Removing user-specified false paths from timing graphs

David Blaauw, Rajendran Panda, Abhijit Das
Motorola, Inc., Austin, TX. E-mail: {blaauw, panda}@advtools.sps.mot.com

Abstract

We present a new method for removing user-specified false subgraphs from timing analysis and circuit optimization. Given a timing graph and a list of specified false paths, false subpaths, or false subgraphs, we generate a new timing graph in which all specified false paths are removed using a process of node splitting and edge removal. We present the necessary and sufficient condition for splitting a node, and show that the number of nodes that must be added to the timing graph is linear with the size of the false path specification. We also present an algorithm for finding the minimum set of nodes that must be split. Since this algorithm requires exponential run time for false subpaths and false subgraphs, we present a heuristic splitting approach which has linear worst-case run time, and where the number of added nodes is linear with the size of the false path specification. The heuristic approach was implemented and results are given for large industrial circuits.

1. Introduction

Static timing analysis has become an integral part of the timing verification and optimization of large digital IC designs. However, static timing analysis may include false paths in its analysis, and this results in an overly pessimistic timing report. Many circuit optimization tools, such as those used for transistor and gate sizing[1], use static timing analysis in the inner loop of their optimization. In this case, the presence of false paths unnecessarily constrains the optimization problem and leads to either a suboptimal solution or a complete failure to meet timing constraints. Therefore, effective removal of false paths from static timing analysis is critical. Furthermore, since static timing analysis is part of the inner loop of the optimization, the false paths must be accounted for efficiently to ensure that overall performance of the optimization is not significantly compromised.

Extensive research has been done on the problem of identifying false paths which arise in a circuit due to reconvergent fanout. These false paths are caused by logic and temporal correlations between the circuit nodes. Although the complete identification of all such false paths in a circuit is an NP-complete problem, a number of exact or approximate methods have been proposed[2-7].

Also important are user-specified false paths. These are paths that may be logically and temporally sensitizable but are unimportant to the intended operation of the circuit. For instance, a path between two latches may be false when the clocks that drive the latches are asynchronous with respect to one another. Another example is a path which is part of a reset or scan circuit. Since such false paths rely on specific information unavailable to the static timing analysis tool, they must be manually identified.

In this paper, we present an efficient approach for removing a set of specified false paths from timing analysis for use in the inner loop of circuit optimization. The set of false paths can be specified as complete paths, subpaths, or subgraphs, and will be referred to as *false paths specification*. The actual identification of false paths can be performed either manually or automatically, and is not discussed in this paper. We propose a method which, given a directed, acyclic, timing graph and a false paths specification, generates a new timing graph (called the *true* timing graph) from which all specified false paths have been removed.

One approach for removing specified false paths from timing analysis is to simply filter them from the timing report. However, most circuit optimization tools require not only the identification of the true critical path, but also the true slack of all nodes in the circuit. True slacks are not available in the path filtering approach. In [8], an approach for removing known false subgraphs from timing analysis is proposed. This method uses additional bookkeeping during the propagation of arrival and required times. However, the method has a worst-case complexity that is exponential with the number of specified false subgraphs. Also, it is not clear how easily other timing algorithms such as incremental timing analysis, multicycle path handling, and top critical path enumeration can be extended with this approach. In [9], an improvement of the algorithm in [8] is proposed. The run time is significantly reduced, but the worst-case run time remains exponential with the number of false subgraphs.

In this paper, we use an approach based on node splitting. By splitting nodes and removing certain edges from the graph, it is possible to isolate and remove the specified false paths without removing any true paths. An advantage of this approach is that all timing analysis algorithms can be directly applied to the newly generated true timing graph without any modification to these algorithms. The cost of generating the true timing graph is incurred only once, and is amortized over the large number of timing analysis invocations during circuit optimization. The runtime penalty during timing analysis is linearly related to the number of added nodes in the new timing graph, which, in turn, is linear with the size of the false path specification. Thus, the proposed approach is very attractive for use in an optimization framework, unless the optimization procedure alters the false paths, either through timing changes or logic changes.

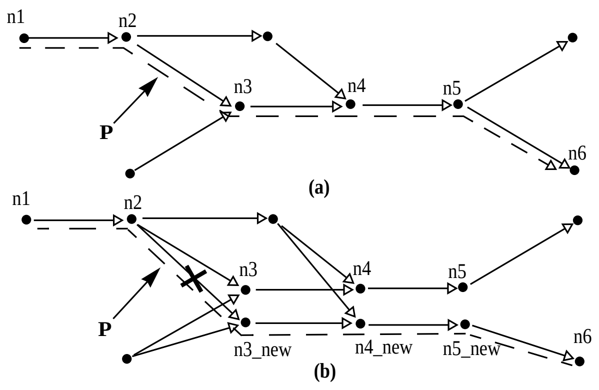
Earlier work in node splitting was reported in [10,11] where it was shown that a path can be removed from a timing graph by splitting nodes along the path from the last node with multiple fanout to the circuit input nodes. This approach yields a number of new nodes that is linear with the number of false paths. However, for false subpaths and false subgraphs, splitting all nodes from the first multiple fanout node to the primary input will yield an exponential number of new nodes in the worst case. Our approach presented in this paper uses a similar splitting method. However, it is shown that *not all* nodes from the last multiple fanout node to the input node need to be split. We present the *necessary and sufficient* condition for splitting a node, and then show how this condition is satisfied for a false subgraph by splitting only the nodes that lie on the false subgraph. Using this method, the number of new nodes added to the timing graph is linear with the size of the false path specification. Finally, in [12], a method for removing false paths from a timing graph through node splitting was introduced for false paths and false subpaths. This method is computationally expensive as it relies on path counting to determine which nodes must be split and which edges can be removed. Also no algorithm is provided to remove false subgraphs from the timing graph.

In this paper, we first present the necessary and sufficient condition for splitting a node in the timing graph. Using this condition, we then show that only a subset of nodes that lie on false subgraphs need to be split to remove all false paths from the timing graph. We then present an algorithm for determining the minimum set of nodes that must be split. This algorithm has worst-case exponential complexity. We therefore present a heuristic that has a quadratic run time and generates a number of nodes that is bounded linearly by the size of the false paths specification.

2. Removing false paths through node splitting

Definition. An edge is a *true edge* if every path through it is a true path. It is a *false edge* if every path through it is a false path. It is a *stained edge* if one or more false paths (and possibly some true paths) pass through it. Thus, the false edges in a timing graph are a subset of the stained edges in that graph.

Our approach for generating a false-path-free timing graph relies on the following simple observation: If an edge is false, it can be deleted, thus removing every false path through that edge without affecting any true timing path in the circuit. Conversely, to remove a false path it must have at least one false edge.We illustrate our approach in Figure 1(a), which shows a timing graph with a single false path P. An edge *e* along this false path can be removed if all nodes on the false path with multiple fanin lie topologically after *e,* and all nodes along the false path with multiple fanout lie topologically before *e*. This condition guarantees that no paths other than the false path use *e,* and by removing *e* from the circuit we remove only the false path. We define the first node along a false path which has multiple fanin as the *first-fanin-node* (FFI) of the path, and the last node along the false path that has multiple fanout as the *last-fanout-node* (LFO) of the path. In order for a false path to contain at least one edge which can be removed, it is necessary that the FFI node of the path lies topologically *after* the LFO node of the path. In Figure 1(a), the FFI node of path P is node n3 and the LFO node is n5. Since the FFI node for path P occurs topologically *before* its LFO node, path P contains no edges that can be removed without also removing some true paths. The following lemmas provide sufficient conditions for removing edges from a timing graph without affecting any true path.



Lemma 1. If, in a false path P, the LFO node lies before the FFI node, then P can be eliminated (and all other paths preserved) by deleting every edge on P between the LFO and FFI nodes.

Proof. For an edge of P lying between the LFO and FFI, there is exactly one path, P, passing through it, as there is no fan-in before and no fan-out after the edge. Thus, every edge between the LFO and the FFI is false and can be removed.

Lemma 2. A fan-in (fan-out) free false path P can be eliminated (and all other paths preserved) by deleting every edge on P occuring after (before) the LFO (FFI) node.

Lemma 3. A fan-in free, fan-out free false path P can be eliminated (and all other paths preserved) by deleting every edge on P.

Lemmas 2 and 3 are simply special cases of Lemma 1, which provides a sufficient condition for removing an edge. We call the false paths that can be eliminated by the application of Lemmas 1-3 as ‘simple false paths’.

The basic idea of our approach is this: When the sufficient condition is not met for some path, we transform the graph such that the condition is satisfied for some edges on the false path, allowing that false path to be eliminated. We do this by splitting all nodes from the LFO node to the FFI node in a manner similar to that described in [10] and [12] and as illustrated in Figure 2.

For each stained fanout edge e_i of node *N* which lies on one or more false paths, we generate a new node, N_i. In Figure 2, e_1 and e_2 are the stained fanout edges of node N. So, we generate two new nodes N_1 and N_2, corresponding to these fanout edges. To N_i, we then create new edges from all fanin nodes of node *N*, and move the fanout edge

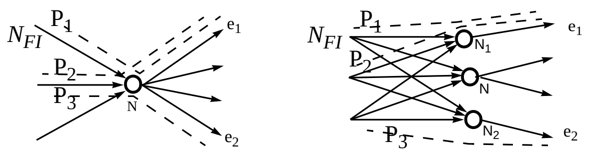


Figure 2. A node before and after node splitting.

e_i from node *N* to node N_i. After splitting node *N*, the false paths P_i (P_1, P_2, and P_3 in the figure) now pass through the newly created nodes N_i (nodes N_1 and N_2 in the figure) and no longer have multiple fanout at this node. Therefore, if node *N* is the LFO node of a path *P,* then after splitting, it is no longer the LFO. Also, the fanin node of N_i along P (node N_{FI}) has multiple fanout edges after splitting, since the fanin edges of *N* have been duplicated. Thus, if node *N* was the LFO node of *P* before splitting, node N_{FI} is the LFO of *P* after splitting node *N*. The LFO node of *P* has been moved exactly one node toward the input along path *P.*

It follows that, by splitting all nodes from the LFO node to the FFI node in reverse topological order, the LFO node will be exactly the fanin node of the FFI node along *P*, and the edge between them can be removed. This is illustrated in Figure 1, where nodes *n5*, *n4,* and *n3* are split in that order. After splitting, node *n2* is the LFO node of *P* and lies before the FFI node of P (*n3_new*). The edge between *n2* and *n3_new* can then be removed to eliminate *P* from the timing graph. The node splitting procedure is given below.

Procedure 1. **Node Splitting.** Consider a node *N* with stained fan-out edges $E = \{e_1,.....,e_k\}$. The splitting of N is done in the following sequence of steps:

(i) Add *k* nodes N_1, ..., N_k. (ii) Connect all fan-in nodes of *N* to each of N_1, ..., N_k. (iii) For $i=1,...k$, move fan-out e_i from *N* to N_i.

The above node splitting procedure has the following properties which we will use to eliminate the false paths:

Path preservation property: The procedure preserves all the paths in the original graph, except that node *N* is replaced by one of N_1 through N_k on some paths.

Fan-out transfer property: The *k* stained fan-outs of *N* are transferred to each fanin node of *N*. N_1 through N_k are now fan-out free, and *N* has only true fan-out edges. Note that the fan-ins of every node in the graph have remained the same.

It may first seem that the node splitting procedure will add an exponential number of new nodes. A careful analysis will reveal that the number of nodes added is bounded by N, where N is the number of edges in the false paths specification. When a node *N* is split, the number of newly generated nodes is equal to the number of fanout edges of *N* that lie on one of more false paths (stained edges). Splitting node *N* also increases the number of stained fanout edges of the fanin nodes of *N*. Thus, by splitting nodes in reverse topological order we increase the number of edges in the timing graph for the next level of nodes that will be split. However, the total number of stained edges itself does not increase as a result of node splitting, as can be seen from Figure 2. Also, the number of stained edges can never exceed the number of edges in the false path specification. Since only one new node is created at the most for every stained edge, the number of nodes added in the true timing graph is bounded by the number of edges in the false path specification.

Theorem 1. Let *P* be a false path in circuit graph ***C*** with N_{FFI} and N_{LFO} as the FFI and LFO nodes on *P*, such that either N_{FFI} and N_{LFO} are one and the same node, or N_{LFO} occurs after N_{FFI}. The false path *P* can be eliminated, and all other paths preserved, by splitting every node from N_{LFO} up to N_{FFI} (both inclusive), in that order, using Procedure 1, and then deleting the stained edge on *P* that is fanning into one of the split nodes of N_{FFI}.

Proof: The path preservation property guarantees that all paths in the original graph are preserved after every splitting. Let S be the split node of N_{FFI} through which P is passing in the new graph obtained after all splitting. Due to the fan-out transfer property of Procedure 1, and the order in which we split the nodes (viz. from N_{LFO} up to N_{FFI} in that order) path P in the new graph is guaranteed to be fan-out free from S onwards. Thus the node fanning into S and lying on P (call it N_{in}) is now the LFO node of P. Moreover, since the fan-ins of no node in the graph has changed, S will be the FFI of P. By lemma 1, P can be eliminated, and all other paths preserved, by deleting the edge between N_{in} and S.

Corollary 1. Suppose $\boldsymbol{C}$ is a circuit graph with no *simple* false paths (those paths which can be eliminated by application of Lemmas 1-3), and $\boldsymbol{P}$ is the set of all false paths in $\boldsymbol{C}$. Let $\boldsymbol{N}$ be the union set of nodes from the FFI up to the LFO (both inclusive) of every path P in $\boldsymbol{P}$. All paths in $\boldsymbol{P}$ can be eliminated, and all other paths preserved, by first splitting every node in $\boldsymbol{N}$ using Procedure 1 in an order guaranteeing that a node is not split after any of its fan-in nodes is split, and then deleting the stained fan-in edges of every path fanning into one of the split nodes of the FFI of that path.

Property. **Fan-out free tail creation**. Call the set of edges of a false path P from a node N through the last edge the *tail* of P at N. The splitting of all nodes from LFO up to and including N in that order creates a tail of P at some N_k (a split node of N) such that this tail is fan-out free.

Procedure 2. **Elimination of all false paths.**

(i) Levelize the circuit graph $\boldsymbol{C}$. (ii) Construct a set of nodes $\boldsymbol{N}$ consisting of all nodes from the FFI to the LFO of every false path in $\boldsymbol{P}$. (iii) Split every node in $\boldsymbol{N}$ in a levelized manner from output to input. (iv) Delete the edges on each path that are fanning into the split nodes of the FFI nodes of every path.

We will now show that we can do better than what is suggested by Procedure 2. We will introduce the notions of 'true FFI' and 'true LFO' to help develop the necessary concepts.

When multiple false paths are present in a timing graph, the FFI and LFO nodes can be defined more precisely as the first node along a false path P where a true path joins path P, and the last node along path P where a true path leaves P, respectively. We earlier defined the *tail* of path P at node N, where N is a node on P, as all edges of P from node N through the last edge of P. Likewise, we define the *head* of P at node N as all edges of P from the first edge of P through the fanin edge of node N. We are now ready for defining the notion of *true FFI* and *true LFO*.

Definition. **True First Fan-in Node:** The true FFI node of a path P is the first node N along P with one or more fanin edges *e_in* that do not belong to P, such that there is at least one true path that lies on *e_in* and the tail of P at N. If all paths that lie on *e_in* and the tail of P at N are false, then there is no true fanin path joining P at node N, and N does not qualify as a true FFI node for P.

Similarly, a node N along path P only qualifies as a true LFO node of P if, for a fanout edge *e_out* which does not belong to P, there is at least one true path that lies on *e_out* and on the head of P at N.

The following theorem is key to develop the necessary and sufficient condition for splitting a node:

Theorem 2. Let N be a node in $\boldsymbol{C}$. Suppose $\boldsymbol{T}$ ($\boldsymbol{H}$) is the set of distinct tails (heads) of all the false paths passing through N, the tails (heads) being constructed from (until) node N. Procedure 2 will eliminate all the false paths without splitting N if N meets the following conditions:

(a) Every fan-in (fan-out) edge of N is stained.
(b) For every t (h) in $\boldsymbol{T}$ ($\boldsymbol{H}$), every fan-in (fan-out) of N has at least one false path through it that has the same tail (head) as t (h).
(c) N is the first (last) fan-in (fan-out) node of all false paths passing through N.
We prove the case for paths with common tails only:

Proof: Consider the situation in which every node from the LFO nodes up to the FFI nodes on every false path, including N have been split using Procedure 2. Consider a fan-out free tail t_i at a new node

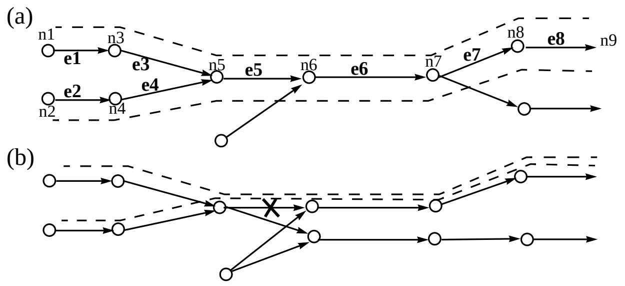


Figure 3. False paths that share a common tail.

N_i. Theorem 1 guarantees that all the false paths sharing t_i can be eliminated by deleting some of the fan-in edges of N_i. This, combined with condition (b), will delete every fan-in edge of N_i, thus making node N_i redundant.

Corollary 2. In theorem 2, condition (c) can be replaced by requiring that no true path entering N has a tail in T.

The essence of theorem 2 and corollary 2 is that any node satisfying their conditions is not a 'true' fan-in (fan-out) node, in the sense that the true paths through it are created only by fan-out (fan-in) nodes lying after (before) that node on any path. Application of theorem 2 to the FFI and LFO nodes and subsequently corollary 2 repeatedly on the topologically-next nodes has the effect of moving the boundaries (FFI and LFO) closer. That is, FFIs are pushed toward the outputs and LFOs pushed toward the inputs, thus reducing the number of nodes to be split significantly.

Figure 3 shows a simple example of a timing graph with two false paths which share a common tail at node *n5*. If each path is considered individually, node *n5* will be the FFI node and node *n7* the LFO node for both paths. Therefore, all nodes from *n7* to *n5* would be split. Using the above definitions, however, the false path on *e3* has a common tail with the false path on *e4* at node *n5*. Therefore, the first FFI node is not *n5* but *n6*, and only nodes *n7* and *n6* need to be split. Figure 3(b) shows the timing graph after both these nodes are split. Edge *e5* can now be removed to eliminate both false paths.

It can easily be shown that the set of nodes identified for splitting by Procedure 1 together with Theorem 2 and Corollary 2 is optimal. That is, the set cannot be further reduced and still cause all false paths to be removed. Thus, the necessary and sufficient condition for splitting a node is that it lies between the first 'true' fan-in and the last 'true' fan-out of any false path.

3. False subpaths and false subgraphs

A *false subpath* P with start node s and end node f potentially consists of an exponential number of *complete false paths*. However, all these false paths share a common tail at node s and a common head at node f. Thus, it is clear that the earliest possible FFI node of P is the fanout node of s, and the last possible LFO node of P is the fanin node of f. Because of this, only the nodes that lie between the start node of P and the end node of P may need to be split.

Likewise, a *false subgraph* with entry nodes S and exit nodes F consists of a set of *false subpaths* P_j, with start node s_j ε S, and end node f_j ε F. Therefore, only the nodes between s_j and f_j need to be split, as explained above. This means that for a false subgraph, only the nodes that lie on the false subgraph between the entry and exit nodes may need to be split.

From the aforesaid discussion, it is thus clear that the maximum number of new nodes that need to be added to the true timing graph is bounded by the number of edges in the false path specification for false paths, false subpaths, or false subgraphs.

The *minimum* set of nodes that need to be split can be determined using the *exact* algorithm shown below in pseudo-code.

FindFaninNodes()

```
for (all edges e) {if (e ε a false path) e->stained=T}
for (all nodes n) {n->fanin=F; if (n ε a false path) n->stained=T}
for (all stained nodes n, in C, in topological order)
    if (for any fanin edge e_in, of n, e->stained == F) n->fanin = T
    for (all fanout edges e_out, of n) {
        e_out->tail_list = tails of paths ε e_out, at n
        for (all paths P, starting at n, ε e_out->tail_list)
    for (all fanin edges e_in, of n)
        if (for all false paths FP ε e_in, FP !ε P) n->fanin = T
        if (n->fanin==T)
            for (all fanout edges e_out, of n) e_out->stained = FALSE
```

The number of paths that lie on the tail of a false subpath or subgraph can grow exponentially with the size of the timing graph. The above algorithm thus has a worst-case run time exponential with the size of the timing graph. To avoid this complexity, we also propose a heuristic approach to identify the common set of tails and heads for the false paths. In this algorithm, a set of paths is considered to have a common tail, only if these paths started at the same node. This is more restrictive than the criteria set forth in Theorem 2, and therefore, it will not find all common heads and tails. The algorithm propagates a set of path-sets along edges, each path-set having a common tail. In order to obtain a linear run time, the list of path sets is limited to a user specified constraint K, set to 50 in our experiments. The algorithm for finding common tails is shown below in pseudo-code:

FindCommonTail()

```
for (all edges e)
    e->tail_l = NULL; e->mark = F;
    e->start_set = set of paths that end at e;
    e->path_set = set of paths the lie on e;
for (all nodes n) n->c_tail=T;
for (all stained edges e, in topo-reverse order) {
    for (all fanout edges e_out of e->fanoutnode)
        for (all path_sets S, in e_out->tail_l)
            add_to_list(e->tail_l, S ∩ e->path_set)
            add_to_list(e->tail_l, e->start_set);
            if (number of elements e->tail_l > K)
                remove all path sets in e->tail_l after K;
    if (a path p ε e->path_set is not ε e->tail_l)
        e->fanin_node->c_tail=F;
    for (all tail sets S in e->tail_l)
        /* check if all fanin edges have a path in S*/
        for (all paths p, in S) p->fanin_edge->mark=T
        for (all fanin edges e_in, of e->fanin_node)
            if (e_in->mark==F) e->fanin_node->c_tail=F;
    e->mark = F;
```

We must point out that the number of new nodes added to the graph is O(N), where N is the number of edges in the specification of false path, false subpath, or false subgraph, even when the set of nodes to be split is determined by the heuristic algorithm. The heuristic (sub-optimal) procedure should not be confused as one that gives rise to the addition of an exponential number of new nodes to the graph.

4. Experimental results

The method described above for generating true timing graphs from a timing graph with user-specified false subpaths was implemented in a timing analysis and circuit optimization tool. The proposed approach was tested on industrial circuits ranging from 1,700 to 22,000 transistors and of both semi-custom and custom design styles. Table 1 shows the results of eliminating false subpaths from the delay graph for the benchmark circuits.

For each circuit in Table 1, the number of false subpaths removed is listed. Some circuits were run multiple times, each time with a different number of false subpaths. The list of false subpaths was generated by the user or through automatic means. The table shows that the number of newly added nodes in the timing graph (#*added nodes*) is easily bounded by the number of edges in the false path specification. In reality, the number of added nodes is dramatically less than the bound given by the size of the false path specification. This is due to the presence of a large number of paths with common tails and heads, and with overlapping edges. The table shows that the number of added nodes is small, even for a very large number of false subpaths. The table also shows that the run time for generating the false-path-free timing graph scales linearly with the number of false subpaths, and is less than 700 seconds for 500,000 subpaths. In practice, timing analysis will be performed many times during circuit optimization, and the run time for generating the true timing graph can easily be amortized over the circuit optimization run time.

5. Conclusions

We presented a new method for generating a false path free timing graph from a timing graph and a list of user-specified false subgraphs, false subpaths, or complete false paths. The necessary and sufficient condition for removing a false path from the timing graph was defined, and from this, the minimum number of nodes that are required to be split was derived. Since determining the minimum number of nodes to be split requires exponential run time for false subgraphs and false subpaths, a linear time heuristic was proposed. This heuristic generates a false path free timing graph where the number of added nodes is bounded by the number of edges in the false path specification. The algorithms were implemented and tested on a number of large industrial benchmark circuits. The results show that the method can easily remove a large set of specified false subpaths and subgraphs with excellent run time performance.

Circuit	# FET	# false subpath	# edges in false path specification	Time for true graph generation (sec).	# added nodes
mrrnd	1,744	49	435	< 1.0	99
mefrm	4,029	28	156	< 1.0	43
		210	2,574	< 1.0	105
		10,210	156,358	14.0	181
		32,932	456,480	36.0	195
mbbus	4,157	1,532	17,220	1.0	158
		4,320	41,362	12.0	454
mereg	4,902	2,398	13,334	5.0	405
vmtx	22,113	96	1,272	6.0	113
		224,128	2,471,936	340.0	268
		426,144	5,270,656	644.0	412

Table 1: False path removal results

References

[1] J. P. Fishburn, et.al. "TILOS: A posynomial programming approach to transistor sizing," ICCAD, Nov 1985.

[2] D. H. C. Du, et. al. "On the general False path problem in timing analysis", DAC, 1989, pp. 555-560.

[3] P. C. McGeer, et. al. "Efficient Algorithms for Computing the Longest Viable Path in a Combinational Network", DAC , 1989, pp. 561-567.

[4] S. Devadas, et. al. "Computation of Floating Mode Delay in Combinational Circuits: Theory and Algorithms", IEEE Trans. on Computer Aided Design, Dec. 1993.

[5] H. Yalcin, et. al. "An Approximate Timing Analysis Method for Datapath Circuits", ICCAD, 1996.

[6] Y. Kukimoto, et. al. "Approximate Timing Analysis of Combinational Circuits under XBD0 Model", ICCAD, 1997, pp. 176-181.

[7] Y. Kukimoto, et. al. “Hierarchical Functional Timing Analysis”, DAC, 1998, pp. 580-585.

[8] K. P Belkhale, et. al. “Timing Analysis with known False Sub-Graphs”, ICCAD, 1995, pp. 736-739.

[9] E. Goldberg, et. al. “Timing Analysis with Implicitly Specified False Path”, Int. Workshop on Timing Issues in the Specification and Synthesis of Digital Designs, T99, 1999.

[10] K. Keutzer, et. al. “Is Redundancy Necesssary to Reduce Delay”, IEEE Trans. on CAD, April 1991.

[11] A. Saldanha et. al, “Circuit structure relations to redundancy and delay: the KMS algorithm revisited”, DAC 1992, pp. 245-248.

[12] D. Blaauw, et. al. “Generation of false path free tming graphs for circuit optimization”, Int. Workshop on Timing Issues in the Specification and Synthesis of Digital Designs, 1999.