ESSCIRC 93

Achieving accuracy and high performance in static sensitizable path analysis

P. Johannes, L. Claesen *, H. De Man *

IMEC, Kapeldreef 75, B-3001 Leuven (Belgium)

Abstract

In this paper we propose the use of reconvergence analysis to boost the performance of static timing analyzers. The method is based on a reconvergence analysis on top of an hierarchical analysis. The proposed analysis is especially useful for analyzing large circuits containing a lot of unsensitizable (false) paths, where an hierarchical approach is not sufficient in itself. An example of the efficiency of the approach is given.

1 Introduction

In the last years, numerous solutions to the unsensitizable path problem have been presented [1, 2, 3, 4],[ICCAD-91/92]. However, these methods are impractical for large complex circuits with many unsensitizable paths. The work presented in this paper presents a feasible hierarchical solution for a sensitizable path analysis based on the algorithms presented in [1] and [5]. To overcome the complexity of the sensitizable path analysis hierarchical approaches were presented in [5, 6]. Its basic strategy is: *use only leaf cells which are fully sensitizable in the hierarchical timing verification.*
This restricts the number of long unsensitizable paths to those that are created during hierarchical assembly, and eliminates those which are created by adding paths to false paths inside cells. This means that the total number of unsensitizable paths can be greatly reduced, using the Timing View Generation (TVG) algorithm, and the performance of the Longest Sensitizable Path (LSP) [1, 5, 6, 7] can be greatly improved. They can not be totally eliminated as it is impossible to predict how leaf cells will be assembled.

The drawback of the presented methods is that the hierarchy used is the one determined by the designer, i.e. probably not the most suitable for an efficient timing verification but rather the most convenient for the the designer.

In [6] examples are given that show the performance benifits obtained from this approach. However, it can easily be shown that the performance boost is not guaranteed. For instance, the example of figure 1 (part of a video processor [8]), where a large bank of multiplexors is controlled by a set of comparators, still contains too many unsensitizable paths at the given level of hierarchy. The analysis does not terminate, hence new levels of hierarchy must be introduced. When we do this, we can analyze this circuit in 4mn 44.5s. As shown in this example, the overall performance depends on a judicious use of the hierarchy: a level of hierarchy must be introduced as soon as unsensitizable paths are introduced. If it is postponed, the possibility exists that it is not possible any more at a certain level either to make timing views or do the LSP analysis in a reasonable time. Also, making timing views at levels where no unsensitizable paths exist is a waste of time. It is our feeling that this problem should be solved automatically, inside the timing verifier, as designers may not be sufficiently aware of the false path problem to perform a reasonably efficient timing verification.

*Professor at Katholieke Universiteit Leuven Belgium

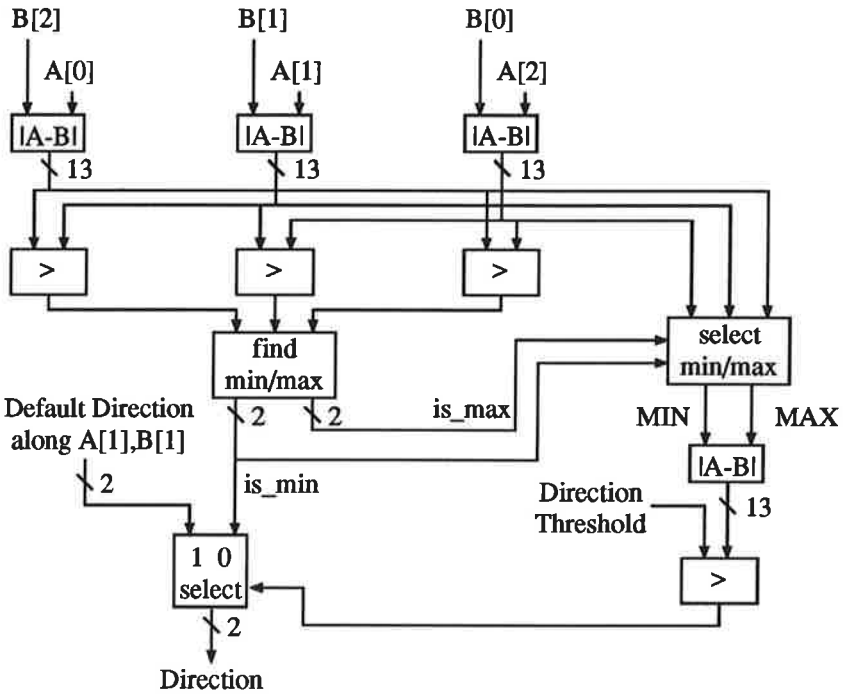

Figure 1: The direction detector example

Therefore it is necessary to develop efficient methods to find the hierarchy that is best suited for the proposed hierarchical timing verification.

In the next section a criterion for automatic generation of a suitable hierarchy is presented. Finally some conclusions are drawn.

2 Reconvergence analysis

It is clear that a cheap criterion is needed. It must be able to forecast the presence of unsensitizable paths and to give good hints as to which parts of the circuit are best modelled in a timing view. One possible criterion is *reconvergence*. Indeed, false paths can only occur in circuits with reconvergent event[1] graphs [5], but not all reconvergent event graphs contain unsensitizable paths. However, at the event graph level, reconvergence is very common. Hence it might not help very much in improving the speed of analysis. To overcome this problem, the reconvergence analysis will be performed on a higher level of abstraction, namely the leaf cell level. This fits perfectly in our hierarchical strategy: as timing views of leaf cells do not contain unsensitizable paths by definition, it is not necessary to check their content. It is only necessary to check their assembly, and the leaf cell level of abstraction is appropriate to do so. Note that the analysis is done incrementally: produced timing views are used in the production of timing views of higher level cells. Therefore the graph we analyse is always related to a certain level in the hierarchy, or *expansion* level.

As reconvergence is a graph theoretical concept, it is necessary to define the graph to which it applies.

Definition 2.1 The *leaf cell graph* or *reachability graph* of a design at a certain level of expansion is a directed graph $\mathcal{R}(L, I, \psi, \Delta)$, where vertices $v \in L$ represent the leaf cells and edges $e \in I$ represent their interconnections, with incidence function ψ and labeling function Δ.

[1] Event graphs are the basis for the LSP and TVG analysis, see [5, 6, 7]

L : $v \in L$ is a leaf cell at the current level of expansion. If a net is connected to more than two leaf cells, an additional vertex is added to L. This is shown in figure 2.

I : $\psi = L \times L \longrightarrow I : \exists i \in I$ between two vertices $v_i, v_o \in L :< v_i, v_o >$ iff leaf cell (or net) v_i produces an input for v_o and if v_o consumes an output of v_i.

$\Delta(L)$: only the vertices $v \in L$ are labeled:

level(v) $\in \mathbb{N}$: used during the search for reconvergences.

ldiv(v) $\subset L$: the set of *divergent* vertices from which exactly one path leads to v.

lrec(v) $\subset L$: the set of *divergent* vertices from which more than one path leads to v.

■

Three additional sets are defined for use in the sorting and reconvergence algorithms:

Definition 2.2 The *Inset(v)*, $\subset L$, which contains all vertices incident to fanin edges of v:
Inset(v) = $\{w \in L | v \in \text{head(fanout}(w))\}$.
Siminlarly, the *Outset(v)*, $\subset L$, is the set of vertices incident to fanout vertices of v:
Outset(v) = $\{w \in L | v \in \text{tail(fanin}(w))\}$.
Level(j $\in \mathbb{N}$) $\subset L$, which contains all vertices with the level number j:
Level(j) = $\{w \in L |$ level $(w) = j\}$ ■

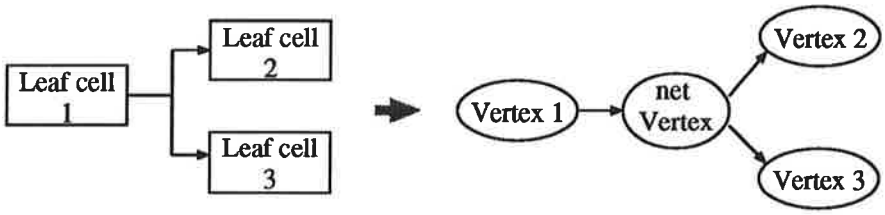

Figure 2: Additional vertices $\in L$ for high fanout or fanin nets

In and *Out* are special vertices for the external ports of the top cell. Note that these definitions are targeted towards combinatorial circuits, or to combinatorial parts of sequential circuits. Therefore reachability graphs are acyclic. The leaf cell graph of the `|A-B|`, which is a part of the direction detector (figure 1), is shown in figure 3.

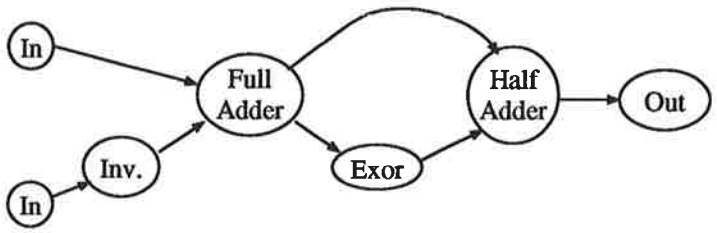

Figure 3: The leaf cell graph of `|A-B|`

Definition 2.3 A cell vertex $v \in L$ is *divergent* if outdegree(v) > 1, and it is *convergent* if indegree(v) > 1. ■

Note that a vertex can be both divergent and convergent. In figure 3 the `Full Adder` is both and the `Half Adder` is convergent.

Definition 2.4 A *path* $\mathcal{P}$ in the leaf cell graph is an ordered set of vertices, starting at an *In* vertex and ending at an *Out* vertex. Each next vertex in the set must be reachable from the current vertex via an interconnectivity edge. ■

Definition 2.5 Two non-equal paths $\mathcal{P}_1$ and $\mathcal{P}_2$ are reconvergent if they have a common divergent vertex v_d followed by a common convergent vertex v_c for which holds: $\exists$ vertex $v_m \in$ {a path from v_d to v_c} such that $(v_m \in \mathcal{P}_1 \wedge v_m \notin \mathcal{P}_2) \vee (v_m \in \mathcal{P}_2 \wedge v_m \notin \mathcal{P}_1)$. ■

Note that two paths $\mathcal{P}_1$ and $\mathcal{P}_2$ are non-equal if $\{\mathcal{P}_1 \setminus \mathcal{P}_2\} \cup \{\mathcal{P}_2 \setminus \mathcal{P}_1\} \neq \phi$. In the example of figure 3, the paths {**In, Full Adder, Half Adder, Out**} and {**In, Full Adder, Exor, Half Adder, Out**} are two reconvergent paths, with **Full Adder** as their common divergent vertex and **Half Adder** as their common convergent vertex. Reconvergent paths define a subgraph of the leaf cell graph referenced further as a reconvergence.

Definition 2.6 A *reconvergence* is an induced subgraph $\mathcal{R}'(L', I', \psi', \Delta')$ defined as follows:
Consider 2 reconvergent paths $\mathcal{P}_1$ and $\mathcal{P}_2$, with a common divergent vertex v_d and a common convergent vertex v_c. Let $\mathcal{P}_t$ be a path from v_d to v_c.
Let $F = \{v \in L | v \in \mathcal{P}_t \cap \mathcal{P}_1 \vee v \in \mathcal{P}_t \cap \mathcal{P}_2\}$
Let $I' = \{i \in I | i :< v, w >$ such that $v \in F \wedge w \in F\}$
Then the graph $\mathcal{R}' = \mathcal{R}[I']$, the induced subgraph of $\mathcal{R}$ by I', defines a reconvergence. ■

In figure 3, the subgraph induced by {**Full Adder, Exor, Half Adder**} is such a reconvergence. To be able to use reconvergence in a meaningful way, one more definition is needed:

Definition 2.7 The *size* of a reconvergence is the maximum number of edges between the divergent and the convergent vertex of the subgraph that defines the reconvergence. ■

An algorithm to detect reconvergences is described in [7]. It works on the leveled (topologically sorted) leaf cell graph, where all the vertices on the same level are labeled with the same number, *level*. The size of a reconvergent subgraph is easily computed as the difference of the level of its convergent vertex and the level of its divergent vertex.

To use the reconvergence in an efficient way for timing view generation, it has to be combined with the existing hierarchy to avoid the loss of the knowledge which might be introduced by the user. Hence a criterion has to be developed that respects the hierarchy as a hint by a knowledgeable user on the one hand, and on the other hand that is able to direct the creation of timing views in a sensible manner if this is not the case. The following arguments have to be taken into account:

1. The generation of a timing view of a subgraph of the leaf cell graph is more expensive in terms of cpu-time if many paths have to be checked and/or eliminated in a large subgraph.

2. It can be very rewarding if a subgraph occurs often: in that case a timing view generation can eliminate a lot of unsensitizable paths in one sweep.

Therefore the new heuristic is:
If there are reconvergences in the leaf cell graph, make timing views of the smallest one. If there are several of the same size, first take the one that occurs most. Should there still remain several possibilities, the one whose divergent vertex has the highest level is chosen arbitrarily.
This new criterion will allow us to manipulate hierarchy. For each vertex in the graph a timing view is available. Hence, creating a new timing view means replacing a subgraph by one vertex.

The procedure to perform an optimal timing view generation according to the reconvergence criterion is described below. It makes timing views of the smallest reconvergences in the graph, and if there are several of equal size it chooses the one closest to the output, i.e. the one whose divergent vertex is at the highest level.

make_optimal_timing_view (*a_leaf_cell*) {
1. set up the leveled leaf cell graph of *a_leaf_cell*: $\mathcal{R}(L, I)$;
  Initialisation: *sub_graph*= $\mathcal{R}(L, I)$;
2. find_reconvergences ($\mathcal{R}(L, I)$);
  $\forall$ vertices $vo \in L$, indegree(vo) > 1, starting at the highest level {
    if (lrec (*vo*) $\neq \phi$) { *vi* = smallest_reconvergent (*vo*);
      if (size (*vi*, *vo*) < size (*sub_graph*) *sub_graph* = find_subgraph (*vi*, *vo*);} }

generate a timing view for *sub_graph*;
create a new vertex *newv* for *sub_graph*;
replace all instances of *sub_graph* in $\mathcal{R}(L, I)$ by *newv*;
re-level $\mathcal{R}(L, I)$;
3. if (no reconvergences left) write the event graph of the leaf cell graph in a timing view;
else goto 2.;
}

The procedures smallest_reconvergent(*vo*) searches the closest divergent vertex that defines a reconvergence with *vo*, and find_subgraph(*vi*, *vo*) extracts the induced subgraph they define. More details on the actual procedures can be found in [7].

3 Conclusion

In this paper a method for fast and accurate timing verification has been presented. The novel use of reconvergence analysis allows to guide hierarchy manipulation for optimal performance. The approach was demonstrated on a complex direction detection circuit. From these results it is clear that our approach to hierarchy in timing verification, targeted towards the LSP algorithm, leads to accurate analysis in acceptable cpu times.

References

[1] 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.

[2] 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.

[3] P. C. McGeer and R. K. Brayton. Provably correct critical paths. In *Decennial Caltech Conference on VLSI*, pages 119–142, 1989.

[4] 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.

[5] P. Johannes, P. Das, L. Claesen, and H. De Man. Slocop-II: improved accuracy and efficiency in timing verification, based on logic functionality and MOS circuit hierarchy. In *Proc. European Solid State Circuits Conference*, pages 248–251, 1989.

[6] P. Johannes, L. Claesen, and H. De Man. Performance through hierarchy in static timing verification. In *Proc. of the 12th World Computer Congress, Madrid, Spain; Vol. 1: Algorithms, Software, Architecture*, pages 703–709, 1992.

[7] P. Johannes. *Delay Characterisation and Hierarchical Timing Verification for Synchronous Circuits.* PhD thesis, Katholieke Universiteit Leuven, October 1992.

[8] A. van der Werf, B. T. McSweeney, J. L. van Meerbergen, P. E. R. Lippens, and W. F. J. Verhaegh. Flexible datapath compilation for PHIDEO. In *Proc. of EURO-ASIC*, May 1991.