Exact Path Sensitization in Timing Analysis

R. Peset Llopis
Centro Nacional de Microelectrónica (CSIC)
Universidad Autónoma de Barcelona
08193 Bellaterra, Spain

Abstract

Timing verification is an important aspect in chip design. However, the growing complexity of combinational circuits increases the total number of false paths, which demands fast and accurate false path elimination methods. Several approaches have been presented in literature, but all are based on approximations of the exact criterion, and offer no exact results. This paper presents the first implementation of the exact criterion. Experiments show that this tool is much more accurate in comparison with other approaches.

1. Introduction

The maximum operational frequency of a circuit is determined by the maximum propagation delay of its combinational parts, which is defined as the longest delay it takes for a signal to propagate from a primary input to a primary output. In other words, the maximum delay of a combinational part is equal to the length of its critical paths. A straightforward approach to compute the length of the critical paths is to simulate the combinational circuit for all possible input vectors (vector dependent approach). However, since the number of different input vectors increases exponentially with the number of inputs, this approach is only feasible for to circuits with only a few primary inputs. The first approach to compute the length of the critical paths, without simulating all input vectors (vector independent approach), was based on the PERT (Program Evaluation and Review Technique) algorithm [1]. Since this approach determines the longest path without taking the logical dependencies into account, it is very likely to find a false path. The PERT delay was therefore used as an upper bound for the length of the critical paths. However, the growing complexity and integration of combinational circuits started to demand tighter bounds. This has resulted in different timing analysis (or critical path finding) approaches, each one based on a different path sensitization criterion. The static criterion [2] provides a lower bound, while others [3-6] offer an upper bound. The viability criterion [7] gives the exact delay of the critical paths.

The first criterion, able to offer the same results as by simulating the circuit for all input vectors was proposed by [8]. It is called the exact or timing simulation criterion. The exact criterion requires knowing timing information at all nodes in the network, which seems to be incompatible with the vector independent path tracing approach. This is why all timing verifiers use approximations of the exact criterion [6], instead of a direct implementation of this criterion. This paper presents the first critical path finding tool based on the exact criterion. It offers therefore better results in comparison with all other approaches, since these are based on approximations of this criterion.

The rest of this paper is organized as follows. Section 2 discusses the exact and other criteria. A description of the proposed algorithm is given in section 3. Then the two steps responsible for eliminating false paths will be described in section 4. Section 5 discusses the results obtained by the proposed tool. Finally we will finish this paper with the conclusions of section 6.

2. Path sensitization criteria

We assume that the reader is familiar with concepts like (non)controlling value, (non)controlling input, on-input, side-input, dominating input, stable time, stable value, sensitizable (or true) path and nonsensitizable (or false) path [6]. The exact criterion is based on the following definition:

Definition 1 (Exact Criterion): Let v be an input vector of the combinational network C, and let P = (G_0, f_0, G_1, f_1, .., G_{n-1}, f_{n-1}, G_n) be a path in this network. Path P is considered to be an exact sensitizable path under v if for each lead f_i ($0 \le i \le n-1$) one of the following two conditions is true:

1) Lead f_i is the controlling input of gate G_{i+1} with the smallest stable time under v.
2) Lead f_i is the noncontrolling input of gate G_{i+1} with the largest stable time, and all inputs of gate G_{i+1} are noncontrolling inputs under v.

The sensitization criteria are characterized by three properties:
Delay correctness: A criterion is delay incorrect if it may underestimate the critical path delay. It is correct otherwise. Delay correctness is crucial for timing verification.
Path correctness: A criterion is path incorrect if it may claim an exactly sensitizable path to be false. It is correct otherwise. Performance optimization is carried out by determining the set of paths with a delay longer than a given threshold, say τ, and by reducing their delays. If a path incorrect criterion is used, it is necessary to guarantee that shortening all sensitizable paths with delays longer than τ, implies that all true paths, which are claimed to be false, are also shortened or become false paths.
Exactness: A criterion is exact if it claims every true path to be true, and every false path to be false. It gives the same results as can be obtained by analyzing the circuit for all input vectors.

A timing analysis tool, based on this criterion, determines not only the critical path, but also the input vector that activates it. Therefore it is possible to use this input vector, in combination with a circuit simulator, to compute accurately the critical path delay. Non-exact criteria can find false paths, which cannot be simulated to obtain more accurate critical path delays.

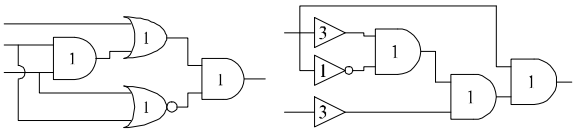
The characteristics of the several criteria are shown in table 1, which is based on [6] (path correctness and delay correctness are denoted as ´correct for the path sensitization problem´ and ´correct for the critical path problem´ by this author). The symbol "X" denotes a violation, while the symbol "√" means that the criterion satisfies a property. Furthermore, the symbols ´≤´, ´≥´ and ´=´ stand for an underestimation, an upperestimation and an exact result of the critical path delay, respectively.

Criterion	Delay Correct	Path Correct	Exact
Static [2]	X(≤)	X	X
BI [3]	√(≥)	X	X
DYG [4]	√(≥)	√	X
PCD [5]	√(≥)	√	X
Approx [6]	√(≥)	X	X
Viable [7]	√(=)	√	X
Exact [8]	√(=)	√	√

Table 1. Summary of path sensitization criteria.

This table shows that the exact criterion [8], is the most accurate one. However, no timing verifier uses this criterion, due to its complexity, and all verifiers are based on approximate criteria.

Consider the circuits shown in figure 1. All gate delays are shown in the corresponding gates.



(a) (b)

Figure 1. Some circuit examples.

The static criterion of [2] requires that all side-inputs of the paths are set to noncontrolling values. This can lead to an underestimation of the critical path delay. The circuit of figure 1a has a critical path delay of three time units; the static criterion claims it to be two. The approximate criterion [6] is a relaxation of the exact criterion, and can lead to an overestimation of the critical path delay. This is demonstrated by the circuit of figure 1b. The critical path delay is four time units, while the approximate criterion claims it to be five.

3. Algorithm

The algorithm is based on tracing the paths from a primary input towards a primary output, using a depth-first-search (DFS) [2]. Let P be a partial path to be expanded, starting at a primary input, and ending at gate G_{i-1}. Let lead f be an output of gate G_{i-1}, and an input to gate G_i. Lead f and gate G_i are added to the partial path if one of the following two conditions is true:

1) Lead f is a noncontrolling input to gate G_i, and setting all side-inputs of lead f to nonlater noncontrolling inputs produces no conflicts.
2) Lead f is a controlling input to gate G_i, and setting all side-inputs of lead f to nonearlier controlling inputs or to noncontrolling inputs produces no conflicts.

The only difference with the approach of [2] is that we take the stable times into account. This is achieved by storing for each gate output, not the logical value, but intervals for the stable times of a logical 0 and a logical 1, as explained by the following definition.

Definition 2 (Stable Times Intervals): Let T_0 and T_1 be the stable times of a gate or lead for a stable value equal to logical 0 and 1, respectively. The stable times of a gate or lead are defined as the stable times at the output of the gate or at the end of the lead, respectively. The stable time intervals of this gate or lead are defined as: $[T_{min0},T_{max0}][T_{min1},T_{max1}]$ with $T_{min0} \le T_0 \le T_{max0}$ and $T_{min1} \le T_1 \le T_{max1}$. If the stable value of the same gate or lead is equal to logical 1, then the corresponding stable time interval satisfies the condition $T_{min0} > T_{max0}$. The stable time interval of a gate or lead with a stable value equal to logical 0, satisfies the condition $T_{min1} > T_{max1}$.

Assume that the stable value and stable time at the end of the previously defined partial path P (of gate G_{i-1}) are equal to logical 0 and T^P respectively. Path P is expanded with lead f (of delay $d(f)$) and gate G_i if one of the following two conditions is true:

1) Lead f is a noncontrolling input of gate G_i, and setting the stable time intervals of lead f to $[T^P+d(f),T^P+d(f)][\infty,0]$ and the stable time intervals of the side-inputs of lead f to $[0,T^P+d(f)][\infty,0]$ produces no conflicts.
2) Lead f is a controlling input of gate G_i, and setting the stable time intervals of lead f to $[T^P+d(f),T^P+d(f)][\infty,0]$ and the stable time intervals of the side-inputs of lead f to $[T^P+d(f),\infty][0,\infty]$ produces no conflicts.

The following two definitions are used to determine how to deal with new conditions for the stable time intervals.

Definition 3 (Intersection Rule): Let $[\mathcal{T}_{min0},\mathcal{T}_{max0}][\mathcal{T}_{min1},\mathcal{T}_{max1}]$ be the current stable time intervals of gate G or lead f, and let $[T_{min0},T_{max0}][T_{min1},T_{max1}]$ be the stable time intervals which must be applied to the same gate or lead. The resulting intervals will be: $[\max(\mathcal{T}_{min0},T_{min0}),\min(\mathcal{T}_{max0},T_{max0})][\max(\mathcal{T}_{min1},T_{min1}),\min(\mathcal{T}_{max1},T_{max1})]$. If after applying the intersection rule we obtain $T_{minX} > T_{minX}$, then we set T_{minX} and T_{maxX} equal to ∞ and 0, respectively (X equal to logical 0 or 1).

Definition 4 (Conflict): Let $[\mathcal{T}_{min0},\mathcal{T}_{max0}][\mathcal{T}_{min1},\mathcal{T}_{max1}]$ be the stable time intervals of a gate or lead, obtained after applying the intersection rule. If $\mathcal{T}_{min0}>\mathcal{T}_{max0}$ and $\mathcal{T}_{min1}>\mathcal{T}_{max1}$ then a conflict arises.

It is clear that the stable value can be derived from the stable time intervals, $[T_{min0},T_{max0}][T_{min1},T_{max1}]$, using the following four rules:

1) $T_{min0}\leq T_{max0}$ and $T_{min1}>T_{max1} \rightarrow$ logical 0.
2) $T_{min0}>T_{max0}$ and $T_{min1}\leq T_{max1} \rightarrow$ logical 1.
3) $T_{min0}>T_{max0}$ and $T_{min1}>T_{max1} \rightarrow$ conflict.
4) $T_{min0}\leq T_{max0}$ and $T_{min1}\leq T_{max1} \rightarrow$ unknown.

In practice, we store only the stable time intervals of the gates, since the intervals of a lead can be derived from those of its driving gate, by adding the delay of this lead. Adding a new interval condition to a lead, is carried out by applying it to its driving gate, after subtraction the lead delay.

The conditions are propagated recursively in backward and forward direction through the network by the implication step, described in the following section. If no conflict is detected, then the extended path can be a exactly sensitizable path. A conflict means that the extended path is an exactly nonsensitizable path.

The stable time intervals can be initialized to [0,∞][0,∞] before starting the expansion of the paths from the primary inputs towards the primary outputs. However, the computation time can be reduced considerably by computing tighter starting intervals. This, because tighter bounds will give the search algorithm more often the possibility to prune a search direction. The minimum and maximum PERT [1] delays between all primary inputs and the output of each gate G_i are computed, and propagated through the network by the implication steps of the next section. The following figure shows how the PERT bounds at the output of a 2 input AND (with unit gate delay) can be improved by forward implication.

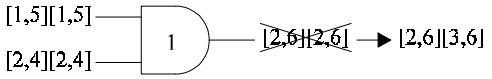


Figure 2. Computing tight initial stable time intervals.

The path search is guided by the esperance value of the partial paths. The esperance value of a partial path is the sum of the path delay and the largest PERT delay from the end of this path to a primary output. It is an upper bound for the delay of the longest sensitizable path containing the partial path.

The following figure shows an example of a path search using the exact criterion. The gate delays are shown in the corresponding gates (leads have no delay). The initial stable time intervals can also be found in this figure.

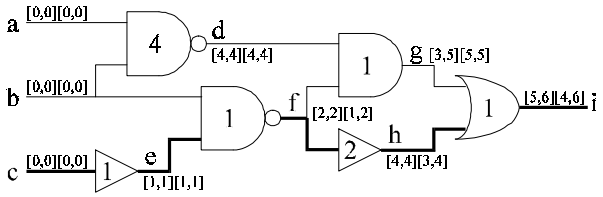


Figure 3. Before tracing the critical path.

The longest exact sensitizable path in this network is c-e-f-h-i, for parity 0 or 1 at input c. Suppose we are tracing this path for parity 0. First we applying [0,0][∞,0] at input c, leading to the intervals [0,0][∞,0] at input c (will be written as c[0,0][∞,0]). Forward and backward implication of this interval results in: e[1,1][∞,0], f[∞,0][1,2], g[5,5][5,5], h[∞,0][3,4] and i[∞,0][4,5]. Now we arrive at the input of the NAND gate of delay 1. Setting [1,1][∞,0] at lead e changes nothing. Lead e is a controlling input of this gate. Therefore we set lead b to a noncontrolling or nonearlier controlling input ([1,∞][0,∞]), leading to b[∞,0][0,0]. Forward and backward implication: f[∞,0][2,2], h[∞,0][4,4] and i[∞,0][5,5]. We are now at the input of the buffer of delay 2. Setting [∞,0][2,2] at lead f changes nothing. Now we apply [∞,0][4,4] at lead h, which again modifies no intervals. Lead h is a controlling input of the OR gate; lead g must be a noncontrolling or nonearlier controlling input ([0,∞][4,∞]), which is already the case. This completes the path search, since the output is reached. The following figure shows the resulting stable time intervals.

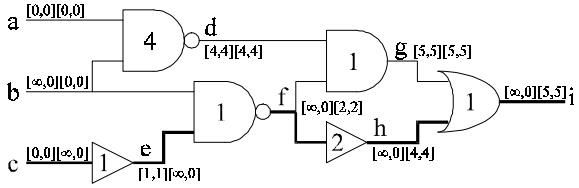


Figure 4. After tracing the critical path.

This example also demonstrates that forward and backward implication may not find the stable values and stable times of all nodes in the network. These can be computed during the justification step.

4. Implication and justification

The logical incompatibilities of the path being traced can be detected by the D-algorithm [9]. Since a complete D-algorithm check on a single path is very time consuming, it is better to split the D-algorithm into an implication and a justification step. The first step detects most false paths, but is not able to detect them all. Therefore some approaches [2] use the time-consuming justification step each time an expanded path reaches a primary output, to remove the remaining false paths. Other approaches [4,6] do not use the justification step to improve the CPU-times, since they propose a conservative timing analysis tool. In fact, a timing verifier without the justification step may claim a false path to be true and can overestimate the critical path delay. Exact results can therefore

only be obtained by using an exact criterion in combination with the justification step. Therefore we present both modified implication and justification steps, which take the stable time intervals into account.

4.1. Implication

There are two different types of implication: forward and backward implication. Both are more complex in comparison with those proposed by [2], since they take the stable time intervals into account.

Let G be a gate with input leads f_i ($1 \le i \le n$). Let the stable time intervals of lead f_i be $[T^i_{min0},T^i_{max0}][T^i_{min1},T^i_{max1}]$, and the stable time interval of gate G be $[T_{min0},T_{max0}][T_{min1},T_{max1}]$.

Forward implication computes the stable time intervals of gate G. The functions $max(T^i,x)$ and $min(T^i,x)$ determine the maximum and minimum values of T^i, for the inputs of gate G with a stable value equal to logical x (= 0, 1, or ?). Variable i goes over all inputs of G ($1 \le i \le n$). The functions MIN and MAX return the minimum and maximum values of its arguments, respectively.

```
if gate G is BUFF gate
  if the input is 0
    apply [T1min0+d(G),T1max0+d(G)][∞,0] at output
  else if the input is 1
    apply [∞,0][T1min1+d(G),T1max1+d(G)] at output
  else /* the input is unknown */
    apply [T1min0+d(G),T1max0+d(G)][T1min1+d(G),
      T1max1+d(G)] at output
if gate G is AND gate
  if all inputs are 1
    apply [∞,0][max(Timin1,1)+d(G),max(Timax1,1)+d(G)] at
      output
  else if any input is 0
    apply [MIN{min(Timin0,0),min(Timin0,?)}+d(G),
      min(Timax0,0)+d(G)][∞,0] at output
  else /* no input is 0 and any input is unknown */
    apply [min(Timin0,?)+d(G),max(Timax0,?)+d(G)][MAX
      {max(Timin1,1),max(Timin1,?)}+d(G), MAX
      {max(Timax1,1),max(Timax1,?)}+d(G)] at output
```

Figure 5. The rules of forward implication.

Backward implication determines the stable time intervals of (some of) the inputs of gate G as follows:

```
if gate G is BUFF gate
  if the output is 0
    apply [Tmin0-d(G),Tmax0-d(G)][∞,0] at input
  else if the output is 1
    apply [∞,0][Tmin1-d(G),Tmax1-d(G)] at input
  else /* the output is unknown */
    apply [Tmin0-d(G),Tmax0-d(G)][Tmin1-d(G),Tmax1-d(G)]
      at input
if gate G is AND gate
  if the output is 1
    if there is only one input with Timax1 ≥ Tmin1-d(G)
      apply [∞,0][Tmin1-d(G),Tmax1-d(G)] at that input
    apply [∞,0][0,Tmax1-d(G)] at all inputs
  else if the output is 0
    if there is only one input with stable value equal to ? or 0
      apply [Tmin0-d(G),Tmax0-d(G)][∞,0] at that input
    else if there is only one input with Timin0 ≤ Tmax0-d(G)
      apply [Tmin0-d(G),Tmax0-d(G)][∞,0] at that input
    else
      apply [Tmin0-d(G),∞][0,∞] at all inputs
  else /* the output is unknown */
    apply [Tmin0-d(G),∞][0,∞] at all inputs
```

Figure 6. The rules of backward implication.

Similar expressions can be derived for the backward and forward implication of NOT, NAND, OR and NOR gates.
The stable time intervals are propagated recursively through the network in forward and backward direction to detect any conflicts.

4.2. Justification

The justification step is used to eliminate the remaining false paths. Justification of the exact criterion is more complex than that of the static criterion [2], which will be described first.

The *static criterion* deals only with stable values. All inputs of noncontrolling gates (gates requiring noncontrolling inputs) must be justified. However, only one input of each controlling gate (gates requiring at least one controlling input) needs to be justified, which implies that a decision must be made for each controlling gate. In this case, one of the inputs is selected and set to the controlling value, followed by backward and forward implications. If no conflict is detected, then the gate driving the selected input is justified. Otherwise, the next unexplored input is tried. If all inputs have been explored, then a backtrack is carried out. A path is sensitizable if all necessary gates have been justified without conflicts. On the other hand, if a conflict is detected and no more backtracks can be carried out, then the path is nonsensitizable.

The *exact criterion* requires justifying also the stable time intervals. Decisions must be made for controlling and noncontrolling gates. For each noncontrolling gate a dominating input must be selected and its side-inputs must be set to nonlater noncontrolling inputs. On the other hand, for each controlling gate a dominating input must be selected and its side-inputs must be set to noncontrolling or nonearlier controlling inputs. Both for controlling as for noncontrolling gates all its inputs must be justified. Justification is carried out in two steps.
The *first step* selects for each gate a dominating input. All inputs, except the side-inputs of the dominating controlling inputs, are justified during this step. The strategy for not justifying these side-inputs is that their stable values are not

important, as long as they are noncontrolling or nonearlier controlling inputs. The final stable values are very often determined by propagating the stable time intervals of other justified leads. This results in a considerable reduction of the justification search space.

During the *second step* we justify all the remaining leads. Most of the side-inputs of dominating controlling inputs have already known logical values, which simplifies the justification task considerably. Furthermore, some of these side-inputs have been justified indirectly during the first step. Both steps will be explained on a AND gate.

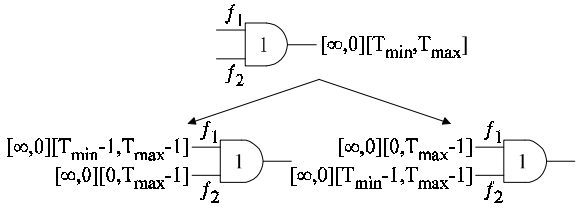


Figure 7. Justifying a AND2 with noncontrolling inputs.

Let gate G be a 2 input AND gate with input leads f_1 and f_2. Suppose we want to justify the stable time intervals [∞,0][T_{min},T_{max}] at its output, as shown in figure 7. Since the stable value at the output of gate G is equal to logical 1, all its inputs must be noncontrolling inputs. However, there are two possibilities in choosing the dominating input. Suppose we choose lead f_1 as the dominating input of gate G. Its stable time interval must satisfy [∞,0][T_{min}-d(G),T_{max}-d(G)]. Lead f_2 is set to an earlier noncontrolling input by the constraint [∞,0][0,T_{max}-d(G)].

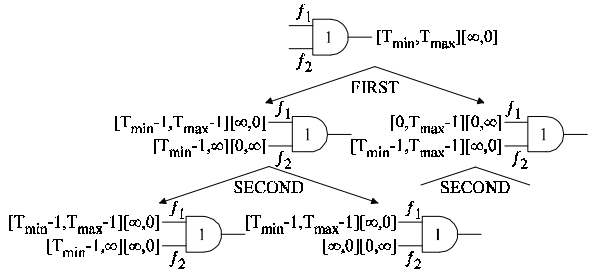


Figure 8. Justifying a AND2 with controlling inputs.

On the other hand, if we want to justify the stable time intervals [T_{min},T_{max}][∞,0], at the output of gate G, its dominating input must be a controlling input. Again there are two possibilities for the dominating input. Suppose we choose again lead f_1 as the dominating input, and apply [T_{min}-d(G),T_{max}-d(G)][∞,0] to its stable time intervals. The difference between the first and second steps of the justification phase is how they deal with the side-inputs of the controlling dominating inputs. During the first step we set lead f_2 to a noncontrolling or a later controlling input: [T_{min}-d(G),∞][0,∞], without carrying out a justification of this lead. The second step sets lead f_2 to either a noncontrolling or a later controlling input, [T_{min}-d(G),∞][∞,0] or [∞,0][0,∞], and justifies it. This is illustrated in figure 8.

The justification step uses forward and backward implications of the decisions made, to detect any conflicts. However, during these implications we must guarantee that a dominating input of a gate will be always dominating. This is achieved by adding another implication rule, called vertical implication. Let G be a gate with input leads f_i ($1 \le i \le n$). Let the stable time intervals of lead f_i be [T^i_{min0},T^i_{max0}][T^i_{min1},T^i_{max1}]. Assume that lead f_k is the dominating input of gate G.

Vertical implication computes the stable time intervals of all side-inputs of f_k as follows:

```
if gate G is AND gate
  if the output is 1
    apply [∞,0][0,T^k_max1] at all side-inputs of f_k
  else if the output is 0
    apply [T^k_min0,∞][0,∞] at all side-inputs of f_k
```

Figure 9. First part of the rules of vertical implication.

Vertical implication is also used to compute the stable time interval of the dominating input f_k. The functions max(T_i, x) and min(T_i, x) determine the maximum and minimum values of T_i for the side-inputs of f_k with a stable value equal to logical x (= 0, 1 or ?). Variable i is $1 \le i \le n$ and $i \ne k$.

```
if gate G is AND gate
  if the output is 1
    apply [∞,0][max(T^i_min1,1),0] at f_k
  else if the output is 0
    apply [0,min(T^i_max0,0)][∞,0] at f_k
```

Figure 10. Second part of the rules of vertical implication.

Similar expressions can be derived for the vertical implication of the inputs of NAND, OR and NOR gates. Vertical implication is used, together with forward and backward implication, recursively during justification to detect any conflicts.

5. Results

The approximate [6], static [2] and the proposed exact criterion have been implemented in C-programs. The ISCAS85 benchmarks with random gate delays have been used as test cases. Surprisingly, all criteria find the same critical path delay for these circuits. In table 2 we show the CPU-time for these benchmarks on a Sun Sparc10 workstation. In this table, PERT and Critical stand for the maximum PERT and critical path delay. Furthermore, the static and exact criteria are a DFS search, while the approximate criterion is a best-first-search (BFS). The latter is not able to compute the c6288 circuit, since it requires too much memory. The exact criterion requires more CPU-time, since it takes not only the stable values, but also the stable time intervals into account.

Circuit	Delay		CPU-seconds		
	PERT	Critical	Approx	Static	Exact
c17	145	145	0	0	0
c880	1414	1414	0	0	0
c1355	1596	1578	1	1	1
c1908	2392	2230	5	4	22
c2670	1943	1818	4	5	8
c3540	2703	2578	7	7	52
c5315	2674	2553	6	7	9
c6288	7532	7182	-	4008	2475
c7552	2575	2472	11	13	16

Table 2. The CPU-times for tracing the critical path.

However, the accuracy of these criteria can be compared by computing the number of paths with delays between 90% and 100% of the critical path delay. Table 3 shows these results for all benchmarks, but the c6288 circuit, which takes too long to complete. These results demonstrate that the exact criterion reduces considerably the set of paths that are claimed to be true. However, since no justification has been applied on these paths, it is only correct to state that the number of paths found by the exact criterion is only an upper bound for the total number of exact sensitizable paths between 90 % and 100 % of the critical path delay. This is because exact results can only be obtained by using the exact criterion, in combination with the justification step.

Circuit	Number of paths			CPU-seconds		
	Static	Approx	Exact	Static	Approx	Exact
c17	8	8	6	0	0	0
c880	392	409	242	3	3	8
c1355	1648	1656	668	12	12	31
c1908	1566	1566	1362	82	81	234
c2670	1579	1798	882	45	49	76
c3540	554	2287	535	90	116	430
c5315	637	651	438	27	28	58
c7552	228	307	206	18	21	35

Table 3. The number of paths for different criteria.

All the paths between 90 % and 100 % of the critical path delay have been justified to determine the number of false paths that are not detected during the path tracing phase. Table 4 shows the number of true and false paths after justification, for the three different criteria. However, justification is very time consuming, since the search space increases exponentially with the number of gates to justify. Therefore we used an upper bound on the number of backtracks to justify a single path. The total number of paths that could not be justified within this bound is shown as "Break". These results show that the number of false paths not detected during the path tracing phase can be important, as demonstrated by circuit c2670. However, the majority of paths are true after justification. Furthermore it is clear that justification is the bottleneck of critical path finding. The justification of the exact criterion is more complicated in comparison with the others, since we justify also the side-inputs of dominating controlling inputs. However, this does not lead to larger justification problems, since the percentage of non justified paths (breaks) of the exact criterion is approximately equal to that of the other criteria.

Circuit	Static Criterion			Approx Criterion			Exact Criterion		
	True	False	Break	True	False	Break	True	False	Break
c17	8	0	0	8	0	0	6	0	0
c880	392	0	0	409	0	0	242	0	0
c1355	1648	0	0	1656	0	0	664	4	0
c1908	1562	0	4	1562	0	4	1355	7	0
c2670	446	157	976	473	215	1110	191	204	487
c3540	475	15	64	1187	475	625	404	109	22
c5315	627	10	0	641	10	0	434	4	0
c7552	223	5	0	293	14	0	206	0	0

Table 4. The number of paths after justification.

6. Conclusions

We have presented the first critical path finding tool based on the exact criterion. This tool is delay and path correct, which means that it will never claim a true path to be false. Furthermore, it is exact when used in combination with the justification step, which means that it will not claim a false path to be true. In the latter case it gives the same results as by simulating a circuit for all possible input vectors.

The results obtained with our approach demonstrate that the approaches proposed in literature are not very accurate. The number of false paths claimed to be true by these approaches is very large. Finally, we have also shown that the justification step is the bottleneck for obtaining exact results for complex circuits. This should be now the topic for further research.

7. References

[1] R.B. Hitchcock et. al., "Timing analysis of computer hardware", IBM J.Res. Develop., vol 26, no. 1, pp. 100-105, Jan. 1982.

[2] J. Benkoski et. al., "Timing verification using statically sensitizable paths", IEEE Trans. on CAD, vol. 9, no. 10, Oct. 1990, pp. 1073-1084.

[3] D. Brand and V. Iyengar, "Timing analysis using functional analysis", IBM Th. J. Watson Res. Ctr., Tech. Rep., 1986.

[4] D. Du et. al., "On the general false path problem in timing analysis", Proc. DAC 1989, pp. 555-560.

[5] S. Perremans et. al., "Static timing analysis of dynamically sensitizable paths", Proc. DAC 19, pp. 568-573.

[6] H.C. Chen and D.H.C. Du, "Path sensitization in critical path problem", IEEE Trans. on CAD, vol. 12, no. 2, Feb. 1993, pp. 196-207.

[7] P. McGeer and R. Brayton, "Efficient algorithms for computing the longest viable path in a combinational network", Proc. DAC 1989, pp. 561-567.

[8] H.C. Chen and D.H.C. Du, "Path sensitization in critical path problem", ICCAD 1991, pp. 208-211.

[9] J.P. Roth, "Diagnosis of automata failures: A calculus and a new method", IBM J. Res. Develop., pp. 278-281, Oct. 1966.