# EXACT REQUIRED TIME ANALYSIS VIA FALSE PATH DETECTION

by

Yuji Kukimoto and Robert K. Brayton

Memorandum No. UCB/ERL M97/44

16 June 1997

# EXACT REQUIRED TIME ANALYSIS VIA FALSE PATH DETECTION

by

Yuji Kukimoto and Robert K. Brayton

Memorandum No. UCB/ERL M97/44

16 June 1997

**ELECTRONICS RESEARCH LABORATORY**

College of Engineering
University of California, Berkeley
94720

# Exact Required Time Analysis via False Path Detection

Yuji Kukimoto* Robert K. Brayton
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720
{kukimoto,brayton}@eecs.berkeley.edu

June 16, 1997

**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. Experimental results show that the technique can extract timing constraints looser than those based on topological analysis for all but three in the ISCAS combinational circuit suite.

*The first author is supported by SRC-97-DC-324.

# 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 subcircuit [4]. Exact computation of the full flexibility is expensive for large networks, so a subset of the flexibility computable efficiently is of great interest, for which various don't care computation techniques [11, 10] have been developed. Recently, Watanabe and Brayton [14] 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 replaceabiliy addressed by

Singhal *et al.* [13] 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 $\Re$. 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. Section 5 shows how to tackle the main problem using the technique of Section 4. Experimental results are given in Section 6. Section 7 concludes the paper.

# 2 Preliminaries

In this section, we review sensitization theory for the false path problem [5, 6, 9]. Specifically, the theory developed in [9] 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*.

## 2.2 The Extended Bounded Delay-0 (XBD0) Delay Model

The extended bounded delay-0 model [9], XBD0 model, is the delay model most commonly used in false path analysis. It is the underlying model for the floating mode analysis [5] and viability analysis [8]. 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. Note that the monotone speedup property introduced in [8] exactly corresponds to the condition that minimum delays of all the gates are zero.

## 2.3 Sensitization under the XBD0 Model

The core idea of [9] 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 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[1]. 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 the 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 $\chi$ 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 $t - d_n$.

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

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.

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

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 $x$.

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

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

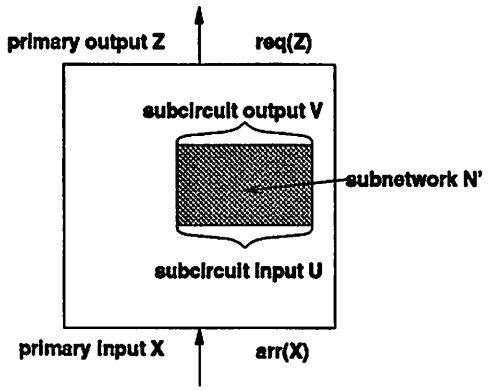


Figure 1: Boolean Network

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 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 [9] 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 outputs and inputs 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 $\mathcal{N}$ and a subnetwork $\mathcal{N}'$ of $\mathcal{N}$[2], our interest is to characterize the timing flexibility of $\mathcal{N}'$ so that resynthesis of the subcircuit can be performed locally without violating the timing constraint of the entire network $\mathcal{N}$. Note that our assumption is that $\mathcal{N} \setminus \mathcal{N}'$ remains unchanged and only $\mathcal{N}'$ is to be resynthesized. 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 $\mathcal{N}$ respectively. Let $U = (u_1, \ldots, u_p)$ and $V = (v_1, \ldots, v_q)$ denote inputs and outputs of $\mathcal{N}'$ respectively.

[2]To be precise, $\mathcal{N}'$ must meet the condition that there is no path leading from a subcircuit output to a subcircuit input.

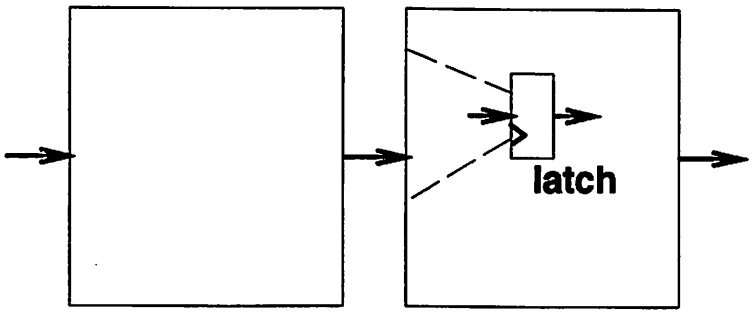


Figure 2: Hierarchical Synthesis

(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 $\mathcal{N} \setminus \mathcal{N}'$ 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 part of critical 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 [12] 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. Optimizing the entire circuit as a single chunk is not desirable in this context because it destroys the hierarchy meaningful to designers and more importantly the whole circuit may be too large to handle for synthesis algorithms. 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

```
    compute_required_time(N)
1       sort all the nodes in N in a reverse topological order
2       for each node n ∈ N s.t. n is not a primary output{
3           req(n) = ∞
4       }
5       for each node n ∈ N {
6           for each fanin m of n {
7               if (req(n) − d_n < req(m))
8                   req(m) = req(n) − d_n
9           }
10      }
```

Figure 3: Algorithm to Compute Required Time based on Topological Delay

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 $\mathcal{N}$ is the combinational portion of the right component and $\mathcal{N}'$ 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, $\mathcal{N}$ corresponds to the entire circuit and $\mathcal{N}'$ the driving circuit.

# 4 Computing Required Time under the Existence of False Paths

In this section we consider the following simpler problem:

> Given a Boolean network, 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 as shown in Figure 3. 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 in 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 $\chi$ functions introduced in Section 2. For each primary output, $\chi$ functions for constants 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 $\chi$ functions almost works. A modification is required only in terminal cases. Since we do not know when a primary input signal arrives, leaf $\chi$ functions at primary inputs remain as unknown variables. Henceforth, we call leaf $\chi$ functions at primary inputs *leaf $\chi$ variables*. Let $x$ be a primary input. Assume that after recursive construction of $\chi$ functions at primary outputs, leaf $\chi$ variables for $x$ are needed at times $t_1 < t_2 < \ldots < t_{p_x}$ for value 1 and at times $t'_1 < t'_2 < \ldots < t'_{q_x}$ for value 0. Remember that leaf $\chi$ variables are characteristic functions of sets of input vectors that are stable by required times. This implies that for any $t_a < t_b$ the set of stable input vectors by time $t_a$ must be contained in the set of stable input vectors by time $t_b$. Therefore, the following ordering conditions among leaf $\chi$ variables must be met.

$$\begin{aligned} \emptyset &\subseteq \chi^{t_1}_{x,1} \subseteq \chi^{t_2}_{x,1} \subseteq \cdots \subseteq \chi^{t_{p_x}}_{x,1} \subseteq x \\ \emptyset &\subseteq \chi^{t'_1}_{x,0} \subseteq \chi^{t'_2}_{x,0} \subseteq \cdots \subseteq \chi^{t'_{q_x}}_{x,0} \subseteq \overline{x} \end{aligned}$$

The formulas above indicate that leaf $\chi$ 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 $\chi$ 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 $\mathcal{N}$ 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 $\chi^t_{z,1}$ and $\chi^t_{z,0}$ are computed in terms of leaf $\chi$ variables at primary inputs, which we call $\chi X$. The goal is to assign Boolean functions of $X$ to unknown $\chi X$ variables so that

$$\begin{aligned} \chi^t_{z,1}(\chi X) &= z(X) \\ \chi^t_{z,0}(\chi X) &= \overline{z(X)} \end{aligned}$$

under the ordering constraints among $\chi X$ variables discussed above, where $z(X)$ denotes the functionality of the primary output in terms of primary inputs $X$[3]. The set

[3]It is possible to extend the theory to the case where $z(X)$ is an incompletely specified function.

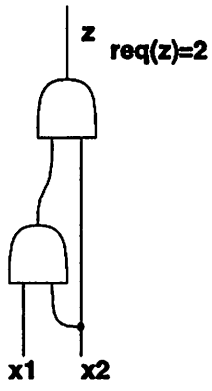


Figure 4: Example

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 $\chi$ variables $\chi_X$. The Boolean constraints to be satisfied are:

$$\begin{aligned} \chi^t_{z,1}(\chi_X) &= z(X) \\ \chi^t_{z,0}(\chi_X) &= \overline{z(X)} \\ \text{for each } x \in X: &\quad \emptyset \subseteq \chi^{t_1}_{x,1} \subseteq \ldots \subseteq \chi^{t_{p_x}}_{x,1} \subseteq x \\ \text{for each } x \in X: &\quad \emptyset \subseteq \chi^{t'_1}_{x,0} \subseteq \ldots \subseteq \chi^{t'_{q_x}}_{x,0} \subseteq \overline{x} \end{aligned}$$

It is easy to transform the above set of Boolean equations to an equivalent single Boolean equation of form $F(X, \chi_X) = 1$ [3] 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 $z$. One method for extracting such a function is Boolean unification [3].

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 4. 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.

$\chi$ functions for $z$ can be computed as follows.

$$\begin{aligned}\chi^2_{z,1} &= \chi^0_{x_1,1}\chi^0_{x_2,1}\chi^1_{x_2,1}\\ \chi^2_{z,0} &= \chi^0_{x_1,0}+\chi^0_{x_2,0}+\chi^1_{x_2,0}\end{aligned}$$

The procedure described above gives the following Boolean relation.

| $x_1x_2$ | $\chi^0_{x_1,1}\chi^0_{x_2,1}\chi^1_{x_2,1}\chi^0_{x_1,0}\chi^0_{x_2,0}\chi^1_{x_2,0}$ |
|---|---|
| 00 | {000100,000101,000001,000011,000111} |
| 01 | {000100,001100,011100} |
| 10 | {000001,000011,100001,100011} |
| 11 | {111000} |

Let us examine the relation to see what kind of timing constraints need to be imposed. For input 00, the first three leaf $\chi$ variables must be zero in all the cases. This is natural since these $\chi$ variables are for constant 1, and neither $x_1$ nor $x_2$ may become stable to 1 in this case. The first two and the last patterns correspond to the case where $\chi^0_{x_1,0}$ is one, i. e. $x_1$ is stable to 0 by time 0. In this case, $z$ is guaranteed to be stable to 0 no matter how $x_2$ behaves. The only constraint to be satisfied is $\chi^0_{x_2,0} \subseteq \chi^1_{x_2,0}$. The third and fourth patterns are for the case $\chi^0_{x_1,0}$ is zero. This time, $x_1$ is not stable to zero at time 0, but as long as $x_2$ becomes zero by time 1, $z$ will be stable by time 2. Again, the ordering condition $\chi^0_{x_2,0} \subseteq \chi^1_{x_2,0}$ must be met.

As mentioned previously, one can think of this relation as a generalization of required time. 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:

$$\begin{aligned}\chi^0_{x_1,1} &= x_1x_2\\ \chi^0_{x_2,1} &= x_1x_2\\ \chi^1_{x_2,1} &= x_1x_2\\ \chi^0_{x_1,0} &= \overline{x_1}\\ \chi^0_{x_2,0} &= 0\\ \chi^1_{x_2,0} &= x_1\overline{x_2}.\end{aligned}$$

To focus on only the stability of signals, we define $\tilde{\chi}^t_n$ as follows.

$$\tilde{\chi}^t_n = \chi^t_{n,1} + \chi^t_{n,0}$$

This $\tilde{\chi}$ 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,

$$\tilde{\chi}^0_{x_1} = \overline{x_1} + x_2$$

$$\tilde{\chi}^0_{x_2} = x_1 x_2$$
$$\tilde{\chi}^1_{x_2} = x_1$$

The interpretation of this is that primary input $x_1$ must be stable by time 0 just for the case $\overline{x_1} + x_2$, and if $x_1\overline{x_2}$, 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 $x_1x_2$. If $\tilde{\chi}^1_{x_2}\overline{\tilde{\chi}^0_{x_2}} = x_1\overline{x_2}$, $x_2$ 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 most pessimistic required time condition.

We have seen that the relation represents all the permissible temporal behaviors, from an aggressive behavior where a signal never arrives under a certain condition to the most stringent condition exactly corresponding to topological analysis. The next question is how to extract the latest required time conditions from the relation 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 $\chi$ 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.

| $x_1x_2$ | $\chi^0_{x_1,1}\chi^0_{x_2,1}\chi^1_{x_2,1}\chi^0_{x_1,0}\chi^0_{x_2,0}\chi^1_{x_2,0}$ |
|---|---|
| 00 | {000100,000001} |
| 01 | {000100} |
| 10 | {000001} |
| 11 | {111000} |

| $x_1x_2$ | $req(x_1)req(x_2)$ |
|---|---|
| 00 | $\{(0,\infty),(\infty,1)\}$ |
| 01 | $\{(0,\infty)\}$ |
| 10 | $\{(\infty,1)\}$ |
| 11 | $\{(0,0)\}$ |

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 $x_1x_2 = 00$. Those two conditions are not comparable and each gives a different limiting condition.

## 4.2 Approximate Approach 1

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

[4]Pick the last output pattern for each minterm

[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 {001, 110, 111}. 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.

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 $\alpha_1^x, \ldots, \alpha_{p_x}^x, \beta_1^x, \ldots, \beta_{q_x}^x$ to encode the ordering constraints in leaf $\chi$ variables[7].

$$\begin{aligned}
\chi_{x,1}^{t_{p_x}} &= x\alpha_1^x \\
\chi_{x,1}^{t_{p_x-1}} &= x\alpha_1^x\alpha_2^x \\
&\cdots \\
\chi_{x,1}^{t_1} &= x\alpha_1^x\alpha_2^x\ldots\alpha_{p_x}^x \\
\chi_{x,0}^{t'_{q_x}} &= \overline{x}\beta_1^x \\
\chi_{x,0}^{t'_{q_x-1}} &= \overline{x}\beta_1^x\beta_2^x \\
&\cdots \\
\chi_{x,0}^{t'_1} &= \overline{x}\beta_1^x\beta_2^x\ldots\beta_{q_x}^x
\end{aligned} \tag{1}$$

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 $\overline{x}$ 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 $\overline{x}$ 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.

$$\begin{aligned}
\chi_{z,1}^{t}(X,\alpha,\beta) &= z(X) \\
\chi_{z,0}^{t}(X,\alpha,\beta) &= \overline{z(X)}
\end{aligned}$$

where $\alpha$ and $\beta$ are the set of all $\alpha$ variables and the set of all $\beta$ variables respectively. In the above all the leaf $\chi$ variables are substituted by the right-hand side expressions in Equations 1. Since these equations must be true regardless of $X$, $X$ should be universally-quantified.

$$\begin{aligned}
F(\alpha,\beta) &= \forall X.[(\chi_{z,1}^{t}(X,\alpha,\beta) \equiv z(X))(\chi_{z,0}^{t}(X,\alpha,\beta) \equiv \overline{z(X)})] \\
&= \forall X.[\chi_{z,1}^{t}(X,\alpha,\beta) \equiv z(X)]\forall X.[\chi_{z,0}^{t}(X,\alpha,\beta) \equiv \overline{z(X)}]
\end{aligned}$$

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

[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.

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 $F(\alpha, \beta)$. The following lemmas and theorems are useful.

**Lemma 1** *$\chi^t_{z,1}(\chi_X)$ and $\chi^t_{z,0}(\chi_X)$ are monotone increasing functions in terms of $\chi_X$.*

**Proof.** By the definition of $\chi$ functions, each $\chi$ function can be represented by a Boolean network where the local functionality of each node is monotone increasing in terms of its fanins. Hence, the claim holds. □

**Lemma 2** *$\chi^t_{z,1}(X, \alpha, \beta)$ and $\chi^t_{z,0}(X, \alpha, \beta)$ are monotone increasing functions in terms of $\alpha$ and $\beta$.*

**Proof.** Let $\hat{\alpha}$ and $\hat{\beta}$ be 0-1 assignments to $\alpha$ and $\beta$ respectively. Let $\hat{\chi_X}$ be the corresponding functions assigned for leaf $\chi$ variables under $\hat{\alpha}$ and $\hat{\beta}$. By changing a zero in $\alpha$ and $\beta$ to one, it is easy to see that one cannot decrease the functions $\chi_X$. Thus from Lemma 1 the claim is proved. □

**Lemma 3**

$$\chi^t_{z,1}(X, \alpha, \beta)\,|_{\alpha=(1,\ldots,1),\beta=(1,\ldots,1)} = z(X)$$
$$\chi^t_{z,0}(X, \alpha, \beta)\,|_{\alpha=(1,\ldots,1),\beta=(1,\ldots,1)} = \overline{z(X)}$$

**Proof.** Let $\tilde{\mathcal{N}}$ be the Boolean network for $\chi^t_{z,1}$. Let $NL$ be the list of all the nodes in the network topologically sorted from primary inputs of $\tilde{\mathcal{N}}$ (leaf $\chi$ variables) to the primary output $\chi^t_{z,1}$. Note that each node is labeled of the form $\chi^t_{n,v}$. The proof is by induction on these sorted nodes.

**Base case** ($n \in X$): By setting $\alpha = (1, \ldots, 1), \beta = (1, \ldots, 1)$, $\chi^t_{n,1} = n$ and $\chi^t_{y,0} = \overline{n}$ for any $t$.

**Induction** ($n \notin X$): From the inductive hypothesis, for any fanin of $\chi^t_{n,v}$, say $\chi^{t-d_n}_{m,v'}$, $\chi^{t-d_n}_{m,1} = m(X)$ and $\chi^{t-d_n}_{m,0} = \overline{m(X)}$, where $m(X)$ is the functionality of node $m$ in terms of $X$ in the original network $\mathcal{N}$. If $v = 1$, then the local function at the node $\chi^t_{n,1}$ in $\tilde{\mathcal{N}}$ is the same as the local function at the node $n$ in $\mathcal{N}$ since the former function is just the sum of all the primes of the latter function. Therefore $\chi^t_{n,1} = n(X)$. Similarly the local function at the node $\chi^t_{n,0}$ in $\tilde{\mathcal{N}}$ is the same as the *complement* of the local function at the node $n$ in $\mathcal{N}$. Thus, $\chi^t_{n,0} = \overline{n(X)}$.

Hence, $\chi^t_{z,1} = z(X)$ and $\chi^t_{z,0} = \overline{z(X)}$. □

**Corollary 1**

$$\begin{aligned} \forall \alpha, \beta : \chi^t_{z,1}(X, \alpha, \beta) &\subseteq z(X) \\ \forall \alpha, \beta : \chi^t_{z,0}(X, \alpha, \beta) &\subseteq \overline{z(X)} \end{aligned}$$

**Proof.** From Lemma 2 and Lemma 3. □

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

**Proof.** Consider a 0-1 assignment to $\alpha$ and $\beta$, say $\hat{\alpha}$ and $\hat{\beta}$ respectively. From Lemma 2 and Corollary 1, it is clear that changing zeroes to ones in $\hat{\alpha}$ and $\hat{\beta}$ does not decrease the function value of $F(\alpha, \beta)$ from one to zero. Therefore, $F(\alpha, \beta)$ 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. Thus, computing the latest required times from $F(\alpha, \beta)$ is equivalent to computing a prime of $F(\alpha, \beta)$. Note that 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.

$$\begin{aligned} \chi^0_{x_1,1} &= x_1 \alpha_1^{x_1} \\ \chi^0_{x_1,0} &= \overline{x_1} \beta_1^{x_1} \\ \chi^1_{x_2,1} &= x_2 \alpha_1^{x_2} \\ \chi^0_{x_2,1} &= x_2 \alpha_1^{x_2} \alpha_2^{x_2} \\ \chi^1_{x_2,0} &= \overline{x_2} \beta_1^{x_2} \\ \chi^0_{x_2,0} &= \overline{x_2} \beta_1^{x_2} \beta_2^{x_2} \end{aligned}$$

$$\begin{aligned} \chi^2_{z,1} &= \chi^0_{x_1,1} \chi^0_{x_2,1} \chi^1_{x_2,1} = \alpha_1^{x_1} \alpha_1^{x_2} \alpha_2^{x_2} x_1 x_2 \\ \chi^2_{z,0} &= \chi^0_{x_1,0} + \chi^0_{x_2,0} + \chi^1_{x_2,0} \\ &= \beta_1^{x_1} \overline{x_1} + \beta_1^{x_2} \overline{x_2} + \beta_1^{x_2} \beta_2^{x_2} \overline{x_2} \\ &= \beta_1^{x_1} \overline{x_1} + \beta_1^{x_2} \overline{x_2} \end{aligned}$$

The $F$ function for this example is:

$$\begin{aligned} F(\alpha,\beta) &= \forall x_1,x_2.(\alpha_1^{x_1}\alpha_1^{x_2}\alpha_2^{x_2}x_1x_2 = x_1x_2)(\beta_1^{x_1}\overline{x_1}+\beta_1^{x_2}\overline{x_2} = \overline{x_1}+\overline{x_2}) \\ &= \alpha_1^{x_1}\alpha_1^{x_2}\alpha_2^{x_2}\beta_1^{x_1}\beta_1^{x_2} \end{aligned}$$

There are two satisfying assignments for the function:
$(\alpha_1^{x_1}\alpha_1^{x_2}\alpha_2^{x_2}\beta_1^{x_1}\beta_1^{x_2}\beta_2^{x_2}) = (111110, 111111)$. The second assignment corresponds to topological analysis. The only prime of $F(\alpha,\beta)$ is $\alpha_1^{x_1}\alpha_1^{x_2}\alpha_2^{x_2}\beta_1^{x_1}\beta_1^{x_2}$, which corresponds to the first assignment. Let us look into the first assignment more carefully. The leaf $\chi$ variables under this assignment are:

$$\begin{aligned} \chi^0_{x_1,1} &= x_1 \\ \chi^0_{x_1,0} &= \overline{x_1} \\ \chi^1_{x_2,1} &= x_2 \\ \chi^0_{x_2,1} &= x_2 \\ \chi^1_{x_2,0} &= \overline{x_2} \\ \chi^0_{x_2,0} &= 0. \end{aligned}$$

This constraint means that $x_1$ has to arrive by time 0 and that $x_2$ has to arrive by time 0 if $x_2 = 1$ but by time 1 if $x_2 = 0$. Notice that this required time is strictly tighter than the following two required times based on the exact algorithm, but is looser than the required time based on topological analysis.

$$\begin{aligned} \chi^0_{x_1,1} &= x_1x_2 \\ \chi^0_{x_1,0} &= \overline{x_1} \\ \chi^1_{x_2,1} &= x_1x_2 \\ \chi^0_{x_2,1} &= x_1x_2 \\ \chi^1_{x_2,0} &= x_1\overline{x_2} \\ \chi^0_{x_2,0} &= 0. \end{aligned}$$

$$\begin{aligned} \chi^0_{x_1,1} &= x_1x_2 \\ \chi^0_{x_1,0} &= \overline{x_1}x_2 \\ \chi^1_{x_2,1} &= x_1x_2 \\ \chi^0_{x_2,1} &= x_1x_2 \\ \chi^1_{x_2,0} &= \overline{x_2} \\ \chi^0_{x_2,0} &= 0. \end{aligned}$$

## 4.3 Approximate Approach 2

The approximation technique in Section 4.2 was a relaxation of the exact formulation. Another approximate analysis will be discussed next, which is based on repeated functional timing analysis.

Assume that for each primary input $x_i$ all the times for which leaf $\chi$ variables are needed are listed. Let $R_i$ be the set which contains all those times for primary input $x_i$. For the sake of simplicity assume that $R_i$ contains all the times needed for value 1 and those for value 0[8]. Let $R = R_1 \times \ldots \times R_n$. Let $r_\perp = (r_{\perp,1}, \ldots, r_{\perp,n}) \in R$ where $r_{\perp,i} = \min_{t \in R_i} t$. Similarly let $r_\top = (r_{\top,1}, \ldots, r_{\top,n}) \in R$ where $r_{\top,i} = \max_{t \in R_i} t$. Let $r, r' \in R$. A partial order $\prec$ is defined over $R$ as follows: for $\forall r, r' \in R$, $r \prec r'$ iff $\forall i = \{1, \ldots, n\}, r_i \leq r'_i$. This partial order forms a lattice over $R$, where the top and the bottom elements are $r_\top$ and $r_\perp$ respectively. Each $r \in R$ represents a candidate for the exact required time.

$r_\perp$ corresponds to the required times at primary inputs obtained by topological analysis. Therefore, if the primary inputs arrive by $r_\perp$, the stability of the primary output by the given required time is guaranteed. Our goal is to find the largest $r \in R$ with respect to $\prec$ such that $r$ guarantees the stability of the primary output. $r$ may not be unique as we saw in the first approximation technique.

One way to find such $r$ is to climb up the lattice gradually from $r_\perp$ by choosing larger $r$'s in a greedy fashion. To test if a certain $r$ is a valid choice, one can simply perform functional timing analysis of the circuit under the arrival times corresponding to $r$ at the primary inputs. If the delay at the primary output is less than or equal to its required time, $r$ is a safe condition. The largest $r$ that meets this requirement gives a limiting condition. The search for $r$ can be refined by the use of backtracking so that all the maximal $r$'s satisfying the condition are enumerated.

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 Computing Timing Flexibility of Subcircuits

Having looked into how to compute required times by taking into account false path effects, we now consider the problem of computing timing flexibility of subcircuits. The timing specification of a subcircuit consists of arrival times at the subcircuit inputs and required times at the subcircuit outputs. One can then pass this information to a logic synthesis tool along with the functional specification of the subcircuit to resynthesize it so that the entire circuit after replacing the existing subcircuit with a new optimized subcircuit meets the top-level functional and temporal specification.

[8]It is possible to extend the idea so that required times for values 1 and 0 are handled separately for each primary input.

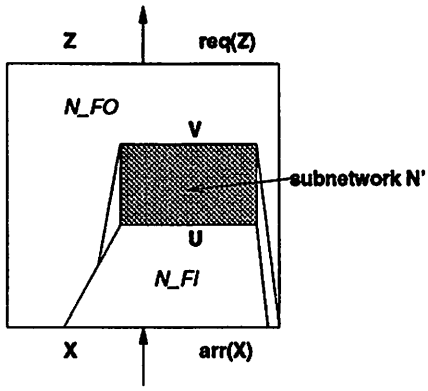


Figure 5: Boolean Network

In this section, we show that the problem can be solved as a combination of the standard false path analysis, where arrival times are propagated forward to subcircuit inputs from primary inputs, and the required time computation technique discussed in Section 4, where required times are propagated backward from primary outputs to subcircuit outputs.

## 5.1 Arrival Time Computation

The first step is to compute arrival times at the subcircuit inputs. The transitive fanin of the subcircuit inputs is extracted from $\mathcal{N}$, which we call $\mathcal{N}_{FI}$. (See Figure 5.) This network is then analyzed with a technique similar to standard false path analysis[9]. Notice that the primary outputs of $\mathcal{N}_{FI}$ are the subcircuit inputs, and the primary inputs of $\mathcal{N}_{FI}$ are a subset of primary inputs $X$ of $\mathcal{N}$. The main difference between this problem and the false path analysis problem is that in the false path problem we only care about the latest arrival time for each output while in our problem interactions among arrival times of different outputs are of much interest to capture timing flexibility accurately[10].

Consider applying $\chi$ function analysis on $\mathcal{N}_{FI}$. For each subcircuit input $u_i \in U$, we list all the *topological* arrival times at $u_i$, which is easily computed by propagating arrival time from primary inputs to the subcircuit inputs while maintaining not a single latest arrival time but a set of arrival times at each node. Then, $\chi^t_{u_i,v}, v \in \{0,1\}$ is computed at all those arrival times. Note that these functions are in terms of primary inputs $X$ of $\mathcal{N}$. Then $\tilde{\chi}^t_{u_i} = \chi^t_{u_i,0} + \chi^t_{u_i,1}$ represents all the primary input vectors at $X$

[9]To be precise the delay of the fanin network is affected by changing its fanout, which is unknown in our setup since the fanout network is to be resynthesized. In this paper we do not take this load effect into consideration to simplify the explanation.

[10][2] proposed a technique to compute input-value dependent delay using ADD's. This can be used as an alternative to the analysis below.

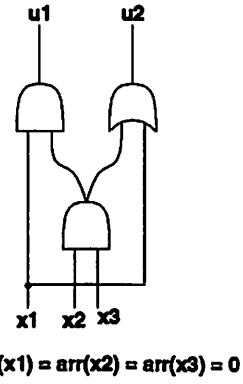


Figure 6: Example – $\mathcal{N}_{FI}$

that make a signal at $u_i$ stable by time $t$. Assume that the list of topological arrival times at $u_i$ is $\{t_1, \ldots, t_l\}$. Now the Boolean space $B^{|X|}$ can be partitioned into $l$ disjoint sets $\{S_1, \ldots, S_l\}$ in terms of arrival times as follows.

$$\begin{aligned} S_1 &= \tilde{\chi}_{u_i}^{t_1} \\ S_k &= \tilde{\chi}_{u_i}^{t_k} \cdot \overline{\tilde{\chi}_{u_i}^{t_{k-1}}} \qquad (k = 2, \ldots, l) \end{aligned}$$

Note that $\tilde{\chi}$ increases as $t$ increases by its construction and $S_k$'s are defined as differences between time-neighboring functions. The set $S_k (k = 1, \ldots, l)$ contains all the input vectors at $X$ that make the signal $u_i$ stable by time $t_k$ but not by time $t_{k-1}$, where $t_0 = -\infty$.

Once a partition of $B^{|X|}$ is computed for each subcircuit input, all the partitions are superimposed on $B^{|X|}$ to form a refined partition. This is equivalent to partitioning $B^{|X|}$ such that any input vector in a class has the same arrival time behavior at $U$.

The final step is to interpret this arrival time in terms of subcircuit inputs $U$ so that temporal specification of the subcircuit is given locally in terms of its inputs. Remember that so far arrival time at $U$ is computed in terms of $X$. The subcircuit, however, cannot distinguish input vectors applied at $X$ unless they yield different vectors at $U$. Therefore, it is necessary to reinterpret this partition in terms of subcircuit inputs $U$ so that one can tell what arrival time behaviors could be observed for each vector at $U$. This is easily computed from the partition of $B^{|X|}$ computed above and the functionality of the transitive fanin network $\mathcal{N}_{FI}$. The Boolean space $B^{|X|}$ can be first partitioned into $B^{|U|}$ sets depending on what vector is driven at $U$. Now for each vector $u$ of $B^{|U|}$, we know the set of all the vectors of $B^{|X|}$ that yield $u$ at $U$. Using the partition of $B^{|X|}$ computed previously, one can list all the possible arrival times for the vector $u$, from which all the latest arrival times are extracted.

Let us illustrate this analysis with an example. Assume that the network in Figure 6 is $\mathcal{N}_{FI}$. For simplicity, we also assume that each gate has a unit delay and all the

primary inputs arrive at time 0. The $\chi$ function analysis gives the following.

$$\begin{aligned}\tilde{\chi}^2_{u_1} &= 1\\ \tilde{\chi}^1_{u_1} &= \overline{x_1}\\ \tilde{\chi}^2_{u_2} &= 1\\ \tilde{\chi}^1_{u_2} &= x_1\end{aligned}$$

The first two equations implies that if $x_1 = 0$, $u_1$ arrives at time 1, but otherwise the signal arrives at 2. The last two equations then describe signal stability of $u_2$. If $x_1 = 1$, then $u_2$ arrives at time 1, but otherwise the arrival time is 2. This can be summarized in the following table.

| $x_1x_2x_3$ | $u_1u_2$ | $arr(u_1)\ arr(u_2)$ |
|---|---|---|
| 000 | 00 | (1,2) |
| 001 | 00 | (1,2) |
| 010 | 00 | (1,2) |
| 011 | 01 | (1,2) |
| 100 | 01 | (2,1) |
| 101 | 01 | (2,1) |
| 110 | 01 | (2,1) |
| 111 | 11 | (2,1) |

Now, notice that the subcircuit which $\mathcal{N}_{FI}$ feeds into cannot distinguish $x_1x_2x_3 =$ 011 and 100 since both yield the same vector 01 at $u_1u_2$. Thus, when the subcircuit is resynthesized, we can only assume that the arrival time at the subcircuit input is either (1,2) or (2,1) when $u_1u_2 = 01$. Although it is possible to approximate this by having a single arrival time (2,2), it is not desirable since this is an overconstraint[11].

The following table is obtained by folding the table above in terms of the values of $u_1u_2$.

| $u_1u_2$ | $arr(u_1)\ arr(u_2)$ |
|---|---|
| 00 | {(1,2)} |
| 01 | {(1,2),(2,1)} |
| 10 | $\{(\infty,\infty)\}$ |
| 11 | {(2,1)} |

$\{(\infty,\infty)\}$ for 10 means no constraints since the subcircuit never observes the vector at the input. This corresponds to a satisfiability don't care among $u_1$ and $u_2$. It is interesting to observe that functional flexibility is captured in this framework naturally.

[11] If an arrival time tuple is strictly earlier than another tuple, the former is dropped since the subcircuit $\mathcal{N}'$ must be synthesized under the worst-case scenario.

## 5.2 Required Time Computation

Computing required times at subcircuit outputs can be performed by analyzing a subnetwork of $\mathcal{N}$, $\mathcal{N}_{FO}$, with the same analysis technique described in Section 4. $\mathcal{N}_{FO}$ is the same network as $\mathcal{N}$ except all the subcircuit outputs $V$ are relabeled as primary inputs. (See Figure 5.) Notice that required times at the subcircuit outputs are of interest. Since we know arrival times at $X$, there is no need to introduce leaf $\chi$ variables for those primary inputs of $\mathcal{N}_{FO}$ which are elements of $X$. As we have seen in Section 4, required time is computed for each vector $v \in B^{|V|}$ at subcircuit outputs.

## 5.3 Towards More Accurate Timing Flexibility

As the final remark in this section, we consider a special case of the problem where no functional flexibility is explored in resynthesizing the subcircuit. In other words, the functional specification given for the subcircuit is the same functionality currently implemented. This allows us to compute timing flexibility more accurately.

For arrival time computation at subcircuit inputs, instead of interpreting arrival time in terms of subcircuit inputs, we can simply keep arrival time in terms of primary inputs $X$. Required times at subcircuit outputs are computed for each vector $v \in B^{|V|}$ in the previous subsection. Since the functionality of the subcircuit is preserved after resynthesis, the functionality of $V$ in terms of $X$ is kept unchanged. Therefore, it is possible to interpret the required times in terms of primary inputs $X$. Now for each primary input vector $x \in B^{|X|}$ we have a single arrival time at the subcircuit inputs and possibly multiple required times at the subcircuit outputs. One can then map this timing constraint to the subcircuit. Since arrival times and required times are coupled through $X$, analysis is more accurate compared to the one described before where arrival times and required times are computed completely independently.

# 6 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 [9] 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[12] in BDD's. The same

[12] In many ISCAS benchmark circuits the number of Boolean variables needed can easily go beyond hundreds.

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 efficiency. Although the second approximate algorithm took 10 hours to complete 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.

# 7 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 [7] how this analysis leads to an abstract delay model for

| circuit | #PI | #PO | CPU time (exact) | CPU time (approximate 1) | CPU time (approximate 2) |
|---|---|---|---|---|---|
| i1 | 25 | 16 | 93.0* | 0.1* | 0.5 |
| i2 | 201 | 1 | memory out | 8.3* | 15.3 |
| i3 | 132 | 6 | 3277.9* | 0.1 | 0.0 |
| i4 | 192 | 6 | - | 0.2 | 0.0 |
| i5 | 133 | 66 | - | 1.9 | 10.7 |
| i6 | 138 | 67 | - | 0.7 | 16.0 |
| i7 | 199 | 67 | - | 0.9 | 31.7 |
| i8 | 133 | 81 | - | 26.8* | 238.7* |
| i9 | 88 | 63 | - | 3.0* | 4.6 |
| i10 | 257 | 224 | - | memory out | 36335.6* |

Table 1: Required Time Computation – Exact vs. Approximate

| circuit | Non-trivial required time? | CPU time first $r \neq r_{\perp}$ (in seconds) | CPU time $r_{max}$ (in seconds) |
|---|---|---|---|
| C432 | Yes | 7.9 | 33.2 |
| C499 | No | - | 40.1 |
| C880 | No | - | 26.7 |
| C1355 | No | - | 26.0 |
| C1908 | Yes | 1.0 | 1356.4 |
| C2670 | Yes | 2.8 | 2298.1 |
| C3540 | Yes | 0.5 | > 12 hours |
| C5315 | Yes | 77.7 | 359.6 |
| C6288 | Yes | 1.0 | > 12 hours |
| C7552 | Yes | 2.5 | 992.5 |

Table 2: Required Time Computation – ISCAS Example

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 part of our implementation.

## References

[1] A. Aziz, R. K. Brayton, F. Balarin, and V. Singhal. Timing-safe replaceability for combinational designs. In *Proceedings of TAU 95: ACM/SIGDA International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, pages 121–128, November 1995.

[2] R. I. Bahar, H. Cho, G. D. Hachtel, E. Macii, and F. Somenzi. Timing analysis of combinational circuits using ADD's. In *Proceedings of the European Design and Test Conference*, pages 625–629, March 1994.

[3] F. M. Brown. *Boolean Reasoning*. Kluwer Academic Pubilshers, 1990.

[4] E. Cerny and M. A. Marin. An approach to unified methodology of combinational switching circuits. *IEEE Transactions on Computers*, C-26(8):745–756, August 1977.

[5] 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.

[6] 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.

[7] Y. Kukimoto and R. K. Brayton. Hierarchical timing analysis under the XBD0 model. In *International Workshop on Logic Synthesis*, May 1997.

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

[9] 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.

[10] S. Muroga, Y. Kambayashi, H. C. Lai, and J. N. Culliney. The Transduction method – design of logic network based on permissible functions. *IEEE Transactions on Computers*, 38(10):1404–1424, October 1989.

[11] H. Savoj, R. K. Brayton, and H. J. Touati. Extracting local don't cares for network optimization. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 514–517, November 1991.

[12] K. J. Singh, A. R. Wang, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Timing optimization of combinational logic. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 282–285, November 1988.

[13] V. Singhal and C. Pixley. The verification problem for safe replaceability. In *Proceedings of 6th International Conference on Computer-Aided Verification, CAV'94*, pages 311–323, June 1994.

[14] Y. Watanabe and R. K. Brayton. The maximum set of permissible behaviors for FSM networks. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 316–320, November 1993.