HIERARCHICAL TIMING VIEW GENERATION INCLUDING ACCURATE MODELING FOR FALSE PATHS.*

P. Das, P. Johannes, L. Claesen, H. De Man †
IMEC, Kapeldreef 75, B-3030 Leuven, Belgium
Phone: +32-16-281220

Abstract

This paper presents a new hierarchical timing verification method consisting of the elimination of local logical incompatibilities and giving accurate and efficient solutions of the false path problem. Experimental results indicate that, with this new hierarchical method, the CPU-times decrease up to 2 orders of magnitude for complex examples.

1 Previous approaches for solving the false path problem

Traditional timing verifiers like Crystal [1] and LEADOUT [2] use a PERT-like algorithm for searching the longest path in a circuit. These methods work correct and efficient as long as there are no false paths in the circuit. But, because they do not take into account the logical incompatibilities, this can result in a large overestimation when there are false paths in the circuit. E.g. for a 24-bit bypass ALU, PERT gives an overestimation of more than 100%.

A first attempt for solving the false path problem is explained in [3]. The false paths are eliminated in a post-processing step. With a modified PERT-algorithm, the n longest paths are calculated. Those paths, sorted by decreasing length, are tested for sensitizability using the D-algorithm [5] on the logic model of the circuit. Unfortunately, the number of false paths to be eliminated before the first sensitizable path is found is fairly large and because this number cannot be predicted, the choice of n is very difficult.

That problem has been solved in [4] by eliminating the false paths as a part of the search and only the longest sensitizable path is presented to the user. This method gives accurate results, but has the drawback that the CPU-times explode for large circuits with many false paths. This algorithm is explained in section 2.

The new method gives a solution for this problem by using the hierarchy in the circuit. The method will be explained in section 3. Experimental results will be given in section 4.

*Work sponsored by the EC under the ESPRIT-1058 project.
†Professor at Kath. Univ. Leuven.

2 The Longest Sensitizable Path (LSP) algorithm

As in other timing verifiers, the timing behavior of the circuit is modeled by a weighted directed graph. The edges in the graph represent the propagation of a signal from one node to another. The weight of the edge is equal to the propagation delay through the circuitry between the two nodes. Associated with each edge are a number of logical conditions for the propagation of the signal[4]. During the search for the longest sensitizable path, before adding a new edge to the path, it is necessary to check that the propagation conditions are compatible with the propagation conditions of all the other edges already in the path. Those conditions must not only be checked locally, but they must be propagated through the logical model of the circuit to all the related nodes.

The problem is thus equivalent to searching the longest path in a conditional graph. A depth first search is used, with following properties :
The search is guided by a low cost heuristic, namely PERT.
The search space is reduced by pruning : if the longest pert-path through a node is shorter than the longest sensitizable path already found, then the subgraph of that node must not be searched because the longest sensitizable path through that node can never be longer than the path already found.

With respect to the CPU-times of this longest sensitizable path algorithm, the following remarks can be made :

- When the longest PERT-path is not a false path, the algorithm is almost as fast as PERT.
- For small circuits where the number of paths to check is small, the CPU-times are of course also manageable.
- But for large, "real life" circuits with many false paths, the CPU-times explode and it takes too long before the longest path is found. For a 24-bit ALU with bypass circuitry, it takes more than one CPU-hour to search the longest sensitizable path[4].

The presented hierarchical method reduces the CPU-times for those "real life" circuits with many false paths by making use of the other two properties. That is described in the following section.

IEEE 1989 CUSTOM INTEGRATED CIRCUITS CONFERENCE

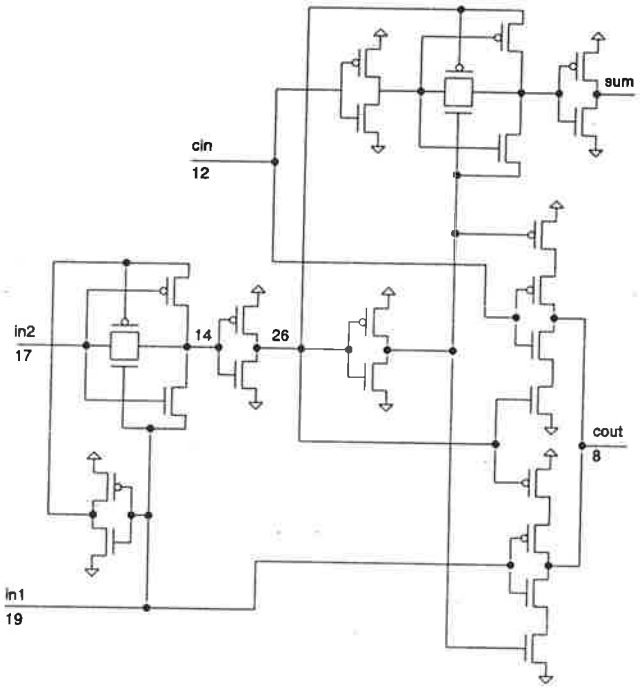

Figure 1: A fulladder circuit

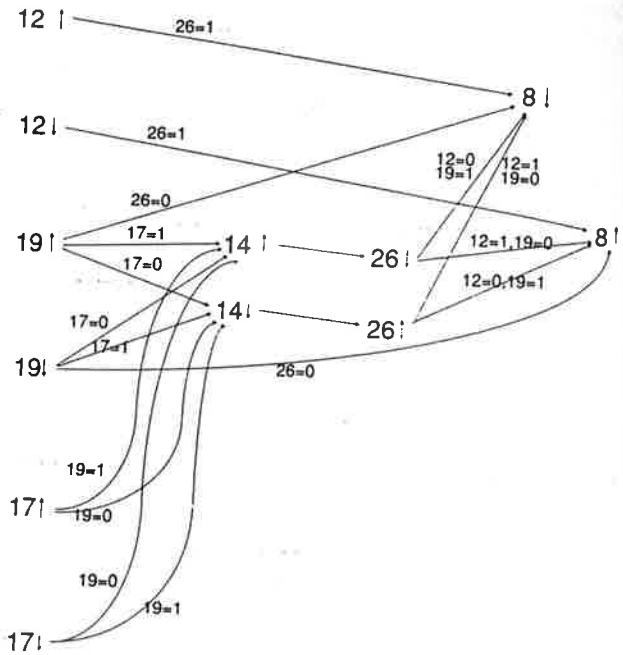

Figure 2: The event graph of the carry generation part of the fulladder

3 The new hierarchical method

3.1 Motivation for this new method

By looking at the occurrences of false paths in the circuit, it can be observed that most false paths occur due to one of the following two reasons :

- false paths due to local logical incompatibilities.
- circuits where the designer intended to create false paths by adding bypass circuitry for speeding up the global circuit.

Examples of local logical incompatibilities can be found in the fulladder circuit of figure 1. Figure 2 gives the event graph and the logical conditions of the carry generation part of this fulladder cell. On this graph can be seen that the logical conditions of some paths are incompatible and these paths are false paths.

An example of a bypass circuit is given in the 16-bit ALU of figure 3. The longest PERT-path through the combination of the 12-bit ALU and 16-bit bypass ALU is the carry ripple path of the 16-bit bypass ALU. However, the longest sensitizable path is the carry ripple path of the 12-bit ALU because the carry ripple path of the 16-bit ALU is false due to the bypass circuitry.

If all these local logical incompatibilities could be eliminated, there would remain less false paths and this would speedup the LSP-algorithm. This can be done by using the hierarchy of the circuit.

The new hierarchical method can be summarized as follows :

Generate the timing views for all the basic cells and for the bypass circuitries together with the bypassed cells.
Compose these timing views to become an event graph for the whole circuit.
Run the LSP-algorithm on this event graph to find the longest sensitizable path in the circuit.

3.2 Timing view generation

The timing view generation method consist of 2 new graph manipulation techniques : the elimination of local logical incompatibilities and a graph reduction by event elimination.

Elimination of local logical incompatibilities.
From the circuit description, the logical functions in the circuit are derived with DIALOG[6]. Also the event graph[2] of the circuit is set up. An event corresponds either to a falling or a rising transition on an electrical node in the circuit. Each edge in the graph, the causality relationship between two events, has a corresponding delay, calculated with a Horowitz type of RC-models[7], and a corresponding set of logical propagation conditions for signal propagation. The total graph is a combined logical and event graph.

False paths can occur in the graph and have to be eliminated. Out of the old event graph, a new event graph without false paths has to be generated. This can be done in several ways [8][9]:

Path enumeration : All the paths are enumerated and the paths with logical incompatibilities are deleted. This method is very easy and would work, but is not efficient because the memory requirements are too large.
Path enumeration with optimal compaction : Take as much events as possible together so that a minimal graph results. This method is not usefull because it would take to much CPU-time to find the optimal solution.
The SLOCOP-method : A method in between the two previous methods is developed. A depth first search is applied with checking for logical incompatibilities as in the LSP algorithm. During the forward search, the path is created and checked for sensitizability. While backtracking, if a sensitizable path is found, the following algorithm is used for compaction : *Two events in the graph are taken together if they refer to the same circuit node, have the same transition, and have the same subgraph.*

Algorithm :

```
1. Initialize : ev = rootevent;
2. while (∃ NOT INVESTIGATED out-edge of ev) {
       /* FORWARD */
       Take edge and mark it INVESTIGATED;
       if (edge compatible with current path) {
           Push current state on stack;
           Add edge to current path;
           ev = out-event of edge;
           }
       }
3. /* BACKWARD */
   if (ev == rootevent) exit;
   Try to combine ev with other events;
   Pop previous state from stack;
   go to 2.
```

This results in a graph without logical incompatibilities where some events have been duplicated. The event graph in figure 2 of the fulladder circuit in figure 1 becomes after elimination of the logical incompatibilities the graph in figure 4.

Graph reduction by event elimination.
As described in the previous section, the number of events has increased due to the elimination of logical incompatibilities. In this section, a method is described to compensate this effect by event elimination in a post processing step.

If an event is eliminated, all the in-edges and all the out-edges of that event are replaced by edges from all the in-events of the eliminated event to all the out-events of the eliminated event. The resulting delay of a created edge is the sum of the delays of the two replaced edges and the resulting logical propagation conditions are the conjunction of the two replaced edges.

Because the events are eliminated in a post processing step, the CPU-time required by the method must be very low

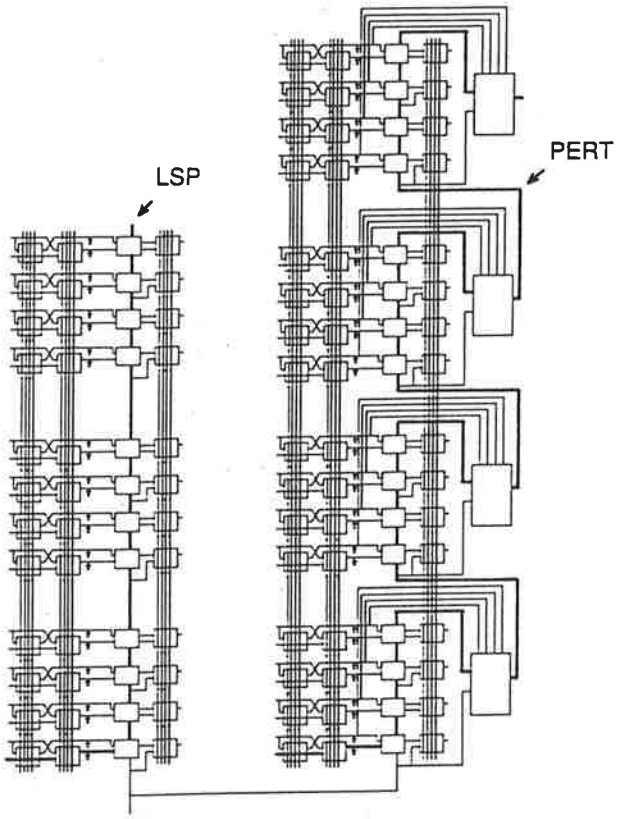

Figure 3: A 12-bit ALU without bypass and a 16-bit bypass ALU

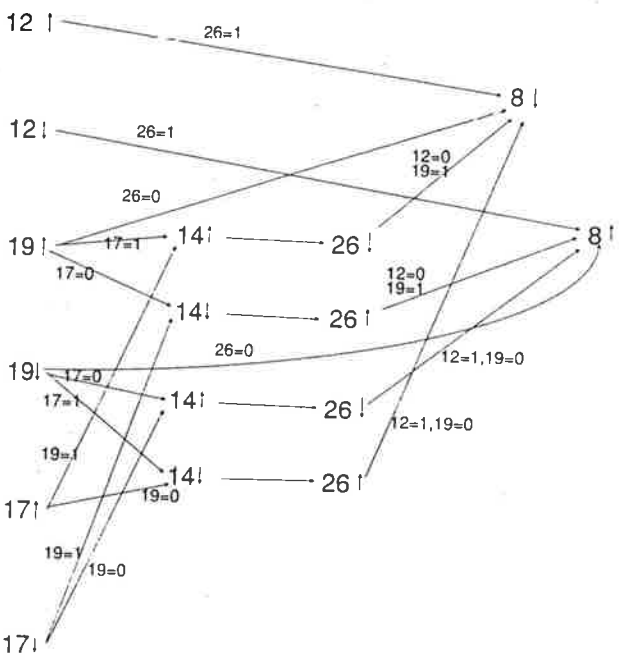

Figure 4: The event graph with logical incompatibilities eliminated

and a very simple algorithm has to be used. The following simple criterion for elimination is used : *eliminate an event if the number of created edges is smaller than the number of replaced edges.* In other words, eliminate an event if the sum of the in-edges and out-edges of the event is larger than their product.

3.3 The hierarchical composition

The timing views can be hierarchically composed and an event graph for the higher level cell is generated. This event graph has less (or none) false paths because all the local logical incompatibilities are eliminated during the timing view generation.

The LSP-algorithm can run on this event graph and the required CPU-times will be much lower than running it directly on the whole flat circuit because all the local logical incompatibilities are eliminated. In many circuits there will be no false paths any more in the hierarchically composed event graph.

4 Experimental results

In table 1 the results of this new method are given for Mead and Conway type ALU's with a different number of bit slices. Alu12 has no false paths and there is of course no improvement with the new method. For the alu's with 14 bit slices or more, each 4 bit slices in the carry ripple chain, a bypass circuitry is included. The bypass circuitry creates false paths and as can be seen in table 1, the CPU-times required by the longest sensitizable path algorithm (LSP) increase rapidly.

circuit	LSP	Hierarchical		longest
		prep.	analysis	path
	CPU-sec	CPU-sec	CPU-sec	nsec
alu12	0.6			6.54e-08
alu14	760	6.0	0.5	5.59e-08
alu16	1070	6.0	0.6	5.83e-08
alu18	2264	6.0	0.7	5.83e-08
alu20	2275	6.0	0.8	6.12e-08
alu22	3280	6.0	0.9	6.12e-08
alu24	4020	6.0	0.9	6.37e-08

Table 1: CPU-times (on an Apollo 3500) of the new hierarchical method (preprocessing + analysis) compared with the flat Longest Sensitizable Path method.

The new hierarchical method consist of 2 steps. First, a timing view has to be generated for 4 carry ripple cells together with the bypass circuitry. The logical incompatibilities in the bypass block are then eliminated. The CPU-times to generate this timing view is indicated as the preprocessing part of the new hierarchical method in table 1.

Second, these timing views are composed and results in an event graph for the whole ALU. The LSP-algorithm can run on this graph and the CPU-times are decreased significantly as can be seen in table 1. E.g. for the 24-bit bypass ALU the CPU-time is reduced from more than one CPU-hour to 15 CPU-seconds, which is a reduction of two orders of magnitude.

5 Conclusions

A new hierarchical method for efficient solving the false path problem, is presented. In a preprocessing step, all the local and the user intended logical incompatibilities are eliminated by generating timing views for these basic cells. These timing views are hierarchically composed and there remain less (or none) logical incompatibilities. Therefore, the CPU-times required by the longest sensitizable path algorithm are much lower and for complex examples, that can be a reduction of two orders of magnitude.

6 References

[1] John K. Ousterhout, *Crystal: a Timing Analyser for nMOS VLSI circuits*, Proc. Third Caltech VLSI Conf., 1983, pp. 58-69.

[2] Thomas G. Szymanski, *LEADOUT: A Static Timing Analyser of MOS Circuits*, Proc. IEEE ICCAD '86, Nov. 1986, pp. 130-133.

[3] E. Vanden Meersch, L. Claesen, H. De Man, *SLOCOP: a Timing Verification Tool for Synchronous CMOS logic*, ESSCIRC '86, pp. 205-207.

[4] J. Benkoski, E. Vanden Meersch, L. Claesen, H. De Man, *Efficient Algorithms for Solving the False Path Problem in Timing Verifiers*, Proc. IEEE ICCAD '87, Nov. 1986.

[5] J. P. Roth, *Diagnosis of automata failures: A calculus and a new method*, IBM J. Res. Develop., Oct 1966, pp 278-281.

[6] I. Bolsens, W. De Rammelaere, L. Claesen, H. De Man, *Electrical Verification Using Rule Based Programming and Symbolic Analysis*, IFIP-workshop: Knowledge based systems for test and diagnosis, Grenoble, September 1988.

[7] M. A. Horowitz, *Timing Models for MOS circuits*, Department of Electrical Engeneering, Stanford University, Stanford CA94305, Tech. report No. SEL83-003, Dec. 1983.

[8] J.P. Schupp, *STIVITS: a high performance timing verification system for VLSI-chips based on compiled code generation*, Engineering thesis Kath. Univ. Leuven Belgium, July 1988 (in Dutch).

[9] L. Claesen, J.P. Schupp, H. De Man, *Accelerated Sensitizable Path Algorithms for Timing Verification based on Code Generation*, Proc. IEEE ISCAS-89, May 9-11, 1989, Portland Oregon.