GLS VLSI -92 Michigan USA

On the use of hierarchy in timing verification with statically sensitizable paths *

P. Johannes

IMEC†
Leuven
Belgium

L. Claesen and H. De Man

IMEC/Katholieke Universiteit
Leuven
Belgium

Abstract

In this paper a novel solution for the efficiency problems encountered in static timing verification is presented. The LSP algorithm is submitted to a critical analysis. A new hierarchy based approach is presented and its advantages and limitations are highlighted. Finally, some results on real life circuits are presented.

1 Introduction

As integrated circuit designs evolve toward higher performance and complexity, there is an increasing need for fast and accurate timing analysis [1]. The presence of many false paths in VLSI circuits calls for complex algorithms to detect the longest true paths in a circuit. Although the viable path concept [2] and other dynamic false path approaches [3, 4] are theoretically more accurate, experimental evidence [2, 3, 4, 5, 6] suggests that static false path analysis is generally as accurate in real circuits. Therefore the basis of the research presented in this paper is the LSP algorithm introduced in [7].

The main performance problem of this algorithm is explained in section 2. In section 3 an hierarchical approach is presented which overcomes these problems. This solution is quite different from the one published in [6], which is potentially limited for certain classes of circuits, e.g. multipliers. To the authors' knowledge no algorithm for the exploitation of hierarchy in the false path problem has been published by other research groups. The results of the new approach and its applicability are discussed in section 4. Finally, some results on real life circuits and some conclusions are given.

*Research sponsored by the EEC-SPRITE project

†Inter University Micro-Electronics Center

2

2 Motivation

For the purpose of timing verification the circuit is transformed in an acyclic event graph $\mathcal{V} \times \mathcal{E}$ in which:

1. $\mathcal{V}$ is the set of vertices v which represent transitions on circuit nodes. These transitions can be either up or down.

2. $\mathcal{E}$ is the set of edges e between vertices v. They are weighted with the delay d and contain the logic conditions on circuit nodes that are necessary for the traversal of e.

For a better understanding the LSP algorithm used in static timing verification is briefly sketched below.

1. In a vertex of the graph, choose the best edge to add to the path, based on a quantity called *esperance* which is the maximal length the global path could attain if this edge is added to the path.

2. If the chosen edge is compatible with the path which was collected up to this point, add the edge to the path. Go to 1 with the starting vertex being the end vertex of the newly added edge. If the edge is not compatible with the recorded path, go to 3.

3. Try the other edges leaving the vertex in decreasing order of esperance until a compatible edge is found, and add it to the path. Go to 1 with the starting vertex being the end vertex of the newly added edge.

4. If an output is reached, stop.

5. If in a given vertex no edge can be found, remove the last edge added to the path and go to 3 with the starting vertex of the removed edge.

The performance problem of this algorithm is caused by step 5. This step causes the algorithm to inspect all possible paths, in decreasing order of maximal attainable length, until a path is found that is completely statically sensitizable. This backtracking makes the complexity of the algorithm linear in the number of false paths that are longer than the longest sensitizable path. Therefore the worst case behavior of the algorithm is exponential with the size of the event graph. An example of this behavior is shown in figure 1 where $\circ$ and $\bullet$ indicate conflicting propagation conditions along edges from A to E. If each edge has length 1, the LSP algorithm will first examine the 36 $\bullet$ paths which cannot reach E before considering one of the $\circ$ paths.

It must be noted that if there are no false paths in the graph, the algorithm performs linearly with the size of the graph.

A solution to this performance bottleneck is to have the algorithm analyse a graph with as few false paths as possible. An in loco removal of the false paths would be unpractical for the same complexity reasons as mentioned above. Therefore the analysis must be performed on a graph which has as few as possible false paths by construction.

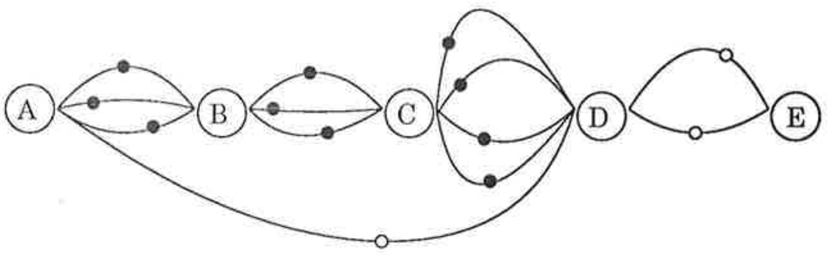

Figure 1: Complexity in the LSP-algorithm

It is possible to use hierarchy to obtain such a graph: if a leaf cell of a circuit is completely free of false paths, then false paths at higher level can only be created through hierarchical assembly. The number of false paths will thus be greatly reduced.

Consider for example the 4 bit carry bypass adder section of figure 2. It consists of 4 full adders and a nand-multiplexor bypass cell. The full adders each contain 24 false paths and the bypass section contain 6 false paths. In a 4 bit carry bypass section they introduce 346 false paths. Starting from leaf cells with no false paths we only have 78 false paths to deal with in the 4 bit section.

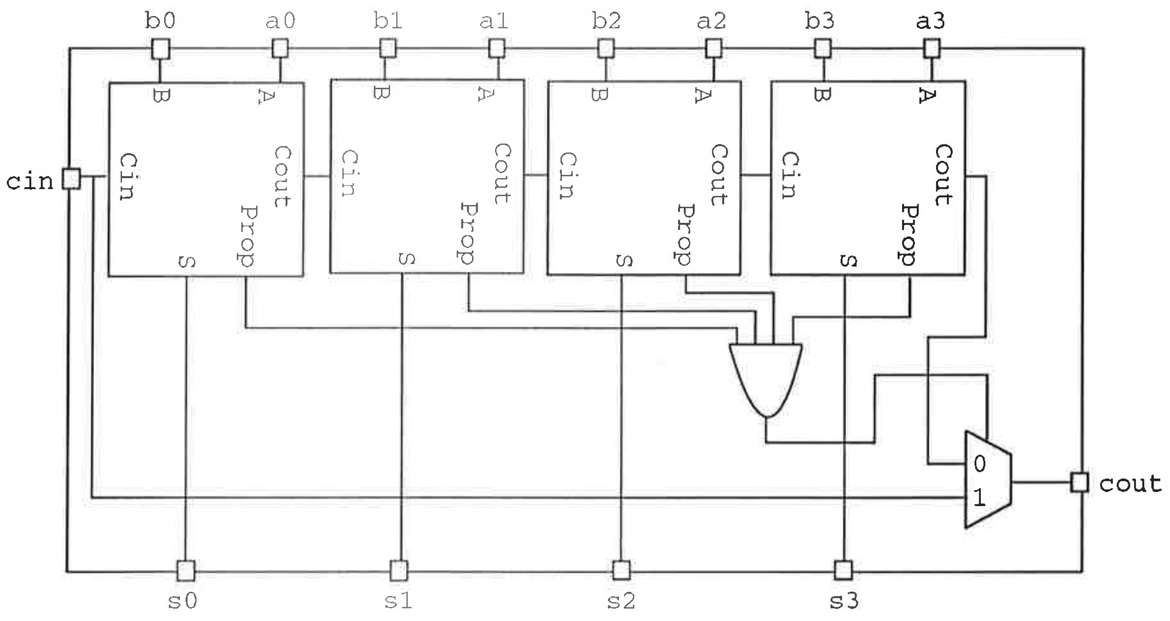

Figure 2: A 4 bit carry bypass section

In the next section the construction of the timing model without false paths is discussed.

3 Model generation

The generation of a timing model that is free of false paths can be done in several ways:

- Path Enumeration: all paths are enumerated, the false ones are rejected. This is the simplest method but has prohibitive memory requirements.
- Path Enumeration with Optimal Compaction: this has been presented in [8]. The graph that is produced is minimal but the cpu-time required to do so is also prohibitive.
- LSP based compaction: a path is created and checked for sensitizability in a forward step. In a backward step the graph is compacted taking together events that refer to the same node, have the same transition and have the same subgraph.

All of the above methods result in a graph without false paths in which events may be duplicated for the preservation of the logic behavior. A small example of such a transformation is shown in figure 3.

The algorithm that offers the best solution, TVG, is presented below. It works in a similar way as the redundancy removal algorithm presented in [9], but acts on the event graph only and does not change the behavior of the circuit in any way.

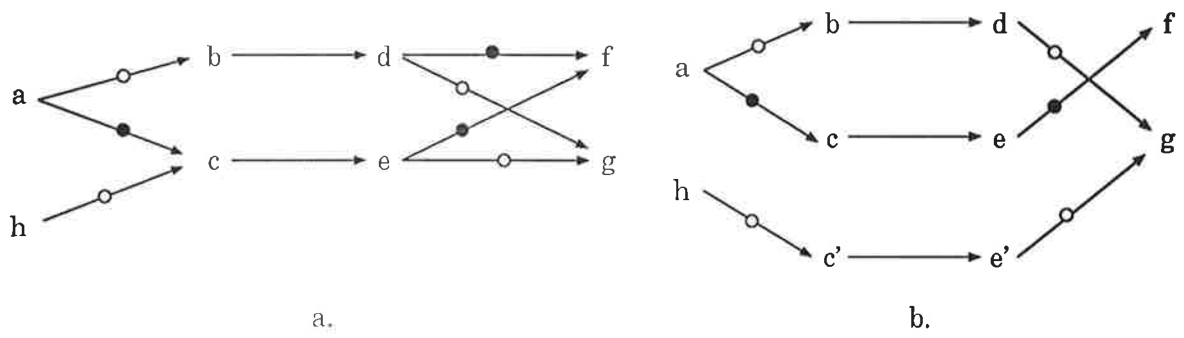

Figure 3: a. graph with logic incompatibilities, b. graph with same logic behavior without logic incompatibilities

A dummy event, *root* is connected to all inputs by edges that have neither delay nor propagation conditions. A path is a collection of pairs of edges and events. *Node* refers to the circuit node connected with an event.

```
TVG(){
Copy the $root$ to $new\_root$.
Start with:
    $event = root$,
    $new\_event = new\_root$,
    $path = \{\}$
1. Forward Step:
   ∀ outgoing edges of $event$
      if ($edge$ is compatible with $path$)
         add $event$ and $edge$ to $path$
         copy $edge$ and append the
                        copy to $new\_event$
         $new\_event$ = copy of the
                  end event of $edge$
         the incoming edge to $new\_event$
                        is the copy of $edge$
      else
         if (all edges of $event$ are examined
               ||$event$ is on an output)
            go to 2.
2. Backward:
   Combine($new\_event$ with the node it refers to)
   if (all edges are visited)
      return($new\_root$)
   pop ($new\_event$,$edge$) from the path
   if (the starting event of $edge$
            has no unvisited outgoing edges)
      go to 1.
   else go to 2.
}
```

The procedure Combine is introduced so that an as reduced as possible event graph results from the addition of *new_event* to the new graph.

```
Combine($new\_event$,$node$){
let $event\_list$ be the list of events
      at $node$ of same transition as $new\_event$.
if ($new\_event$ has outgoing edges)
   if ( ∃$event$ ∈ $event\_list$ such that:
      {the set of outgoing edges of $event$}
      ≡ {the set of outgoing edges of $new\_event$})
      add the incoming edges of
         $new\_event$ to those of $event$
      delete $new\_event$ and its outgoing edges
else
```

6

```
    if (new_event is on an output)
        if (event_list != NULL)
            add the incoming edges of new_event
                to those of event_list
            delete new_event
        else add new_event to node
    else delete new_event and its incoming edge
}
```

In the next section the result of applying this algorithm in the false path analysis is presented.

4 Experimental Results[1]

First two cases which pinpoint the usefulness of the introduction of the hierarchy in the LSP analysis are discussed. Next the application on some benchmarks and a discussion of the applicability is given.

4.1 24 bit carry bypass alu

The circuit consists of 116 instances of 5 leaf cells: a bypass, a carry even, a carry odd, a general function block even and odd cell. The generation of timing views for these cells from transistor level descriptions takes 18s [10], simulations included. The LSP analysis of the graph of the full ALU takes 22804s (> 6 hrs.).
A close look at the graph reveals that all false paths are generated in the bypassed 4 bit sections of the carry chain. Therefore a new timing view is generated of those 4 carry cells and a bypass cell, in 1.25s.
The introduction of this new level of hierarchy reduces the number of instances to 100, and the LSP analysis time to 2.6s. The longest paths remain the same.
Gain on the LSP analysis: $22804/(1.25 + 2.6) = 3591$.

4.2 16*16 booth multiplier

This is a standard cell design, in the MIETEC 2.4μm library with worst case parameters. A particularity is that the last row in the carry save adder matrix is a 32 bit carry bypass adder. The multiplier consists of 524 instances of 13 leaf cells, whose timing views were generated in less than one cpu minute. The LSP analysis of this multiplier did not terminate, the PERT analysis took 5s. The resulting path was 380ns long. In a multiplier many are false paths present. Usually, the longest real path is approximately of the same length as the longest false path. However, with the bypass sections at the bottom of the carry save matrix the longer false paths become much larger than the longest real path and thus

[1]All the times mentioned are cpu times on a DecStation 3100.

the LSP algorithm is swamped, as explained in section 2. An extra level of hierarchy is introduced and a timing view for the carry bypass sections is generated. The LSP analysis with this new level of hierarchy (446 instances) lasts 6s and yields a path of 260ns.

4.3 ISCAS-85 benchmarks

Several of the circuits of the ISCAS-85 benchmark suite [11] were analyzed. The results are presented in table 1. The circuits C7552 and C6288 which were not easily analyzed in [5, 6] were analyzed in very small cpu times, despite the fact that no additional hierarchy was introduced. In our implementation the C7552 circuit contains 248399 false paths. Generating a timing view for this circuit took 31mn 51s.

4.4 Applicability

From the two examples above it is clear that the hierarchical timing view approach can yield very large improvements in the performance of the LSP algorithm. It is also clear that the user must be aware of the false path problem in order to make the largest gain: it is of no use to make timing views of larger portions of the circuit than those where the false paths occur, as it will probably take longer to do so. Therefore the user should know rather well where relevant false paths occur.

In general, as the user is usually the designer of the circuit, this is not too much of a problem.

5 Conclusion

Some additional results of the hierarchical analysis on real life examples are presented in table 1. The type column indicates if the designs were made with standard cells (MIETEC 3u, worst case parameters, hence the large delays) or with the Cathedral [12, 13] system. 5xp1 is a well known logic synthesis benchmark. 5xp1_area is optimized for optimal area, C7552, C6288 and 5xp1_orig are only mapped to the library.

From these results it is clear that our new approach to hierarchy in timing verification, targeted towards the LSP algorithm, leads to accurate analysis of real life circuits in acceptable cpu times.

References

[1] T. G. Szymanski. LEADOUT: a static timing analyzer for MOS circuits. In *Proc. of the IEEE Int'l Conf. on CAD*, pages 130–133, 1986.

[2] P. C. McGeer and R. K. Brayton. Efficient algorithms for computing the longest viable path in a combinational network. In *Proc. 26th Design Automation Conference*, pages 561–567, 1989.

8

circuit	type	♯ cells	critical path	cpu-time
ERDIF	St.C.	931	166ns *	21s
REC3	St.C.	2031	217ns *	32s
16*16 MULT	St.C.	446	260ns *	6s
ARCODEC	St.C.	589	331ns *	9s
C7552	St.C.	1588	140ns *	11s
C6288	St.C.	2368	466ns *	12s
5xp1_orig	St.C.	86	49ns	0.08s
5xp1_area	St.C.	74	109ns *	0.11s
24 bit ALU	Cath.	100	53ns *	2.6s
APLUSB	Cath.	52	32ns	0.5s
ARCODEC	Cath.	589	90ns *	9s

Table 1: Results of the hierarchical analysis. A * indicates that the longest(PERT) paths were false.

[3] S. Perremans, L. Claesen, and H. De Man. Static timing analysis of dynamically sensitizable paths. In *Proc. 26th Design Automation Conference*, pages 568–573, 1989.

[4] H. C. Yen, S. Ghanta, and H. C. Du. On the general false path problem in timing analysis. In *Proc. 26th Design Automation Conference*, pages 555–560, 1989.

[5] P. C. McGeer, R. K. Brayton, A. Saldanha, P. Stephan, and A. L. Sangiovanni-Vincentelli. Timing analysis and delay fault test generation using path recursive functions. In *Proc. of the Int'l. Workshop on Logic Synthesis, MCNC North-Carolina*, 1991.

[6] Yun-Cheng Ju and Resve A. Saleh. Incremental techniques for the identification of statically sensitizable critical paths. In *Proc. 28th Design Automation Conference*, pages 541–546, 1991.

[7] J. Benkoski, E. vanden Meersch, L. Claesen, and H. De Man. Timing verification using statically sensitizable paths. *IEEE Trans. on Computer Aided Design*, Vol. CAD-9: pages 1073–1084, October 1990.

[8] J.-P Schupp, P. Das, P. Johannes, S. Perremans, L. Claesen, and H. De Man. Efficient false path elimination algorithms for timing verification by event graph preprocessing. *INTEGRATION, the VLSI Journal*, Nr. 8: pages 173–187, 1989.

[9] K. Keutzer, S. Malik, and A. Saldanha. Is redundancy necessary to reduce delay? *IEEE Trans. on Computer Aided Design*, Vol. CAD-10: pages 427–436, April 1991.

[10] P. Johannes, P. Das, L. Claesen, and H. De Man. Slocop-II: a versatile timing verification system for MOS VLSI. In *Proc. of IEEE EDAC*, pages 518–523, 1990.

[11] F. Brglez and H. Fujiwara. Neutral netlist of ten combinational benchmark circuits and a target translator in FORTRAN. In *Proc. IEEE Int. Symp. Circuits and Systems*, June 1985.

[12] J.Rabaey H. De Man, P. Six, and L. Claesen. Cathedral-II: a silicon compiler for digital signal processing. *IEEE Design and Test*, pages 13–25, December 1986.

[13] S. Note, F. Catthoor, G. Goossens, and H. De Man. Combined hardware selection and pipelining in high performance data-path design. In *Proc. IEEE ICCD*, pages 328–331, September 1990.

10