# An Upper-Bound Algorithm for Gate-level Delay Analysis

*Sunshin Lu and Feipei Lai*

Department of Computer Science
National Taiwan University
Taipei, Taiwan, R. O. C.
Tel: 02-3630231-ext 3230

*Abstract*

An upper-bound algorithm for gate-level delay analysis dynamically updates path timing information. When conflicting logic value occurs, the algorithm switches to the most promising path according to the updated path delay. The execution time has been greatly shortened by reducing the number of paths to be traced.

## 1. Introduction

The number of gates in a chip increase dramatically due to the fast progress in semiconductor technology. Since the design of digital circuit becomes more and more expensive and difficult, product verifications can never be over emphasized. There are three major parts of product verification, functional design verification, physical design verification, and timing verification. Timing verification is validating the path delays (from primary input or storage elements to primary output or storage elements) to ensure that they are within the specified delay range.

Most earlier works of timing analysis can be roughly divided into the following two approaches:

1. Block oriented algorithm [1-4].
2. Path enumeration algorithm [5-8].

Path enumeration algorithm enumerates all the possible paths in the digital networks and calculates the delay of each path. The complexity of this algorithm is $O(e^n)$ and $n$ is number of nodes in the digital network. The path found by this algorithm is not necessarily sensitive.

Block oriented algorithm is a *PERT* (Performance Evaluation Review Technique) based algorithm for finding critical path. The complexity of this algorithm is $O(n)$, where $n$ is number of nodes in the digital network. Though it is fast, the path found by this algorithm could be too conservative. The reason is that it does not consider the functional relations between signals; therefore, it may find a path that would never be activated.

As the critical path found by block oriented algorithm is not necessarily activated, there are some algorithms that consider the functional relationships between signals in order to find a more accurate critical path. In this paper, we proposed an algorithm that uses the results obtained from block oriented algorithm to select the most promising critical path for performing functional analysis.

## 2. Upper-Bound Algorithm

We know that the delay analysis concerns if there is a path which is too short or too long that would not meet the timing requirements. As the short path is easy to fix up, we will focus on the long path. Given a digital network, we perform the block oriented algorithm from inputs to outputs. We can find critical path by checking the delays of the output nodes. If the critical path is not too long then the digital network will work properly. However, it is seldom the case, so by performing the functional analysis, we can see whether the critical path is sensitive or not.

Let us describe our data structure and algorithm briefly now. The analyzed circuit can be viewed as a network. We first perform the PERT analysis of the analyzed circuit from primary inputs to primary outputs and we will get the maximum delay (max_in(p)) from primary inputs to each node $p$ in the circuit. We will treat each primary output nodes separately as we know that the critical path starts from one of the primary input nodes and ends at one of the primary output nodes. Upper-bound algorithm will dynamically update the maximum delay of each node in the network. We will create a network for a primary output node if it has the maximum delay among all primary output nodes. For example, in Figure 1(a) $J$, $K$, and $L$ are primary output nodes. The number below each node is the maximum delay from primary inputs to that node. The delay value, 11, of node $J$ is maximum so a network is created for it (Figure 1(b)). After performing functional analysis of the network created for node $J$ the maximum delay of node $J$ is updated from 11 to 8 and the delay node $K$, 10, becomes the maximum. The numbers in parentheses (Figure 1(b)) are the updated maximum delay from primary inputs to that node. This time a network (Figure 1(c)) is created for node $K$ and we perform functional analysis for that network. This process will be continued until we succeed in finding the critical path by performing functional analysis.

For each primary output node, there is a critical path starts from some primary input node and ends at it while performing the functional analysis on a network of primary output node $J$, we start from node $J$ and trace back to the primary input nodes. As critical path is traced back from some primary output node $J$, we say that the critical path is led by node $J$. The most important part of our algorithm is the procedure which will dynamically update the maximum delay of each node when the functional analysis was blocked. The following example can explain how this works. A network created for primary output node $Q$ is Figure 2(a), the critical path being traced is node $Q$, $N$, $K$, [illegible]. The number below the name of each node is the maximum delay from primary inputs. Node $P$, $Q$, and $R$ are primary outputs; node $A$, $B$, and $C$ are primary inputs. The next node to be included in the critical path is $D$. If the functional analysis fails

when node $D$ is included then the maximum delay of the nodes ($Q$, $N$, $K$, and $H$) in the critical path must be updated. In Figure 2(b) the maximum delay of node $H$ changed from 8 to 5 as its maximum input node changed from node $D$ (maximum delay 5) to node $E$ (maximum delay 2). Dotted line means the node had been visited before. Maximum delay of node $K$ must be updated from 11 to 9 as its maximum input delay had been changed from 8 to 6. The maximum delay of node $N$ should be changed from 14 to 13 and node $Q$ changed from 16 to 15 for the same reason. Note that the maximum input node of node $N$ had changed from node $K$ to node $J$ and node $K$ has the new maximum input node $G$; such nodes, as $K$ and $G$, we call them separate nodes. If separate nodes exist, then we define tail node to be the separate node with the maximum delay; otherwise, tail node is the last node of the critical path.

In this example the current tail node is node $N$. Assume that the path led by node $Q$ is still the most promising critical path after we have updated the maximum delays, then the functional analysis will start from the tail node ($N$) of the path that is led by node $Q$. We will not stop the functional analysis until the process fails or reaches the primary input node. If a primary input node is reached then the critical path has been found and the program will be terminated. If the new critical path (node $Q$, $N$, and $J$) is blocked by node $J$ (Figure 2(c)), then the maximum delays of node $J$, $N$, and $Q$ must be updated. We have two critical paths now, one is the new critical path (node $J$, $N$, and $Q$) the other is the previous one (node $H$, $K$, $N$, and $Q$). However, we can just keep track of one critical path in a network. If the new critical path is still the most promising one and this time node $H$ is chosen to be included in the new critical path then the next node to be chosen after node $H$ would be node $E$ instead of node $D$ (Figure 2(d)). It is because at node $H$ the input node $D$ has been marked as visited. In the presence of more than one critical path, we may choose the wrong one, node $E$, instead of node A so we can have only one critical path in a network at one time.

Pruning-subpaths, tracing all the paths that come after the tail node($N$) of the previous critical path, is our approach to solve the problem. The procedure of pruning-subpaths is the same as the modified DFS algorithm [11] except that the root is the tail node ($N$) and the direction is from primary output to primary inputs. Pruning-subpaths of previous critical path starts from the last node ($H$) and ends at the tail node ($N$) along the path. The procedure will also keep track of the lower bound of the analyzed circuit which is at first set to be zero. Pruning-subpaths will not include a node to the critical path if the maximum delay plus the gate delays along the current critical path is less than the lower bound, and this node would be marked as visited. We will update the lower bound when a primary input node is successfully included in the path. We will include a node in the critical path only if the inclusion produces a path delay which is greater than the current lower bound.

As pruning-subpaths searches for the lower bound and the tracing of critical path of each network finds the upper bound, the upper-bound algorithm is somewhat like a branch-and-bound algorithm.

One of the important parts of our algorithm is that the procedure which collects node signal information derives more conclusions to check for logic consistency. We use the same procedure as in [9], and define a set of propagation rules as follows:

A = not(B)
1. If B is known, set A to $\bar{B}$.
2. If A is known, set B to $\bar{A}$.

$A$ = and($B_1$, $B_2$, ..., $B_n$)
1. If $B_i$ = 1 for all $i$, set A to 1.
2. If A = 1, set $B_i$ to 1 for all $i$.
3. If $B_i$ = 0 for any $i$, set A to 0.

$A = or(B_1, B_2, ..., B_n)$
1. If $B_i$ = 0 for all $i$, set A to 0.
2. If A = 0, set $B_i$ to 0 for all $i$.
3. If $B_i$ = 1 for any $i$, set A to 1.

$A = nand(B_1, B_2, ..., B_n)$
1. If $B_i$ = 1 for all $i$, set A to 0.
2. If A = 0, set $B_i$ to 1 for all $i$.
3. If $B_i$ = 0 for any $i$, set A to 1.

$A = nor(B_1, B_2, ..., B_n)$
1. If $B_i$ = 0 for all $i$, set A to 1.
2. If A = 1, set $B_i$ to 0 for all $i$.
3. If $B_i$ = 1 for any $i$, set A to 0.

$A = xor(B_1, B_2, ..., B_n)$
1. If $\sum_1^n B_i$ is odd, set A to 1.

$A = xnor(B_1, B_2, ..., B_n)$
1. If $\sum_1^n B_i$ is even, set A to 1.

Before describing the algorithm, we will define some terminologies used in the algorithm first. Let $Q$ be the queue containing pairs of output node and its corresponding delay, *lower_bound* be the lower bound of critical path delay, and $L$ be the maximum path delay that would not cause timing violation.

*delay*($p$) - gate delay of node $p$

*max_in*($p$) - delay of the longest path from primary inputs to node $p$

*latest_to*($p$) - the unmarked node that has largest *max_in* value among all the input nodes to node $p$

*mark_node*($q$, $p$) - mark the input node $q$ of node $p$ as visited.

*tail*($H$) - tail node of the path which is led by primary output node $H$.

*prenode*($p$) - node that comes before node $p$ in the critical path which direction is from some primary output to primary inputs.

*postnode*($p$) - node that comes after node $p$ in the critical path which direction is from some primary output to primary inputs.

*funalys*($p$) - return true if the inclusion of the node $p$ into current critical path does not cause logical confliction.

```
Begin
lower_bound := 0  /* initial value */
For all node p in the circuit compute max_in(p).
/* use PERT method */
Sort queue Q according to path delay by descending order.
/* every pair in Q contains a primary output node and the max-
imum delay from primary inputs to that node */
(H, upper_bound) := remove(Q)
/* (H, upper_bound) is the first pair in Q */

WHILE ((upper_bound > L) && (upper_bound > lower_bound)){
/* continue when upper bound is still too large and the current
upper bound is greater than lower bound */
  IF (H has never been visited before)
  THEN
       Create a network for the primary output node H.
       /* for holding functional values while analyzing the path */
  p := tail(H)
  /* p is the tail node of the network that is created for H */
  r := postnode(p)
  /* r points to the subpath to be pruned */
  q := latest_to(p)
  continue := funalys(q)
  /* if continue is true than node q can be included in the path that
  is led by node H  */

  WHILE (continue && (q is not a primary input node)){
  /* the path is not blocked and not reach the primary input */
    add node q to the current critical path
    q := latest_to(q)
    continue := funalys(q)
  }
  IF (q is a primary input node) /* the critical path is sensitive */
  THEN exit

  IF (r != NULL) /* if r is NULL it means nothing to be pruned
  */
  THEN
        lower_bound := prune_subpaths(r, lower_bound)
        /* pruning_subpaths returns the updated lower bound
        trace back and update timing delays */
  p := prenode(q)
  mark_node(q, p)
  /* q cause violation when adds to the critical path */

  WHILE (p != H){
  /* there are more nodes to be traced in the path */
    q := latest_to(p) /* q is the new max_in of p */
    s := prenode(p)
    IF (q == NULL)
    THEN /* all the input nodes of p had been traced */
      mark_node(p, s) /* mark node p as visited */
    ELSE
      max_in(p) := delay[p] + max_in(q)
      /* compute new max_in of node p in the path */
    p := s
  }
  max_in(H) := delay(H) + max_in(latest_to(H))
  /* compute new upper bound of the path that is led by H */
  Insert (H, max_in(H)) into Q according to the new path delay.
  (H, upper_bound) := remove(Q)
  /* (H, upper_bound) is now the most promising critical path and
  critical path delay */
```

## 3. Examples

Consider the circuit shown in Figure 3 [10]. After performing *PERT* analysis we get the maximum delay from primary inputs of each node. The results are also shown in Figure 3. Primary output node Y has the maximum delay (4), so a network is created for it. The path which composed of edges $Y$, $X$, $E$, $D$, and $B$ is the critical path. The result after performing functional analysis is shown in Figure 4. We observe that at edge $E$, there is a logic inconsistence, so the path is blocked. As we have only one critical path, so we do not have to do prune-subpaths. The max_in(i) of each node $i$ that goes between edge $Y$ and edge $E$ must be reevaluated and the result is shown in Figure 5. Path (Y, X, E, C) is now the new critical path. The result from functional analysis is shown in Figure 6, and this time the path is not blocked.

## 4. Conclusion

We have proposed an algorithm that will always select the most promising critical path for performing functional analysis. Previous algorithms used to choose the alternative of the last edge to replace the one that violates the functional requirements. Sometimes the real critical path is far away from the current blocked path, previous algorithms will still waste time on those hopeless ones. Nevertheless, the upper-bound algorithm will always select the most promising one according to updated path delay information after it prunes subpaths. Ten circuits from ISCAS testing benchmark have been run on VAX 8530 and the results in Table 1 show the advantages of efficiency and accuracy over the PERT approach.

## References


[1]. T. I. Kirkpatrick and N. R. Clark, "PERT as an aid to Logical Design," *IBM Journal of Research and Development*, Vol. 10, No. 2, pp. 135-141, March 1956.

[2]. T. M. Mcwilliams, "Verification of Timing Constraints on Large Digital Systems," *Proceedings of the 17th Design Automation Conference*, Minneapolis, pp. 139-147, 1980.

[3]. Robert B. Hitchcock, Sr. "Timing Verification and the Timing Analysis Program," *Proceedings of the 19th Design Automation Conference*, pp. 594-604, 1982.

[4]. Lionel C. Bening , Thomas A. Lane, and Curtis R. Alexander, "Developments in Logic Network Path Delay Analysis," *Proceedings of the 19th Design Automation Conference*, pp. 605-615, 1982.

[5]. D. J. Pilling, and H. B. Sun, "Computer Aided Prediction of Delays in LSI Logic Systems," *Proceedings of the 10th ACM/IEEE Design Automation Workshop*, Portland, Oregon, pp. 182-186. 1973.

[6]. M. A. Word, "Design Verification and Performance Analysis," *Proceedings of the 15th Design Automation Conference*, Las Vegas, pp. 264-270, 1978.

[7]. T. Sasaki, A. Yamada, T. Aoyama, K. Hasegawa, S. Kato, and S. Sato, "Hierarchical Design Verification for Large Digital Systems," *Proceedings of 18th Design Automation Conference*, Nashville, pp. 105-112, 1981.

[8]. R. Kamikawai, M. Yamada, T. Chiba, K. Furumaya and Y. Tsuchiya, "A Critical Path Delay Check System," *Proceedings of the 18th Design Automation Conference*, Opryland, pp. 118-123, 1981.

[9]. D. Brand, "Redundancy and Don't cares in Logival Synthesis," *IEEE Transactions on Computers*, Vol. 1. C-32, No. 10, October, pp. 947-952, 1983.

[10]. D. Brand, and V. S. Iyengar, "Timing Analysis using Functional Relationships Verification," *IEEE Digest of Technical Report, ICCAD'86,* pp. 126-129, 1986.

[11]. J. Benkoski, E. Vanden Meersch, L. Claesen, and H. De Man, "Efficient Algorithms for Solving the False Path Problem in Timing Verification," *IEEE Digest of Technical Report, ICCAD'87,* pp. 44-47, 1987.

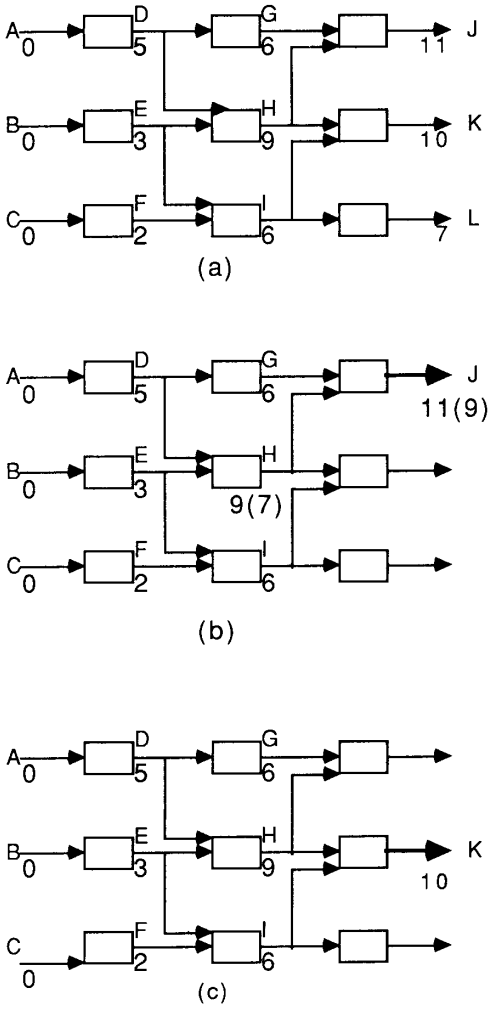


Figure 1. (a) is the sample circuit, (b) is the network created for node J, and (c) is the network created for node K.

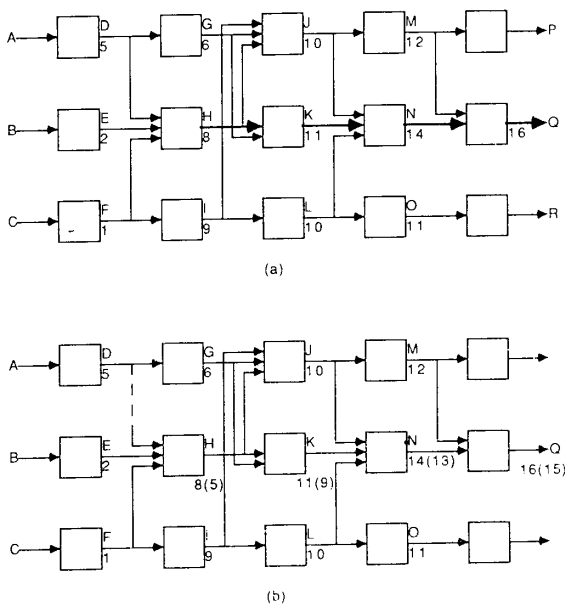


Figure 2. (a) network created for node Q, number below each node is the maximum delay from primary input to that node.
(b) numbers in parentheses are the updated delays, dotted line marks the node as visited.

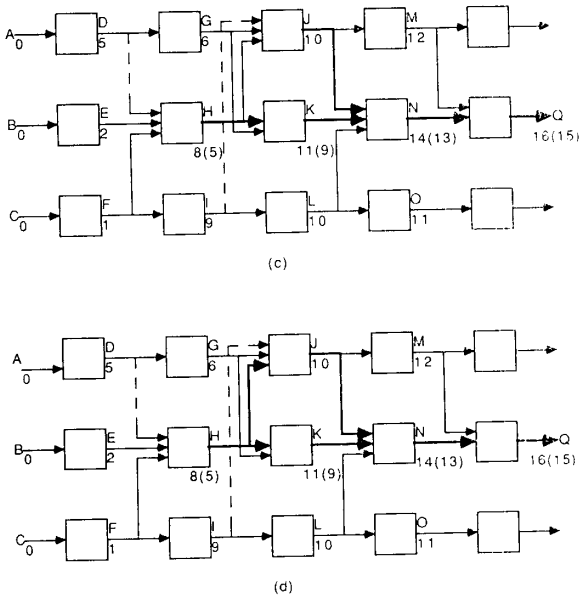


Figure 2. Continued

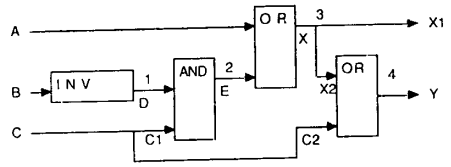


Figure 3. Sample circuit and the results after performing PERT analysis.

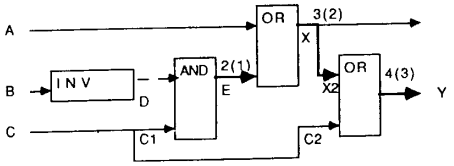


Figure 5. Circuit after the reevaluation of max-in(i) of each node i in the traced path.

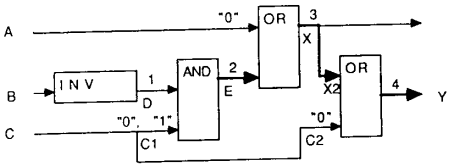


Figure 4. Results of performing functional analysis, where the number in quotes represents logical values.

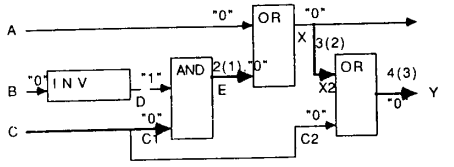


Figure 6. Circuit after performing functional analysis on the new critical path.

| circuit | # gates | exe-time (pert) 1/60 second | delay (pert) unit delay | exe-time (u-bound) 1/60 second | real-delay unit delay |
|---|---|---|---|---|---|
| c432 | 160 | 5 | 17 | 5 | 17 |
| c499 | 202 | 6 | 11 | 1 | 11 |
| c880 | 383 | 9 | 25 | 2 | 25 |
| c1355 | 546 | 12 | 25 | 22 | 25 |
| c1908 | 880 | 19 | 45 | 339 | 43 |
| c2670 | 1193 | 33 | 37 | 648 | 35 |
| c3540 | 1669 | 35 | 57 | 9637 | 51 |
| c5315 | 2307 | 53 | 51 | 6552 | 46 |
| c6288 | 2416 | 53 | 124 | 1208 | 124 |
| c7552 | 3512 | 78 | 46 | 16799 | 40 |

Table 1. Result of ISCAS testing benchmarks running on VAX 8530, all the gates have 1 unit delay except that BUFF has 2 units.