the second condition two circuits C_1 and C_2 are built for $f(x, y+1)$ and $f(x, y) + x$ respectively and are verified for equivalence. C_1 and C_2 are intractable to OBDD-based verification. We also found that these two circuits have no internal equivalences. Therefore, the circuits were partitioned to create internal equivalences, after which they could be verified in 481 seconds.

6 Conclusions

We have proposed a *fully automated*, filter-based approach for Boolean Comparison where numerous verification techniques are arranged according to their fundamental characteristics. We have presented intuitive explanations and experimental evidence to show that our approach is extremely efficient. Verification results have been presented on the ISCAS 85 benchmark circuits and a large number of industrial circuits. Many of these industrial circuits could not be verified using several available published techniques and popular commercial verification tools. Detailed comparison with several published techniques shows the superiority of our approach.[3] In order to prove that the filter configuration proposed by us has the best performance, we have reported verification results comparing the proposed configuration with several other possible arrangements. Experimental evidence to show that our arrangement of various verification techniques allows each verification instance to be solved with the least computational cost has been presented. As future work, we plan to apply the filter framework to provide efficient solutions to other NP-hard problems. We also plan to investigate the interaction of the filters and use of partial information produced by one filter by other filters in order to improve the performance of the verifier.

References

[1] C. Berman *et. al.*, "Functional Comparison of Logic Designs for VLSI Circuits," ICCAD 1989.

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

[3] R. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," IEEE Trans. Comp., Aug. 1986.

[4] R. Bryant *et. al.*, "Verification of Arithmetic Circuits with Binary Moment Diagrams," DAC 1995.

[5] Cerny *et al.*, "Tautology Checking Using Cross-Controllability and Cross-Observability Relations," ICCAD 1990.

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

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

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

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

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

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

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

[13] R. Mukherjee *et al.*, "Efficient Combinational Verification Using BDDs and a Hash Table," ISCAS 1997.

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

[2] This is the average runtime required by any given filter per verification instance given to it.

[3] Non-disclosure agreements do not allow us to publish verification results comparing our approach with commercial verification tools.