(21)

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: NW-LIS-89-12-06	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): GeminiII: A Second Generation Layout Validation Program		5. TYPE OF REPORT & PERIOD COVERED: Technical
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Carl Ebeling		8. CONTRACT OR GRANT NUMBER(s): N00014-88-K-0453
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Northwest Laboratory for Integrated Systems, University of Washington, Dept. of Comp. Science, FR-35 Seattle, WA 98195		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: DARPA-ISTO, 1400 Wilson Boulevard, Arlington, VA 22209		12. REPORT DATE: December 1989
		13. NUMBER OF PAGES: 4
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office): Office of Naval Research - ONR, Information Systems Program - Code 1513: CAF, 800 North Quincy Street, Arlington, VA 22217		15. SECURITY CLASS. (of this report): Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this report is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Layout, layout validation, graph isomorphism, bipartite graphs, graph partitioning, VLSI circuits.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Gemini is a circuit comparison program that is widely used to compare circuit layout against a specification. In this paper we describe recent extensions made to Gemini that make it faster, enable it to isolate errors better, and extend its domain of application. This has been done by changes to the labeling algorithm, extensions to the local matching algorithm, better handling of symmetrical circuits and the accommodation of series-connected transistors. GeminiII's

Continued on back....

#20 Abstract

(continued from front page)

algorithm is separated into global labeling and local matching phases. GeminiII dynamically switches between the two depending on the amount of local structure contained in the circuit, taking advantage of the speed of the local matching algorithm when possible and relying on the power of the more general algorithms when the simple algorithm fails. This blending of algorithms also allows differences between two circuits to be better contained so that defects can be pinpointed.

GeminiII: A Second Generation Layout Validation Program

Carl Ebeling

Technical Report #89-12-06

Department of Computer Science and Engineering, FR-35
University of Washington, Seattle, WA 98195 USA

GeminiII: A Second Generation Layout Validation Program

Carl Ebeling

Technical Report #89-12-06

Department of Computer Science and Engineering, FR-35
University of Washington, Seattle, WA 98195 USA

A

GeminiII: A Second Generation Layout Validation Program

Carl Ebeling
Department of Computer Science
University of Washington

ABSTRACT

Gemini is a circuit comparison program that is widely used to compare circuit layout against a specification. In this paper we describe recent extensions made to Gemini that make it faster, enable it to isolate errors better, and extend its domain of application. This has been done by changes to the labeling algorithm, extensions to the local matching algorithm, better handling of symmetrical circuits and the accommodation of series-connected transistors. GeminiII's algorithm is separated into global labeling and local matching phases. GeminiII dynamically switches between the two depending on the amount of local structure contained in the circuit, taking advantage of the speed of the local matching algorithm when possible and relying on the power of the more general algorithm when the simple algorithm fails. This blending of algorithms also allows differences between two circuits to be better contained so that defects can be pinpointed.

Introduction

One of the crucial steps in the design of VLSI circuits is that of determining whether the layout of the circuit geometry corresponds to the specification of the circuit. This problem is often addressed by simulating the circuit extracted from the layout. This method is both inefficient and ineffective. Simulation requires many hours for large systems and still is not guaranteed to find all errors. Moreover, subtle differences such as transistor sizing typically go undetected.

Gemini validates layout by comparing two circuits, the specification circuit and the circuit extracted from the layout. Gemini determines whether the two circuits match exactly, and if not, what the differences are. Gemini has been used to validate a wide range of chips, including the HITECH, CHIPTEST and SLAP chips at CMU, and the CRISP microprocessor at Bell Labs. The program is very fast, comparing circuits at the rate of about 300 transistors/second on a Sun-3 workstation. Thus even very large chips with 100,000 transistors take only a few minutes to compare.

The approach that Gemini takes is to treat the circuit comparison problem as an instance of the graph isomorphism problem. The circuits are represented as graphs and a heuristic algorithm based on *node invariants* is used to partition the graphs into groups of devices and nets which have the same characteristics. The partitioning is refined until each device or net is in a separate partition. At this point Gemini can match partitions between graphs to obtain the isomorphism mapping of the elements of one circuit onto the elements of the other.

This basic partitioning algorithm does very well at determining whether the circuits are identical. When circuits are different, however, Gemini attempts to isolate the differences. This is in general a difficult problem, since there are typically many different explanations for circuit differences. Gemini's approach is to match as many devices and nets as possible.

Gemini's graph matching algorithm is very powerful and efficient, achieving performance that is almost linear in the size of the circuit in most cases. However, some problems with Gemini became apparent over time. First, for some fairly simple circuits, Gemini was very slow to converge, in spite of 'obvious' structure in the circuits. This was especially apparent for highly symmetric circuits like register files and data paths. Second, Gemini was often unable to pinpoint the differences between circuits and offered the designer no help on how to proceed. Finally, Gemini did not allow series-connected transistors to be interchanged. In CMOS circuits where almost every complementary gate contains transistors in series, this became a serious drawback.

The improvements described in this paper address these problems. A new matching algorithm has been incorporated that uses local structure information to match nodes in the two graphs. Gemini uses this more efficient algorithm where possible, relying on the more general partitioning algorithm only when necessary. Gemini dynamically switches between the two algorithms based on the success of the weaker algorithm. A preprocessing step has also been added to allow the order of series-connected transistors to be ignored by the isomorphism algorithm.

The Partitioning Algorithm

Gemini determines whether two graphs are isomorphic using a partitioning algorithm[1, 2]. While the graph isomorphism problem appears to be very difficult in general[4], partitioning algorithms have been shown to be effective for a wide range of graph applications where pathological graphs do not occur in practice. Digital circuits fall in this category.

Circuits are represented in Gemini as bipartite graphs as shown in Figure 1, with devices and nets forming the two class of nodes. Devices are connected to nets through terminals which are divided into different classes. For example, the source and drain of a transistor are in the same terminal class since their connections can be interchanged without affecting the circuit. This very general way of representing circuits has several advantages. First, the representation is compact, using only $O(N)$ space for a set of N completely connected devices instead of $O(N^2)$ as required if only devices are represented. Second, this representation makes it convenient for the partitioning algorithm to alternate labeling

[1]This research was supported by the Defense Advanced Research Projects Agency, ARPA Contract Number MDA903-85-K-0072, and by NSF, Contract Number CCR-8657589.

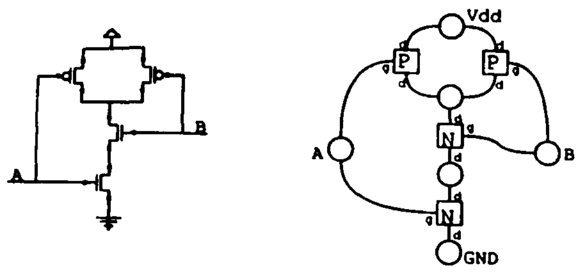

Figure 1: *A simple circuit and its graph representation. Devices are represented by squares and nets by circles. There are two terminal classes, drain/source represented by* d *and gate represented by* g.

the two classes of nodes. Finally, this representation is technology independent since it makes no assumptions about the devices or their interconnection.

The partitioning of a graph is done implicitly by assigning a *label* to each node in the graph. This labeling is done using *node invariants*, which are properties that are maintained under an isomorphism mapping. Gemini uses the device type as the invariant for device nodes and the number of connections as the invariant for net nodes. An initial partitioning is done using these invariants as labels. This partitioning is refined by relabeling each node based on the labels of its neighbors, suitably modified by the terminal classes used to make the connections.

It is easy to show that if an isomorphism mapping exists between two graphs, then there is also a mapping between the partitions that are created by this labeling procedure. Moreover, the mapping can be easily derived from the labels. If all partitions contain a single node, then an isomorphism mapping has been found. The goal therefore is to relabel the nodes of the graphs until only singleton partitions remain.

Most circuits admit to this isomorphism algorithm and singleton partitions are in fact generated by repeated labeling. Symmetric circuits, for which many distinct isomorphism mappings are possible, are an exception. Thus the partitioning algorithm cannot reduce the size of partitions that contain equivalent nodes. When Gemini detects symmetry, it arbitrarily matches nodes in corresponding partitions in the two circuits. Gemini's algorithm for doing this did not always work, often making only very slow progress and sometimes failing completely. GeminiII makes use of local matching as described in the next section to always make steady, if somewhat slow progress.

When a node is relabeled, all the information in the labels of neighboring nodes must be kept. This requires either progressively larger labels which are time-consuming to compute, or a renormalizing of labels after each step which is also time-consuming. Gemini instead uses approximate labels with values in the range $[0 \ldots N-1]$ with $N = 2^{32}$. Labels are then computed according to the hash function: $L^+ = \sum_i c_i L_i$ where L^+ is the new label, the L_i's are the labels of neighboring nodes and c_i is a factor that depends on the terminal class through which the neighbor is connected. It can be shown that with suitably chosen factors and initial labels, these approximate labels behave like the full labels with high probability.

Unfortunately, for very large circuits and circuits with a large effective diameter, collisions did in fact occur in Gemini resulting in non-singular partitions with differing nodes. These collisions occur because labels cannot maintain all information over a long number of relabelings. When this happened, Gemini was not able to reach any conclusion about the circuits. The labeling procedure was modified in GeminiII by assigning matching nodes new, unique labels as they are matched instead of relying on the unique labels generated by the labeling algorithm. This, in connection with the local matching algorithm, serves to minimize the loss of information in the approximate labeling.

Local Matching

The strength of Gemini's graph isomorphism algorithm is that it does not rely solely on local structure in the circuit to map the elements from one circuit to another. Many circuit comparison algorithms start from known nets such as inputs and progress from there by straightforward deduction about which nodes must match. This *local matching* approach works very well where circuits are locally structured. However, the kinds of large, regular circuits typically found in VLSI chips do not admit to this type of algorithm.

Gemini, by contrast, uses labeling to combine the local information over a large area until it finds nodes that must match. This algorithm can be inefficient, however, especially for large symmetric circuits since many nodes must be relabeled many times before being labeled uniquely.

GeminiII takes advantage of the strengths of the two different approaches by adding a simple, local matching algorithm to the general partitioning algorithm, using whichever is appropriate at each step of the matching process. The global partitioning algorithm is first used until some nodes are found to match. At this point, Gemini focuses on the neighbors of each pair of matching nodes and tries to match them based on strictly local information. This is done by partitioning just the neighbor nodes according to their values. These partitions must match in just the same way as the partitions of the graph itself and the nodes in singleton partitions can be immediately added to the isomorphism mapping. This local matching continues until there are no unprocessed matching nodes.

The two algorithms blend extremely well since they are cast in very similar terms. One examines partitions in the entire graph while the other examines partitions in a very small neighborhood. Thus most of the same data structures and procedures can be used for both algorithms. There is in fact a range of alternatives for local matching. An intermediate algorithm matches all the nodes in the local neighborhoods together instead of in each individual neighborhood. The performance of these alternatives are similar, but matching over restricted neighborhoods isolates circuit differences better.

The local matching algorithm does not discover anything that the global partitioning algorithm does not discover. However, it is more efficient since the work of global relabeling is not done. For most circuits over 95% of the labels can be found using local matching. However, the global partitioning algorithm is essential to provide the local algorithm 'anchor' points from which to start.

Isolating errors

The partitioning algorithm works extremely well when two circuits are the same, but it offers little help if the two circuits are different. In this case, the labels in the two graphs diverge rapidly and little useful information can be extracted. It is useful to think of one circuit as the standard and differences between the circuits as defects in the second. A single defect such as a switched or missing connection will cause all nodes within a distance D to be mislabeled after D relabelings. Since most circuits have a small diameter (typically less than 10), the entire circuit will be labeled differently after only a few relabelings. Gemini avoids this problem by matching all partitions after each relabeling. This allows singleton partitions to be matched as soon as possible and also allows non-matching partitions to be detected early and isolated so that differing labels do not propagate into the rest of the circuit.

Local matching also has a very beneficial effect on pinpointing defects. Matching is propagated without the global relabeling that causes large sections of the circuit to be mislabeled. This tends to localize the area affected by defects, and it also gives the designer a way to find defects when Gemini has trouble pinpointing them.

Gemini produces a node equivalence file as part of the matching process which can then be used as input to tell Gemini which nodes to match *a priori*. By starting out with matching nodes, the local matching algorithm can propagate matches into that part of the graph which contains the defect, narrowing down the area in which the problem occurs. Iterating this procedure typically narrows down the area containing the defect until it is isolated.

Symmetric Circuits

The running time of the partitioning algorithm increases dramatically for highly symmetric circuits. In these cases Gemini must make sure that partitions do in fact contain equivalent nodes before performing an arbitrary match. After every such match, Gemini continued with the labeling algorithm to allow partitions to subdivide if necessary. Many times, of course, this labeling did not accomplish anything.

GeminiII uses the local matching algorithm to quickly determine the result of arbitrarily matching two nodes in symmetric circuits. If no other nodes are affected, then no relabeling is required and the matching procedure can continue. This improved handling of symmetric circuits has dramatically improved the running time of GeminiII for symmetric circuits. For example, Gemini took 8 hours to compare one large circuit containing very fine-grained symmetry: GeminiII compares these in less than 5 minutes.

Series Transistors

Gemini compares circuits based on strict topological structure alone. Thus Gemini will consider two topologically different circuits different even if their functionality is the same. In the general case, it is extremely difficult to determine whether two topologically different circuits are functionally equivalent. However, series-connected transistors are a simple instance that is easy to recognize and that is particularly common in CMOS circuits.

The NOR gate of Figure 1 illustrates the problem with series transistors. Clearly the functionality of the circuit does not change if signals A and B are interchanged. However, the graph structure *is* changed and Gemini would report such a switch as an error. In practice, we have found that many designers want to know when such an interchange has been made since even though the circuit function remains the same, its performance

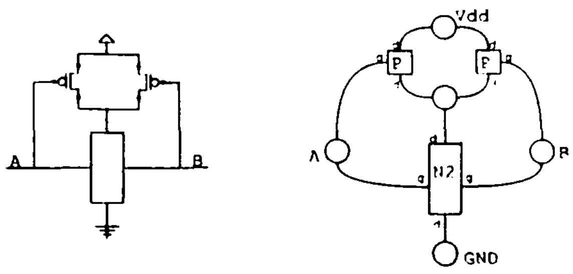

Figure 2: *The replacement of series-connected transistors by a composite device.*

can be affected substantially. For example, the size of transistors in a stack is typically increased towards the rail and late-changing signals are connected near the output. However, there are many designers who choose to ignore these issues, especially in CMOS designs where series transistors occur much more frequently than in NMOS.

The solution takes advantage of the fact that Gemini uses a general graph isomorphism algorithm that is oblivious to the actual types of devices contained in the circuit. A set of series-connected transistors, which we call a *chain*, can be thought of as a composite device that implements the AND switch function. That is, a chain comprising N transistors can be replaced with a pseudo-device with two source/drain terminals and N gate terminals as shown in Figure 2. The gate connections can now be permuted on this composite device without affecting the graph structure.

A pre-processing step that is part of the input procedure performs this graph reduction. As composite chain devices are matched, Gemini checks the order of the transistors and prints warnings if it differs in the two circuits. Isolating series transistors also serves to reduce the number of isomorphism errors, which can be more easily pinpointed by GeminiII.

Results

The graph in Figure 3 plots the running time for circuits of varying size and symmetry. The time measured is CPU time including I/O time for a Sun-3/180 workstation with local disk and 16Mbytes of memory. None of the circuits represented here suffered from substantial paging activity. We have chosen cases from the two extremes with respect to symmetry to indicate a performance envelope for all types of circuits.

For circuits with little or no symmetry, the running time is given by the equation $T = 2.6(\frac{N}{1000})^{1.09}$ where N is the number of transistors in the circuit. This is almost linear in the size of the circuit and extrapolating to a circuit with 1,000,000 transistors yields a running time of less than one hour.

The presence of symmetry increases the running time of GeminiII substantially. However, GeminiII is able to compare symmetrical circuits with predictable running times, in contrast to the original Gemini which often was unable to make a comparison or was very inefficient. The running time for highly symmetrical circuits is given by the equation $T = 3(\frac{N}{1000})^{1.85}$. This almost quadratic behavior is a result of the repeated scanning of partitions that Gemini performs when searching for the nodes most likely to be equivalent. The user can reduce the symmetry of the circuit by giving GeminiII a list of matching nodes such as inputs,

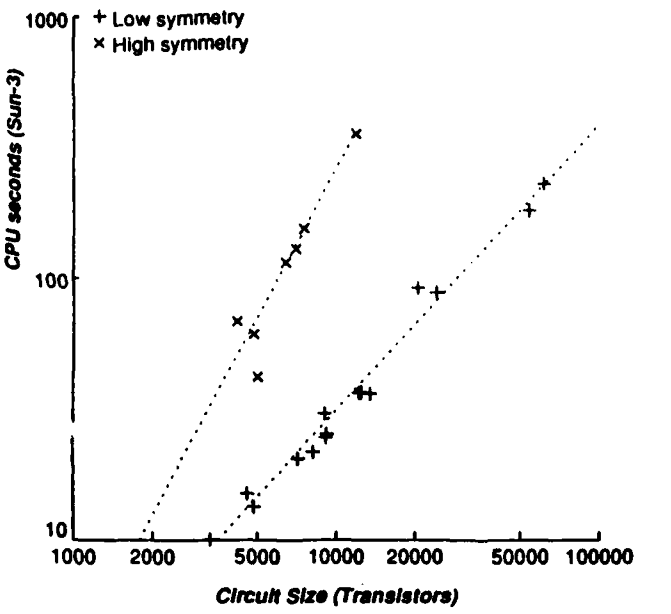

Figure 3: *GeminiII performance over circuits of varying size and symmetry.*

Type of error	CPU time	Spurious Errors reported
None	91	0
1 switched connection	116	0
Vdd/Gnd shorted	121	0
10 shorted connections	118	+5/-4
10 open connections	135	+2/-1
10 shorts, 10 opens	143	+10/-3
10 missing transistors	113	0
20 missing transistors	133	+3
100 missing transistors	181	+88
1000 missing transistors	450	+731

Table 1: *Performance of GeminiII when locating defects of various types. The circuit is a 25,000 transistor microprocessor with about 1% symmetry. Spurious errors indicates the number of non-defect nodes reported (+) and the number of defect nodes not reported (-).*

outputs, and bus lines. However, large chips that contain components such as memory with a high degree of symmetry, usually contain interface circuitry that contains sufficient information to make the symmetry of the overall chip negligible.

Table 1 indicates the ability of GeminiII to find different kinds of defects. The ability of GeminiII to pinpoint errors is enhanced by the use of the local matching algorithm. If the difference between the two circuits is not substantial, then GeminiII typically pinpoints the error precisely. Sometimes, however, GeminiII will report nodes in error that are in fact correct. The usual procedure in this case is to rerun GeminiII while indicating which of the reported nodes in fact match. This forces GeminiII to find an alternative 'explanation' for the error. Depending on the number [illegible] errors, the user may have to repeat this procedure several ti[illegible] to find the location of the 'real' defect. In the limit, the co[illegible]t explanation can be generated by prematching all nodes ex[illegible] those in error. The local matching algorithm allows one to approach this in effect by only prematching a few nodes in the vicinity of the reported error. These anchors are used to propagate matches, reducing the area in which errors are reported.

The type of error that continues to be the most difficult to analyze is that of shorted and switched connections. This type of defect affects many devices and nets unrelated to the cause of the error making it very difficult to isolate the defect in spite of the improvements described here. The better isolation of defects is the subject of continuing research.

Conclusion

GeminiII is a very efficient program for validating circuit layout against an independent specification. GeminiII retains the same general partitioning algorithm as Gemini, but makes better use of local matching to isolate errors and match symmetric circuits more efficiently. Collisions in the labeling hash function have been effectiveiy eliminated by randomizing labels as nodes are matched. Finally, GeminiII recognizes permutations of series-connected transistors to be equivalent in function, increasing its utility for CMOS circuit designs.

References

[1] D. G. Corneil and C. C. Gotlieb. An algorithm for graph isomorphism. *Journal of the ACM*, 17:51–64, January 1970.

[2] D. G. Corneil and D. G. Kirkpatrick. A theoretical analysis of various heuristics for the graph isomorphism problem. *SIAM Journal of Computing*, 9:281-297, May 1980.

[3] C. Ebeling and O. Zajicek. Validating vlsi circuit layout by wirelist comparison. In *Proceedings of ICCAD*, pages 172-173, 1983.

[4] R. C. Read and D. G. Corneil. The graph isomorphism disease. *Journal of Graph Theory*, 1:339-363, 1977.