Abstract
This paper addresses how to compute required times at intermediate nodes in a combinational network given required times at primary outputs. The simplest approach is to compute them based on topological delay analysis without any consideration of false paths. In this paper, however, we take into account false paths between the intermediate nodes and the primary outputs explicitly to characterize the timing constraints at the nodes more accurately. We show that this approach leads to a technique for computing a more refined and relaxed timing constraint than that obtained by topological analysis. We generalize the notion of required times from a single constant to a relation where a signal is required at different times depending on the values of the other signals.

1 Introduction
An optimization strategy common in most logic optimization algorithms is repeated applications of the following two steps: 1) characterize a set of permissible implementations at a subcircuit (or a node) and 2) select a behavior of better quality from the set in consideration of area, speed, testability, power dissipation or a combination of the four. This paper is concerned with the first step, especially how to characterize permissible implementations which preserve temporal behaviors of the original circuit. The temporal behaviors of interest in this paper can be captured by required times at primary outputs and arrival times at primary inputs. Suppose a combinational network and its arrival/required times at primary inputs/outputs are given. Assume that a subnetwork of this circuit is to be optimized. When resynthesizing this subnetwork, arrival times at subcircuit inputs and required times at subcircuit outputs must be specified to a logic synthesis tool along with the functional specification of the subcircuit so that replacing the existing subcircuit with an optimized circuit automatically preserves the original functional and temporal specifications. This scheme enables us to resynthesize subcomponents locally without violating the functional and temporal requirements of the whole system.

A naive solution for this problem is to compute arrival times and required times using topological delays. This approach is commonly used in most timing optimization algorithms in the literature. Although this conservative approach gives a fast and safe approximation to the true timing constraint, the resulting timing requirement may be tighter than necessary since false paths in the surrounding network are completely ignored. Therefore, the timing constraint computed in this manner may prevent a synthesis tool from exploring the entire temporal flexibility, thereby leading to an unsatisfactory circuit. The goal of this work is to solve this problem more rigorously by taking false path effects into account so that a more accurate and thus more flexible timing constraint is computed for the subnetwork.

There have been many theoretical and practical results published in the literature for functional flexibility. Functional flexibility of a subnetwork can be characterized as the set of functionalities which can replace the current implementation without changing the I/O functionality of the entire system. For the case where the subcircuit is a combinational circuit, the full flexibility can be captured by a Boolean relation of the inputs and the outputs of the subnetwork [2]. Recently, Watanabe and Brayton [11] resolved the case where the subcircuit is a sequential circuit by showing that the complete functional flexibility of the subcircuit is expressible by a single non-deterministic FSM, called the E-machine.

Although a significant effort has been made towards computing functional flexibility, little has been done for timing flexibility. The main objective of this paper is to leverage the theory of temporal flexibility up to the same level as functional flexibility.

One related work published recently is the notion of timing-safe replaceability proposed by Aziz et al. [1], which is an extension of safe replaceability addressed by Singhal et al. [10] in the context of sequential synthesis. The core idea of [1] is to characterize the set of all temporal behaviors exhibited by a combinational circuit using the linear logic of \mathcal{R}. If the set of all behaviors exhibited by another combinational circuit is a subset of the original set, no surrounding environment can detect a difference between the two circuits. Thus, the replacement of the original circuit with the new circuit is timing-safe; it works in all environments. This notion of replaceability, however, is often too stringent in realistic design scenarios since timing optimization of a component is typically performed given a particular surrounding environment.

This paper is organized as follows. Section 2 summarizes false path analysis, which forms a basis of this work. Section 3 gives the overview of the problem of computing temporal flexibility and illustrates how this problem arises in practical setups. Section 4 discusses a novel technique to propagate required times backwards by taking into account false path effects. Due to space limitation we will not discuss a technique to propagate arrival times forward, which can be found in [5]. Experimental results are given in Section 5. Section 6 concludes the paper.

2 Preliminaries
In this section, we review sensitization theory for the false path problem [3, 4, 8]. Specifically, the theory developed in [8] is
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 characteristics of the network more accurately. Those paths along which signals never propagate are called false paths.

The extended bounded delay-0 model [8], XBD0 model, is the delay model most commonly used in false path analysis. It is the underlying model for the floating mode analysis [3] 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 the input vectors by the required time. The overall scenario of computing true delay is to start by setting the primary inputs and the off-set of the primary output respectively to see if the output becomes stable to a constant by a given required time. Once these sets are identified, the overall scenario of computing true delay is to start by setting the primary inputs and the off-set of the primary output respectively to see if the output becomes stable to a constant by a given required time. Once these sets are identified, the overall scenario of computing true delay is to start by setting the primary inputs and the off-set of the primary output respectively to see if the output becomes stable to a constant by a given required time. Once these sets are identified.

Let us illustrate how we can compute these sets. Let \( n \) and \( d_n \) be a node (gate) in a Boolean network \( N \) and the maximum delay of the node \( n \) respectively. Let \( \chi_{n,v} \) 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 \( f_n = m_1 \land m_2 \), 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

\[
\chi_{n,1}^t = \chi_{m_1,1}^{t-d_n} \land \chi_{m_2,1}^{t-d_n}
\]

Note that the two \( \chi \) 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 \( t - d_n \).

\[
\chi_{n,0}^t = \chi_{m_1,0}^{t-d_n} \lor \chi_{m_2,0}^{t-d_n}
\]

Here the two \( \chi \) 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.

\[
\chi_{n,v}^t = \bigcup_{p \in P_n^1} \chi_{m_1,v}^{t-d_p} \land \bigcup_{m \in P_n^0} \chi_{m,v}^{t-d_m}
\]

where \( P_n^1 \) and \( P_n^0 \) 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 \( P_n^1 = \{m_1, m_2\} \) and \( P_n^0 = \{m_1 \land m_2\} \). The terminal cases are given when the node \( n \) is a primary input \( x \).

\[\text{Figure 1: Boolean Network}\]

where \( \text{arr}(x) \) 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 onset of the output, and the other for the characteristic function for 0 against the offset 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. Experimental results in [8] showed that this approach can analyze large networks in reasonable computation time.

3 Overview

In this paper we restrict our attention to combinational circuits. Sequential circuits using edge-triggered latches, however, can be easily handled with the same framework by assuming all the latch inputs and outputs as primary inputs and outputs respectively, where the required times and arrival times of those are determined by the clock edge minus the setup time and the clock edge itself respectively.

Given a Boolean network \( N \) and a subnetwork \( N' \) of \( N \), our interest is to characterize the timing flexibility of \( N' \) so that synthesis of the subcircuit can be performed locally without violating the timing constraint of the entire network \( N \). Note that our assumption is that \( N \setminus N' \) remains unchanged and only \( N' \) is to be synthesized. Let us introduce some notation for ease of explanation. Let \( X = (x_1, \ldots, x_n) \) and \( Z = (z_1, \ldots, z_m) \) be primary inputs and primary outputs of \( N \) respectively. Let \( U = (u_1, \ldots, u_p) \) and \( V = (v_1, \ldots, v_q) \) denote inputs and outputs of \( N' \) respectively. (See Figure 1.) We assume that arrival times at primary inputs \( X \) and required times at primary outputs \( Z \) are given. Our goal is to compute arrival times at subcircuit inputs \( U \) and required times at subcircuit outputs \( V \) by considering the effects of false paths in \( N \setminus N' \) explicitly. One can think of this as mapping the timing requirement of the entire circuit onto the subcircuit.

This problem has several practical applications. The first is performance-oriented resynthesis. Suppose a combinational circuit was synthesized from a specification. Although one can optimize the entire circuit further to speed up late outputs, another promising approach is to extract a subcircuit containing a part of critical

1 Although it is possible to differentiate rise delays from fall delays, we do not distinguish between them in this paper to simplify exposition.

2 To be precise, \( N' \) must meet the condition that there is no path leading from a subcircuit output to a subcircuit input.
paths and optimize it locally. This scheme is more likely to give a faster circuit because the circuit fed to synthesis is smaller. A similar approach is in fact taken in timing optimization techniques [9] published in the literature, but their delay computation is based on topological longest paths thereby failing to capture some of existing timing flexibility. Since our approach computes the timing flexibility of the subcircuit by considering false path effects from the surrounding circuit, larger flexibility, i.e. less stringent timing requirement, is obtained, which makes resynthesis easier. An interesting subproblem of this application is to compute the true slack of a gate output, where the slack is calculated by taking false path effects into account.

The second practical application is in hierarchical synthesis. Assume that a set of communicating sequential circuits does not meet a timing requirement, e.g., they do not satisfy a cycle time constraint. We now want to optimize component circuits one by one to speed up late signals. Since the boundaries of components are not necessarily latch inputs or outputs, one may have to map arrival/required times for latch inputs/outputs of the other components to the interface nodes of the component to be optimized. Figure 2 shows such a situation, where two sequential circuits are cascaded. Assume that a cycle time is given as a timing specification and we want to optimize only the left component with the right component unchanged. For simplicity, assume that there is a single latch in the right component. The input of this latch must become stable before the cycle time. This constraint can be translated to that of the left component by propagating the required time at the latch input backward through the combinational gates in the transitive fanin of the latch to the boundary of the two components. Note that this problem is equivalent to our problem where N is the combinational portion of the right component and N' is the set of all the boundary variables between the two components.

A similar scenario can arise in pure combinational synthesis. Consider a cascaded combinational circuit, where the driven circuit contains a fair amount of false paths. To resynthesize the driving circuit effectively for improved performance it is critical to characterize the required times of the signals feeding the driven circuit as accurately as possible. Required times computed by topological analysis may completely mislead resynthesis due to the unawareness of false paths in the driven circuit. In this setup, N corresponds to the entire circuit and N' the driving circuit.

Due to space limitation we will focus on the computation of required times in the rest of this paper. The full treatment of the problem including arrival time computation can be found in [5].

4 Computing Required Time under the Existence of False Paths

In this section we consider the following simpler problem: Given a Boolean network A, maximum delay of each gate, and required times at its primary outputs, compute required times at its primary inputs. This is a special case of the general problem where the subcircuit under analysis only contains all the primary inputs of the network. The problem can be solved efficiently if delays are defined as topological longest delays. The procedure first sorts all the nodes in a reverse topological order and initializes required times of all the nodes, except primary outputs, to infinity. It then propagates required times of the output of each node backward to the fanins of the node. If a signal has more than one fanout, only the earliest required time is recorded. The procedure runs in time linear to the size of the network. Note that required times are uniquely determined in this algorithm, which is not necessarily true once false path effects are taken into account as we see later in this section.

The approach taken in the following makes use of χ functions introduced in Section 2. For each primary output, χ functions for constraints 0 and 1 are computed for a given required time and they are compared against the onset and the offset of the output respectively to extract conditions on required times at primary inputs. The main difference between this problem and the functional delay analysis problem discussed in Section 2 is that arrival times at primary inputs are unknown variables in our problem while they are given in the other. In spite of this difference the original recursive formulation for computing χ functions almost works. A modification is required only in terminal cases. Since we do not know when a primary input signal arrives, leaf χ functions at primary inputs remain as unknown variables. Henceforth, we call leaf χ functions at primary inputs leaf χ variables. Let x be a primary input. Assume that after recursive construction of χ functions at primary outputs, leaf χ variables for x are needed at times t1 < t2 < ... < tnp for value 1 and at times t'1 < t'2 < ... < t'n'p for value 0. Remember that leaf χ variables are characteristic functions of sets of input vectors that are stable by required times. This implies that for any t < t0 the set of stable input vectors by time t0 must be contained in the set of stable input vectors by time t. Therefore, the following ordering conditions among leaf χ variables must be met.

\[
\emptyset \subseteq \chi_{x,1}^1 \subseteq \chi_{x,1}^2 \subseteq \ldots \subseteq \chi_{x,1}^{t_1} \subseteq x \\
\emptyset \subseteq \chi_{x,0}^{t'_1} \subseteq \chi_{x,0}^{t'_2} \subseteq \ldots \subseteq \chi_{x,0}^{t'_{n'p}} \subseteq \overline{x}
\]

The formulas above indicate that leaf χ variables are 1) monotone non-decreasing with respect to time and 2) bounded above by x and \(\overline{x}\) for value 1 and 0 respectively. The first constraint is imposed since, once an input vector becomes stable, it must continue to be stable. The second constraint is required so that leaf χ variables are compatible with the onset and the offset of the primary input x.

Let us go back to our problem. For simplicity, assume that a Boolean network N has a single primary output z, whose required time t is given. Generalization to multiple primary outputs is trivial. We are interested in computing required times at primary inputs of the network. Suppose that \(\chi_{x,1}^i\) and \(\chi_{x,0}^i\) are computed in terms of leaf χ variables at primary inputs, which we call χ. The goal is to assign Boolean functions of X to unknown χ variables so that \(\chi_{x,1}^i(x) = z(X)\) and \(\chi_{x,0}^i(x) = \overline{z(X)}\) hold under the ordering constraints among χ variables discussed above, where z(X) denotes the functionality of the primary output in terms of primary inputs X. The set of input vectors which make the output stable at value 1 or 0 by time t are constrained to be equal to the onset or the offset of the output function respectively.

4.1 Exact Approach

One can formulate this problem as solving a Boolean equation where unknown variables are leaf χ variables χ. The Boolean constraints to be satisfied are:

\[\chi_{x,1}^i(x) = z(X)\]

n It is possible to extend the theory to the case where \(z(X)\) is an incompletely specified function.
It is easy to transform the above set of Boolean equations to an equivalent single Boolean equation of form \( F(X, \chi X) = 1 \) by AND'ing all these constraints together. In this equation, \( \chi X \) are variables to be solved while \( X \) are Boolean constants. One can think of \( F(X, \chi X) \) as the characteristic function of a Boolean relation where \( X \) is the inputs and \( \chi X \) is the outputs. Any function in terms of \( X \) compatible with \( F \) satisfies the timing requirement at \( z \).

Notice that the notion of required times at primary inputs is significantly generalized here. For each primary input, its required time is not simply a single constant, but the input signal can arrive at different times depending on signal values of the other inputs.

Let us illustrate this in the circuit shown in Figure 3. For simplicity, assume that the maximum delay of the AND gate is 1 and the required time at the primary output \( z \) is 2. The required time computed by topological delay analysis is time 0 for both inputs. The procedure described above gives the following Boolean relation.

\[
\begin{array}{c|c|c}
X_{x2} & X_{x1} & \chi x_0 \\
00 & 000100 & 000000 \\
01 & 000100 & 000101 \\
10 & 000100 & 000001 \\
11 & 111000 & 000001 \\
\end{array}
\]

Any signal behavior at primary inputs that is compatible with this relation meets the required time at the primary output. For example, if we pick 000100, 000100, 000001, and 111000 for input minterms 00, 01, 10, 11 respectively, then leaf \( \chi \) variables will be: \( \chi x_1 = x_2 x_2; \chi x_2 = x_2 x_2; \chi x_2 = x_2 x_2; \chi x_0 = x_2 x_2; \chi x_0 = x_2 x_2 \). To focus on only the stability of signals, we define \( \chi x \) as follows.

\[ \chi x = x_{x1} + x_{x0} \]

This \( \chi \) function of a node \( n \) at time \( t \) is the characteristic function of the set of all input vectors that make the signal \( n \) stable either to 0 or to 1 by time \( t \). For the \( \chi \) functions above, \( \chi \) functions are computed as: \( \chi x_1 = x_2 x_2; \chi x_0 = x_2 x_2; \chi x_0 = x_2 x_2 \). The interpretation of this is that primary input \( x_1 \) must be stable by time 0 just for the case \( x_2 x_2 \) and if \( x_2 x_2 \), it can delay forever without violating the given functional and temporal requirements. Notice that in topological analysis it always has to arrive before time 0. Let us look into how signal \( x_2 \) should behave. It must be stabilized by time 0 for the case \( x_2 x_2 \). If \( x_2 x_2 \), \( x_2 x_2 \) has to become stable by time 1. For all the other cases, i.e., if \( x_1 = 0 \), however, the signal can be infinitely delayed. One can easily see that the relation contains a compatible function corresponding to the required time computed by topological analysis \( ^4 \), which gives the tightest required time condition.

\( ^4 \) Pick the last output pattern for each minterm since the later the required times are, the easier synthesis of prelogic becomes.

For each input minterm the relation gives a set of permissible vectors for leaf \( \chi X \) variables. Since a 1 in a vector means that the corresponding leaf \( \chi \) variable must be stable, having fewer 1’s requires less stability. Therefore, the latest required time is characterized by a subset relation of the original relation where each input minterm can be mapped only to vectors with the least number of 1’s \( ^5 \). For the working example, the subset relation is shown below on the left while its interpretation as required times is shown on the right.

\[
\begin{array}{c|c|c}
X_{x2} & X_{x1} & \chi x_0 \\
00 & 000100 & 000000 \\
01 & 000100 & 000101 \\
10 & 000100 & 000001 \\
11 & 111000 & 000001 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
x_{x1} & x_{x1} & x_{x0} \\
00 & 000100 & 000000 \\
01 & 000100 & 000101 \\
10 & 000100 & 000001 \\
11 & 111000 & 000001 \\
\end{array}
\]

Note that there may be more than one latest required time. In this particular example, either \( x_1 \) arriving by time 0 or \( x_2 \) arriving by time 1 is required for \( x_1 x_2 = 00 \). Those two conditions are not comparable and each gives a different limiting condition.

### 4.2 Approximate Approach

In this subsection, we will consider an approximate approach. In the exact approach, a primary input signal can arrive at different times depending on signal values of the other inputs. Here, we simply assume that a primary input signal arrives at a certain time no matter what signal values are present in the other inputs. Arrival times for values 0 and 1, however, are still distinguished \( ^6 \).

In the exact approach, we explicitly impose the ordering constraints among leaf \( \chi \) variables as Boolean constraints. Here, instead of keeping the constraints around, we introduce additional 0-1 Boolean variables \( \alpha_{x1}, \alpha_{x2}, \beta_{x1}, \beta_{x2} \) to encode the ordering constraints in leaf \( \chi \) variables \( ^7 \).

\[
\begin{align*}
\chi x_1 & = x_{x1} + \chi x_0 \\
\chi x_2 & = x_{x2} + \chi x_2 \\
\chi x_1 & = x_{x1} + \chi x_0 \\
\chi x_0 & = x_{x1} + \chi x_0 \\
\end{align*}
\]

Notice that all the ordering constraints are automatically satisfied by the use of the Boolean variables. The side effect of this encoding is that leaf \( \chi \) variables can now either take \( x \) or 0 for value 1, and either \( \chi \) or 0 for value 0 under a 0-1 assignment to the \( \alpha \) and \( \beta \) variables while they can take any function between 0 and \( x \) for value 1 and between 0 and \( \chi \) for value 0 in the exact approach. This, however, directly corresponds to our new constraint that each primary input arrives at a fixed time no matter how the other inputs behave. The remaining condition to be satisfied is that the two \( \chi \) functions for the primary output are equal to the onset and the offset of the output respectively.

\[
\chi x_1, (X, \alpha, \beta) = z (X), \quad \chi x_0, (X, \alpha, \beta) = \overline{z (X)}
\]

\( ^5 \) To be precise, extracting all the minterms with the least number of 1’s is not enough depending on how stability is defined. Consider a set [00, 11, 11]. Since 110 is strictly less stable than 111, 111 can be safely thrown away, but 001 and 110 are incomparable even though 001 has less number of 1’s. Therefore all the minimal elements in a given set under the Boolean lattice should be extracted. In the example above this also gives the same subset relation.

\( ^6 \) This distinction can be removed to design a more aggressive approximation scheme.

\( ^7 \) One can employ a log-based encoding to decrease the number of Boolean variables introduced although this makes it difficult to extract the latest required times later.
equations must be true regardless of X, X should be universally-quantified.

\[ F(\alpha, \beta) = \forall X. [z^{\text{r}}(X, \alpha, \beta) \equiv z(X)] \forall X. [z^{\text{r}}(X, \alpha, \beta) \equiv z(X)] \]

Any satisfying assignment of \( F(\alpha, \beta) \) meets the timing requirement.

Let us go one step further, as we did in the exact approach, to see how we can compute the latest required time at primary inputs from \( F(\alpha, \beta) \).

**Theorem 1** \( F(\alpha, \beta) \) is a monotone increasing function in terms of \( \alpha \) and \( \beta \).

We have shown that \( F(\alpha, \beta) \) captures all the required times that meet a given timing constraint. Since having less 1’s in an assignment to \( \alpha \) and \( \beta \) requires less stability, we are interested in a satisfying assignment where no assignment of 1 to a variable can be changed to 0 without making the assignment non-satisfying. Since \( F(\alpha, \beta) \) is a monotone increasing function, such an assignment has a one-to-one correspondence with a prime of \( F(\alpha, \beta) \). Notice that any prime of the function has only positive literals. The variables with positive literals in a prime are those which must be set to 1. Each prime gives a different limiting condition as in the exact algorithm.

Let us go back to the previous example. By introducing \( \alpha \) and \( \beta \) variables, leaf \( \chi \) variables can be expressed as follows.

\[
\begin{align*}
x_{x_1,1}^0 &= x_1 \alpha_1^1 \\
x_{x_1,1}^1 &= x_1 \beta_1^1 \\
x_{x_2,1}^0 &= x_2 \alpha_2^1 \\
x_{x_2,1}^1 &= x_2 \beta_2^1 \\
x_{x_2,1}^{-1} &= x_2 \alpha_2^0 + x_2 \beta_2^0 + x_{x_2,0}^0 = \beta_2^1 x_2 + \beta_2^2 x_2 \\
\end{align*}
\]

The \( F \) function for this example is:

\[ F(\alpha, \beta) = \alpha_1^1 \alpha_2^1 \alpha_2^0 \beta_1^1 \beta_2^2 \]

There are two satisfying assignments for the function:

\[ (\alpha_1^1 \alpha_2^0 \beta_1^1 \beta_2^2) \]

The second approach gives a looser constraint than the first approximation technique. The advantages of this second approximation technique are twofold. First, one can directly use state-of-the-art timing analysis tools as a subroutine. Secondly even if an entire analysis takes a huge amount of time, any intermediate \( r \) looser than topological analysis gives useful information immediately.

5 Experimental Results

We have implemented on top of SIS the exact and the two approximate algorithms for required time computation discussed in Section 4. The delay model we used in the experiments is the unit delay model. In all the experiments we set the required times of all the primary outputs to zero and computed required times at primary inputs. All the Boolean operations in the exact and the first approximate methods are done using BDD’s while in the second approximate method a SAT-based timing analysis tool [8] is used.

The efficiency of the algorithms is dependent on how much reconvergence a given circuit has. In the exact algorithm, we introduce one Boolean variable for each pair of a primary input and a potential required time. Thus, the existence of many reconvergences implies manipulation of \( \chi \) functions of many input variables in BDD’s. The same observation is also true for the first approximate method, where a Boolean parameter variable is introduced for each such pair.

The second approximate algorithm is more scalable than the first one since the computation engine is a SAT solver. As mentioned before, an advantage of this approach is that any intermediate required time validated can be used as a safe approximation to the exact solution.

Table 1 shows a comparison between the exact and the approximate algorithms on MCNC benchmark circuits. CPU times are measured in seconds on DEC AlphaServer 8400 5/300. The exact algorithm was run with dynamic variable reordering being set. * in the table denotes that the analysis gives a non-trivial required time looser than topological analysis. The reason why the first approximate algorithm gives a looser constraint than the second algorithm in some examples is that the required times of values 0 and 1 for each primary input are distinguished in the first algorithm while the current implementation of the second algorithm only searches for value-independent required times for an efficiency reason.

\footnote{In many ISCAS benchmark circuits the number of Boolean variables needed can easily go beyond hundreds.}
<table>
<thead>
<tr>
<th>circuit</th>
<th>#PI</th>
<th>#PO</th>
<th>CPU time (exact)</th>
<th>CPU time (approximate 1)</th>
<th>CPU time (approximate 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>i1</td>
<td>25</td>
<td>16</td>
<td>93.0s</td>
<td>0.1s</td>
<td>0.5</td>
</tr>
<tr>
<td>i2</td>
<td>201</td>
<td>1</td>
<td>memory out</td>
<td>8.3s</td>
<td>15.3</td>
</tr>
<tr>
<td>i8</td>
<td>133</td>
<td>81</td>
<td>-</td>
<td>26.8s</td>
<td>238.7s</td>
</tr>
<tr>
<td>i9</td>
<td>88</td>
<td>63</td>
<td>-</td>
<td>3.0s</td>
<td>4.6</td>
</tr>
<tr>
<td>i10</td>
<td>257</td>
<td>224</td>
<td>memory out</td>
<td>36335.6s</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Required Time Computation – Exact vs. Approximate

<table>
<thead>
<tr>
<th>circuit</th>
<th>Non-trivial required time?</th>
<th>CPU time first ( r \neq r_L ) (in seconds)</th>
<th>CPU time ( r_{max} ) (in seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>Yes</td>
<td>7.9</td>
<td>33.2</td>
</tr>
<tr>
<td>C880</td>
<td>No</td>
<td>-</td>
<td>26.7</td>
</tr>
<tr>
<td>C1355</td>
<td>No</td>
<td>-</td>
<td>26.0</td>
</tr>
<tr>
<td>C1908</td>
<td>Yes</td>
<td>1.0</td>
<td>1356.4</td>
</tr>
<tr>
<td>C2670</td>
<td>Yes</td>
<td>2.8</td>
<td>2298.1</td>
</tr>
<tr>
<td>C3540</td>
<td>Yes</td>
<td>0.5</td>
<td>&gt; 12 hours</td>
</tr>
<tr>
<td>C5315</td>
<td>Yes</td>
<td>77.7</td>
<td>359.6</td>
</tr>
<tr>
<td>C6288</td>
<td>Yes</td>
<td>1.0</td>
<td>&gt; 12 hours</td>
</tr>
<tr>
<td>C7552</td>
<td>Yes</td>
<td>2.5</td>
<td>992.5</td>
</tr>
</tbody>
</table>

Table 2: Required Time Computation – ISCAS Example

the analysis of i10, the first non-trivial required time was obtained in 134.9 seconds.

Table 2 shows CPU times of the second approximate algorithm on ISCAS combinational benchmark circuits. CPU times are measured in seconds on the same machine. The second column shows whether the algorithm could find non-trivial required times or not. The third and fourth columns show CPU time spent until the first non-trivial required time was found, and CPU time for the entire analysis respectively. Although the algorithm could not finish on C3540 and C6288 within 12 hours of CPU time, it found non-trivial required times within a second.

6 Conclusions

We have studied how to compute required times of combinational circuits more accurately than topological delay analysis, by taking false path effects into account. The technique proposed in this paper, which is designed on top of the theory of false path analysis, computes a more relaxed yet correct timing constraint.

Even though this approach captures larger temporal flexibility, existing timing optimization algorithms are not able to exploit the flexibility fully since timing specifications handled by timing optimization algorithms are of much simpler form than value-dependent constraints computed by our technique. A more sophisticated timing optimization algorithm compatible with the refined timing constraint proposed here is needed to fill this gap. Another avenue for future research is to improve the computational complexity of the algorithm by further approximations. In the current algorithms we distinguish between all potential required times at primary inputs. One possible approximation is to group them into clusters of neighboring required times conservatively. Controlling the number of clusters gives a trade-off between accuracy and CPU time for a more realistic delay model.

We have recently shown [6] how this analysis leads to an abstract delay model for black boxes. The delay model can be accurate taking into account false paths, without giving the internal details of the box.

Acknowledgments

The authors wish to thank Alexander Saldanha for kindly allowing us to use his timing analysis tool as a part of our implementation.

References