An Improved Path Enumeration Method Considering Different Fall and Rise Gate Delays[†]

José Luís Güntzel [1], Ana Cristina Medina Pinto [1], Fernando Moraes [2], Ricardo Reis [1]

[1] Universidade Federal do Rio Grande do Sul
Instituto de Informática
Porto Alegre - RS - BRAZIL
e-mail: <guntzel,cpinto,reis>@inf.ufrgs.br

[2] Pontifícia Universidade Católica do Rio Grande do Sul
Instituto de Informática
Porto Alegre - RS - BRAZIL
e-mail: moraes@andros.inf.pucrs.br

Abstract

Most of path enumeration-based timing analysis tools use a single delay per gate for path delay calculation. However, the timing analysis of current submicronic designs demands more accurate delay calculation methods, which can improve path enumeration accuracy and especially, critical delay estimation accuracy.
This article presents modifications to the classical best-first procedure proposed by Yen and collaborators [1] in order to consider a pair of delays per gate. The increase in the accuracy of path delay calculation is evaluated by running both the original and the improved path delay calculation methods on the ISCAS'85 circuits.

1. Path tracing in timing analysis

Timing analysis is a widely used method for estimating the maximum delay of a circuit, thus determining its maximum operation frequency. Its greatest advantages over circuit simulation come from the facts that it is much faster and no input vector is required. Instead of evaluating the state of each circuit component for each input vector (and at each time step), most of existing timing analysis approaches take a DAG (direct acyclic graph) representation of the circuit and enumerate the most critical paths in a non-increasing order of their delays. At the same time, the sensitizability of each path may be checked using one of the various existing sensitization criteria. Whenever an unsensitizable path (also called false path) is found it may be immediately discarded from the list because no transition can travel down it and thus the associated delay will never occur.

In recent years, substantial advances have been achieved in component delay modeling and also in path sensitization theory. However, most of timing analysis tools still use a single delay value per gate for calculating path delays in the path enumeration phase. Considering the need for even more accurate timing evaluation of submicronic CMOS designs, this is an unrealistic and too rough assumption. In reality, the delay of CMOS gates depends on the input that is undergoing a transition and on the type of transition (fall or rise). Furthermore, any CMOS gate inverts the signal phase and thus the delay of a path is found by the alternate sum of fall and rise gate delays (disregarding collisions). According to this assumption, each circuit path exhibits at least two (possibly different) delay values: one for a falling edge on its input and another for a rising edge.

The classical best-first procedure proposed by Yen and partners [1] use only one delay per gate for calculating path delays and no reference is made on how to consider more sophisticated path delay calculations while tracing paths. This path delay calculation method will be referred as *single gate delay* (**sgd**). In order to improve path enumeration accuracy, we propose the use of a *single pair gate delay* (**spgd**) method for enumerating paths of CMOS combinational circuits. This method assumes a pair of delays per gate for calculating the delay of a path, being one value associated to the output falling transition of the gate and the other, associated to the output rising transition.

2. A modified best-first procedure that considers a pair of delays per gate

The best-first procedure is the most appropriate graph traversal method for tracing paths in timing analysis because its overhead is at most logarithmic, while the overhead of the depth-first procedure (the other possible method) is at most exponential [2]. This is the main reason for choosing the best-first procedure presented in [1]. The procedure has three main steps: the creation of the circuit graph, the pre-processing phase and the path enumeration phase. Our modified best-first procedure is described in the sequel. The basic differences between the original procedure, which uses the sgd method, and the modified procedure, using the spgd method, are highlighted.

[†] This work was supported by CAPES and CNPq Brazilian Agencies.

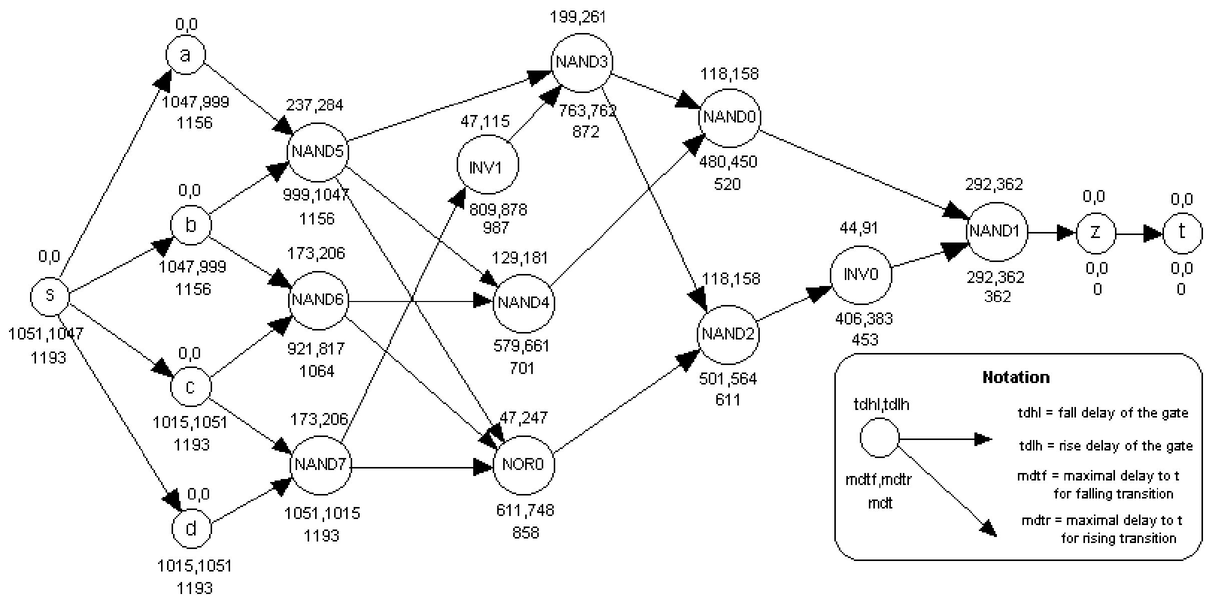


Figure 1 - Example of a circuit graph.

In the first phase, a DAG (direct acyclic graph) is created to represent the circuit under analysis. In this DAG, each node represents a CMOS gate and each edge represents a connection. Dummy nodes (i.e., with zero delay) are added to represent primary inputs and primary outputs. A source node **s** and a terminal node **t** are also created; a dummy edge is added between node **s** and each primary input node and between each primary output node and node **t**. With this canonical DAG representation, any circuit path assumes the form P= (v_0, v_1, v_2, v_3, ...v_n, v_{n+1}), where v_n is a graph node. If v_0 = **s** and v_{n+1} = **t**, then P is said to be a **complete** path. If $v_0 \neq$ **s** or $v_{n+1} \neq$ **t**, then P is said to be a **partial** path.

To consider different fall and rise gate delays (tdhl and tdlh, respectively) in path delay calculation the nature of the transition traveling down a path must be considered. In the spgd method this is accomplished by decomposing each topological path into two distinct logical paths, each one with its own path delay. The delay of a partial logical path P = (Q, v_{n+1}), with Q = (**s**, v_1, v_2, v_3, ...v_n) is calculated by:

$$d(P) = d(Q) + tdhl(v_{n+1}) \qquad (1)$$

if the output of v_{n+1} is undergoing a falling transition and

$$d(P) = d(Q) + tdlh(v_{n+1}) \qquad (2)$$

if the output of v_{n+1} is undergoing a rising transition.

For ordering path tracing, the original best-first procedure of Yen goes through a pre-processing phase in which the **maximal delay to node t** (**mdt**) is calculated for each node and stored in the data structure. However, if different fall and rise gate delays are to be dealt, two values of mdts must be calculated: **mdtf** and **mdtr**. For a given node *v*, **mdtf**(*v*) and **mdtr**(*v*) give the maximum of the delays of all possible partial logical paths starting at node *v* and ending at node **t**, for a falling transition and for a rising transition at the output of *v*, respectively, and are calculated by:

$$mdtf(v) = tdhl(v) + \max\{mdtr(u_i)\} \qquad (3)$$

$$mdtr(v) = tdlh(v) + \max\{mdtf(w_i)\} \qquad (4)$$

with $u_i \in$ **adjf**(*v*) (the list of successors of *v* for a falling transition) and $w_i \in$ **adjr**(*v*) (the list of successors of *v* for a rising transition).

The procedure for calculating the mdts of all nodes in the graph starts from node **t** and goes backwards in a breadth-first order. A node *v* will have its mdtf and mdtr computed only after all of its successors' mdtf and mdtr have been computed. The procedure ends when node **s** has its mdtf and mdtr calculated. At the same time the mdtf and mdtr of a node *v* are calculated, its lists of successors, **adjf**(*v*) and **adjr**(*v*), are sorted in a non-increasing order of each successor's mdtf and mdtr, respectively.

The mdt is a valuable information that allows to avoid searching branches that will not result in paths having greater delay than the smallest delay among the paths already discovered. It also permits to keep the partial logical path candidates ordered according to their esperances. The esperance of a partial logical path P = (Q, v_{n+1}), obtained by appending node v_{n+1} to the partial logical path Q = (**s**, v_1, v_2, v_3, ...v_n), is the maximum delay among all complete paths that have P as a prefix [3] and is calculated by:

$$e(P) = d(Q) + mdtf(v_{n+1}) \qquad (5)$$

if the output of v_{n+1} is undergoing a falling transition and

$$e(P) = d(Q) + mdtr(v_{n+1}) \qquad (6)$$

if the output of v_{n+1} is undergoing a rising transition.

In the enumeration phase the k most critical logical paths will be found and stored in a linear structure named k-list in the non-increasing order of their delays (k is a user provided value). Each position of k-list stores a logical path (initially, a partial logical path), its delay, d(P), its esperance, e(P), the current number of elements, level(P) and the transition type applied to its input, type(P). In the case of the sgd method, level and type were not necessary. The k-list structure is initialized by storing the first 2**fo(s)** partial logical paths made up from node **s** and its successors (from either **adjf**(s) or **adjr**(s)). **fo(s)** is the outdegree of node **s**, i.e., the number of primary inputs. The nodes taken from **adjf**(s) and **adjr**(s) are selected following a unique order. If 2**fo(s)**>k, then only the first k partial paths will be kept.

A function called **extend_and_insert** is then called for each partial path, beginning from the path stored in k-list[0]. This function takes the partial path stored in k-list[i] and extends it until node **t** has been reached. Consider that partial logical path P = (Q, v_{n+1}), with Q = (**s**, v_1, v_2, v_3, ...v_n), has an associated input transition such that the output of v_{n+1} is undergoing a falling transition. Assume also that **adjf**(v_{n+1}) = { u_0, u_1, u_2, u_3, ...} is the ordered list of successors for a falling transition at the output of v_{n+1}. Extending P through u_0 creates path P_0, with the same esperance of P (because u_0 is the first successor of v_n). Thus, P_0 will replace P in position k-list[i]. However, extending P through node u_1 creates a new partial path P_1 with esperance:

$$e(P_1) = d(P) + mdtf(u_1) \qquad (7)$$

Similarly, extending P through node u_2 creates a new partial path P_2 with esperance:

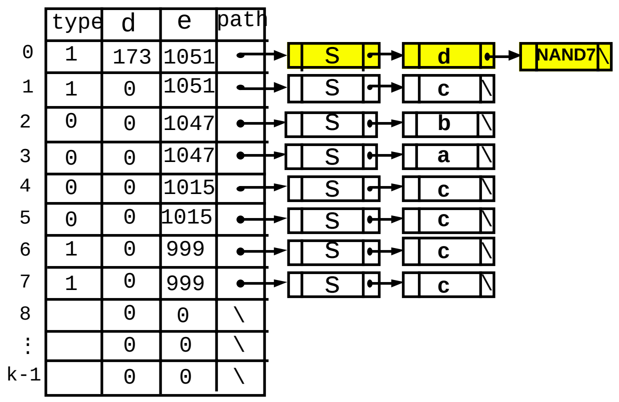
$$e(P_2) = d(P) + mdtf(u_2) \qquad (8)$$

and so on. Before extending path P by appending node u_0, the esperance of each new partial path (except P_0) is calculated: if the new partial path's esperance is greater than esperance(k-list[k-1]), then the new partial path is inserted in k-list, in an ordered manner. Otherwise, neither this partial path nor the next ones will be inserted, because partial paths are examined in the same order of the elements in **adjf**(v_{n+1}) (**adjr**(v_{n+1}), case the last node is undergoing a rising transition). This results in a prune of the search space. Case a new path is to be inserted, the insertion position j will be within the range (i<j£k-1) and the path stored in position k-list[k-1] will be automatically discarded. The delay of each new partial path P_i is calculated either by equation (1) or (2).

Once path P, stored in k-list[i], is completely extended, function **extend_and_insert** will be called for the next partial path, k-list[i+1]. The path enumeration phase will end when the path stored in k-list[k-1] is completely extended. Figure 2 shows the contents of k-list for the circuit represented by the graph of figure 1, when the first path begins to be extended.

3. Accuracy analysis of the sgd method

For investigating the loss in accuracy produced by using a single delay per gate in path delay calculation (the sgd method) two versions of the best-first procedure were implemented in C language: the original one, using the sgd method and the modified one, using the spgd method. Both versions were run on the combinational circuits of the ISCAS'85 suite. These circuits had their layout generated in 0.8_m CMOS technology using TROPIC layout generator [4] (except those marked with * in table 1). CMOS gates of these circuits had their fall and rise delays estimated using the formulation presented in [5].



	type	d	e	path
0	1	173	1051	S → d → NAND7
1	1	0	1051	S → c
2	0	0	1047	S → b
3	0	0	1047	S → a
4	0	0	1015	S → c
5	0	0	1015	S → c
6	1	0	999	S → c
7	1	0	999	S → c
8		0	0	\
⋮		0	0	\
k-1		0	0	\

Figure 2 - k-list structure for the modified best-first procedure running on the circuit example.

The sgd method was evaluated by considering the following cases for defining the single delay td of each gate *v*: td(*v*)=max{tdlh(*v*),tdhl(*v*)}, td(*v*)=(tdlh(*v*)+tdhl(*v*))/2 and td(v)=min{tdlh(*v*),tdhl(*v*)}. These cases will be referred as **max**, **typical** and **min**, respectively. The original best-first procedure (sgd-based) was used to determine the critical delay for the **max**, **typical** and **min** cases, while the modified one (spgd-based) was used to find the critical delay for the **fall/rise** case. The sensitizability of paths was not checked because the target of these experiments was to emphasize the importance of path delay calculation in the enumeration phase. The (topological) critical path delays are listed in table 1 (in picoseconds). The percentage error is with respect to the **fall/rise** case.

Table 1 - Results for fall/rise, max, typical and min cases on ISCAS'85 circuits.

Circuit	fall/rise	max		typical		min	
	delay	delay	error%	delay	error%	delay	error%
c432	12809	14767	15.2	12231	4.5	9727	24.0
c499	9638	13579	40.8	8566	11.1	4781	50.3
c880	9033	10670	18.1	8712	3.5	7032	22.1
c1355	10390	12684	22.0	9915	4.5	8176	21.3
c1908	11157	13478	20.8	10980	1.5	8582	23.0
c2670*	7021	8741	24.4	6954	0.9	5190	26.0
c3540	16826	21080	25.2	16607	1.3	12281	27.0
c5315*	8691	10443	20.1	8543	1.7	6691	23.0
c6288	40009	59055	47.6	38972	2.5	19116	52.2
c7552*	7510	9001	19.8	7459	0.6	5939	20.9

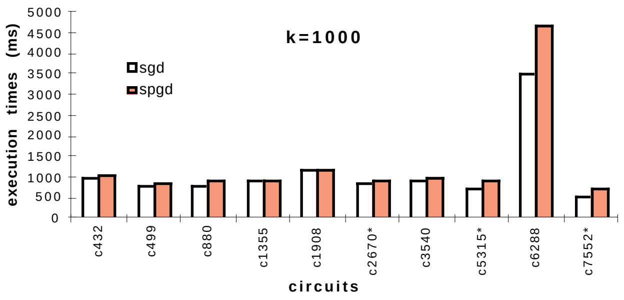
Min case underestimated the critical path delay for all circuits by a percentage ranging from 15% to 52%, showing the uselessness of this value. On the other hand, **max** case overestimated the critical delay for all circuits, with most of them exhibiting about 20% of error.

However, **max** case does not lead to an underestimation of path delays, what would be incorrect for timing analysis. Satisfying delay correctness is crucial for timing analysis algorithms that intend to be accurate.

Some authors argue that it is not necessary to account for different fall and rise gate delays because the accumulated error tends to be small. According to them, using a single gate delay td, being td a "typical" delay value is sufficient. But the problem of determining what would be a "typical" delay value is never addressed. We have decided to investigate this issue by creating a **typical** case where td is the arithmetic mean of tdhl and tdlh, what would be an acceptable assumption. Indeed, results indicate that the error of the **typical** case was significantly smaller for all examples. However, the **typical** case underestimated the critical path delay for all examples and thus is not appropriate for timing analysis. It is also interesting to notice that when layout information is not available (cases marked with * in table 1), the error of **typical** case is smaller. This may confirm the need for more sophisticated path delay calculations under the presence of layout-extracted information (back annotation).

4. Comparative performance evaluation

The graphics of figure 3 show the execution times for both **sgd** and **spgd**-based best-first procedures running on a Sun Ultra2™ workstation for the ISCAS'85 circuits, with k=1000 and k=10000. It was observed that the execution times for the spgd case are equal or greater than the execution times for the sgd case. This behavior was expected since the number of partial path insertions is potentially greater in the spgd case because the total number of paths is doubled. As a consequence, the time for inserting partial paths may become dominant quicker than in the sgd case. This effect is more evident for large values of k, as showed by figure 3. However, it is important to note that, for a given k, the number of insertions in the spgd case is not doubled because the nodes of each partial path present the same successors as in the sgd case. Concerning the total number of paths, the spgd method has to explore two topologically identical graphs, but in a possibly different order. By plotting the curves *k x number of partial path insertions* for both methods, one conclude that, although the difference in the number of insertions tends to increase with the increase of k, the number of insertions in the spgd case is still linear. This shows that it is possible to reduce the execution time of the spgd method by reducing the time required to insert partial paths in the list. This may be accomplished by using a more sophisticated list insertion scheme. Finally, it is important to remark that the biggest execution times among all cases considered in this article corresponds to circuit c6288 with k=10000: 343 seconds for the spgd case and 272 seconds for the sgd case. This is also the case presenting the biggest difference between spgd and sgd execution times: 71 seconds.



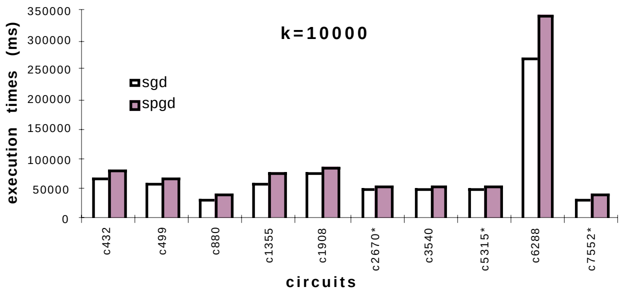


Figure 3 - Execution times for the sgd and spgd best-first procedures.

5. Conclusions

A path enumeration procedure that considers different fall and rise gate delays was presented. This method is based on modifications to the best-first procedure of Yen in order to use the spgd method in path delay calculation.

Results show that the error produced by using the sgd method in path delay calculation, instead of spgd method, cannot be neglected. In the sgd method, the use of the arithmetic mean of tdhl and tdlh as gate delay has lead to small errors. Even though, the critical delay has been underestimated, showing that the sgd method is not appropriate for timing analysis. The overhead in execution time of the spgd-based best-first procedure is small and may be reduced if a faster list insertion scheme is adopted.

References

[1] YEN, S.; DU, D.; GHANTA, S. Efficient Algorithms for Extracting the K Most Critical Paths in Timing Analysis In: ACM/IEEE DAC, 26.,1989. p.649-652.

[2] MCGEER, P.; BRAYTON, R. Integrating Functional and Temporal Domains in Logic Design: The False Path Problem and its Implications. Kluwer Academic Publishers - 1991.

[3] BENKOSKI, J. et al. Efficient Algorithms for Solving the False path Problem in Timing Verification In: ICCAD, 1987 p.44-47.

[4] MORAES, F.; REIS, R.; LIMA F. An Efficient Layout Style for Three-Metal CMOS Macro-Cells. IFIP Intl. Symposium on VLSI, 1997. Gramado-Brazil, p 415-426.

[5] DAGA, J.-M.; TURGIS, S.; AUVERGNE, D. Design Oriented Standard Cell Delay Modeling. In Proceedings of PATMOS'96,. p.265-274