JOURNAL OF SEMICONDUCTOR TECHNOLOGY AND SCIENCE, VOL.16, NO.6, DECEMBER, 2016
https://doi.org/10.5573/JSTS.2016.16.6.771

ISSN(Print) 1598-1657
ISSN(Online) 2233-4866

Extracting the K-most Critical Paths in Multi-corner Multi-mode for Fast Static Timing Analysis

Deok-Keun Oh, Myeoung-Woo Jin, and Ju-Ho Kim

Abstract—**Detecting a set of longest paths is one of the crucial steps in static timing analysis and optimization. Recently, the process variation during manufacturing affects performance of the circuit design due to nanometer feature size. Measuring the performance of a circuit prior to its fabrication requires a considerable amount of computation time because it requires multi-corner and multi-mode analysis with process variations. An efficient algorithm of detecting the K-most critical paths in multi-corner multi-mode static timing analysis (MCMM STA) is proposed in this paper. The ISCAS'85 benchmark suite using a 32 nm technology is applied to verify the proposed method. The proposed K-most critical paths detection method reduces about 25% of computation time on average.**

Index Terms—**Process variation, critical path, static timing analysis, path pruning, MCMM**

I. Introduction

Due to the rapid progress in circuit design technology, the complexity of a digital system and its size have increased dramatically. Thus, timing analysis and verification play an important role to satisfy the given timing constraint. The maximum operation speed of digital integrated circuit is determined by the slowest path that is so called the critical path. Detecting and identifying a set of critical paths are crucial steps in timing analysis and optimization. There exist many techniques [1-4] of extracting critical paths followed by delay analysis of digital circuits. The depth-first search and breadth-first search [1] are two major critical path search algorithms. An efficient algorithm of detecting K-most critical paths was proposed in [6] and the benefit comes from pruning unnecessary searches. Applying the above search algorithms to recent circuit design suffers from path explosion problem due to large circuit size and multi-corner multi-mode (MCMM) analysis. In the proposed method, modification and improvement are made to the existing K-most critical paths algorithm [5] for efficient timing analysis.

As complementary metal-oxide semiconductor (CMOS) technology has been scaled down to the nanometer range, process variations of device parameters have great impact on circuit performance. Taking process variations into account in timing analysis requires more computational effort. Multi-corner multi-mode static timing analysis (MCMM STA) is another variation-aware approach to timing analysis that can serve as a compromise between traditional STA [7] and statistical static timing analysis (SSTA) [8]. In timing analysis, MCMM STA is very efficient in a sense that it propagates only minimum and maximum delay values. MCMM STA can simultaneously analyze a circuit's performance at various modes and corners while considering process variations.

The remainder of the paper is organized as follows. Section II describes the basic concept of K-most critical path search algorithm and multi-corner multi-mode static timing analysis. Modified K-most critical path detection method using MCMM STA is presented in Section III. Section IV presents the experimental results. Finally,

Manuscript received Jan. 24, 2016; accepted Nov. 1, 2016
Department of Computer Science and Engineering, Sogang University, Seoul 04107, Korea
E-mail : pci123@sogang.ac.kr

concluding remarks are given in Section V.

II. Background

Performance has been one of the major design constraints along with area and power. Timing analysis can be performed at various design stages to meet the timing constraint. In this section, we briefly describe K-most critical path algorithm [5] and basics of multi-corner multi-mode static timing analysis.

1. K-Most Critical Paths Algorithm

The well-known critical path search methods include the breadth-first search, depth-first search, and the PERT method [9]. The critical path search algorithms normally report only the critical path to the designer and it fails to give sufficient information for correcting the timing violations.

Based on either a breadth-first search or depth-first search, path enumeration algorithms have been proposed in [6, 7]. Although these algorithms provide more information than the critical path algorithms, they suffer from the path explosion problem.

The K-most critical path algorithm [5] is proposed to generate the K longest paths for a given acyclic directed graph. It is an efficient approach to find a set of long paths [12]. The pseudo code of the algorithm [5] is summarized in Fig. 1. The algorithm consists of four phase: creating the source and sink node, computing the maximum delay to sink, sorting the successors of each vertex, and finally K longest path enumeration.

In the first phase, a source node s and a sink node t are added to the graph for the purpose of easy handling and simplifying the boundary conditions. A dummy edge is added from node s to each of the starting vertices and from each of the ending vertices to the node t. Then, the longest path of the graph with multiple primary inputs and outputs becomes the longest path from the source node s to the sink node t. The second phase calculates the maximum delays of all the possible partial paths starting from vertex s and ending with the sink node t. In order to prune the unnecessary traversals in the path enumeration phase, we need to know the maximal delay to sink node t from each vertex in the graph. This information is useful when we traverse a partial path P starting from the sources and ending in a vertex v, we can find the maximum delay of all possible full paths. Thus, this phase calculates the maximum delay from all the vertices to sink in the graph. This computation starts from sink node t and works backward until it reaches the source node s. Third phase sorts the successors of each vertex to further simplify the pruning process. When we traverse the graph, we can immediately prune the path that is no larger than given threshold.

```
Algorithm
Let K be the number of most critical paths in graph G

1. Create the source s and the sink t
2. Compute the maximum delays to the sink
3. Sort the successors of each vertex
4. Enumerate the K longest paths
   repeat
       Take out the partial path
   until
       there are K completed paths
```

Fig. 1. The pseudo code presented in [5].

Enumerating the K longest paths is final phase. In order to store the K longest paths in the graph, this phase maintains a data structure PATHS. There are K entries in PATHS where each entry hold a full path and its delay. First, the longest path can be extracted by function nextnode() recursively. The function nextnode(v_i, v_{i+1}) will return the vertex next to v_{i+1} in the sorted successors list of v_i. If v_{i+1} is the last successor in the sorted list, it will return NULL. Next, the longest path is found and inserted into PATHS. Backward tracing is used to get the next partial path that has a greater delay than threshold T. Once a partial path with its delay greater than the threshold T is identified, it use forward tracing to extend the partial path to a full path. The newly generated full path is stored in PATHS. This enumeration process will terminate if all the possible paths are generated or pruned.

2. Multi-corner Multi-mode Static Timing Analysis

Static timing analysis (STA) estimates the circuit's performance prior to manufacturing. The STA is reasonably accurate and efficient because test vectors are not required during the analysis. The SPICE-like circuit level simulations always produce the most accurate

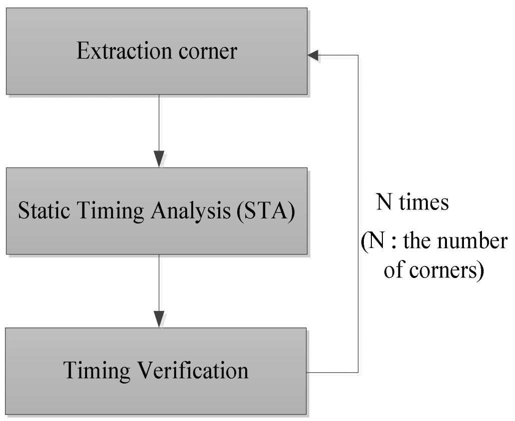


Fig. 2. Timing verification.

results. However, it is not practical to apply to the large circuit design. Thus, STA is widely accepted method of verifying circuit performance at various design steps.

The performance of a circuit is normally different at different temperatures. Therefore, timing analysis at various temperatures must be conducted to measure the operational speed of the design at different environment. Similarly, variations on supply voltage may alter operational speed of the circuit. Such conditions are so called "corners" and they include temperature, voltage, process parameters, and so on. Fig. 2 illustrates a simple flow of timing analysis and verification. For N parameters, the number of corners is 2^N. Thus, we must perform the STA 2^N times. Recently, the number of corners to consider for timing analysis is increased and as a result it becomes time consuming.

Multi-corner multi-mode static timing analysis (MCMM STA) is another variation-aware approach to timing analysis that can serve as a compromise between traditional STA and statistical static timing analysis (SSTA) [14]. Unlike traditional STA, MCMM STA can analyze a circuit's performance at various modes and corners while considering process variations. Multi-corner analysis consists of process parameters, supply voltage, temperature, and so on. For example, the worst-case timing analysis can be performed under conditions such as a low supply voltage, high temperature, and slow process parameters. Fig. 3 shows the best and worst corners consisting of the process, voltage, and temperature (PVT). The types of corners are categorized as best, typical, or worst. The modes include the following: functional mode, test mode, and sleep mode.

Gate delays are described by affine functions [10] of

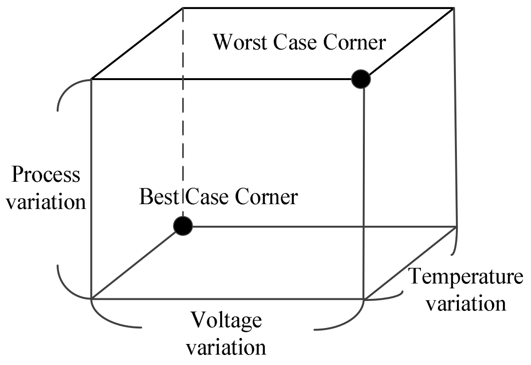


Fig. 3. Best and worst case corner that consists of process, voltage, temperature.

the parameters as follows:

$$\begin{aligned} Delay_{gate} &= a_0 + a_1P_1 + a_2P_2 + \cdots a_nP_n \\ &= a_0 + \sum_{i=1}^{n} a_iP_i \end{aligned} \tag{2.1}$$

In Eq. (2.1), the term $Delay_{gate}$ is defined as gate delay and a_0 is the nominal delay computed at nominal condition. Here, $a_i\,(1 \le i \le n)$ denotes the sensitivity of P_i. The term P_i represents the process parameters such as the oxide thicknesses, effective channel length, and width. The coefficient of each parameter is defined as the sensitivity that represents degree of impact on the gate delay. The larger sensitivity implies larger impact on gate delay among other process parameters. The sensitivity can be determined as the following equation:

$$\begin{aligned} a_i &= \frac{Delay@P_{i,MAX} - Delay@P_{i\ MIN}}{P_{i,MAX} - P_{i\ MIN}}, \\ i &= 1,2,3,\cdots \end{aligned} \tag{2.2}$$

The term $P_{i,MAX}$ is defined as the maximum value of P_i, and $P_{i,MIN}$ is defined as the minimum value of P_i. To obtain the maximum delay, all the process parameters with positive sensitivity are set to 1 and all the process parameters with negative sensitivity are set to -1. Thus, the shortest and longest delays are obtained by setting each parameter to one of its extremes [11, 13].

III. MODIFIED K-MOST CRITICAL PATHS ALGORITHM

The objective of the proposed method is to extract the

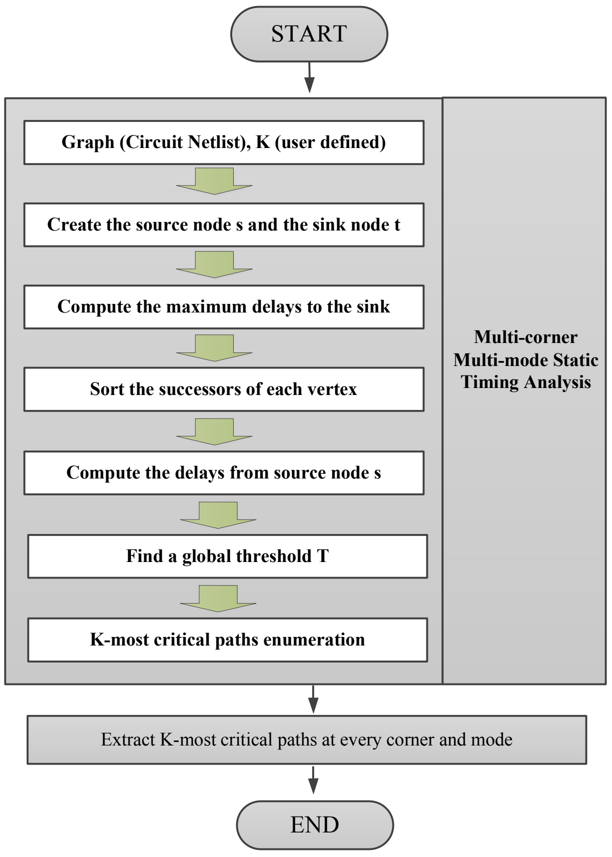

START
Graph (Circuit Netlist), K (user defined)
Create the source node s and the sink node t
Compute the maximum delays to the sink
Sort the successors of each vertex
Compute the delays from source node s
Find a global threshold T
K-most critical paths enumeration
Multi-corner Multi-mode Static Timing Analysis
Extract K-most critical paths at every corner and mode
END

Fig. 4. The overall flow of modified K-most critical paths search method.

K-most critical paths in MCMM static timing analysis. At the same time, process variations must be considered for accurate analysis. The modified K-most critical path algorithm is described next.

1. Overall Flow for Modified K-most Critical Paths Search Method

Fig. 4 represents the overall flow of modified K-most critical paths search method. The proposed method extracts the K-most critical paths at various corners and modes. Throughout this paper, the K is defined as the number of longest paths and it is normally given by the user. As the first step of timing analysis, the circuit netlist is converted to a timing graph. In the graph, interconnection or input/output pins are represented as edges while logic gates are represented as nodes. In order to handle the boundary conditions, we modify the timing graph by adding a dummy source node and a dummy sink node in the first step. Then finding a longest path from primary inputs to primary outputs becomes the

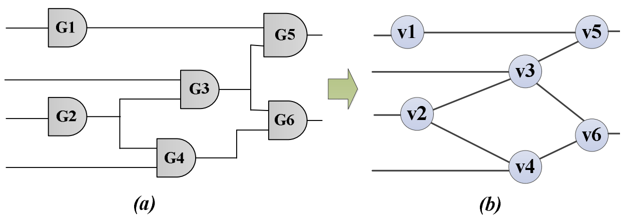


Fig. 5. Converting the c17 circuit (a) from the ISCAS85' benchmark to acyclic timing graph (b).

problem of finding the longest paths from the node s to t. As the second step, the maximum delay to sink node t from each vertex is calculated from the graph. At the third step, the successors of each vertex are sorted according to their delay. The fourth step is to obtain the maximum delays from source node s to each vertex. Then, the fifth step finds global threshold T using delay information obtained in second and fourth step. As the final step, the depth first search is performed with backward/forward tracing. The K-most critical paths at various corners and modes are extracted during the path enumeration phase.

2. K-most Critical Paths Extraction using Multi-corner Multi-mode Static Timing Analysis

This section explains how the K-critical paths search algorithm is modified to apply in MCMM static timing analysis in details.

A. Generating the Timing Graph

The proposed method requires a user-defined K which is the number of most critical paths among all possible paths and a circuit netlist as inputs. The circuit is firstly converted to acyclic timing graph as shown in Fig. 5. In order to apply a depth-first search for the whole graph, we modify the acyclic timing graph by adding a source node and a sink node. Then, a dummy edge is added from the source node to each of the starting vertices and from each of the ending vertices to the sink node. We assume the delay of dummy edges is zero and the weight of the vertex is zero. Fig. 6 represents a graph after source node, sink node, and dummy edges (dotted line) are inserted. This example will be used throughout this section. A full path P is defined as [v_o, v_1, v_2, …, v_n] if the node v_0 is equal to the dummy source node s and v_n is equal to the dummy sink node t. A partial path P is

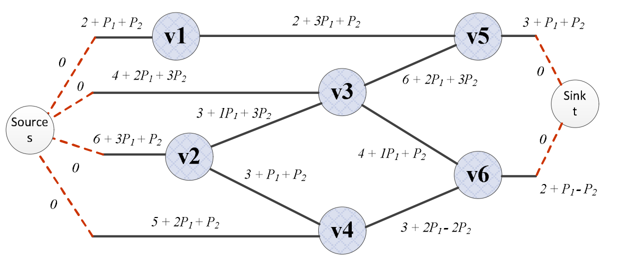


Fig. 6. Graph representation after source node, sink node and dummy edges are added in Fig. 5(b).

defined as [v_0, v_1, v_2, …, v_n] if the node v_0 is not the dummy source node s or v_n is not the sink node t. For example, [s, v1, v5, t], [s, v2, v3, v6, t] are both full paths and [s, v2, v3], [v4, v6, t] are both partial paths. The delay of path is summation of the edges. For example, the delay of full path [s, v1, v5, t] is summation of the edges <s, v1>, <v1, v5>, <s, t>. The delay of the edge is calculated using Eq. (2.1). The delay of edge <v1, v5> is $2 + 3P_1 + P_2$. If the parameters P_1 and P_2 are equal to 1, its delay becomes 6 from $2 + (3 \times 1) + 1$.

B. Calculating the Maximum Delay

After creating the graph, the maximum delay from all vertices to the sink node t is calculated as the next step. The delay of gate is represented as the canonical form instead of a constant value considering various corners and modes.

When we find the K-most critical paths, the maximum delay from each vertex to sink node t is used for pruning unnecessary searches. We can find the maximum delay of all possible full paths that has partial path P as a prefix. As a result, the large amount of search space can be reduced by pruning. Computing maximum delay from each vertex to sink node t starts backward from sink node t until the source node s is reached. The maximum delay of the v_i is define as *MaxDelay* (v_i) and it is equal to *MAX { MaxDelay (u) + d <v_i, u> }* where u is successors of v_i. For example, when we calculate the maximum delay of v2 in Fig. 7, the maximum delays of v2's successors have to be computed beforehand. The maximum delay of v5 which is a successor of v2, MaxDelay (v5), can be represented as MaxDelay (t) + d <v5, t>. Thus, its delay becomes 5 from $0 + (3 + 1 \times 1 + 1 \times 1)$. The maximum delay of v6 is 2 by similar calculation. Then, the maximum delay of the next node v3, MaxDelay(v3), can be calculated as MAX

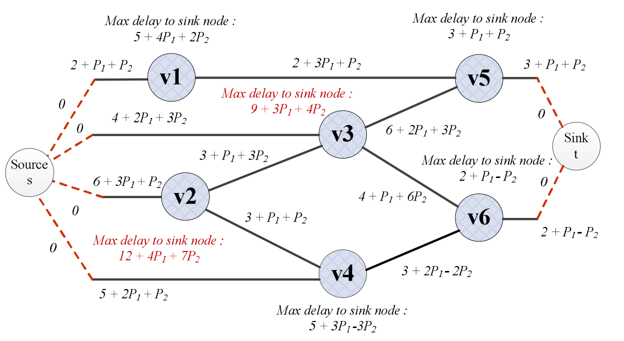

Max delay to sink node : 5 + 4P1 + 2P2
Max delay to sink node : 3 + P1 + P2
Max delay to sink node : 9 + 3P1 + 4P2
Max delay to sink node : 2 + P1 - P2
Max delay to sink node : 12 + 4P1 + 7P2
Max delay to sink node : 5 + 3P1 -3P2
Source s
Sink t
v1
v2
v3
v4
v5
v6
2 + P1 + P2
2 + 3P1 + P2
3 + P1 + P2
4 + 2P1 + 3P2
3 + P1 + 3P2
6 + 2P1 + 3P2
6 + 3P1 + P2
4 + P1 + 6P2
3 + P1 + P2
2 + P1 - P2
5 + 2P1 + P2
3 + 2P1 - 2P2
0

Fig. 7. The example of the maximum delay from each vertex to sink node t using linear function.

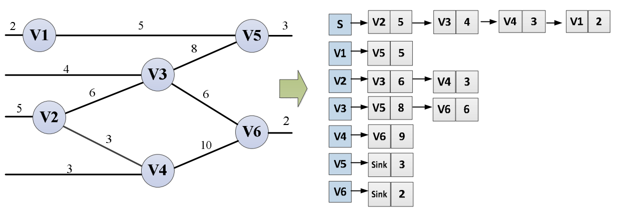


Fig. 8. Successor sort with decreasing order.

{ MaxDelay (v5) + d<v3,v5> , MaxDelay (v6) + d<v3,v6> }. It is equal to MAX { $(3 + P_1 + P_2) + (6 + 2P_1 + 3P_2)$, $(2 + P_1 - P_2) + (4 + P_1 + 6P_2)$ }. Its delay becomes { $9 + 3P_1 + 4P_2$ }. Hence, the maximum delay from v2 to sink is $12 + 4P_1 + 7P_2$. Likewise the maximum delay of each vertex is calculated. The order of the MAX operation [11] is a linear function and it is used to calculate delay of each node in our method.

C. Sorting the Successors

Sorting successors is an efficient way of reducing search space. Thus, the successors of each vertex are stored at adjacent list in decreasing order. Fig. 8 provides an example of the sorted successors list used in this paper. For example, the v3's successors are v5 and v6. Because the delay of v5 is larger than that of v6, the v5 is stored next to v3 at adjacent list in Fig. 8. When the current path is [s, v2, v3] in Fig. 7 and the delay of [s, v2, v3, v6] is smaller than threshold, the vertex v6 will be pruned.

D. Calculating the maximum delay from source node s and finding a global threshold T

Calculating the delay from source node s to each vertex is used for finding a global threshold T. The global threshold T is defined as Kth delay of a path among all corners and modes. Computing the delay from source node s to each vertex starts forward from source node s until the sink node t is reached. The maximum delay from source node s to the v_i is define as *SrcToDelay (v_i)* and it is equal to *MAX { SrcToDelay (u_n) + d <u_n, v_i>}* where u_n are predecessors of v_i. The calculation process is similar to the calculation of maximum delay. This information which is not represented in existing methods is very useful.

Finding a global threshold T is a crucial step in the modified K-most critical paths algorithm. If the initial threshold is set to a proper value, the large amount of search space can be reduced. In existing methods, the threshold T is set to zero or user-define value (n% delay of critical paths). Thus, the search time to find critical paths become increased or unpredictable. The proposed method obtains the global threshold T using MaxDelay() function and SrcToDelay() functions. At all vertices, the maximum delay from source node s to each vertex is added to the maximum delay from each vertex to sink node t, and it is expressed as {SrcToDelay(v_i) + MaxDelay(v_j)}. Also, this function is calculated at all corners and modes. When the number of corners/modes is n and the number of vertices in a circuit is m, the delay of each vertex at 2^n corners is calculated. Then, they are stored at Threshold Bound (TB) array (2^n x m entries) in decreasing order. The Kth entry of TB array is set to the global threshold T. The critical delay path at one corner may not be the critical delay at a different corner. Namely, at a corner, the longest path delay can become the second longest or the third longest path delay at a different corner. Thus, setting threshold T among all corner and modes is efficient. We can reduce search space to K-critical paths. We need not to update threshold T since the global threshold T at first is determined once. We can reduce search time by removing unnecessary update process. For example, Fig. 10 is example of global threshold presented in this paper. At each vertex, the {SrcToDelay(v_i) + MaxDelay(v_j)} function is calculated at 4 corners. At v4, since the number of predecessors is 2 and the number of successors is 1, the number of combination is 2 (2x1). As a result, Total 8 delays (2 combinations x 4 corners) are generated at v4. In a graph, this process is repeated at all vertices and generated delays are stored in TB array with decreasing order. Finally, 3rd entry TB array is set to the global threshold T.

E. Extracting the K-Most Critical Paths at Multi-corner Multi-mode

After computing maximum delay at each node, a data structure array MKPATH that contains K entries is created. The path information is stored in MKPATH in decreasing order. At first, we extract a longest path from the source node s to the sink t using greedy algorithm [2]. Existing methods find one partial or full path at beginning stage. However, to search K-critical paths after finding a longest path can reduce amount of search time. The delay of Kth full path is set as the global threshold T and it is stored in MKPATH[0]. The modified K-most critical paths algorithms utilizes the non-zero threshold value that eventually reduces search space. The newly extracted path that has a delay greater than threshold T is stored in MKPATH. The backward tracing at sink node begins to get the full path which has a delay greater than the current threshold T. When a full path [s, v1, v2, …, v_{n-1}, v_n, t] is generated, the backward trace moves first from sink node t to the vertex v_n. The function nextnode(v_i, v_{i+1}) returns the vertex next to v_{i+1} in the sorted successors list. Otherwise, when v_{i+1} is the last successor in the sorted successor list, the next node will be vertex v_{i-1}. If nextnode(v_i, v_{i+1}) return the vertex v_{nex} next to v_{i+1}, and MaxDelay(P[s, v_i, …,v_i, v_{nex}, t]) is a delay greater than threshold T, the backward tracing is terminated and the forward tracing is used to get the full path that is a delay greater than threshold T. For example, when the function nextnode(v_i, v_{i+1}) is v_3 in Fig. 9, the backward tracing is terminated. The delay from source node s to v3 is added MaxDelay(v3) in Fig. 9. It is equal to A+B+C. If this is greater than threshold T, the forward tracing will progress to v_6. Otherwise, the path that includes v_6 is pruned. The path pruning can reduce large amount of search space. The enumeration stage will be terminated if all the possible paths are generated or pruned. Fig. 11 is the pseudo code of modified K-most critical paths algorithm. For example, when we find the three critical paths in Fig. 6, the first step is to find optimal threshold T for reducing search space. The greedy algorithm will find the full path [s, v2, v3, v5, t] which has the delay 18 + $7P_1$ + $8P_2$. This path and delay are inserted into MKPATH[0]. Then, the backward tracing is started at sink nodes. To determine whether the path is pruned or not, the following conditions have to be checked.

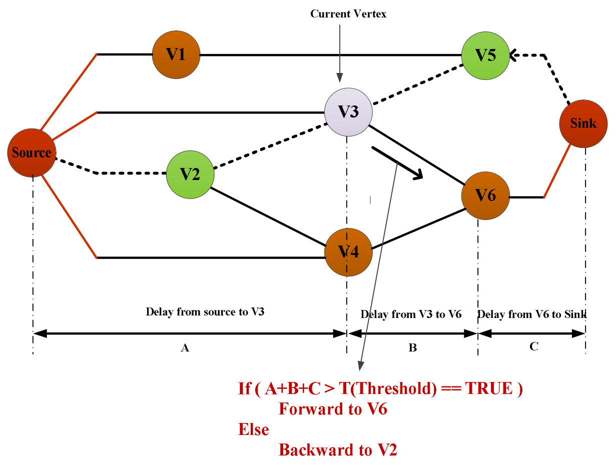


Fig. 9. The example of path pruning with given threshold, backward and forward tracing.

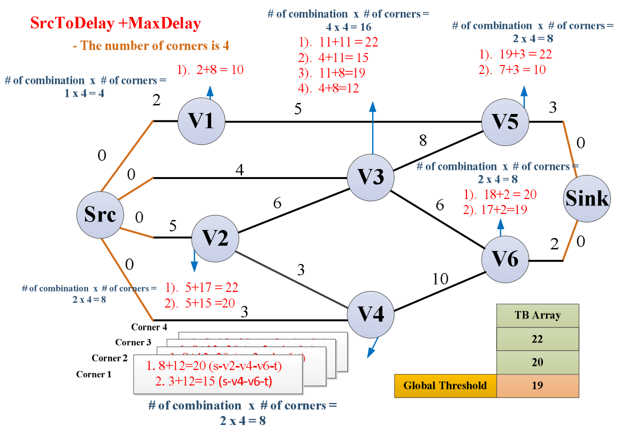

SrcToDelay +MaxDelay
- The number of corners is 4
# of combination x # of corners = 1 x 4 = 4
1). 2+8 = 10
# of combination x # of corners = 4 x 4 = 16
1). 11+11 = 22
2). 4+11= 15
3). 11+8=19
4). 4+8=12
# of combination x # of corners = 2 x 4 = 8
1). 19+3 = 22
2). 7+3 = 10
# of combination x # of corners = 2 x 4 = 8
1). 18+2 = 20
2). 17+2=19
# of combination x # of corners = 2 x 4 = 8
1). 5+17 = 22
2). 5+15 =20
Corner 4
Corner 3
Corner 2
Corner 1
1. 8+12=20 (s-v2-v4-v6-t)
2. 3+12=15 (s-v4-v6-t)
# of combination x # of corners = 2 x 4 = 8
Src
V1
V2
V3
V4
V5
V6
Sink
TB Array
22
20
Global Threshold
19

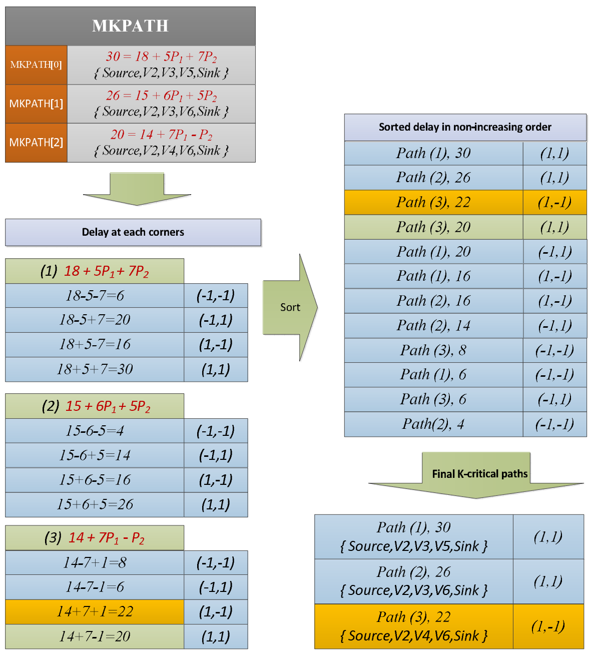
Fig. 10. The example the global threshold T is calculated among 4 process corners (K=3).

Threshold T is 10 + $5P_1$ - $2P_2$, nextnode(v_3, v_5) is v_6, and the delay of P[s, v2, v3, v6] is 15 + $6P_1$ + $9P_2$. Because the delay of P[s, v2, v3, v6] is greater than threshold T, the forward tracing will find the full path P[s, v2, v3, v6, t]. And this path is inserted into MKPATH[1]. The second iteration, backward tracing will trace back to v4. Threshold T is 10 + $5P_1$ - $2P_2$, nextnode(v_2, v_3) is v_4, and the delay of P[s, v2, v4, t] is 14 + $7P_1$ -P_2. Because the delay of P[s, v2, v4, t] is greater than threshold T, the forward tracing will find the full path P[s, v2, v4, v6, t] and its delay is inserted into MKPATH[2]. This process is terminated until all the possible paths are generated or pruned.

Fig. 12 shows the final results after the K-most critical paths in a graph is generated. The parameters p1 and p2 are assigned a value of either -1 or 1. The maximum condition of each parameter is normalized as 1, and the minimum condition of each parameter is normalized as -

```
Algorithm
Let K be the number of most critical paths in graph G
Let MKPATH be the array structure
Let i be the number of predecessors in each ending vertex
Let j be the number of successors in each ending vertex
 1. Create the source node s and the sink node t
 2. Compute the maximum delays to the sink
 3. Compute the delays from source node s
 4. Sort the successors of each vertex
 5. Find a global threshold T
         In the set of all ending vertex V
                   at Multi-corner Multi-mode
            SrcToDelay($V_i$) + MaxDelay($V_j$)
               is stored in TB array in decreasing order
         Update a threshold T to K-th entry in TB array
 6. Extract the K-most critical paths
         Take a longest path using greedy algorithm
         repeat
            Backward tracing
            if (current path delay > global threshold T)
               Forward tracing
               A full path is stored in MKPATH array
                           in decreasing order
         until
            There are K-most completed paths
 7. Final results that K-most critical paths
                  in multi-corner multi-mode scenario
```

Fig. 11. The pseudo code of the modified K-most critical paths algorithm.



MKPATH	
MKPATH[0]	30 = 18 + $5P_1$ + $7P_2$ { Source,V2,V3,V5,Sink }
MKPATH[1]	26 = 15 + $6P_1$ + $5P_2$ { Source,V2,V3,V6,Sink }
MKPATH[2]	20 = 14 + $7P_1$ - P_2 { Source,V2,V4,V6,Sink }

Delay at each corners

(1) 18 + $5P_1$ + $7P_2$	
18-5-7=6	(-1,-1)
18-5+7=20	(-1,1)
18+5-7=16	(1,-1)
18+5+7=30	(1,1)

(2) 15 + $6P_1$ + $5P_2$	
15-6-5=4	(-1,-1)
15-6+5=14	(-1,1)
15+6-5=16	(1,-1)
15+6+5=26	(1,1)

(3) 14 + $7P_1$ - P_2	
14-7+1=8	(-1,-1)
14-7-1=6	(-1,1)
14+7+1=22	(1,-1)
14+7-1=20	(1,1)

Sort

Sorted delay in non-increasing order	
Path (1), 30	(1,1)
Path (2), 26	(1,1)
Path (3), 22	(1,-1)
Path (3), 20	(1,1)
Path (1), 20	(-1,1)
Path (1), 16	(1,-1)
Path (2), 16	(1,-1)
Path (2), 14	(-1,1)
Path (3), 8	(-1,-1)
Path (1), 6	(-1,-1)
Path (3), 6	(-1,1)
Path(2), 4	(-1,-1)

Final K-critical paths

Path (1), 30 { Source,V2,V3,V5,Sink }	(1,1)
Path (2), 26 { Source,V2,V3,V6,Sink }	(1,1)
Path (3), 22 { Source,V2,V4,V6,Sink }	(1,-1)

Fig. 12. The K-most critical paths extraction and enumeration in all corner/modes according to decreasing order.

1. Through this information, we can obtain the delay in all corners and modes. For example, when two process parameters are considered as p1 and p2, the total number of corners are $2^2 = 4$ corners such as (1, 1), (1, -1), (-1, 1), (-1,-1). Thus, timing results can be attained from multi-corner multi-mode scenario.

F. Time Complexity Analysis

Let n be the number of vertices and m be the number of edges in the graph. The creation of source node s and sink node t takes a constant time. In the step of computing maximum delay, each edge of the graph is traversed exactly once. Hence, the overall complexity of computing maximum delay for all the vertices is $O(m+n)$. The time complexity of sorting phase is $O(m+n+mlogm)$. The time complexity of calculating maximum delays from source node t is same as that of computing maximum delay. It becomes $O(m+n+mlogm) \approx O(mlogm)$. In every iteration step, there is one deletion and d_{max} insertions. If the number of iterations is N_{itr}, the time complexity of the enumeration phase becomes $O(N_{itr} \cdot logK)$. Therefore, the overall time complexity of the modified K-most critical paths algorithm is $O(m+n+mlogm+N_{itr} \cdot logK) \approx O(mlogm+N_{itr} \cdot logK)$.

IV. EXPERIMENT RESULTS

The modified K-most critical paths algorithm using multi-corner multi-mode static timing analysis (MCMM STA) is implemented in C/C++ language. We verified the proposed method using a 32mm technology and used a predictive technology model (PTM) [15] with HSPICE. In our experiment, we considered the following six process parameters: temperature, voltage, oxide film thickness, threshold voltage, channel length, and width. Process variations have a normal distribution and the standard deviation is set to 5% of their average. Table 1 shows the run time when the number of paths to find K is arbitrarily set to one hundred in a typical corner. We applied our algorithm to ISCAS'85 benchmark circuits: c432, c499, c880, c1355, c1908, c2670 and compared with breadth first search with branch and bound (BFSB&B), depth first search with branch and bound (DFSB&B). The BFSB&B and DFSB&B [16, 17] are strategies that combined basic DFS/BFS with B&B (pruning). The result shows that our method take less

Table 1. The run time of the proposed method compared to breadth-first search (BFS) and depth-first search (DFS), with K=100

ISCAS'85 CIRCUIT	Run time for Critical Path Search (sec) , K=100				
	BFS with B&B	DFS with B&B	Proposed Method	Improvement (%)	
C432	5.983	6.415	4.193	29.92%	34.64%
C499	5.366	5.234	3.466	35.41%	33.78%
C880	5.817	5.909	5.317	8.60%	10.02%
C1355	5.292	5.532	3.592	32.12%	35.07%
C1908	14.328	15.433	10.428	27.22%	32.43%
C2670	18.439	18.638	12.439	32.54%	33.26%

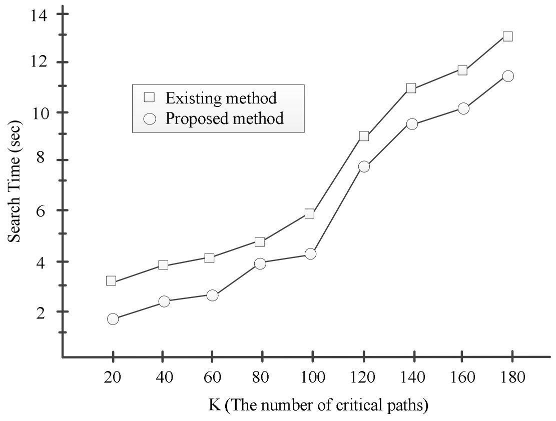


Fig. 13. The global threshold T used in modified K-most critical paths algorithm.

time than the others. Our proposed method improve search time up to 30%.

The global threshold proposed in our method is compared with the threshold of existing method. The existing method set its initial threshold to 0 and updated it. Fig. 13 is the search time on c432 benchmark circuit when the number of critical paths K is set from 20 to 180 and the number of process parameter is set to two. The global threshold T used in modified K-most critical paths algorithm more reduces the search space than existing method. Thus, the global threshold T presented in the modified K-most critical paths algorithm is efficient and fast when we find the K-most critical paths. Also the modified K-most critical paths algorithm is more efficient in multi-corner multi-mode scenario.

To show the efficiency of the proposed method, we compared a global threshold T presented in this paper with existing methods. Existing methods that include original K-critical path generally use a threshold T as zero or n% delay of critical path. The number of pruning

Table 2. Critical path search time in MCMM scenario

ISCAS'85 CIRCUIT	Run time for Critical Path Search (sec)														
	K=100, No. of corners = 2^4					K=100, No. of corners = 2^6					K=120, No. of corners = 2^6				
	BFS with B&B	DFS with B&B	Proposed Method	Improvement (%)		BFS with B&B	DFS with B&B	Proposed Method	Improvement (%)		BFS with B&B	DFS with B&B	Proposed Method	Improvement (%)	
C432	26.72	25.21	22.31	16.51	11.50	38.21	37.30	29.67	22.35	20.47	47.41	44.43	35.04	26.11	21.14
C499	35.76	33.14	24.27	32.13	26.76	47.14	40.10	32.28	31.52	19.49	60.91	48.95	38.13	37.41	22.11
C880	45.97	38.72	31.85	30.71	17.73	58.14	48.97	38.18	34.34	22.03	69.54	55.47	46.53	33.09	16.12
C1355	57.75	54.49	39.42	31.75	27.65	78.59	66.64	53.34	32.13	19.96	88.19	75.37	64.54	26.82	14.37
C1908	77.62	73.23	56.64	27.03	22.65	101.00	97.40	70.37	30.33	27.75	113.67	117.85	97.18	14.50	17.54
C2670	98.24	94.38	77.46	21.16	17.93	140.48	120.03	96.28	31.47	19.79	172.76	140.46	112.74	34.74	19.73
Average	-			26.55	20.70	-			30.36	21.58	-			28.78	18.50

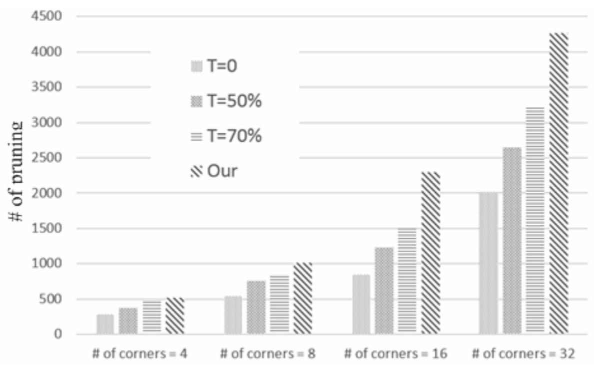


Fig. 14. Thu number of pruning operation used in modified K-most critical paths algorithm according to the number of corners (K=100) at c432 circuit.

happened during traversing a graph is small in existing methods. Therefore, they are time-consuming and it is difficult to predict search time. Fig. 13 represents the number of pruning operation used in modified K-most critical paths algorithm according to the number of corners when K is one hundred at c432. The number of pruning in proposed method is smaller than existing methods based on DFS with pruning. Also, as the corners increased, our method is fast and efficient.

In Table 2, to prove this, we applied our method to circuits in multi-corner multi-mode and compared the run-time in our method with breadth first search with branch and bound (BFSB&B) and depth first search with branch and bound (DFSB&B). The number of corners is set to 2^4 and 2^6. The number of paths to find K is arbitrarily set to 100 and 120. The proposed method reduces more search space than other methods. Through this experiment, as the number of corners increased, our method is more efficient than existing methods. Because our proposed method assigns initial threshold to Kth delay of a path among all paths in all the given corners, the number of pruning in traversal process is more increased than BFSB&B and DFSB&B. Thus, the results show that the proposed method reduced the search time more by approximately 30.36%, 21.58%, compared to BFSB&B and DFSB&B respectively. Therefore, the modified K-most critical paths algorithm is faster than existing methods.

V. CONCLUSIONS

The modified K-most critical paths algorithm in multi-corner multi-mode is presented. It reduces the search space through path pruning. Also, to further prune paths, we propose initial threshold bound that is called global threshold T and our algorithm uses the global threshold T that is obtained in multi-corner multi-mode scenario. Thus, we can quickly extract the K-most critical paths in MCMM scenario and reduce the time required to analyze circuits. The experiment results demonstrate that the proposed method reduces the search time by up to 30.36% than existing methods.

ACKNOWLEDGMENTS

This research was supported by the MISP, Korea, under the National Program for SW in Excellence (R7718-16-1004) supervised by the IITP and supported by IDEC(MPW, EDA Tool). Also, we thank Samsung Electronics LSI team for their support with the research grant.

REFERENCES

[1] Y. C. Ju and R. A. Saleh, "Incremental techniques for the identification of statically sensitizable

critical paths," in *Proc. ACM/IEEE Design Automation Conf.*, pp. 541–546, 1991.

[2] L. Liu, D. Du, and H.-C. Chen, "An efficient parallel critical path algorithm," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, Vol. 13, No. 7, pp. 909–919, Jul. 1994.

[3] L. Xie and A. Davoodi, "Bound-based statistically-critical path extraction under process variations," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, Vol. 30, No. 1, pp. 59–71, Jan. 2011.

[4] K. Heloue et al., "Efficient block-based parameterized timing analysis covering all potentially critical paths," *IEEE Trans. on CAD*, Vol. 31, pp. 472–484, 2012.

[5] S. H. C. Yen, D. C. Du, and S. Ghanta, "Efficient Algorithms for extracting the K most critical paths in timing analysis," *26th ACM/IEEE Design Automation Conference*, pp. 649-654, June 1989.

[6] W. Qiu and D. M. H. Walker, "An efficient algorithm for finding the k longest testable paths through each gate in a combinational circuit," *in Proc. IEEE Int. Test Conf.*, pp. 592–601, Oct. 2003.

[7] J. Bhasker and R. Chadha, *Static Timing Analysis for Nanometer Designs: A Practical Approach* (1st ed.), New York, NY: Springer Science & Business Media, 2009.

[8] C. Visweswariah, K. Ravindran, K. Kalafala, S. Walker, and S. Narayan, "First-order incremental block-based statistical timing analysis" *in Proc. Des. Autom. Conf.*, pp. 331–336, Jun. 2004.

[9] H. Chang and S. S. Sapatnekar, "Statistical timing analysis considering spatial correlations using a single PERT-like traversal," *in Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, pp. 621–625, Nov. 2003.

[10] S. V. Kumar et al., "A Framework for Block-Based Timing Sensitivity Analysis," *in Proc. Des. Autom. Conf*, pp. 688-693, Jun.2008.

[11] L. M. Silveira and J. R. Phillips. "Efficient computation of the worst-delay corner," in *Proc. Design Automation and Test in Europe*, pp. 1617–1622, 2007.

[12] H. Li Z. He T. Lv and X. Li. "Test path selection for capturing delay failures under statistical timing model," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.,* Vol. 21, No. 7, pp.1210-1219, 2013.

[13] S. Onaissi and F. N. Najm, "A linear-time approach for static timing analysis covering all process corners," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, Vol. 27, No. 7, pp. 1291–1304, Jul. 2008.

[14] J. J. Nian, S. H. Tsai, and C. Y. Huang. "A unified multi-corner multi-mode static timing analysis engine," *in ASP-DAC*, pp. 669–674, 2010.

[15] Predictive Technology Model, downloaded from http://ptm.asu.edu/.

[16] N. R. Vempaty, V. Kumar and R. E. Korf, "Depth-first vs best-first search," *in: Proceedings AAAI-91, Anaheim, CA,* 434-440, 1991

[17] W. Zhang, *State-Space Search: Algorithms, Complexity, Extensions, and Applications*, Springer, New York, NY, 1999.

Deok-Keun Oh received the B.S. degree and Master degree in Computer Science and Engineering from Sogang University, Korea in 2012 and 2014. He is currently pursuing a Ph.D degree in Computer Science and Engineering at Sogang University, Korea. His research interests are variation-aware timing analysis, clock tree synthesis, Monte-Carlo analysis, design for reliability enhancement.

Myeoung-Woo Jin received the B.S. degree and Master degree in Computer Science and Engineering from Sogang University, Korea in 2012 and 2014. He is currently pursuing a Ph.D degree in Computer Science and Engineering at Sogang University, Korea. His research interests are variation-aware timing analysis, design for reliability enhancement, interconnect variation.

Ju-Ho Kim received B.S degree and Ph.D degree in Computer and Information Science from University of Minnesota in 1987 and 1995, respectively. After getting Ph.D egree, he worked as a senior member of technical staff at Cadence Design System until 1997. Professor Kim joined the department of computer science and engineering in Sogang University, Seoul, Korea in 1997, and he was a department chair from 2005 to 2008. His research interests are variation-aware timing anaysis, low power design.