CLEVER: Divide and Conquer Combinational Logic Equivalence VERification with False Negative Elimination

John Moondanos[1], Carl H. Seger[2], Ziyad Hanna[3], and Daher Kaiss[4]

[1] Logic Validation Technologies, Intel Corporation
john.moondanos@intel.com
[2] Strategic CAD Labs, Intel Corporation
carl.seger@intel.com
[3] Logic Validation Technologies, Intel Corporation
ziyad.hanna@intel.com
[4] Logic Validation Technologies, Intel Corporation
daher.kaiss@intel.com

Abstract. Formal equivalence verifiers for combinational circuits rely heavily on BDD algorithms. However, building monolithic BDDs is often not feasible for today's complex circuits. Thus, to increase the effectiveness of BDD-based comparisons, divide-and-conquer strategies based on cut-points are applied. Unfortunately, these algorithms may produce false negatives. Significant effort must then be spent for determining whether the failures are indeed real. In particular, if the design is actually incorrect, many cut-point based algorithms perform very poorly. In this paper we present a new algorithm that completely removes the problem of false negatives by introducing *normalized functions* instead of free variables at cut points. In addition, this approach handles the propagation of input assumptions to cut-points, is significantly more accurate in finding cut-points, and leads to more efficient counter-example generation for incorrect circuits. Although, naively, our algorithm [1] would appear to be more expensive than traditional cut-point techniques, the empirical data on more than 900 complex signals from a recent microprocessor design, shows rather the opposite.

1 Introduction

The design process of complex VLSI systems can be thought of as a series of system-model transformations, leading to the final model that is implemented in silicon. In this paper, we concentrate on formal verification techniques establishing logic functionality equivalence between circuit models at the RTL and schematic levels of abstraction. Traditionally, such techniques operate under the assumption that there is a 1-1 correspondence between the state nodes of the two circuit models, in effect transforming the problem of sequential circuit verification into one of combinational verification. Therefore, they are able to exploit

[1] US Patents are pending for this algorithm.

G. Berry, H. Comon, and A. Finkel (Eds.): CAV 2001, LNCS 2102, pp. 131–143, 2001.

the power of Reduced Ordered Binary Decision Diagrams [1] (from now on called simply BDDs). Although BDDs are very useful in this domain, they still suffer from exponential memory requirements on many of today's complicated circuits.

To overcome this, many researchers have investigated alternative solutions based on a divide-and-conquer approach. They attempt to partition the specification and implementation circuits along **frontiers** of equivalent signal pairs called **cut-points**. The goal of the overall equivalence verification is now transformed into one of verifying the resulting sub-circuits. This situation is depicted in Fig. 1.

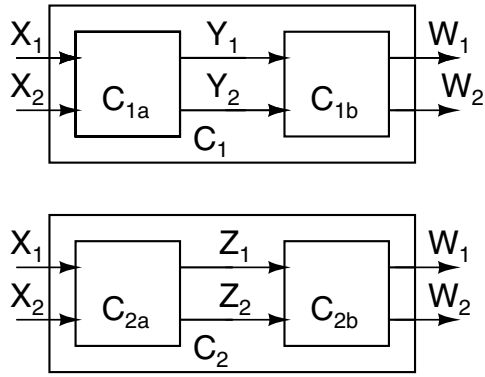

Fig. 1. Circuit Partitioning across Cut-Points.

The two circuits C_1 and C_2 compute their outputs (W_1, W_2) from their inputs (X_1, X_2). If the BDDs for the outputs as functions of the primary inputs grow exponentially in size, one could hope to reduce the complexity of the problem by exploiting the fact that internal nodes Y_1 and Y_2 of C_1 are equivalent to Z_1 and Z_2 of C_2, respectively. If this were the case, one could prove the equivalence of C_1 and C_2, by first establishing the equivalence of C_{1a} and C_{2a} and then the equivalence of C_{1b} and C_{2b}. It would be expected that potentially the sizes of the BDDs that correspond to the sub-circuits C_{1a}, C_{1b}, C_{2a} and C_{2b} are considerably smaller than the intractable sizes of C_1 and C_2, so that the verification of the original circuits could complete. The motivation behind such cut-point based techniques is the desire to exploit the potentially large numbers of similarities between the two circuit models.

Unfortunately, CP-based techniques suffer from some serious limitations. More specifically, when we perform the verification of C_{1b} against C_{2b} in Fig. 1, we consider Y_1, Y_2, Z_1, and Z_2 to be free variables (i.e., they can assume arbitrary boolean values, with $Y_1 = Z_1$ and $Y_2 = Z_2$). This can lead to problems in the verification of C_{1b} and C_{2b}. For example, consider the circuits in Fig. 2.

Let us assume that we could prove the equivalence of Y_1 and Y_2 to Z_1 and Z_2 respectively. Then if we introduced the same free variable A for (Y_1, Z_1) and

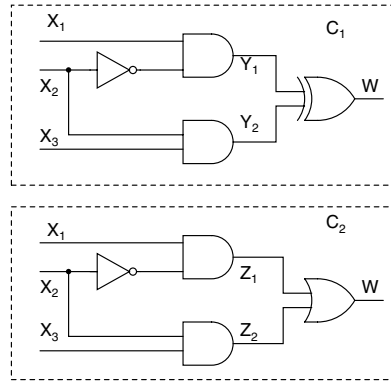

Fig. 2. False negative as a result of the free variables at the cut-points variables (Y_1, Z_1) and (Y_2, Z_2).

B for (Y_2, Z_2), we would compute that C_1 calculates $W = A \oplus B$, while C_2 calculates $W = A + B$. This does not allow us to conclude that the two circuits are equivalent. However, this is actually a false negative. The reason is that, due to the nature of the logic that generates them, Y_1 and Y_2 cannot be both 1 at the same time, i.e., they are **mutually exclusive**. The same is true for Z_1 and Z_2. As a result, the two circuits actually produce the same W functions, because for mutually exclusive input signals, XOR and OR gates produce the same results.

Given this problem, cut-point based verification algorithms usually perform the following operations:

1. Discover as many cut-points as possible, hoping to produce the smallest possible sub-circuits whose functions will be computed and compared with BDDs **(cp-identification)**.
2. Choose out of these cut-points the ones that (based on various criteria) simplify the task of sub-circuit verification **(cp-selection)**.
3. Perform the actual verification of the resulting sub-circuits **(eq-check)**, and
4. Attempt to determine whether the corresponding circuit outputs that appear as inequivalent, are truly different or the algorithm produced false negatives **(fnr, false negative reduction)**.

A comprehensive review of the existing cut-point based algorithms appears in [8]. For *cp-identification* traditionally **random simulation**, automatic test pattern generation (**ATPG**) or **BDD-based** techniques are employed. Out of the cut-points so identified, some are rejected in the cp-selection stage, according to various criteria that are outside the scope of this paper. The remaining are used to form the boundaries of the sub-circuits to be verified. Then the resulting sub-circuits are verified independently, most frequently with the use of BDDs. If all these sub-circuits are verified equal, then the original circuits are equal. Nevertheless, as we have showed in Fig. 2, the cut-point based algorithms can indicate that the circuits are different as a result of false negatives.

Thus, the presently known cut-point algorithms perform a final stage of false negative reduction (fnr). One method that is employed is that of re-substitution of the cut-point functions [9]. In the example of Fig. 2, we have for C_1 that $W = (Y_1 \oplus Y_2)$ and for C_2 that $W = (Y_1 + Y_2)$ given that $Y_1 = Z_1$ and $Y_2 = Z_2$. Although, the two circuits appear to calculate different outputs based on cut-points, if we compose into the expressions for W the functions of Y_1 and Y_2, we prove circuit equivalence. The main difficulties with this technique are that in the worst case we might have to compute the entire function of a circuit's output, which may be the very problem that we attempted to avoid by using the cut-points algorithm. The method presented in [8] is based on maintaining multiple cut-point frontiers from which to choose the functions to be composed into the output functions, with the hope that some of the frontiers will lead to the desired results.

Other false negative reduction techniques [4] are based on the idea of maintaining the set of values that the cut-points are allowed to assume. Again for the case of the circuit of Fig. 2, we can see that the cut-point variables (Y_1, Y_2) or (Z_1, Z_2) can belong only to the set $\{(0,0),(0,1),(1,0)\}$ since they are mutually exclusive. Such sets are encoded by BDDs and are used to restrict the comparisons of circuit node functions within the regions of allowed cut-point values. Unfortunately, maintaining and propagating these sets are often very difficult and computationally expensive. One other approach to the problem of false negative reduction is based on Automatic Test Pattern Generation techniques (ATPG) as in [5]. There, for each value combination of the cut-points that causes the circuit outputs to mismatch, they attempt to find (using an ATPG tool) the input pattern that can generate the cut-point value combination in question. The drawback of this algorithm is that one must call the ATPG tool for each cut-point value combination that causes a mismatch, until one determines whether the mismatch is a false negative or a real design problem. This can be time-consuming for a large number of cut-point value combinations.

The fundamental limitation of these approaches is that they fail to identify the cause of the false negatives. In this paper we identify the cause and we design an algorithm that allows us to eliminate false negatives early during the cp-identification stage of the algorithm. As the careful reader will realize this leads to a simpler algorithm, since we do not perform a final false negative reduction stage with potentially very expensive BDD operations.

2 Cut-Point Algorithm in CLEVER

2.1 Avoiding False-Negative Generation

Through the example we presented in the previous section, one can understand that the key reason for false negatives is the fact that not all value combinations can appear at the nodes of a cut-point frontier. In this section we restate the following observation from [6] which forms the basis for the development of our false negative elimination approach.

Proposition 1. *In a minimized circuit without re-convergent fanout, verification algorithms based on cut-point frontiers cannot produce false negative results.*

Proof. (*Sketch only) Intuitively, if there are no re-convergent fanouts then each cut-point can be completely controlled by its fanin cone. Additionally if the circuit is minimized it cannot have constant nodes. As a consequence all possible value combinations can appear at the signals of any cut-frontier, and therefore false negatives cannot happen.* □

Proposition 1 identifies the reason for the false negative result. Clearly, the reconvergent fanout of node X_2 makes Y_1 and Y_2 to be correlated. As a result not all possible value combinations can appear on Y_1 and Y_2, as implied by the introduction of free variables by some cut-point based techniques. A more sophisticated algorithm should not assign free variables on Y_1 and Y_2, since this will not allow the W signals in the two circuits to be identified as equal. Here we present an alternative approach, where instead of assigning a free variable to a cut-point, we assign a function that captures the correlation between cut-points. To make the BDD representation of this function as small as possible, we attempt to exclude from it the effect of the non-reconverging signals in the support of the cut-point, based on Proposition 1.

Let V be a cut-point that will be used to calculate additional cut-point frontiers. The logic gates from the previous cut-point frontier that generate V, implement a logic function $g(\boldsymbol{r}, \boldsymbol{n})$. The variables $\boldsymbol{r}$ and $\boldsymbol{n}$ correspond to cut-points from the previous frontier. However, here, we have partitioned all these variables into two groups. The $\boldsymbol{r}$ variables correspond to cut-points with reconvergent fanout that leads outside of the cone of the signal V. On the other hand, the $\boldsymbol{n}$ variables correspond to cut-points that do not have fanout re-converging outside the cone of V. The goal here is to capture the effect of the re-converging signals on V, so that the $\boldsymbol{r}$ variables and the free variable we introduce for V do not assume incompatible values. We hope that by doing this, we can avoid introducing false negatives in the process of calculating the cut-points belonging to the next frontier.

Now, to capture the relationship between the $\boldsymbol{r}$ variables and signal V, we examine the situations where they force V to assume a specific value, either 1 or 0. The $\boldsymbol{r}$ signal values that can force V to be 1 are the ones for which, for every value combination of the $\boldsymbol{n}$ variables , V becomes 1. These values of $\boldsymbol{r}$ are captured by universally quantifying out the $\boldsymbol{n}$ variables from $g(\boldsymbol{r}, \boldsymbol{n})$ as in the following function F_g :

$$F_g = F_g(\boldsymbol{r}) = \forall \boldsymbol{n}.g(\boldsymbol{r}, \boldsymbol{n})$$

Subsequently, we call this function the **forced term** to be intuitively reminded of its meaning in the context of cut-points algorithms (although other naming conventions exist in the literature). Now, let us examine when the V signal equals 0. This happens for all those $\boldsymbol{r}$ values that make $g(\boldsymbol{r}, \boldsymbol{n}) = 0$ regardless of the values of the $\boldsymbol{n}$ signals. So V should be 0 whenever the following function P_g is 0:

$$P_g = P_g(\boldsymbol{r}) = \exists \boldsymbol{n}.g(\boldsymbol{r}, \boldsymbol{n})$$

This function is the result of existentially quantifying out the $\boldsymbol{n}$ variables from $g(\boldsymbol{r}, \boldsymbol{n})$ and will subsequently be called the **possible term**. Thus, if for a given value combination of the $\boldsymbol{r}$ variables, all the possible combinations of the $\boldsymbol{n}$ variables make $g(\boldsymbol{r}, \boldsymbol{n}) = 0$, then the free variable assigned to V in CP-based algorithms must obtain only the 0 value. Otherwise, we will have to cope with the potential appearance of false negatives. On the other hand, if some of the combinations of the $\boldsymbol{n}$ variables make $g(\boldsymbol{r}, \boldsymbol{n}) = 0$ and some others make it $g(\boldsymbol{r}, \boldsymbol{n}) = 1$ for a specific $\boldsymbol{r}$ variable value combination, then V can assume either 0 or 1 independently from the $\boldsymbol{r}$ variables.

From this discussion it becomes apparent that a more appropriate assignment to the V signal for the calculation of the cut-points in the next frontier and the avoidance of false negatives is:

$$vP_g + F_g = v.(\exists \boldsymbol{n}.g(\boldsymbol{r}, \boldsymbol{n})) + (\forall \boldsymbol{n}.g(\boldsymbol{r}, \boldsymbol{n})) \tag{1}$$

We call expression (1) the **normalized function** for the signal V, as opposed to the free variable that is assigned to it in other implementations of cut-point based algorithms. We also call the variable v that we introduce for the signal V the **eigenvariable** of V. To illustrate the use of *normalized functions* we consider the circuit of Fig. 2. X_2 has reconvergent fanout that affects both Y_1 and Y_2. The possible term for Y_1 is X_2', while the forced term is 0. Therefore, the normalized function for Y_1 is $v_1.X_2' + 0 = v_1.X_2'$. Similarly, for Y_2 the possible term is X_2, while the forced term is 0. So, Y_2 gets a normalized function of: $v_2.X_2+0 = v_2.X_2$. Now, signals Z_1 and Z_2 of C_2 get the same normalized functions as Y_1 and Y_2 respectively, since they implement the same functions in C_2 as their counterparts in C_1. So, the function for W in C_1 becomes $v_1.X_2' \oplus v_2.X_2$, while in C_2 we have that W implements $v_1.X_2' + v_2.X_2$. These two expressions are clearly equal since the two terms comprising them cannot be 1 at the same time. One can prove that the use of normalized functions solves the problem of false negatives in the general case as well. This is based on the fact that the range of a vector of Boolean functions is identical to the range of the corresponding vector of normalized functions. A rigorous proof of this claim appears in Appendix A.

2.2 Cut-Point Algorithm Implementation

For the comparison of the functions implemented by nodes n_s and n_i of the specification (spec) and the implementation (imp) models, we set the cut-point frontier to be initially the primary inputs of the cones that are driving the two signals. Then repeatedly we attempt to identify more cut-points lying ahead of the current cut-point frontier closer to n_s and n_i with the hope that eventually we will build the BDDs for these two nodes, so that we can compare them for equivalence. These BDDs are built not by assigning free variables for the cut-points on the present frontier but by assigning to every cut-point its corresponding normalized function.

As we see, our CLEVER cut-point based algorithm departs from the classical approach by **combining** the *cp-identification* and *false negative* reduction phases. The main benefit with respect to pre-existing algorithms is that

CLEVER correctly identifies the root cause of false negatives and tries to avoid creating them, so that it does not perform expensive operations to correct them. Furthermore, employing normalized functions allows us to correctly identify all internal signals of equal functionality that exist within the cones of the signals being compared. False negative elimination is usually not done for internal cut-points, and previous algorithms fail to identify every pair of sub-circuits with identical functionality. Thus, the algorithm presented here has the opportunity to work on smaller verification sub-problems, since it identifies all possible cut-points. In addition we do not have to perform any circuit transformations to increase the similarity of the circuit graphs.

One additional area where the presented approach with the use of normalized functions is fundamentally different from previous techniques is the generation of counter examples and the debugging of faulty circuits. In methods like the ones in [4], [5], [8], [9], when the outputs are not proven equal based on the last frontier, one does not know whether this is due to a false negative or a real circuit bug. Here, we do not have to perform a false negative elimination step, since we know that the difference of the outputs must always be due to the presence of a bug. In contrast to other algorithms that require the resubstitution of the cut-point variables by their driving functions, when we employ normalized functions there exists an efficient and simple algorithm that does not require large amounts of memory.

The validity of this algorithm is based again on the theory of Appendix A, where we show that the range of a function is identical to the range of its normalized version. Intuitively, the counter-example that we can produce for the outputs based on the signals of the last frontier will be in terms of values of eigenvariables and reconverging primary inputs. The goal is to use these values of eigenvariables and reconverging inputs to compute the corresponding values of the non-reconverging primary inputs. These must be computed to be compatible with the internal signal values implied by the cut-point assignment that was selected to expose the difference of the outputs.

Finally, one additional area where our cut-point based techniques can be contrasted with pre-existing approaches is the area of **input assumption** handling. This topic is not usually treated in publications of cut-point based algorithms. However, in our experience logic models of modern designs contain many assumptions on input signal behavior, without which the task of formal equivalence verification is impossible. In the case of our cut-point based algorithms we employ *parametric representations* to encode input assumptions as described in [10]. Normalized functions are ideally suited to capture the effect of boolean constraints on the inputs. This is the case because the validity of our algorithm still holds if it is invoked on *parametric variables* encoding the inputs of a circuit rather than the actual inputs themselves. One could even argue that the normalized functions are a parametric representation of the function driving a cut-point in terms of its eigenvariable and its reconverging input signals.

3 Results

The algorithms that are presented in this paper were developed to enable the equivalence verification of a set of difficult to verify signals from a next-generation microprocessor design. The results of the application of CLEVER on these complex circuit cones appear in Table 1. Note that this table lists results for the RTL to schematic netlist comparison. In Table 1 the numbers of the problematic signals for each circuit appear in the **Prb** column. The term problematic here refers to signals whose comparison was not possible by means of monolithic BDDs even though all possible ordering heuristics were exhausted (static and dynamic ordering). The column **IPs** lists the average number of inputs per signal cone. The next six columns of the table are partitioned into two sections, one for the SPEC model (the RTL) and one for the IMP (the transistor netlist). The **Comp** column refers to the average size in composite gates of the cones of the problematic signals. The **CP%** column lists the percentage of nodes in a model that are found by the classic cut-point algorithm to have nodes of identical functionality in the other model. Similarly, the **NCP%** column lists the percentage of nodes in a model that are found by the cut-point algorithm with normalized functions to have corresponding nodes of identical functionality in the other model. Finally, the **CP** and **NCP** columns list how many signal comparisons were completed by the classic cut-point algorithm with resubstitution and the cut-point algorithm with normalized functions, respectively.

Table 1. Statistics about the logic cones driving various problematic signals.

			SPEC (RTL)			IMP (Netlist)				
Ckt	Prb	IPs	Comp	CP%	NCP%	Comp	CP%	NCP%	CP	NCP
C_1	388	343	579	60%	74%	378	27%	61%	272	388
C_2	400	226	365	57%	66%	221	26%	56%	352	400
C_3	8	212	980	25%	69%	570	24%	39%	0	8
C_4	96	130	1040	72%	84%	410	48%	51%	76	96
C_5	15	260	810	20%	60%	650	25%	45%	0	15

One important detail becomes immediately evident from Table 1. The use of normalized functions in our cut-points techniques helps us identify a higher number of cut-points than the classic algorithm. This is the case because our cut-point based techniques do not produce false negatives. Clearly we identify more cut-points in the RTL model, but the difference becomes much more dramatic in the case of the logic model for the transistor netlist. This is to be expected because the logic model for the transistor netlist is more compact, since it is coming from the minimized model for the circuit implementation and as a result has many nodes with reconverging fanout. These nodes cause the classic cut-point algorithm to produce many false-negatives and fail to correctly identify

many cut-points. As a result, the cut-point algorithm with normalized functions manages to complete the verification of approximately 200 more signals out of 900. In addition the C_4 netlist contained 14 output signals which initially were not equivalent with their specification models. These signals were debugged using Algorithm 2, since the classic algorithm with BDD resubstitution could not handle them.

The plot in Fig. 3 indicates the time requirements (in cpu sec) for the signal comparisons on HP Unix workstations with 512MB of RAM. The horizontal axis corresponds to the time it takes the classic cut-point algorithm (CP) for the comparisons of the signals in Table 1. The vertical axis corresponds to the time it takes the cut-points algorithm with normalized functions (NCP) for the same comparisons. The 200 signal comparisons for which the CP algorithm timed out are arbitrarily placed at the bottom right of the plot only to indicate the time it took the NCP algorithm to finish them. The diagonal line partitions the plot into two areas indicating which algorithm performed better.

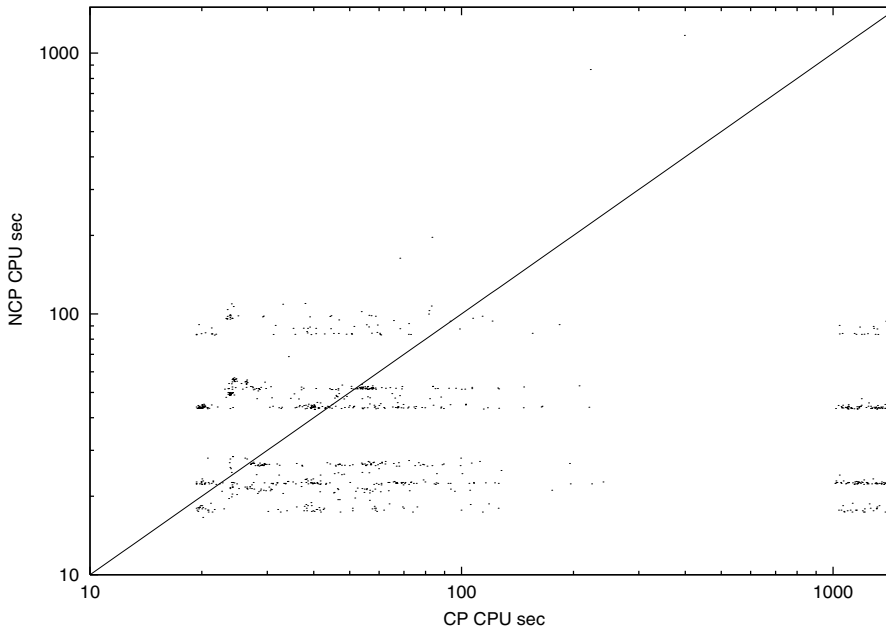

Fig. 3. Time Comparison of Cut-Points with Resubstitution and Cut-Points with
Normalized Functions.

Figure 4 shows the memory requirements (in MB) for the signal comparisons from Table 1. The dashed lines correspond to the classic cut-points algorithm and the dots to the one with normalized functions. Signals are sorted accord-

ing to increasing verification time and we can identify the point where memory requirements become exponential for the classic cut-points algorithm. The key observation here is that cut-point techniques with normalized functions require constant amounts of memory for a larger number of signals. This happens because normalized functions detect every possible cut-point, thus resulting in smaller verification problems and more controlled memory requirements. The second observation is that the classic cut-point algorithm with BDD resubstitution requires more memory. This is happening for two reasons. First, it may not produce a cut-point frontier close to the output because of failing to identify cut-points due to false negatives. So it would create bigger BDDs for the circuit output. The second reason is that if the outputs could not be proven equal, the classic cut-point algorithm needs to perform BDD resubstitution. This is necessary to determine whether the signal in-equivalence is real or a false negative. As a result the memory requirements for the BDDs that get created are increased.

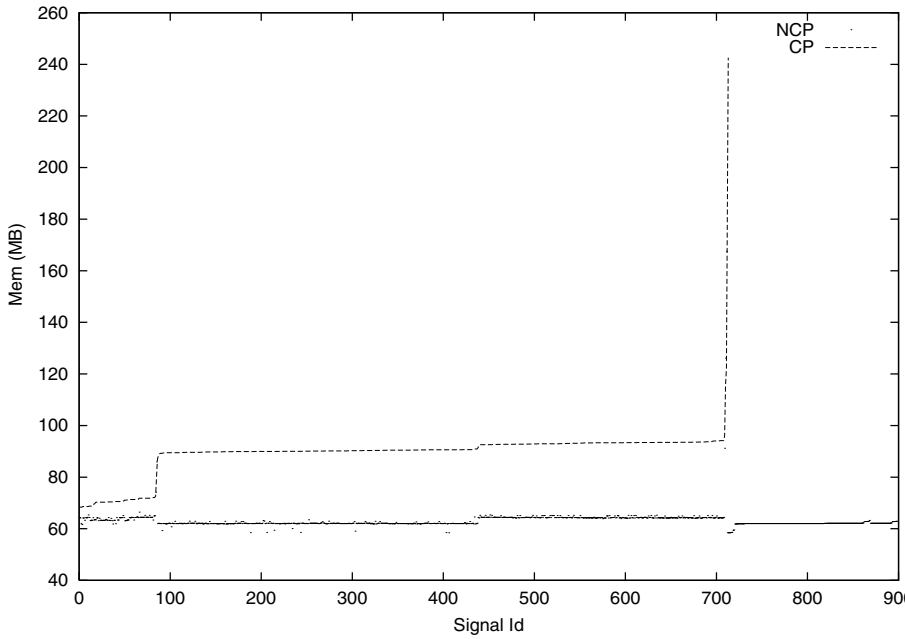

Fig. 4. Memory Consumption Comparison between Cut-Points and Cut-Points with Normalized Functions.

4 Summary

CLEVER is a tool for the formal equivalence verification of next generation microprocessor designs and employs a number of different engines for the ver-

ification of combinational circuits. Among them, one is based on BDD technology using the state of the art algorithms for monolithic BDDs. As it is well known monolithic BDDs suffer from certain limitations. For this reason CLEVER employs circuit divide and conquer techniques based on BDDs. In this paper, we have presented the main idea behind the divide and conquer algorithm in CLEVER. This algorithm is based on the concept of function normalization to provide an efficient means for avoiding the "false negatives problem" that appears in other combinational verification techniques based on circuit partitioning. In addition function normalization readily lends itself to simple counter-example generation and comprehensive handling of input assumptions. As a result, we are able to apply divide and conquer techniques for the comparison of complicated combinational signals, even in cases where the degree of similarity between the circuit models is limited between 20% to 30%.

References

1. R.E. Bryant: *Graph-based algorithms for boolean function manipulation*, IEEE Transactions on CAD, 1986
2. M. Fujita, H. Fujisawa, and N. Kawato: *Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagrams*, ICCAD 1988
3. R. Rudell: *Dynamic variable ordering for ordered binary decision diagrams*, ICCAD 1993, pp. 2-5
4. Y. Matsunaga: *An efficient equivalence checker for combinatorial circuits*, DAC 1996, pp. 629-634
5. S.M. Reddy, W. Kunz, and D.K. Pradhan: *Novel verification framework combining structural and OBDD methods in a synthesis environment*, DAC 1995, pp. 414-419
6. M. Abramovici , M.A. Breuer, and A.D. Friedman: *Digital Systems Testing and Testable Design*, Computer Science Press, 1990
7. T. Nakaoka, S. Wakabayashi, T. Koide, and N. Yoshida: *A Verification Algorithm for Logic Circuits with Internal Variables*, ISCAS 1995, pp. 1920-1923
8. A. Kuhlmann and F. Krohm: *Equivalence Checking using Cuts and Heaps*, DAC 1997
9. Pixley, et al. *Method for determining functional equivalence between design models* United States Patent 5754454, May 19, 1998
10. M.D. Aagaard, R.B. Jones, C.H. Seger: *Formal Verification Using Parametric Representations of Boolean Constraints*, DAC 1999, pp. 402-407.

Appendix A: Proofs

Let $B = \{0, 1\}$ and R be a subset of B^k. Now R contains vectors of the form $\boldsymbol{s} =< s_1, s_2, \ldots, s_k >$. Traditionally, such a set is represented by its characteristic function $\Re(\boldsymbol{s}) = \Re(s_1, s_2, \ldots, s_k)$ which becomes 1 iff $\boldsymbol{s} \in R$. If we consider the variables s_i to model signal values of a logic circuit model, we can view any set $\Re(\boldsymbol{s}) = \Re(s_1, s_2, \ldots, s_k)$ as a signal relation. If there is actually no relation between the signals, then $\Re(\boldsymbol{s}) \equiv 1$. Also, let $\boldsymbol{G}(\boldsymbol{s}) =< g_1(\boldsymbol{s}), g_2(\boldsymbol{s}), \ldots, g_k(\boldsymbol{s}) >$ be a Vector Boolean Function, where $\boldsymbol{s} =< s_1, s_2, \ldots, s_m >$ are the function inputs. Each $g_i(\boldsymbol{s})$ can be written as $g_i(\boldsymbol{r_i}, \boldsymbol{n_i})$, where:

- $\boldsymbol{r_i}$ are the variables on which some other $g_j, j \neq i$ depends. These will be called the re-converging variables.
- $\boldsymbol{n_i}$ are the variables on which no other $g_j, j \neq i$ depends. These will be called the non-reconverging variables.

Now let us re-introduce the concepts of Forced and Possible Terms of Boolean Vector Functions. Let $\{\boldsymbol{x}\}$ stand for all possible value combinations of $\boldsymbol{x} =< x_1, x_2, \ldots, x_m >$ in $B^m = \{0,1\}^m$. There are 2^m such combinations.

For $g_i(\boldsymbol{r_i}, \boldsymbol{n_i})$ we define its Possible Term P_{g_i} as:

$$P_{g_i} = P_{g_i}(\boldsymbol{r_i}) = \exists \boldsymbol{n_i}.g_i(\boldsymbol{r_i}, \boldsymbol{n_i}) \tag{2}$$

and its Forced Term F_{g_i} as:

$$F_{g_i} = F_{g_i}(\boldsymbol{r_i}) = \forall \boldsymbol{n_i}.g_i(\boldsymbol{r_i}, \boldsymbol{n_i}) \tag{3}$$

The following lemmas follow directly from the properties of existential and universal quantification.

Lemma 1.

$$P_{g_i}(\boldsymbol{r_i}) = 0 \Rightarrow F_{g_i}(\boldsymbol{r_i}) = 0$$

Lemma 2.

$$F_{g_i}(\boldsymbol{r_i}) = 1 \Rightarrow P_{g_i}(\boldsymbol{r_i}) = 1$$

Also, let $\partial \boldsymbol{G} =< \partial g_1, \partial g_2, \ldots, \partial g_k >$ stand for the Normalized Function of $\boldsymbol{G}$, where $\boldsymbol{G}$ is a boolean vector function. More specifically, let us define $\partial \boldsymbol{G}$ as

$$< \partial g_1(\boldsymbol{r_i}, \boldsymbol{n_i}), \partial g_2(\boldsymbol{r_i}, \boldsymbol{n_i}), \ldots, \partial g_k(\boldsymbol{r_i}, \boldsymbol{n_i}) >$$

where

$$\partial g_i(\boldsymbol{r_i}, \boldsymbol{n_i}) = v_i.P_{g_i}(\boldsymbol{r_i}) + F_{g_i}(\boldsymbol{r_i})$$

The variable v_i is a free Boolean variable, and is called the eigenvariable of g_i.

Also, let $[\boldsymbol{G}]$ stand for the Range of the vector Boolean Function $\boldsymbol{G}(\boldsymbol{s}) =< g_1(\boldsymbol{s}), g_2(\boldsymbol{s}), \ldots, g_k(\boldsymbol{s}) >$. Then $[\boldsymbol{G}]$ is defined as:

$$[\boldsymbol{G}] = \{\boldsymbol{b} \in B^k | \exists \boldsymbol{s} : \boldsymbol{b} =< g_1(\boldsymbol{s}), g_2(\boldsymbol{s}), \ldots, g_k(\boldsymbol{s}) >\}$$

where $\boldsymbol{s} =< s_1, s_2, \ldots, s_m > \in B^m$, and $\boldsymbol{b} =< b_1, b_2, \ldots, b_k > \in B^k$.

The main result of our algorithm is captured in the following theorem.

Theorem 1. *The Range of a function $\boldsymbol{G}(\boldsymbol{s})$ is identical to the range of its normalized function $\partial \boldsymbol{G}(\boldsymbol{s})$,*

$$[\boldsymbol{G}(\boldsymbol{s})] \equiv [\partial \boldsymbol{G}(\boldsymbol{s})]$$

We repeat that ∂g_i is a function of the eigenvariable v_i of g_i and its reconverging variables $\boldsymbol{r_i}$, i.e. $\partial g_i = \partial g_i(v_i, \boldsymbol{r_i})$.

Similarly, we can say $\partial \boldsymbol{G} = \partial \boldsymbol{G}(\boldsymbol{v}, \boldsymbol{r})$ where $\boldsymbol{v} =< v_1, v_2, \ldots, v_k >$ the eigenvariables of $g_1, g_2, \ldots, g_k$, and
$\boldsymbol{r} =< \boldsymbol{r_1}, \boldsymbol{r_2}, \ldots, \boldsymbol{r_k} >$ are the reconverging variables of $g_1, g_2, \ldots, g_k$, respectively.

Keep in mind $\boldsymbol{G}$ has k functions of at most m variables each.

We will attempt now to prove Theorem 1.

Proof: Initially we will prove that $[\boldsymbol{G}] \subseteq [\partial \boldsymbol{G}]$.

Let $\boldsymbol{b} =< b_1, b_2, \ldots, b_k >\in [\boldsymbol{G}]$, where $b_i \in \{0, 1\}$.
$\Rightarrow \exists \boldsymbol{s'} = (s'_1, s'_2, \ldots, s'_m) : b_i = g_i(s'_1, s'_2, \ldots, s'_m), i = 1, \ldots, k$
$\Rightarrow \exists \boldsymbol{r'_i}, \boldsymbol{n'_i}$ derived from $\boldsymbol{s'} : b_i = g_i(\boldsymbol{r'_i}, \boldsymbol{n'_i}), i = 1, \ldots, k$.

Now, if $b_i = 1 \Rightarrow g_i(\boldsymbol{r'_i}, \boldsymbol{n'_i}) = 1 \Rightarrow P_{g_i}(\boldsymbol{r'_i}) = 1$ according to formula 2. If we select the eigenvariable value $v'_i = 1$ we get that $\partial g_i(v'_i, \boldsymbol{r'_i}) = v'_i.P_{g_i}(\boldsymbol{r'_i}) + F_{g_i}(\boldsymbol{r'_i}) = 1.1 + F_{g_i}(\boldsymbol{r'_i}) = 1 = b_i$

On the other hand, if $b_i = 0 \Rightarrow g_i(\boldsymbol{r'_i}, \boldsymbol{n'_i}) = 0 \Rightarrow F_{g_i}(\boldsymbol{r'_i}) = 0$ according to formula 3. If we select the eigenvariable value $v'_i = 0$ we get that $\partial g_i(v'_i, \boldsymbol{r'_i}) = v'_i.P_{g_i}(\boldsymbol{r'_i}) + F_{g_i}(\boldsymbol{r'_i}) = 0.P_{g_i}(\boldsymbol{r'_i}) + 0 = 0 = b_i$

So, for the bit pattern $\boldsymbol{b} =< b_1, b_2, \ldots, b_k >\in [\boldsymbol{G}]$, we have created a pattern $< \partial g_i(v'_1, \boldsymbol{r'_1} >, \ldots, \partial g_i(v'_k, \boldsymbol{r'_k}) >\in [\partial \boldsymbol{G}]$, which is identical to $\boldsymbol{b}$. So, $\boldsymbol{b} \in [\partial \boldsymbol{G}]$ and $[\boldsymbol{G}] \subseteq [\partial \boldsymbol{G}]$.

To complete the proof of Theorem 1, we also need to prove: $[\partial \boldsymbol{G}] \subseteq [\boldsymbol{G}]$

To prove this, let $\boldsymbol{b} =< b_1, b_2, \ldots, b_k >\in [\partial \boldsymbol{G}]$.
$\Rightarrow \exists (\boldsymbol{v'}, \boldsymbol{r'}) : \boldsymbol{b} = \partial \boldsymbol{G}(\boldsymbol{v'}, \boldsymbol{r'})$
$\Rightarrow \exists v'_i, \boldsymbol{r'_i} : b_i = \partial g_i(v'_i, \boldsymbol{r'_i}) = v'_i.P_{g_i}(\boldsymbol{r'_i}) + F_{g_i}(\boldsymbol{r'_i}), i = 1, \ldots, k.$

Now, in case $b_i = 0$. Then $b_i = \partial g_i(v'_i, \boldsymbol{r'_i}) = 0 \Rightarrow F_{g_i}(\boldsymbol{r'_i}) = 0$
$\Rightarrow \forall \boldsymbol{n_i}.g_i(\boldsymbol{r'_i}, \boldsymbol{n_i}) = 0 \Rightarrow \exists \boldsymbol{n'_i} : g_i(\boldsymbol{r'_i}, \boldsymbol{n'_i}) = 0$
$\Rightarrow \exists \boldsymbol{n'_i} : g_i(\boldsymbol{r'_i}, \boldsymbol{n'_i}) = 0 = b_i$

On the other hand, if $b_i = 1 \Rightarrow P_{g_i}(\boldsymbol{r'_i}) = 1$. Otherwise, if $P_{g_i}(\boldsymbol{r_i}) = 0 \Rightarrow F_{g_i}(\boldsymbol{r_i}) = 0$ according to the lemmas presented previously. This would make $v_i.P_{g_i} + F_{g_i} \equiv 0$ while we assumed it's a 1.

But, if $P_{g_i}(\boldsymbol{r'_i}) = 1 \Rightarrow \exists \boldsymbol{n_i}.g_i(\boldsymbol{r'_i}, \boldsymbol{n'_i}) = 1 \Rightarrow \exists \boldsymbol{n'_i} : g_i(\boldsymbol{r'_i}, \boldsymbol{n'_i}) = 1 = b_i$.

So, for $\boldsymbol{b} =< b_1, b_2, \ldots, b_k >\in [\partial \boldsymbol{G}]$, we can construct another bit pattern $< g_i(\boldsymbol{r'_1}, \boldsymbol{n'_1}), \ldots, g_i(\boldsymbol{r'_k}, \boldsymbol{n'_k}) >\equiv< b_1, b_2, \ldots, b_k >$, which $\in [\boldsymbol{G}]$. Therefore, $[\partial \boldsymbol{G}] \subseteq [\boldsymbol{G}]$, which completes our proof. □

To establish the false-negative elimination claim, consider now the function G that is computed by forming the exclusive-OR of every pair of outputs from the two circuits to be compared. Since the range of $\partial \boldsymbol{G}$ is equal to the range of $\boldsymbol{G}$, our claim follows trivially.