Timing Analysis with Implicitly Specified False Paths

Eugene Goldberg
Cadence Berkeley Labs,
Berkeley,California
egold@cadence.com

Alexander Saldanha
Cadence Berkeley Labs,
Berkeley, California,
saldanha@cadence.com

Abstract

We consider the problem of timing analysis in the presence of known false paths. The main difficulty in adaptation of classical breadth-first search to the problem is that at each node one has to store the number of delays which is proportional to that of false paths going through the node. We propose a reduction technique that allows one to drastically reduce the number of delays to store. In particular, the technique can be applied when false paths are implicitly specified by a set of through-path exceptions or false sub-graphs. In addition, we introduce a new data structure for representing false paths called abstract false graphs which are as expressive as false sub-graphs but are as compact as through-path exceptions. A preliminary prototype implementation illustrates the potential benefits of our reduction technique by showing up to exponential reduction in memory usage and run-time over previous work.

Introduction

Given a logic circuit with a delay assignment on each of the gates and wires in the circuit, timing analysis is employed to determine the longest (or critical) paths of the design. It is well known that a depth-first (DFS), breadth-first (BFS), or best-first search algorithm may be utilized for this task [2,3,4]. However, it is often required that false paths be neglected from the timing analysis. In this paper we address the problem of timing analysis in the presence of user-specified false paths. There are two issues to be addressed in solving this problem efficiently: (i) the false paths must be specified and represented compactly (ii) the search algorithm must be modified to account for the false paths. This paper makes contributions on both issues. Note that we assume that the false paths have been identified prior to timing analysis using manual or automatic techniques.

Since BFS is the fastest of the known timing analysis algorithms, it is natural to try to adapt it to the case of timing analysis with false paths. The problem of modifying BFS to account for false paths specified by false sub-graphs during timing analysis has been addressed in [1]. It is shown that the number of delay values to be stored at a node is bounded by the number of false paths including this node. This potentially limits the application of the BFS algorithm. In this paper we develop methods to reduce the number of delay values to be stored at a node.

The first contribution of this paper is to present a timing analysis algorithm for which the number of delay values to be stored at a node can be reduced. We describe and prove the conditions under which the reductions retain the accuracy of the timing analysis. We formulate our reduction rules for the case when false paths are specified explicitly. We extend the rules to the case when false paths are specified implicitly by false sub-graphs or through-path exceptions.

Application of our reduction technique to implicitly represented false paths is based on the fact that a false sub-graph or a through-path exception specify a Cartesian set of false paths. We show that if the set of false paths is specified by *k* Cartesian representations the number of delays to store at a node is bounded by 2^k. (This result generalizes the one obtained in [1] for false sub-graphs.) So to make timing analysis efficient it is crucial to represent the set of false paths as the union of the minimum number of Cartesian sets of paths. From this point of view it is important to have a representation such that any set of paths specified by it is Cartesian and any Cartesian set of paths can be specified by this representation. We show that false sub-graphs satisfy these two requirements and through-path exceptions do not. However false sub-graphs are too redundant. The second contribution of our paper is that we introduce a new data structure for representing false paths called *Abstract False Graphs* (AFGs) which are as expressive as false sub-graphs but are as compact as through-path exceptions.

In the paper we focus only on the problem of finding the delay of the longest path that is not false. Another problem of great interest in timing analysis is to compute the required arrival time (RAT) for each node in the circuit [3]. The RAT and delay at a node *n* can then be used to compute the slack-time at *n,* which is an important measure of timing violation. All the theory we develop further for finding delays can be easily extended and applied to computing the RAT.

Problem formulation

A circuit *C* is represented by a directed acyclic graph *G* (termed the *circuit graph*) whose nodes correspond to gates in *C* and whose edges correspond to connections between the gates. While a delay may be associated with each gate and connection in *C*, without any loss in generality, we assign delays only on edges of *G*. Primary inputs (outputs) of C correspond to sources (sinks) of *G*. An example of the circuit graph with the source node *a* and the sink node *c* is shown in Figure 1.

A complete path in *G* starts at a source node and terminates at a sink. In this paper the term false path is only applied to complete paths and denotes any user specified path that must be ignored in timing analysis. An incomplete path will be called sub-path. We use the notation ${}_nH$ to denote a sub-path that starts at a source node and terminates at node *n* and T_n to denote a sub-path from *n* that terminates at a sink node. The notation ${}_nH.T_n$ denotes the complete path formed by the catenation of the sub-paths ${}_nH$ and T_n.

True paths: Given a set Φ of false paths, a path is called true if it is not contained in Φ.

Problem: Given a graph *G* with delays assigned to edges and a set Φ of false paths, find the longest true path.

Reduction for explicit representation

False sub-paths: Given a set of false paths Φ**,** a sub-path is false if there is a false path in Φ that contains this sub-path. Otherwise the sub-path is called true.

In the absence of false paths, a single delay value equal to the length of the longest sub-path up to the node needs to be stored at each node. If we want to disregard a set of false paths Φ, several delay values may have to be stored at a node. Let $\Delta({}_nH)$ denote the delay of ${}_nH$. Suppose that ${}_nH$ and ${}_nH'$ are two sub-paths such that $\Delta({}_nH) > \Delta({}_nH')$. If we drop delay $\Delta({}_nH')$ it may turn out that the longest true path is ${}_nH'.T_n$. This may happen if ${}_nH.T_n$, which is longer than ${}_nH'.T_n$, is false (and so ${}_nH$ is a false sub-path). It means that in general, given $\Delta({}_nH) > \Delta({}_nH')$, we can't drop $\Delta({}_nH')$ if ${}_nH$ is a false sub-path. The simplest solution is to store *r* extra delays at *n* where *r* is the total number of false sub-paths ending at node *n*.

Proposition 1: [1] Given a set of false paths Φ the total number of extra delays to store at a node *n* is no more than the number of false paths that include *n*.

We can reduce the number of values to store required by Proposition 1 by applying a form of dominance between false sub-paths. This reduction approach is new.

False (true) tails: T_n is a false (true) tail of a sub-path ${}_nH$ if ${}_nH.T_n$ is a false (true) path.

Let *False_Tails*$({}_nH)$ (*True_Tails*$({}_nH)$) denote the set of all false (true) tails of ${}_nH$. Obviously, if *False_Tails*$({}_nH)$ is not empty then ${}_nH$ is a false sub-path. Similarly, if ${}_nH$ is a true sub-path then all its tails are true.

Let ${}_nH$ and ${}_nH'$ be two paths such that $\Delta({}_nH) > \Delta({}_nH')$. We need to store the value $\Delta({}_nH')$ at *n* only if there exists a path from *n*, say T_n such that path ${}_nH.T_n$ is false while ${}_nH'.T_n$ is not. However, this will never happen if *False_Tails*$({}_nH') \supseteq$ *False_Tails*$({}_nH)$. In this case we can safely drop delay $\Delta({}_nH')$.

We can generalize the reduction rule based on the idea of false sub-path dominance.

Proposition 2: Let ${}_nH^1, {}_nH^2, .., {}_nH^k$ be the set of all sub-paths such that $\Delta({}_nH^i) \geq \Delta({}_nH^1)$, $i=2, \ldots, k$. We do not need to store the value $\Delta({}_nH^1)$ at *n* if $\cup_{i=2,\ldots,k}$ *True_Tails*$({}_nH^i) \supseteq$ *True_Tails*$({}_nH^1)$.

Cartesian representations

The set Φ of false paths is usually specified implicitly. Let us consider two representations commonly used for specifying false paths.

Through-Path Exception: A through-path exception F_T is a subset of nodes in the circuit graph G such that any path of *G* that includes all the nodes in F_T is specified to be a false path.

In Figure 1 through-path exceptions *{a,b,s}* and *{a,b,p}* specify the 4 false paths listed in the top right corner.

False Sub-Graph [1]**:** A false sub-graph F_S is a connected sub-graph of the circuit graph *G* with a begin set $B(F_S)$ and end set $E(F_S)$ of nodes such that each node in $B(F_S)$ has no incoming edges in F_S and each node in $E(F_S)$ has no outgoing edges in F_S. A complete path P of G is specified to be a false path if P contains at least one path in F_S that starts at a node in $B(F_S)$ and ends at a node in $E(F_S)$.

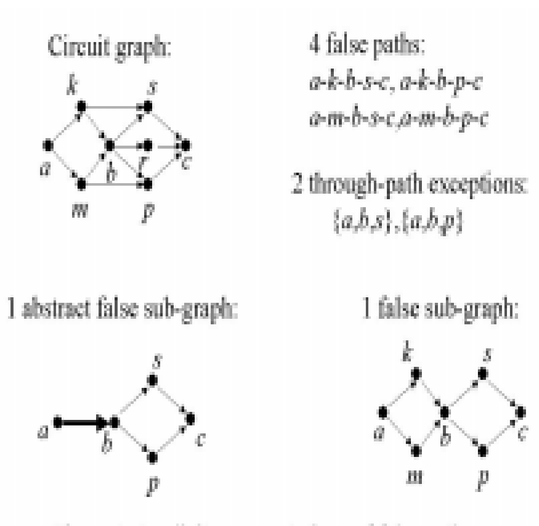

Figure 1: Implicit representations of false paths

An example of the false sub-graph specifying the same 4 false paths as the through-path exceptions *{a,b,s}* and *{a,b,p}* is shown in Figure 1.
Let F be a false sub-graph or a through-path exception. Denote by $\Phi(F)$ the set of all false paths specified by F. A sub-path in G satisfies F if it is contained in a path from $\Phi(F)$.

Cartesian set of paths: A set of false paths Φ is Cartesian if for any node n of circuit graph G and any two false sub-paths ${}_nH$, T_n, ${}_nH.T_n$ is a false path from Φ.

If Φ is Cartesian then the subset of paths from Φ going through a node n can be represented as ${}_n\boldsymbol{H^*} \times \boldsymbol{{}^*T_n}$ where ${}_n\boldsymbol{H}^*$ and $\boldsymbol{{}^*T_n}$ are the sets of all false sub-paths ending and starting at n, respectively. Sympol '$\times$' denotes the Cartesian product yielding the set of all paths that can be obtained by catenation of a sub-path ${}_nH$ from ${}_n\boldsymbol{H^*}$ and T_n from $\boldsymbol{{}^*T_n}$.

The set of 4 false paths shown in Figure 1 is Cartesian and can be represented as $\{a\text{-}k\text{-}b,\ a\text{-}m\text{-}b\}\times$ $\{b\text{-}s\text{-}c,\ b\text{-}p\text{-}c\}$.

Cartesian representation: An implicit representation is called Cartesian if it specifies only Cartesian sets of paths.

From the definitions it follows that the set of false paths represented by a single false sub-graph or a through-path exception is Cartesian. This means that false sub-graphs and through-path exceptions are Cartesian representations. We use this fact in the next section.

Reduction for implicit representation

Let F be a Cartesian representation specifying the set $\Phi(F)$ of paths to disregard. Then for any two false sub-paths ${}_nH$, ${}_nH'$ satisfying F $\quad$ $False\_Tails({}_nH) = False\_Tails({}_nH')$. Since the set of false tails of any false sub-path ${}_nH$ satisfying F is the same, we will denote the set by $False\_Tails(n,F)$.

Let the set of false paths be specified by a set F_1, ..., F_k of Cartesian representations *i.e.* the set of false paths is $\cup_{i=1,\dots,k}\ \Phi(F_i)$.

Denote by ${}_nH(S)$ a false sub-path that satisfies each representation from a subset S of $\{F_1,..,F_k\}$. *Let* $False\_Tails(n\ ,\ S) = \cup_{F_i \in S} False\_tails(n,\ F_i)$. Since every F_i specifies a Cartesian set of false paths $False\_Tails({}_nH(S))= False\_Tails(n,\ S)$. This means that if ${}_nH(S)$ and ${}_nH'(S')$ are two sub-paths ending at n and $S\subseteq S'$ then $False\_Tails({}_nH(S)) \subseteq False\_Tails({}_nH'(S'))$. Hence by Proposition 2 we can drop delay $\Delta({}_nH'(S'))$ if $\Delta({}_nH'(S')) \leq \Delta\ ({}_nH(S))$.

This means in particular that for all paths satisfying the same set S of representations, we need one corresponding delay value at a node. So if we apply Proposition 2 only to paths satisfying the same set of false graphs then the number of extra delays at a node n is equal to the number of subsets S of $\{F_1,..,F_k\}$ such that there is at least one sub-path ${}_nH$ satisfying only representations from S. Since the number of different subsets of S is 2^k, it is an upper bound on the number of values to store at a node. This result was obtained for false sub-graphs in [1]. We now show that Proposition 2 yields further dramatic reduction of the number of values to store.

Denote by S^* the set of all representations F_i satisfied by at least one path ${}_nH$. Denote by $True\_Tails(n,\varnothing)$ the set of tails T_n not satisfying any of representations F_i. It means that if T_n is in $True\_Tails(n,\varnothing)$ then for any sub-path ${}_nH$ path ${}_nH.T_n$ is true. If S is the set of representations satisfied by ${}_nH$ then $True\_Tails({}_nH(S)) = False\_Tails(n,\ S^* \setminus S\ ) \cup True\_Tails(n,\varnothing)$. The latter is true because ${}_nH.T_n$, where $T_n \in False\_Tails(n,\ S)$, is true iff T_n does not satisfy any representation in S. That is T_n either satisfies only representations from $S^* \setminus S$ or doesn't satisfy any representation from S^* at all.

Proposition 3: Let ${}_nH^1(S_1)$, ${}_nH^2(S_2)$,.., ${}_nH^k(S_k)$ be the set of all sub-paths such that $\Delta({}_nH^i(S_i)) \geq \Delta({}_nH^1(S_1))$, $i=2, \ldots, k$. We do not need to store the value $\Delta({}_nH^1(S_1))$ at n if $(S^* \setminus S_2) \cup .. \cup (S^* \setminus S_k) \supseteq S^* \setminus S_1$.

The larger S_1 is (i.e. the more representations are satisfied by ${}_nH^1(S_1)$) the smaller $S^* \setminus S_1$ is and the more likely it is that $\Delta({}_nH^1(S_1))$ will be dropped.

Abstract false sub-graphs

Let the set of false paths to disregard is specified by k Cartesian representations $F_1, \ldots, F_k$. Though application of Proposition 3 allows a drastic reduction in the number of delay values stored at a node, the worst case storage requirement remains at 2^k. It is thus very important to minimize the value of k. Given a set of false paths, a minimum value of k corresponds to a representation that is the union of the smallest number of Cartesian sets of false paths.

If false paths are specified implicitly it is impor-tant to select a Cartesian representation that is expressive enough to specify any Cartesian set of false paths. False sub-graphs satisfy this requirement while through-path exceptions does not.

Proposition 4. Any Cartesian set of false paths can be represented by a false sub-graph.
Proposition 5. There is a Cartesian set of paths that cannot be specified by only one through-path exception.

It not hard to check that the Cartesian set of 4 paths shown in Figure 1 can't be represented by a single through-path exception.

Though false sub-graphs are more expressive than through-path exceptions they may not be very compact. Suppose we want to disregard the set of all paths going through nodes a and b. It can be represented by the through-path exception consisting of just two nodes a, b. On the other hand, to represent this set of false paths by a false sub-graph requires the inclusion of the nodes and edges of the circuit graph included in all sub-paths of G starting at a and ending at b.

We describe a new data structure named Abstract False Graph (AFG) which is as expressive as false sub-graphs and as compact as exceptions. The key idea is to supplement false sub-graphs with abstract edges.

Abstract edge: The edge (x, y), where x, y are nodes of the circuit graph G, is called abstract if there is no edge between the nodes x and y in G.

An example of the AFG specifying the same set of false paths as the false sub-graph on the right is shown in Figure 1. In the AFG we replace edges (a,k),(k,b),(a,m),(m,b) with one abstract edge (a,b) since all sub-paths in the circuit graph beginning at a go through node b. Adding abstract edges allows the AFG to use an optimal combination of means employed by through-path exceptions and false sub-graphs to specify false paths.

Satisfiability of an AFG by a path: A path P in G satisfies an AFG F if there is a path R in F such that:

1. if an edge of (a, b) of R is abstract then nodes a and b are contained in P; and
2. if an edge (a, b) of R is contained in G then (a, b) is an edge of P.

An AFG is not just a false sub-graph supplemented with abstract edges. If we do not restrict the use of abstract edges, there is a chance that the set of paths specified by an AFG will not be Cartesian. Suppose, for example, that the set of false paths is specified by AFG F consisting of two abstract edges (a, b) and (c, d). Assume that in the circuit graph G there is node n and edges (a, n), (n, b) and (c, n), (n, d). Then F specifies any path containing sub-path a-n-b or c-n-d as false while a path containing sub-paths c-n-b or a-n-d is true. Let sub-paths ${}_nH$ and T_n contain edges (c, n) and (n, b) respectively. Either sub-path is false because there is a false path containing it. On the other hand, path ${}_nH.T_n$ is true and so the set of paths specified by F is not Cartesian.

The definition of AFG below is meant to guarantee that AFG's are equivalent to false-sub-graphs in their expressiveness and so they can represent any Cartesian set of paths and any set of paths specified by an AFG is Cartesian.

Let $level(n)$ denote the level of node n in graph G. The level of n is equal to 1 if n is a source of G. Otherwise $level(n)$ is equal to 1 plus the maximum level of its fanin nodes. If L is a set of nodes, $min\_level(L) = min \{ level(n) \mid n \in L\}$ and $max\_level(L) = max \{ level(n) \mid n \in L\}$.

Abstract False Graph: F is an abstract false graph (AFG) of a circuit graph G with a begin set $B(F)$ and end set $E(F)$ of nodes such that each node in $B(F)$ has no incoming edges in F and each node in $E(F)$ has no outgoing edges in F, if the set of nodes in F can be partitioned into p subsets of nodes, called *layers* and denoted as $L_1,..,L_p$, such that for each i, $1 \leq i < p$:

(i) $min\_level(L_{i+1}) > max\_level(L_i)$; and

(ii) nodes a, b of the same layer L_i are not connected by a path in G; and

(iii) if $a \in L_i$, $b \in L_{i+1}$ and there is no edge (a, b) in F, then (a, b) is an edge in G and there is no other path in G from a to b; and

(iv) if $a \in L_i$, $b \in L_j$ where $j > i+1$ and there is edge (a, b) in F, then (a, b) is an edge in G.

The AFG can be thought of as a subset of nodes of G partitioned into layers with nodes of adjacent layers connected by edges (abstract or "real"). The essence of the last two conditions is that we restrict what one can do with abstract edges. One cannot drop the abstract edge (a, b) between nodes a and b of adjacent layers. One cannot connect nodes a, b of non-adjacent layers by an abstract edge.

Proposition 6: Let F_S be a false sub-graph. Then we can find an AFG F such that $\Phi(F) = \Phi(F_S)$.

Proposition 7: Let F be an AFG. Then we can find a false sub-graph F_S such that $\Phi(F_S) = \Phi(F)$.

Experimental results

Our reduction technique is demonstrated using a prototype program written in APL2 on a Pentium II 300 MHz computer. Programs in APL2 are about two orders of magnitude slower than those written in C. We applied our program on rectangular meshes of size $m \times m$ having one source and one sink located at nodes (0, 0) and (m, m) respectively, where (x, y) specifies the Cartesian coordinates in the mesh. Each node (x, y) is connected to the node $(x+1, y)$ (if $x < m$) and node $(x, y+1)$ (if $y < m$). All edges have unit delays. The results of the experiment are given in Table 1. In each run, represented by a row of the table, we generated a set of AFG's and ran the program to find the longest delay path. Each AFG consisted only of two nodes: begin node (x_b, y_b) and end node (x_e, y_e).

Each pair of begin and end nodes was selected randomly to satisfy the two conditions: i) $x_b \leq x_e$, $y_b \leq y_e$; ii) $x_e + y_e - (x_b + y_b) = C$ where C is a constant. The first condition guarantees that there is a path in the mesh going through nodes (x_b, y_b) and (x_e, y_e). The second condition allows one to control the number of paths satisfying the AFG. C is the "length" of the AFG and the larger it is the more paths the AFG satisfies.
We ran the program in two modes: with and without the reduction based on Proposition 3. We compare run times and average numbers of delays to store. The results show that the run time of the algorithm in the reduction mode grows close to linear due to the fact that the number of values stored at each node is very small. On the other hand, without reduction the run time tends to grow exponentially because of the blow-up of the number of delays to store. Note that without the reduction the algorithm mimics the one of [1].

Table 1

Name	Size	Number of AFGs	Length of AFGs	Runtime (sec)		Average number of delays per node	
				w/o reduction	with reduction	w/o reduction	with reduction
Rand1	24 x 24	40	10	17.61	9.64	11.44	1.04
Rand2	24 x 24	80	10	40.25	15.73	25.25	1.58
Rand3	24 x 24	40	15	124.91	11.13	54.73	1.34
Rand4	24 x 24	80	15	1129.67	16.77	177.88	1.34
Rand5	34 x 34	60	10	30.33	24.64	5.56	1.07
Rand5	34 x 34	120	10	68.38	41.53	15.6	1.07
Rand7	34 x 34	60	20	1358.25	27.67	140.76	1.24
Rand8	34 x 34	120	20	> 30 min	48.64	284.82	1.99
Rand9	34 x 34	60	30	> 30 min	32.70	268.56	1.34
Rand10	34 x 34	120	30	> 30 min	70.73	365.50	5.59

References

1. K. P. Belkhale, A. J. Suess. "Timing Analysis with Known False Sub Graphs," *Proc of ICCAD-95*, pp.736-740.
2. J. Cherry. "Pearl: A CMOS Timing Analyzer", *Proc. of the DAC-88,* pp. 148-153.
3. R. B. Hitchcock, "Timing Verfication and the Timing Analysis Program," *Proc. of the DAC-82,* pp. 594-604
4. T. G. Szymanski. "LEADOUT: A Static Timing Analyzer for MOS Circuits", *Proc of ICCAD-86*, pp.130-1