Approximate Timing Analysis of Combinational Circuits under the XBD0 Model[1]

Yuji Kukimoto† Wilsin Gosti† Alexander Saldanha‡ Robert K. Brayton†

Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720†
Cadence Berkeley Laboratories, Berkeley, CA 94704‡

{kukimoto,wilsin,brayton}@eecs.berkeley.edu saldanha@cadence.com

Abstract

This paper is concerned with approximate delay computation algorithms for combinational circuits. As a result of intensive research in the early 90's [3, 8] efficient tools exist which can analyze circuits of thousands of gates in a few minutes or even in seconds for many cases. However, the computation time of these tools is not so predictable since the internal engine of the analysis is either a SAT solver [8] or a modified ATPG algorithm [3], both of which are just heuristic algorithms for an NP-complete problem. Although they are highly tuned for CAD applications, there exists a class of problem instances which exhibits the worst-case exponential CPU time behavior. In the context of timing analysis, circuits with a high amount of reconvergence, e.g. C6288 of the ISCAS benchmark suite, are known to be difficult to analyze under sophisticated delay models even with state-of-the-art techniques. For example [8] could not complete the analysis of C6288 under the mapped delay model. To make timing analysis of such corner case circuits feasible we propose an approximate computation scheme to the timing analysis problem as an extension to the exact analysis method proposed in [8]. Sensitization conditions are conservatively approximated in a selective fashion so that the size of SAT problems solved during analysis is controlled. Experimental results show that the approximation technique is effective in reducing the total analysis time without losing accuracy for the case where the exact approach takes much time or cannot complete.

1 Introduction

During late 80's and early 90's significant progress [2, 8] was made in the theory of exact gate-level timing analysis. In this, false paths are correctly identified so that exact delays can be computed. As the theory progressed, the efficiency and size limitation of actual implementations of timing analysis tools were dramatically improved [3, 8]. Although state-of-the-art implementations can handle circuits composed of thousands of gates under mapped delay models, it is evident that the current size limitation is far from satisfactory for analyzing industrial-strength circuits. Furthermore, even if they can handle large circuits, the computation time is often prohibitively large especially when delay models are elaborate.

To alleviate this problem several researchers have proposed approximate timing analysis algorithms. The goal is to compute a conservative yet accurate enough approximation of true delays in less computation time to make analysis of large circuits tractable.

Huang *et al.* [4, 6] proposed, as part of optimization techniques used in exact analysis, a simple approximation heuristic, in which a complex timed Boolean calculus expression at an internal node is simplified to a new independent variable arriving at the latest time referred to in the original expression. This simplification is applied only when the number of terms in the Boolean calculus expression exceeds a certain limit, to control the computational complexity. Accuracy loss comes from the fact that the original functional relationship is completely lost by the substitution. They also investigated a more powerful approximation technique in [5], in which each timed Boolean calculus formula is under- and over-approximated by sum of literals and products of literals respectively so that each sensitizability check, which is a satisfiability problem in the exact analysis, can be performed conservatively in polynomial time. Since this approximation is fairly aggressive to guarantee the polynomial time complexity, estimated delays do not seem accurate enough to be useful. Unfortunately their results, shown in [5], are not clear about the accuracy of approximate delays. They merely showed ratios of internal nodes whose delays match the exact delays at the nodes. No result was shown on the accuracy of circuit delays.

More recently Yalcin *et al.* [11] proposed an approximation technique, which utilizes user's knowledge about primary inputs. They categorize each primary input either as data or control and label all the internal nodes either data or control using a certain rule. The sensitization condition at each node is then simplified conservatively so that it becomes independent of the data variables. The intuition behind this is that the delay of a circuit is most likely determined by control signals while data signals have only minor effects in the final delay. [11] shows experimentally that a dramatic speed-up is possible without losing much accuracy for unit-delay timing analysis based on static sensitization. Unfortunately this sensitization criterion is known to underapproximate true delays, i.e. it is not a safe criterion, which defeats the whole purpose of timing analysis. More recently they confirmed that a similar speed-up and accuracy can be achieved for a correct sensitization criterion (the floating mode) under the unit-delay model [9]. Although an application of the same technique to more sophisticated delay models is theoretically possible, it is not clear whether their algorithm can handle large circuits under those delay models. Moreover, their CPU times for exact analysis are much worse than state-of-the-art implementations available, which cancels some of the speed-up since their speed-up is reported relative to this slower algorithm[1].

In this paper we apply their idea of using data/control separation to a state-of-the-art timing analysis technique [8] to design an approximate algorithm. The sensitization criterion here is the XBD0 model [8], which is one of the well-accepted delay models shown to be correct and accurate. In addition a novel technique to control the complexity of the analysis is proposed. The combination of these two ideas leads to a new approximation scheme, which for

[1]This work was supported by SRC-97-DC-324.

[1]One of the reasons why their exact algorithm is slower is that they try to represent in BDD all the input minterms that activate the longest sensitizable delay while most of the state-of-the-art techniques determine the delay without representing those input minterms explicitly.

0-89791-993-9/97 $10.00 © 1997 IEEE

some extreme cases shows a speed-up of 70x, while maintaining accuracy within the noise range.

This paper is organized as follows. Section 2 summarizes false path analysis, which forms a basis of this work. We specially focus on the technique proposed in [8]. Section 3 proposes two approximation schemes and discusses how they can be selectively applied to trade off accuracy and speed-up. Experimental results are given in Section 4. Section 5 concludes the paper.

2 Preliminaries

In this section, we review sensitization theory for the false path problem. Specifically, the theory developed in [8] is detailed below since the analysis following this section is heavily based on this particular theory.

2.1 Functional Delay Analysis

Functional delay analysis, or *false path analysis*, seeks to determine when all the primary output signals of a Boolean network become stable at their final values given maximum delays of each gate and arrival times at the primary inputs. Since some paths may never be sensitized, the stable time computed by functional delay analysis can be earlier than the time computed by topological delay analysis, thereby capturing the timing characteristic of the network more accurately. Those paths along which signals never propagate are called *false paths*.

The extended bounded delay-0 model [8], the XBD0 model, is the delay model most commonly used in false path analysis. It is the underlying model for the floating mode analysis [1] and viability analysis [7]. Under the XBD0 model, each gate in a network has a maximum positive delay and a minimum delay which is zero. Sensitization analysis is done under the assumption that each gate can take any delay between its maximum value and zero.

The core idea of [8] is to characterize recursively the set of all input vectors that make the signal value of a primary output stable to a constant by a given required time. Once these sets are identified both for constants 0 and 1, one can compare these against the on-set and the off-set of the primary output respectively to see if the output is indeed stable for all input vectors by the required time. The overall scenario of computing true delay is to start by setting the required time to the longest topological delay minus $\delta > 0$ and gradually decrease it until some input vector cannot make the output stable by the required time. The next to the last required time gives an approximation to the true arrival time at the output. This process of guessing the next required time can be sped up and refined by making use of a binary search.

Let us illustrate how we can compute these sets. Let n and d_n be a node (gate) in a Boolean network $\mathcal{N}$ and the maximum delay of the node n respectively[2]. Let $\chi^t_{n,v}$ be the characteristic function of the set of input minterms under which the output of the node n becomes stable to a constant $v \in \{0,1\}$ by time t. Let f_n be the local functionality of the node n in terms of immediate fanins $m_1, \ldots, m_k$ of n. For ease of explanation, let $f_n = m_1 m_2$, i.e., n is a two-input AND gate. It is clear from the functionality of the AND gate that to set n to a constant 1 by time t, both of the fanins of n, m_1 and m_2, are required to be stable at 1 by time $t - d_n$. This is equivalent to

$$\chi^t_{n,1} = \chi^{t-d_n}_{m_1,1} \cdot \chi^{t-d_n}_{m_2,1}.$$

Note that the two χ functions for the fanins are AND'ed to take the intersection of the two sets. Similarly, to set n to a constant 0 by time t, at least one of the fanins must be stabilized to 0 by time $t - d_n$.

$$\chi^t_{n,0} = \chi^{t-d_n}_{m_1,0} + \chi^{t-d_n}_{m_2,0}$$

Here the two χ functions are OR'ed to take the union of the two conditions. It is easy to see that the above computations can be generalized to the case where the local functionality of n is given as an arbitrary function in terms of its fanins as follows.

$$\chi^t_{n,v} = \sum_{p \in P^v_n} [\prod_{m_i \in p} \chi^{t-d_n}_{m_i,1} \cdot \prod_{\overline{m_i} \in p} \chi^{t-d_n}_{m_i,0}]$$

where P^1_n and P^0_n are the sets of all primes of f_n and $\overline{f_n}$ respectively. One can easily verify that the recursive formulations for the AND gate shown above are captured in this general formulation by noticing $P^1_n = \{m_1 m_2\}, P^0_n = \{\overline{m_1}, \overline{m_2}\}$ for $f_n = m_1 m_2$. The terminal cases are given when the node n is a primary input x.

$$\begin{aligned} \chi^t_{x,1} &= x && \text{if } t \geq arr(x) \\ &= 0 && \text{otherwise} \\ \chi^t_{x,0} &= \overline{x} && \text{if } t \geq arr(x) \\ &= 0 && \text{otherwise} \end{aligned}$$

where $arr(x)$ denotes the arrival time of x. The above formulas simply say that a primary input is stable only after its given arrival time. The key observation of this formulation is that characteristic functions can be computed *recursively*.

Once characteristic functions for constants 0 and 1 are computed at a primary output, two comparisons are made: one for the characteristic function for 1 against the on-set of the output, and the other for the characteristic function for 0 against the off-set of the output. Each comparison is done by creating a Boolean network which computes the difference between two functions and using a SAT solver to check whether the output of the network is satisfiable. The Boolean network is called a χ-network.

2.2 Optimal Construction of χ-Networks

To argue the approximation algorithms presented in this paper, further details on the construction of χ-networks need to be understood. We have mentioned that a χ-network is constructed recursively from a primary output. In [8] further optimization to reduce the size of χ-networks is discussed.

Given a required time at a primary output, assume that a backward required-time propagation of $\mathcal{N}$ is done to primary inputs so that the list of all required times at each internal node is computed. The propagation is done so that all the potential required times are computed at each node instead of the earliest required time. If the χ-network is constructed naively, for each internal node in $\mathcal{N}$, a distinct node is to be created for each required time in the list. This, however, is not necessary since it is possible that different required times exhibit the same stability behavior, in which case having a single node in the χ-network for the required times is enough. To detect such a case a forward arrival-time propagation from primary inputs to primary outputs is performed to compute the list of all potential arrival times at each node. Note that each potential arrival time corresponds to the topological delay of a path from a primary input to the internal node. Therefore the stability of the node can only change at those times. In other words between two adjacent potential arrival times, one cannot see any change in the stability.

Consider an internal node $n \in \mathcal{N}$. Let $R = (r_1, \ldots, r_p)$ and $A = (a_1, \ldots, a_q)$ denote the sorted list of required times and that of arrival times respectively at node n. Consider χ function $\chi^{r_i}_{n,v} (v = 0, 1)$. Let $a_j \in A$ be the maximum arrival time such that $a_j \leq r_i$. Since there is no event happening between time a_j and r_i, $\chi^{r_i}_{n,v} = \chi^{a_j}_{n,v}$. Matchings from required times to arrival times are

[2] It is possible to differentiate rise delays from fall delays. In this paper, however, we do not distinguish between them to simplify exposition.

performed in this fashion to identify the subset of A that is required to compute the final χ function. This optimization avoids creating redundant nodes in the χ network thereby reducing the size of the χ network without losing any accuracy in analysis. Only those arrival times which have a match with required times yield nodes in the χ network.

Another type of optimization suggested in [8] is to generate the list of arrival times more carefully. For each potential arrival time, equivalence between the corresponding χ function and the on-set or the off-set (whichever suitable) is checked by a satisfiability call and a new node is created in χ network only if the two functions are different. Otherwise, the original function or its complement is used as it is. Although this requires additional CPU time spent on satisfiability calls, it is experimentally confirmed that the size reduction of the final χ network is so significant that the the total run-time decreases in most cases.

3 Approximation Algorithms

3.1 Limitation of the Exact Algorithm

Although the exact algorithm proposed in [8] can handle many circuits of thousands of gates, it still has a size limitation. If a large network is given and timing analysis is requested under a detailed delay model like the technology mapped delay model, it is likely that the algorithm runs practically forever[3]. Even if timing analysis is tractable, the computation time can be too large to be practical.

As seen in the previous section, the exact timing analysis consists of repeated SAT solver calls. More precisely, for each time tested at a primary output, a χ-network is constructed such that the network computes the difference between the on-set (off-set) of the primary output and the set of input vectors which make the primary output stable to value 1 (0) by the given time. If the output never becomes 1 for any input assignment, i.e. it is not satisfiable, we know that the output becomes stable completely by the time tested. To test whether this condition holds, a SAT formula which is satisfiable only if the output is satisfiable is created directly from the χ network, and a SAT solver is called on it. The size of the SAT formula is roughly proportional to the size of the χ network. The main difficulty in the analysis of large networks is that due to a potentially large size of the χ networks, the size of SAT formulas generated can be too large for a SAT solver to solve even after the optimization discussed in the previous section has been applied[4]. In the following we discuss how to control the size of χ networks without losing much accuracy.

3.2 Reducing the Size of χ Networks for Effective Approximation

The main reason why χ networks become large in the exact approach is that χ functions at many distinct arrival times must be computed for internal nodes. This size increase occurs when there are many distinct path delays to internal nodes due to the reconvergence of the circuit. Therefore our goal is to control the number of distinct arrival times considered at each internal node. More specifically we only create a small number of χ functions at each internal node. This strategy avoids the creation of huge χ networks thereby controlling the size of SAT formulas generated.

Although this idea certainly helps reduce the size of χ networks, it must be done carefully so that the correctness of the analysis is guaranteed. We must never underapproximate true delays since otherwise the timing analysis could miss timing violations when used in the context of timing verification. Overapproximation is acceptable as long as reasonable accuracy is maintained. We guarantee this property by selectively underapproximating stability of signals. This underapproximation in turn overapproximates instability of signals thereby guaranteeing that estimated delays are never underapproximated.

The key idea on approximation is to modify the mapping from required times to arrival times discussed in Section 2.2 so that only a small set of arrival times forms the image of the mapping. Given the sorted set of required times $R = (r_1, \ldots, r_p)$ and the sorted set of arrival times $A = (a_1, \ldots, a_q)$ at an internal node n, the mapping $f : R \mapsto A$ used in the exact analysis is defined as

$$f(r) = \begin{cases} \max a_i \in A \text{ such that } a_i \leq r & \text{if } r \geq a_1 \\ -\infty & \text{otherwise} \end{cases}$$

Since the stability of the signal at the node increases monotonically as time elapses by the definition of χ functions, it is safe to change the mapping so that it maps a required time to a time earlier than the time defined in the above. This corresponds to underapproximation of the signal stability. Thus, by modifying the mapping under this constraint so that only a small set of arrival times is required, one can control the number of nodes to be introduced in the χ network without violating the correctness of the analysis. Depending on how the original mapping in the exact analysis is changed several conservative approximation schemes can be devised. Two such approximation schemes are described next.

3.2.1 Topological Approximation

The most aggressive approximation, which we call *topological approximation*, is to map required times either to the topological arrival time (a_q[5]) or to $-\infty$. More formally, the mapping f^T is defined as follows.

$$f^T(r) = \begin{cases} a_q & \text{if } r \geq a_q \\ -\infty & \text{otherwise} \end{cases}$$

It is easy to see that f^T is a conservative approximation of f. Since $\chi_{n,1}^{a_q} = n$ and $\chi_{n,0}^{a_q} = \overline{n}$, there is no need to create a new node for the χ function in the χ network[6]. Instead the node function or its complement of the original network can be used for the χ function. For the other arrival time $-\infty$, $\chi_{n,v}^{-\infty} = 0$ for $\forall v \in \{0, 1\}$. Therefore it is sufficient to have a constant zero node in the χ network and use it for all the cases where the zero function is needed. Since neither of the arrival times needs any additional node in the χ network, this approximation never increases the size of the χ network. If this reduction is applied at all nodes, the analysis simply becomes pure topological analysis. Therefore, this approximation makes sense only if it is selectively invoked on some subset of nodes. A selection strategy is described later.

3.2.2 Semi-Topological Approximation

The second approximation scheme, called *semi-topological approximation*, is slightly milder than the first in terms of the power of simplifying χ networks. In this, required times are mapped to two arrival times again, but the times chosen are different. The times to be picked are 1) the arrival time, say a_e, matched with r_1 in the exact mapping f and 2) the topological arrival time a_q, which is the same as in the first approximation. The first approximation and this one are different only if $a_e \neq -\infty$, in which case the second one

[3] The algorithm is CPU intensive rather than memory intensive since the core part of the algorithm is SAT.

[4] Theoretically it is not necessarily true that a smaller SAT formula is easier to solve. However we have observed that the size of SAT formulas is well correlated with the time the solver takes.

[5] To be precise, a_q can be earlier than the topological arrival time if an intermediate satisfiability call has already verified that by time a_q the signal is stabilized completely.

[6] Notice that the χ network always includes the original circuit.

gives a more accurate approximation. To be precise, the definition of the new mapping function f^S is as follows.

$$f^S(r) = \begin{cases} a_e & \text{if } r < a_q \\ a_q & \text{otherwise} \end{cases}$$

If $a_e \neq -\infty$, the χ function for time a_e is now computed explicitly, and the corresponding node is added to the χ network. Similar extensions which give tighter approximations are possible by allowing more arrival times to remain after the mapping. A set of various approximations gives a tradeoff between compactness of χ networks and accuracy of analysis.

3.3 Control/Data Dichotomy in Approximation Strategies

In [11] Yalcin *et al.* proposed to use designer's knowledge on control-data separation of primary inputs for effective approximate timing analysis. They applied this idea to speed up their timing analysis technique using conditional delays [10] by simplifying signal propagation conditions of data variables. We adapt their idea, of using this knowledge, to the XBD0 analysis to develop a selection strategy of various approximation schemes.

3.3.1 Labeling Data/Control Types

Given data/control types of all primary inputs, each internal node is labeled data or control based on the following procedure. All the nodes in the network are visited from primary inputs to primary outputs in a topological order. At each node the types of its fanins are examined. If all of them are data, the node is labeled data; otherwise it is labeled control. Hence nodes labeled data are *pure* data variables with no dependency on control variables, while those labeled control are all the other variables with some dependency on control variables. This labeling policy is different from the one used in [11], where a node is labeled data if at least one of its fanins is labeled data. In their labeling, nodes labeled data are variables with some dependency on data whereas nodes labeled control are pure control variables. The difference between the two labelings is whether pure data variables or pure control variables are distinguished. Our labeling will lead to tighter approximations.

3.3.2 Applying Different Approximations based on Data/Control Types

Once all the nodes are labeled, different approximation schemes are applied at nodes based on their types. The strategy is as follows.

If a node is a control variable, the semi-topological approximation f^S is applied while if a node is a data variable, the topological approximation f^T is applied. The intuition is to use a tighter approximation for control variables to preserve accuracy while performing maximum simplification for data variables assuming they have less impact on delays than control variables.

3.3.3 Extracting Control Circuitry for Further Approximation

If the approximation so far is not powerful enough to make analysis tractable, further approximation is possible by extracting only the control-intensive portion of the circuit and performing timing analysis on the subcircuit. The extraction of the control portion is done by stripping off all pure data nodes from the original network under analysis. Note that any circuit can be decomposed into a cascade circuit where the nodes in the driving circuit are labeled as data and those in the driven circuit control by the definition of data variables. Therefore, the primary inputs of the subcircuit are the boundary variables which separate the subcircuit from the pure data portion. We assume conservatively that delays of the pure data portion of the circuit are the same as topological delays, which gives arrival times at the primary inputs of the extracted circuit. Analysis is then performed on this subcircuit as if it were the circuit given. Notice that this has a similar flavor to the approximation proposed in [4].

The difference between this approximation and the previous method is that the subcircuit has a new set of primary inputs, which are assumed independent. However, it is possible that in the original circuit only a certain subset of signal combinations appears at the boundary variables. Since this approximation assumes that all signal combinations can show up, the analysis becomes pessimistic[7]. For example, if a signal combination which does not appear on the cut makes a long path sensitizable, it can make delay estimation unnecessarily pessimistic. Although this method is more conservative than the one without subcircuit extraction, it reduces the size of a circuit to be analyzed much more significantly than the other one.

4 Experimental Results

We implemented the new approximation scheme on top of the implementation of [8] under SIS environment. To evaluate the effectiveness of the approximation, we focused on timing analysis of mapped ISCAS combinational circuits, which is generally much more time-consuming than analysis based on simpler delay models. In Table 1[8] the results on three circuits whose exact analysis takes more than 20 seconds on a DEC Alpha Server 7000/610 are shown[9]. Each circuit is technology-mapped first with the option specified in the second column using the `lib2.genlib` library. The delay of the circuit is then analyzed using three techniques. The first one (exact) is the exact method presented in [8]. The remaining two are approximate methods; the second, called approx(1), is the technique in Section 3.3.2 and the third, called approx(2), is the one in Section 3.3.3 which involves subcircuit extraction. Control/Data specification for the primary inputs of these circuits are the same as those in [11][10]. For each of the three analyses, estimated delay and CPU time are shown in the last two columns. One can observe that accuracy is preserved in the three examples in both of the approximation methods while CPU time is reduced significantly.

Table 2 summarizes a similar experiment for C6288, an integer multiplier, which is known to be difficult for exact timing analysis due to a huge amount of reconvergence. Since all the primary inputs are data variables, the approximate techniques proposed are degenerated into topological analysis. To avoid this inaccuracy all the primary inputs were set to control. Note that this sets all intermediate nodes to control. We then applied the first approximate method under this labeling. Although the approximation is not so powerful as the original algorithms, this at least enables us to reduce the size of χ networks without giving up accuracy completely. Since there is no data variable in the network, only approx(1) was tried. Significant time saving was achieved with only a slight overapproximation in terms of analysis quality. The exact analysis is not only more CPU-time intensive but also much more memory-intensive than the approximate analysis. In fact we could not complete any of the three exact analyses within 150MB of memory. They ran out of memory in a couple of minutes. These exact analyses were possible after

[7] If the set of all possible signal combinations at the boundary variables can be represented compactly, one can safely avoid this pessimism by multiplying the additional constraint to the SAT formula generated.

[8] Timing analysis was done in the linear search mode [8] where the decrement time step is 0.1 and the error tolerance is 0.01.

[9] If exact analysis is already efficient, approximation cannot make significant improvement in CPU time; in fact the overall performance can be degraded due to additional tasks involved in approximation.

[10] More precisely, C1908(1) and C3540(1) in [11] were used.

circuit	tech.map	#gates	topological delay	type of approx.	estimated delay	CPU time
C1908	`-m 1`	536	39.25	exact	34.77	29.1
				approx(1)	34.77	8.9
				approx(2)	34.77	5.4
C1908	`-m 0`	602	40.76	exact	35.76	41.2
				approx(1)	35.76	12.0
				approx(2)	35.76	5.2
C3540	`-n 1 -AFG`	1113	35.88	exact	35.66	727.0
				approx(1)	35.66	559.5
				approx(2)	35.66	502.9

Table 1: Exact analysis vs. Approximate analysis (CPU time in seconds on DEC AlphaServer 7000/610)

circuit	tech.map	#gates	topological delay	type of approx.	estimated delay	CPU times
C6288	`-m 1`	2429	127.23	exact	123.87	7850.2
				approx(1)	123.94	169.2
	`-m 0`	2371	123.51	exact	119.16	18956.2
				approx(1)	119.21	257.1
	`-n 1 -AFG`	2911	114.62	exact	112.92	15610.5
				approx(1)	112.86	1690.9

Table 2: Exact analysis vs. Approximate analysis on C6288 (CPU time in seconds on DEC AlphaServer 7000/610)

the memory limit was expanded to 1GB. The last example needs an additional explanation. In this example the estimated delay by the approximate algorithm is smaller than that by the exact algorithm although in Section 3 we claimed that the approximation algorithm never underapproximates exact delay. The reason for this is that the SAT solver is not perfect. Given a very hard SAT problem, the solver may not be able to determine the result under a given resource, in which case the solver simply returns Unknown. This is conservatively interpreted as being satisfiable in the timing analysis. In this particular example the SAT solver returned Unknown during the exact timing analysis, which resulted in an overapproximation of the estimated delay, while in the approximate analysis the SAT solver never aborted because of the simplification of χ networks and gave a better overapproximation. This example shows that the approximate analysis gives not only computational efficiency but also better accuracy in some cases.

To compare the exact and the approximate methods further, we examined the total CPU time of the exact analysis to see how it can be broken down. For the first example of C6288 the exact analysis took 714.7 seconds to conclude that any path of length 123.93 is false, which is about four times longer for the approximate analysis to conclude that the delay of the circuit is 123.94. The situation is much worse in the second example, where the exact analysis took 18390.8 seconds to conclude that any path of length 119.21 is false while the approximate method took only about 1.4% of this time to finish off the entire analysis.

5 Conclusions

We have proposed new approximation algorithms as an extension to the XBD0 timing analysis [8]. The core idea of the algorithms is to control the size of sensitization networks to prevent the size of SAT formulas to be solved from getting large. The use of knowledge on data/control separation of primary inputs originally proposed in [11] was adapted to choose an appropriate approximation at each node. We showed experimentally that the technique helps simplify the analysis while maintaining accuracy well within the accuracy of the delay model.

Acknowledgments

Hakan Yalcin kindly offered detailed data on ISCAS benchmark circuits.

References

[1] H.-C. Chen and D. H.-C. Du. Path sensitization in critical path problem. *IEEE Transactions on Computer-Aided Design*, 12(2):196–207, February 1993.

[2] S. Devadas, K. Keutzer, and S. Malik. Computation of floating mode delay in combinational circuits: Theory and algorithms. *IEEE Transactions on Computer-Aided Design*, 12(12):1913–1923, December 1993.

[3] S. Devadas, K. Keutzer, S. Malik, and A. Wang. Computation of floating mode delay in combinational circuits: Practice and implementation. *IEEE Transactions on Computer-Aided Design*, 12(12):1924–1936, December 1993.

[4] S.-T. Huang, T.-M. Parng, and J.-M. Shyu. A new approach to solving false path problem in timing analysis. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 216–219, November 1991.

[5] S.-T. Huang, T.-M. Parng, and J.-M. Shyu. A polynomial-time heuristic approach to approximate a solution to the false path problem. In *Proceedings of 30th ACM/IEEE Design Automation Conference*, pages 118–122, June 1993.

[6] S.-T. Huang, T.-M. Parng, and J.-M. Shyu. Timed boolean calculus and its applications in timing analysis. *IEEE Transactions on Computer-Aided Design*, 13(3):318–337, March 1994.

[7] P. C. McGeer and R. K. Brayton. *Integrating Functional and Temporal Domains in Logic Design*. Kluwer Academic Publishers, 1991.

[8] P. C. McGeer, A. Saldanha, R. K. Brayton, and A. Sangiovanni-Vincentelli. Delay models and exact timing

analysis. In T. Sasao, editor, *Logic Synthesis and Optimization*, pages 167–189. Kluwer Academic Publishers, 1993.

[9] H. Yalcin. Private communication, March 1997.

[10] H. Yalcin and J. P. Hayes. Hierarchical timing analysis using conditional delays. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 371–377, November 1995.

[11] H. Yalcin, J. P. Hayes, and K. A. Sakallah. An approximate timing analysis method for datapath circuits. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 114–118, November 1996.