Techniques for Higher-Performance Boolean Equivalence Verification

Harry D. Foster

The techniques and algorithms presented in this paper are a result of six years' experience in researching, developing, and integrating Boolean equivalence verification into the HP Convex Division's ASIC design flow. We have discovered that a high-performance equivalence checker is attainable through careful memory management, the use of bus grouping techniques during the RTL-to-equation translation process, hierarchical to flat name mapping considerations, subequivalence point cone partitioning, solving the false negative verification problem, and building minimal binary decision diagrams.

In 1965 Gordon Moore observed that the complexity and density of the silicon chip had doubled every year since its introduction, and accompanying this cycle was a proportional reduction in cost. He then boldly predicted—what is now referred to as Moore's Law—that this technology trend would continue. The period for this cycle was later revised to 18 months. Yet the performance of simulators, the main process for verifying integrated circuit design, has not kept pace with this silicon density trend. In fact, as transistor counts continue to increase along the Moore's Law curve, and as the design process transitions from a higher-level RTL (Register Transfer Language) description to its gate-level implementation, simulation performance quickly becomes the major bottleneck in the design flow.

In 1990, the HP Convex Division was designing high performance systems that used ASICs with gate counts on the order of 100,000 raw gates. During this period the HP Convex Division used a third-party simulator for both RTL and gate-level verification. This simulator had the distinction of being the fastest RTL simulator on the market, but it also suffered the misfortune of being one of the market's slowest gate-level simulators in our environment. Hence,

Harry D. Foster

Harry Foster joined the HP Convex Division in 1989 after receiving his MSCS degree from the University of Texas. An engineer/scientist, he has been responsible for CAD tool research and development, formal verification, resynthesis, and a clock tree builder. Before coming to HP, he worked at Texas Instruments. He was born in Bethesda, Maryland, and likes traveling, weight training, and inline skating.

running gate-level simulations quickly became a major bottleneck in the HP Convex Division system design flow. In addition, these regression simulations could not completely guarantee equivalence between the final gate-level implementation and its original RTL specification. This resulted in a lack of confidence in the final design.

To address these problems, the HP Convex Division began researching alternative methods to regression simulation, resulting in a high-performance equivalence checker known as lover (LOgic VERifier).

Today, the HP Convex Division is delivering high-performance systems built with 0.35-μm CMOS ASICs having on the order of 1.1 million raw gates.[1] lover has successfully kept pace with today's silicon density growth, and has completely eliminated the need for gate-level regression simulations. Our very large system-level simulations are now performed entirely at the faster RTL level when combining the various ASIC models. This has been made possible by incorporating our high-performance equivalence checker into our design flow. We now have confidence that our gate-level implementation completely matches its RTL description.

Boolean Equivalence Requirements

Our experience has been that last-minute hand tweaks in the final place-and-route netlist require a quick and simple verification process that can handle a complete chip RTL-to-gate comparison. Such hand tweaks, and all hand-generated gates, are where most logic errors are inadvertently inserted into the design. The following list of requirements drove the development of the HP Convex Division's high-performance equivalence checker:

- Must support RTL-to-RTL (flat and hierarchical) comparisons during the early development phase.
- Must support both synthesizable and nonsynthesizable Verilog RTL constructs for RTL-to-gate comparisons.
- Must support a simple one-step process for comparison of the complete chip design (RTL-to-gate, gate-to-gate, hierarchical-to-flat).
- Must support the same Verilog constructs and policies defined for the entire HP Convex Division design flow (from our cycle-based simulator to our place-and-route tools), along with standard Verilog libraries.

Boolean Equivalence Verification

Boolean equivalence verification is a technique of mathematically proving that two design models are logically equivalent (e.g., a hierarchical RTL description and a flat gate-level netlist). This is accomplished by the following steps:

1. Compile the two designs. Convert a higher-level Verilog RTL specification into a set of lower-level equations and state points, which are represented in some internal data structure format. For a structural or gate-level implementation, the process includes resolving instance references before generating equations.

2. Identify equivalence points. Identify a set of controllability and observability cross-design relationships. These relationships are referred to as equivalence points, and at a minimum consist of primary inputs, primary outputs, and register or state boundaries.

3. Verify equivalence. Verify the logical equivalence between each pair of observability points by evaluating the following *equivalence equation*:

$$\neg\left(m_1(x_i) \oplus m_2(x_i)\right) \vee \left(x_i \cap D_e(x)\right) = 1. \qquad (1)$$

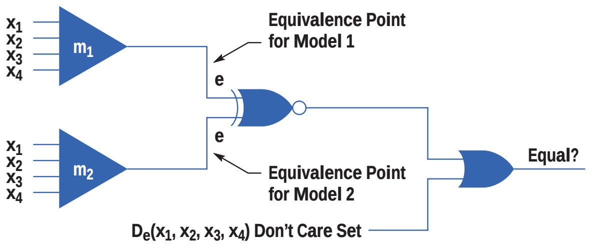
In the equivalence equation (equation 1), $\neg$ is the propositional logic NOT or negation operator, $\vee$ is the propositional logic OR or disjunction operator, $\oplus$ is the Boolean XOR operator, $\cap$ is the set intersection operator, m_1 represents the logic expression (or cone of logic*) for an observability equivalence point found in design model 1, and m_2 is the expression for the corresponding point found in design model 2. The variables x_i are the cone's input or controllability equivalence points for both models' logic expressions. Finally, $D_e(x)$ is known as the *don't care set* for the equivalence point e, and consists of all possible values of x for which the logical expressions m_1 and m_2 do not have to match. **Figure 1** graphically illustrates the process of proving equivalence between two observability equivalence points.

Another way to view **Figure 1** is to observe that if a combination of x_i can be found that results in $m_1(x_i)$ evaluating to a different value than $m_2(x_i)$, and x_i is not contained within the don't care set $D_e(x)$, then the two models are

* A cone of logic is the set of gates or subexpressions that fan into a single point, either a register or an equation variable.

Figure 1

Proving equivalence.


x_1
x_2
x_3
x_4
m_1
Equivalence Point for Model 1
e
e
x_1
x_2
x_3
x_4
m_2
Equivalence Point for Model 2
Equal?
$D_e(x_1, x_2, x_3, x_4)$ Don't Care Set

formally proved different by the following *nonequivalence equation*:

$$\left(m_1(x_i) \oplus m_2(x_i)\right) \wedge \neg\left(x_i \cap D_e(x)\right) = 1, \qquad (2)$$

where $\wedge$ is the propositional logic AND or conjunction operator. Finding an x_i that will satisfy the nonequivalence equation, equation 2, is NP-complete.* In general, determining equivalence between two Boolean expressions is NP-complete. This means, as Bryant[2] points out, that a solution's time complexity (in the worst case) will grow exponentially with the size of the problem.

Instead of trying to determine inequality by finding a value for x that satisfies the nonequivalence equation, a better solution is to determine the equality of m_1 and m_2 through a specific or unique symbolic or graphical representation, such as an *ordered-reduced binary decision diagram* (OBDD).

Ordered-Reduced Binary Decision Diagrams

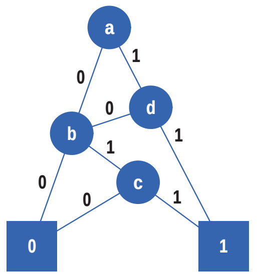
An ordered-reduced binary decision diagram (OBDD) is a directed acyclic graph representation of a Boolean function.[2] The unique characteristic of an OBDD is that it is a canonical-form representation of a Boolean function, which means that the two equations m_1 and m_2 in our previous example will have exactly the same OBDD representation when they are equivalent. This is always true if a common ordering of the controllability equivalence points is used to construct the OBDDs for m_1 and m_2.

Choosing an equivalence point input ordering can, in some cases, influence the resulting size of the OBDD. In addition, finding an optimal ordering of equivalence

* An NP-complete or co-NP-complete problem is a relatively intractable problem requiring an exponential time for its solution. See reference 3.

Figure 2

OBDD (ordered-reduced binary decision diagram) with good ordering.



(a & d) | (b & c)

points in an attempt to minimize an OBDD is itself a co-NP-complete problem.[2,3] However, for most cases it is only necessary to find a good ordering, or simply one that works. Techniques for minimizing the size of an OBDD will be described later in this paper.

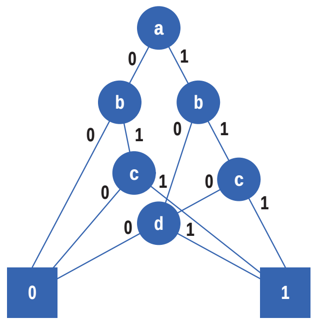
Figures 2 and 3 show two examples of an OBDD for the Boolean function (a & d) | (b & c), where & is the Boolean AND operator and | is the Boolean OR operator.

Performance Techniques

This section provides details and techniques for achieving high performance in the Boolean equivalence verification process. Some of the techniques can be incorporated into a user's existing design flow to achieve higher performance from a commercial equivalence checker.

Figure 3

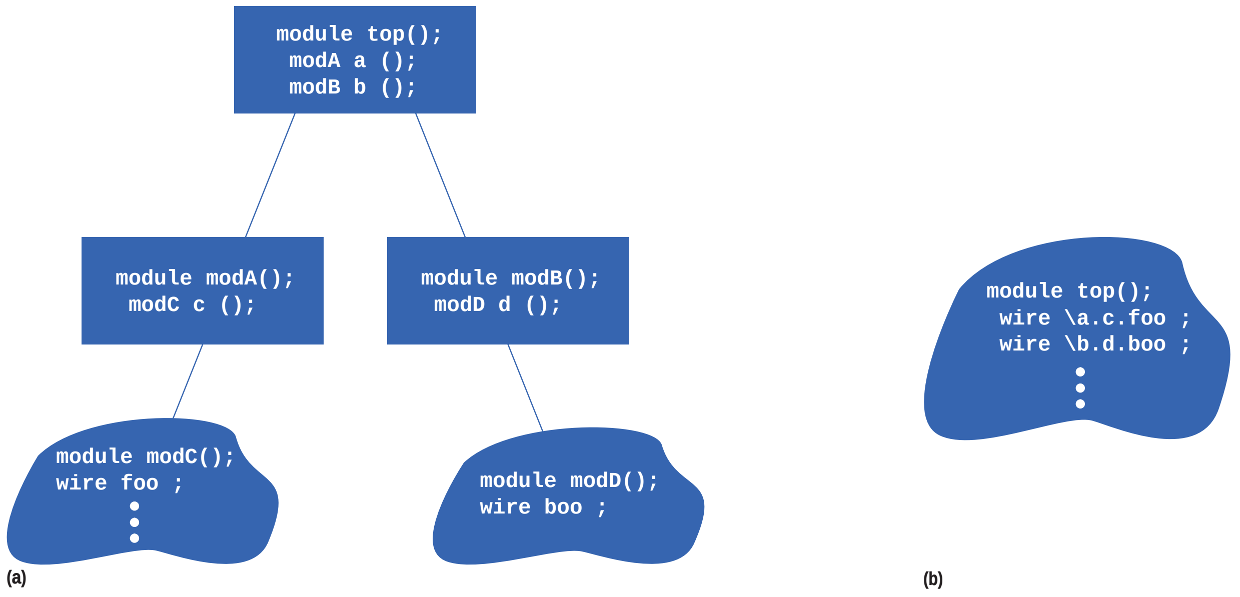
OBDD with not-so-good ordering.



(a & d) | (b & c)

Figure 4

Hierarchical to flat name mapping. (a) Hierarchical RTL and gates. (b) Flat gate description.



Don't Throw Away Useful Information. When establishing an ASIC design flow, it is a benefit to view the entire flow globally—not just the process of piecing together various CAD tools (simulators, equivalence checkers, place-and-route tools, etc.). Why throw out valuable information from one process in the design flow and force another process to reconstruct it at a significant cost in performance?

At the HP Convex Division, we've designed our flow such that the identification of registers and primary inputs and outputs is consistent across the entire flow. The same hierarchical point in a Verilog cycle-based simulation run can be referenced in a flat place-and-route Verilog netlist without having to derive these common points computationally.

Name Mapping. lover will map the standard cross-design pairs of controllability and observability equivalence points directly as a result of the the HP Convex Division flow's naming convention. **Figure 4** helps illustrate how name mapping can be preserved between a hierarchical RTL Verilog tree of modules and a single flat gate-level description.

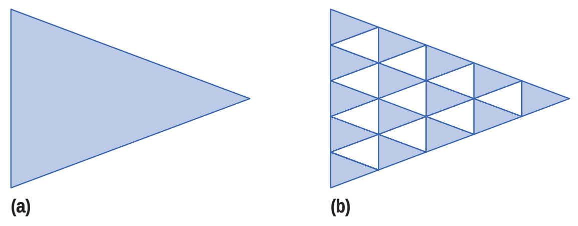
To support non-name-based mapping of equivalence points, that is, designs that violate the HP Convex Division flow naming conventions, lover will accept equivalence mapping files containing the two designs' net pair relationships. In addition, the user can provide a special filter function, which will automate the process of resolving cross-design name mapping for special cases.

For example, let f_1 be a special mapping function for design model 1 and f_2 be a special mapping function for design model 2. Then an equivalence point will be established whenever $f_1(\xi) = f_2(\zeta)$. This allows the two models' Verilog wire and register names (or strings ξ and ζ) to differ, but resolve to the same equivalence point through their special mapping functions.

Cone Partitioning. A technique known as *cone partitioning* is used to minimize the size of the OBDDs built during the verification process, since smaller OBDDs require significantly less processing time and consume much less memory than larger ones. Cone partitioning is the process of taking a large cone of logic and dividing it into a set of smaller sized cones. **Figure 5** helps illustrate this concept.

Figure 5

Cone partitioning.(a) Large cone of logic. (b) Set of partitioned cones.



The two designs verified by lover are stored internally in a compact and highly efficient net/primitive relational data structure. OBDDs are built from the relational data structure only on demand for the specific partitioned cone of logic being proved, and then immediately freed after their use. This eliminates the need to optimize the specific cone's OBDD into the entire set of OBDDs for a design. In addition to achieving higher performance during the verification process, this ensures that any differences found between the partitioned cones tends to be isolated down to either a handful of gates or a few lines of RTL code. This greatly simplifies the engineer's debug effort.

Another advantage of cone partitioning is that it becomes unnecessary to spend processing time minimizing equations for large cones of logic, since they are automatically decomposed into a set of smaller and simpler cones.

In general, cones of logic are bounded by *equivalence points*, which consist of registers and ASIC input and output ports. However, lover takes advantage of a set of pairs of internally cross-design equivalent relationships (e.g., nets or subexpressions), which we refer to as *subequivalence points*, to partition large cones. Numerous methods have been developed to compute a set of cross-circuit subequivalence points based on the structural information or modular interfaces.[4] Costlier ATPG (automatic test pattern generator) techniques are commonly used to identify and map subequivalence points between designs lacking a consistent naming convention. lover determines subequivalence points directly without performing any intensive computations by taking advantage of the consistent name mapping convention built into the the HP Convex Division design flow. Numerous subequivalence points are derived directly from the module's hierarchical boundaries.

In addition, lover attempts to map the Verilog module's internal wire and register variable names between designs. For example, Synopsys will unroll the RTL Verilog wire and register bus ranges as follows:

```
wire [0:3] foo;
```

will be synthesized into gates with the following unrolled names:

```
wire \foo[0], \foo[1], \foo[2], \foo[3];
```

lover recognizes Synopsys' unrolled naming convention and will map these points back to the original RTL description. This results in a significantly better cone partition than limiting the subequivalence points to only the structural or module interface.

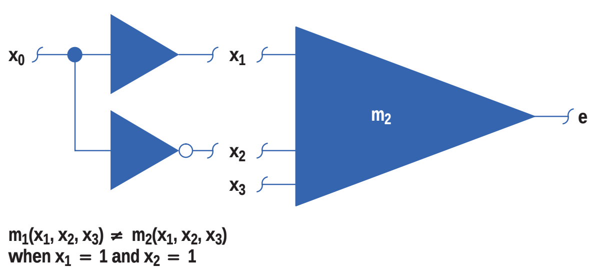
The performance gains achieved through cone partitioning are highly variable and dependent on a circuit's topology. Some modules' measured performance gains have been on the order of 20 times, while other modules have suffered a performance loss of 1.5 times when applying a maximum cone partition. The performance gains tend to increase as the proof moves higher up the hierarchy of modules (due to larger cones). In general, we've observed that cone partitioning contributes to about a 40% increase in performance over the entire chip. More important, cone partitioning allows us to prove certain topologies containing large cones of logic that would be impractical to prove using OBDDs.

OBDD Input Ordering. Cone partitioning has the additional advantage of simplifying the ordering of OBDDs by reducing the size of the problem. lover uses a simple topological search through these smaller partitioned cones, which yields an excellent variable ordering for most cases. This search method was first proposed by Fujita,[5] and consists of a simple depth-first search through a circuit's various logic levels, starting at its output and working backwards towards its controllability equivalence points. The equivalence points encountered first in the search are placed at the beginning of the OBDD variable ordering.

Solving the False Negative Problem. Cone partitioning techniques used to derive cross-circuit subequivalence points can lead to a proof condition known as a *false negative*. This quite often forces the equivalence checker into a more aggressive and costlier performance mode to complete the proof. We have developed a method of identifying a false negative condition while still remaining in the faster

Figure 6

False negative.


x_0
x_1
x_2
x_3
m_2
e
$m_1(x_1, x_2, x_3) \neq m_2(x_1, x_2, x_3)$
when $x_1 = 1$ and $x_2 = 1$

name-mapping mode throughout the entire verification process.

One type of false negative can occur when the RTL specifies a don't-care directive to the synthesis tool, and the equivalence checker does not account for the don't care set $D_e(x)$ in equation 1.

Another more troublesome type of false negative can occur when the synthesis process recognizes a don't care optimization opportunity not originally specified in the RTL. This can occur when the synthesis step is applied to a subexpression, which then takes an optimization advantage over the full cone of logic (e.g., generating gates for a subexpression with the knowledge that a specific combination of values on its inputs is not possible).

Figure 6 provides a simple example of this problem. It is possible that the partitioned cone m_2's synthesized logic will be optimized with the knowledge that x_1 and x_2 are always mutually exclusive. This can lead to a false negative proof on the partitioned cones $m_1(x_1, x_2, x_3)$ and $m_2(x_1, x_2, x_3)$ for the impossible case $x_1 = 1$ and $x_2 = 1$.

However, if the subequivalence points that induced the false negative condition are removed from the cone partition boundaries of m_1 and m_2 (e.g., x_1 and x_2), the resulting larger cone partition $m_1'(x_0, x_3)$ is easily proved to be equivalent to $m_2'(x_0, x_3)$. **Figure 7** helps illustrate how the false negative condition can be eliminated by viewing a larger partition of the cone.

There are numerous situations that can induce a false negative condition, and most are much more complex than the simple example provided in **Figure 6**. lover has algorithms built into it that will detect and remove all false negative conditions. These algorithms are invoked only when nonequivalence has been determined between observability points.

The algorithm used to solve a false negative performs a topological or breadth-first search through the levelized cone of logic, removing the first input or controllability equivalence point encountered. It then reproves the resulting larger cone. The following steps describe the algorithm used by lover to remove a false negative condition:

1. Levelize the cone of logic by assigning the observability equivalence point the number 0. Then, step back to the next level of logic in the cone, assigning a logic level number to each logic primitive and net. Continue this process back through all levels of logic until reaching the cone's controllability equivalence points.

2. Remove the lowest level of logic (or numbered) controllability subequivalence point, resulting in a larger cone partition.

3. Relevelize the new large cone of logic.

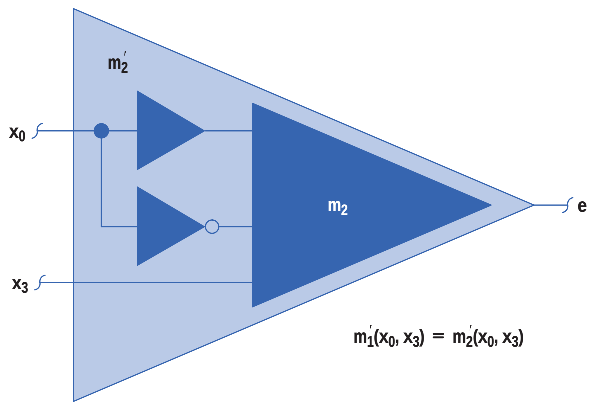
4. Identify and order the new set of input controllability equivalence points.

5. Try reproving the new larger cone partition with the new set of controllability subequivalence points.

6. If the new cone partition is proved nonequivalent and we can continue removing subequivalence points (i.e., we haven't reached a register or ASIC port boundary) go to

Figure 7

Solving the false negative problem.



step 2. Otherwise, we are done. If step 5 is proven, the two cones are equivalent.

Process Memory Considerations. Most high-performance tools require their own memory management utility to reduce the system overhead time normally associated with searching a process's large memory allocation table. lover implements three methods of managing memory: (1) recycle high-use data structure elements during compilation, (2) ensure that memory is unfragmented when building OBDDs (i.e., at the start of each cone's proof), and (3) maintain and manipulate a single grouped structure representation for equations containing buses.

- High-Use Data Structure Recycling. The various data structures (or C typedefs) used during the compilation process can be recycled when it becomes necessary to free them. Recycling is a technique of linking the specific structure types together into a *free list*. Later compilation steps can tap into these lists of high-use data structure elements and not incur any of the system overhead normally associated with allocating or freeing memory. We have observed performance gains in the order of 1.25 times for small designs and up to 2 times for larger designs by using data structure recycling techniques.
- OBDD Unfragmented Memory Management. A block of memory should be reserved for use by the OBDD memory management utilities. Once a proof is complete for a partitioned cone, its memory block can be quickly reset to its original unfragmented state by simply resetting a few pointers. Working with a block of unfragmented memory increases the chances of fitting a cone's OBDD into the system's cache. The performance gains achieved by controlling the fragmentation of memory are significant, but hard to quantify. In general, we have observed that the performance of manipulating OBDDs degrades linearly as memory fragmentation increases.
- Grouping Structures for Verilog Buses. Care should be taken to retain the buses within an equation as a single grouped data structure element for as long as possible. Expanding an equation containing buses into its individual bits too soon will result in a memory explosion during the compilation process and force unnecessary elaboration on the equation's replicated data structures (i.e., process duplication while manipulating the individual bits for a bused expression).

The Don't Care Set. As a final comment on performance techniques, we need to point out that the HP Convex Division design flow has a requirement of simulating the ATPG vector set on the RTL. This helps flush out any ATPG model library problem and provides an additional sanity check and assurance that Boolean equivalence verification was performed on the entire design. An additional benefit of verifying the ATPG vectors at the RTL level is a potential tenfold speedup in simulation performance compared to a gate-level simulation of the vector set.

To support RTL simulation of the ATPG vectors, the Verilog control structures (e.g., case and if) must be fully specified or defaulted to a known value. lover takes advantage of this flow requirement and makes no attempt to gather and process the don't-care set $D_e(x)$ in equation 1. We have developed linting tools within our flow to ensure that these control structures are fully specified. In addition, lover detects violations of this ATPG RTL requirement.

Performance Gains

This section demonstrates the performance gains that can be achieved through techniques described in this paper. The benchmarks for **Tables I** and **II** were performed on an HP S-Class technical server (a 16-way symmetric multiprocessing machine based on the HP PA 8000 processor with 4G bytes of main memory). The single-threaded performance times provided for lover include a composite of the RTL and gate-level model compilation times, as well as the verification step (unlike most commercial tools, compilation and verification are performed within a single process).

Table I describes four recently designed 0.35-μm CMOS ASICs built for Hewlett-Packard's Exemplar S-Class and X-Class technical servers. These times represent a comparison of a full hierarchical RTL Verilog model to its complete-chip flat gate-level netlist.

Table II presents the gate-to-gate run-time performance for the same four ASICs. These times represent a complete chip hierarchical gate-level model (directly out of synthesis) compared to its final hand-tweaked flat place-and-route netlist.

Table I

RTL-to-Gate lover Results

Chip Name	Size (kgates)	Minutes	GBytes
Processor Interface	550	116	1.3
Crossbar	500	26	1.1
Memory Interface	570	68	1.3
Node-to-Node Interface	300	56	1.0

Table II

Gate-to-Gate lover Results

Chip Name	Size (kgates)	Minutes	GBytes
Processor Interface	550	20	1.2
Crossbar	500	9	0.9
Memory Interface	570	20	1.2
Node-to-Node Interface	300	10	0.9

Additional Performance Gains

The Hewlett-Packard Convex Division is in the business of researching and developing symmetric multiprocessing high-performance servers. Historically, most vendors' CAD tools have lagged behind the design requirements for our next-generation systems. To solve the vast and escalating problems encountered during the design of these systems, the HP Convex Division has begun research in the area of parallel CAD solutions.[6] A prototype multithreaded equivalence checker (p-lover) has been developed to investigate the potential performance gains achievable through parallel processing.

The prototype multithreaded equivalence checker is based on lover's single-threaded proof engine. A front-end input compiler and data structure emulation engine was developed to feed the parallel threads with partitioned cones of logic for verification.

Figure 8 shows the speedup factors we obtained with p-lover when launching two, four, and six threads. Note the superlinear performance we were able to achieve with two threads (see reference 6 for a discussion of superlinear behavior). Each thread only contributed a 4% increase in the overall process memory size. This can be attributed to a single program image for the compiled net and primitive relational data structures, which were stored in globally shared memory.

The following is a list of multithreaded tool design considerations we've identified while developing our prototype equivalence checker:

- Long Threads. To reduce the overhead incurred when launching threads, the full set of observability equivalence points needs to be partitioned into similar-sized subsets or lists for each thread (i.e., threads should be launched to prove a list of cones and not a single point). **Figure 9** helps illustrate this idea.
- Thread Balancing. When a thread finishes proving its list of observability equivalence points, the remaining threads should rebalance their list of unproven cones to maintain maximum tool performance.
- Thread Memory Management. Each thread must have its own self-contained memory management and OBDD utility.

Figure 8

Multithreaded performance.

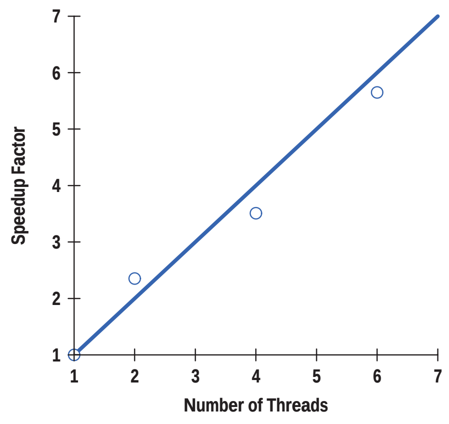
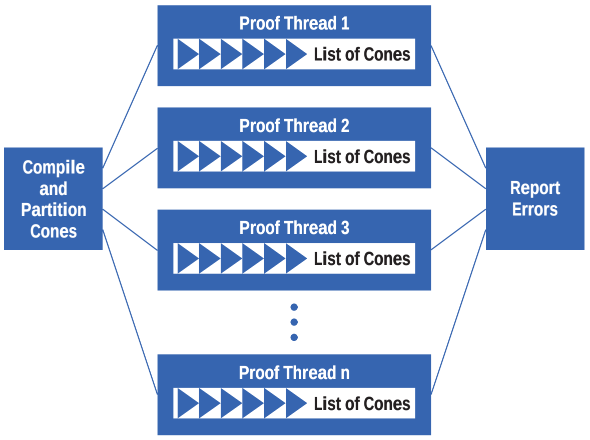


Figure 9

Multithreaded equivalence checker.



- System Resources and Locking. All I/O and logging must be eliminated from the individual launched threads. Otherwise, the locking and unlocking schemes built into the system resource's critical sections will dramatically degrade the tool's performance. Errors can be flagged in internal data structures and reported after all threads have finished processing their individual lists of equivalence points.

A production version of p-lover will require additional research to eliminate some of the locking requirements necessary when addressing globally shared memory. In particular, solving the false negative problem in a multithreaded environment will require some additional thought. However, the potential performance gains obtainable through a multithreaded equivalence checker are attractive.

Conclusion

Boolean equivalence verification, an integral process within the HP Convex Division's ASIC design flow, bridges the verification gap between an ASIC's high-level RTL used for simulation and its place-and-route gate-level netlist. We have presented techniques in this paper that have contributed to the development of a Boolean equivalence checker with performance on the order of 100 times faster than many currently available commercial tools. Even a commercial equivalence checker will benefit substantially if its users understand a few of the techniques we have presented and apply them directly to their design flow (e.g., name mapping, subequivalence points, and cone partitioning concepts). Finally, we have presented data from a prototype multithreaded equivalence checker to illustrate that an even higher performance level is attainable through a parallel solution.

References

1. L. Bening, T. Brewer, H. Foster, J. Quigley, B. Sussman, P. Vogel, and A. Wells, "Physical Design of 0.35-µm Gate Arrays for Symmetric Multiprocessing Servers," *Hewlett-Packard Journal*, Vol. 48, no. 2, April 1997, pp. 95-103.

2. R. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, August 1986, pp. 677-691.

3. M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, 1979.

4. E. Cerny and C. Mauras, "Tautology Checking Using Cross-Controllability and Cross-Observability Relations," *IEEE Transactions on Computers*, January 1990, pp. 34-37.

5. M. Fujita, H. Fujisawa, and N. Kawato, "Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagrams," *International Conference on Computer-Aided Design*, November 1988, pp. 2-5.

6. L. Bening, "Putting Multi-Threaded Simulation To Work," accessible on the World Wide Web at URL:
http://www.hp.com/wsg/tech/supercon96/index.html

▶ Go to Next Article
▶ Go to Journal Home Page