Functional Timing Analysis Made Fast and General

Yi-Ting Chung

Graduate Institute of Electronics Engineering; Jie-Hong R. Jiang

Department of Electrical Engineering
National Taiwan University, Taipei 10617, Taiwan

{r99943080@ntu.edu.tw, jhjiang@cc.ee.ntu.edu.tw}

ABSTRACT

Functional, in contrast to structural, timing analysis is accurate, but computationally expensive in refuting false critical paths. Although satisfiability-based analysis using timed characteristic functions has been proposed, its efficiency and generality remain room for improvement. This paper shows functional timing analysis on industrial designs can be made up to several orders of magnitude faster and more generally applicable than prior methods.

Categories and Subject Descriptors

B.8.2 [Performance and Reliability]: Performance Analysis and Design Aids

General Terms

algorithms, design, verification

Keywords

false path, satisfiability solving, timed characteristic function, timing analysis

1. INTRODUCTION

In modern synthesis flow of very large scale integration (VLSI) design, timing analysis is essential in identifying timing critical regions for re-synthesis, determining operable clock frequencies, and avoiding wasteful over-optimization and thus accelerating design closure in meeting stringent timing constraints. As timing analysis often has to be repeatedly performed, how to make the computation efficient and accurate becomes a crucial task.

There are two main approaches to timing analysis. Static timing analysis (STA), based on pure structural (or topological) analysis, though fast with linear-time complexity, can be too pessimistic in estimating circuit delay due to the ignorance of false or nonsensitizable paths [2]. Functional timing analysis (FTA), on the other hand, provides accurate delay calculation, but is computationally intractable, i.e., NP-hard, in identifying false critical paths [7].

Many FTA algorithms, e.g., [7, 11, 2, 4, 10, 1, 14, 13, 6, 3], have been proposed. When delay-dependency is concerned, an FTA algorithm can be delay-independent [3] or delay-dependent [2]. The former (latter) identifies true and false paths without (with) respect to some timing library. Whereas the former is incomplete in that not every delay path can be concluded true or false regardless of arbitrary delay assignments, this paper focuses on the latter analysis.

When the underlying computation engine is concerned, an FTA algorithm can be powered by an automatic test pattern generator, e.g., [4, 1], or by a satisfiability (SAT) solver, e.g., [10, 14, 6]. Since ATPG-based computation involves sophisticated circuit transformation and multi-fault testing, it is difficult to implement and scale. In contrast, SAT-based computation allows simple implementation due to its clean separation between timed characteristic function (TCF) construction [7] and SAT solving. Although recent advances in SAT solving techniques [9, 8, 5] make SAT-based FTA a viable approach, FTA for large industrial designs remains challenging due to the massive numbers of variables and clauses when translating a complex TCF into a conjunctive normal form (CNF) formula for SAT solving. Moreover, modern SAT-based FTA algorithms [14, 6] cannot handle arbitrary gate types. Although formulation for general gate types has been proposed in [10], its complex formulas make SAT solving inefficient.

This work aims to develop a scalable and general FTA framework. The main results include 1) a generalized TCF framework supporting arbitrary complex gate types for both combined and separate rise/fall-time analysis, 2) an implication-based TCF construction and its linear-time translation to CNF without extra variables being introduced, 3) a TCF reduction technique with an improved equivalence relation based on table look-up, 4) a model generation mechanism, which produces a true critical path along with its sensitization condition if the target delay is sensitizable, and 5) an algorithm to identify timing critical regions of a circuit for potential timing optimization. Experimental results show substantial speedup over prior SAT-based delay computation methods and show effective critical region identification.

The rest of this paper is organized as follows. We give a brief description of our sensitization criteria and satisfiability model in Section 2. Our general TCF formulation is introduced and compared with prior formulations in Section 3. Section 4 presents efficient algorithms for timing delay computation and critical region identification. Section 5 shows experimental evaluation. Finally, conclusion and future work are given in Section 6.
of an AND gate is of controlling value when it is clear from the context. We assume the gate delay is of controlling value, (respectively, non-controlling value, denoted \(v_0, v_1\)) of \(f\) if the output value of \(f\) can (respectively, cannot) be completely determined by \(g\) with \(v_0\) (respectively, \(v_1\)) regardless of the truth assignments to other inputs. For example, any input of an AND gate is of controlling value 0.

For a complex gate, such as XOR, its inputs may likely have no controlling values at all. Nevertheless the notion of controlling values can be generalized to controlling cubes. For a complex gate \(f\), a truth assignment to a minimal (strict) subset \(S \subset FI(f)\) that determines the output value of \(f\) independent of other fanins forms a controlling cube. A literal in a controlling cube \(c\) is called a controlling literal of \(c\). For example, the controlling cubes of the gate \(f\) with function \(ab \lor c\) are \(\{a, b, \neg a, c, \neg b, c\}\), where cubes \(ab\) and \(c\) make \(f = 1\) and cubes \(-a, c\) and \(-b, c\) make \(f = 0\). In addition, \(-a\) is a controlling literal of cube \(-a, c\).

### 2.2 Sensitization Criteria

Among the various modes of circuit operation when functional timing analysis is concerned, floating-mode operation [7], which we adopt, is the most popular due to its simplicity and robustness. Under this mode of operation, the signals of a circuit are of unknown initial values and stabilize to their final values induced by a set of truth assignments on the PIs.

Under the floating-mode operation, various path sensitization criteria can be defined. The exact criterion [2] and viable criterion [7] are two commonly studied criteria. When the truth and falsity of a single path is concerned, the analysis of the former is exact whereas that of the latter is conservative [2]. Nevertheless, when the timing analysis is performed for all paths of a circuit without tracing a particular path, the viable criterion becomes exact as was shown in [11]. This paper is mainly concerned with computing the longest true delay among all paths.

### 2.3 Satisfiability of Timing Requirement

To perform satisfiability testing on whether there exists a PI assignment that exercises a target circuit delay through some unknown true path, the condition can be translated into the so-called timed characteristic function (TCF) [7]. Specifically, the set of PI assignments that makes the output value of \(f\) stabilize \(\leq t\) is characterized by a \((t, t)\) TCF, denoted \(\chi_f^{t,t}\). In other words, a PI assignment satisfying \(\chi_f^{t,t}\) makes the output value of \(f\) remains unknown (under the floating-mode assumption) until time \(t\). When the stabilization value of \(f\) is specific to value 0 (respectively 1), the corresponding 0/1-specified TCF is denoted as \(\chi_f^{0,1}\) (respectively \(\chi_f^{1,1}\)). Likewise one can define an early TCF, denoted \(\chi_f^{t,t-}\), characterizing the set of PI assignments that make the output value of \(f\) stabilize earlier than time \(t \geq 0\). Note that \(\chi_f^{t,t} \rightarrow \chi_f^{t,t+1}\) for \(t_1 \geq t_2\), and \(\chi_f^{t,t} = \neg \chi_f^{t,t-}\).

The circuit delay computation can therefore be formulated as searching the maximum \(D\) such that the formula

\[
\chi_f^{t,t} = 1 \quad \text{if} \quad \chi_f^{t,t+1} = 0 \quad \text{or} \quad \chi_f^{t,t-} = 0
\]

is satisfiable. (If Formula (1) is satisfiable, the circuit delay must be equal to or larger than \(D\) because there exists some PO whose value remains unknown before time \(D\). Otherwise, the circuit delay is strictly smaller than \(D\).) As to be discussed in Section 3, these TCFs of Formula (1) can be constructed recursively from POs to PIs of the circuit, and Formula (1) can be converted to CNF for SAT solving.

### 3. TCF CONSTRUCTION

In this section we consider TCF formulations without and with 0/1-specificity. Our formulations are then compared with prior methods [14], [6] and [10]. Finally, TCF equivalence reduction techniques are proposed.

#### 3.1 TCF without 0/1-Specificity

**3.1.1 Prior Formulation**

Prior work [14] reformulated the exact [2] and viable [7] sensitization criteria (with path tracing) for circuit delay computation (without path tracing) with the following TCFs

\[
\chi_f^{t,t} = \bigvee_{g_i \in FI(f)} \chi_{g_i}^{t,t-d_i} \land \bigwedge_{g_j \in FI(f)} (g_j = v_{n_j}) \lor (g_j = v_{n_j} \land \chi_{g_j}^{t,t-d_j} \lor (g_j = v_{n_j}))
\]

\[
\chi_f^{t,t} = \bigvee_{g_i \in FI(f)} \chi_{g_i}^{t,t-d_i} \land \bigwedge_{g_j \in FI(f)} (\chi_{g_j}^{t,t-d_j} \lor (g_j = v_{n_j}))
\]

respectively, where \(d_i\) is the pin-to-pin delay from \(g_i\) to \(f\) and \(v_{n_j}\) are the controlling and non-controlling values of \(g_j\). Equations (3) and (4) were considered in [14] as exact and approximative circuit delay computation, respectively.

The recursive definition of \(\chi_f^{t,t}\) naturally translates to a combinational circuit. For a \(k\)-input simple gate \(f\), Equations (3) and (4) result in \((k^2 + 13k + 2)\) and \((5k + 3)\) clauses with \((4k + 1)\) and \((k + 1)\) extra variables being introduced, respectively, by Tseitin’s circuit-to-CNF conversion [15]. The satisfiability of such a TCF can be difficult to solve especially when the corresponding circuit is large. (Note that the number of nodes in the circuit is bounded from above by the number of possible arrival times of all nodes.)

#### 3.1.2 Our Formulation

A close examination of Equations (3) and (4) reveals that they are essentially equivalent in circuit delay computation. In fact, as has been shown earlier in [11], Equation (4) yields exact (rather than approximative, as interpreted in [14]) analysis when path tracing is not performed.

\(^1\)Equation (3) looks different from the one in [14] as it was previously expressed by both exact and viable TCFs.
Building upon Equation (4), we propose a general and compact TCF formula for arbitrary complex gates as follows.

PROPOSITION 1. For a node \( f \) with a set \( C \) of controlling cubes, its TCF can be expressed as

\[
\chi_{f,t} = \bigwedge_{g_i \in F_L(f)} \chi_{g_i,t-d_i} \land \bigvee_{c \in C} (\chi_{g_i,t-d_i} \lor \neg \text{lit}(g_i)),
\]

where \( d_i \) is the pin-to-pin delay from \( g_i \) to \( f \) and \( \text{lit}(g_i) \) denotes the literal of \( g_i \).

PROOF. There are exactly two possible cases for the value of \( f \) being determined before time \( t \). First, the value of every \( g_i \in F_I(f) \) is determined before time \( t-d_i \). Second, every constituent input \( g_i \) of some controlling cube \( c \) is determined to its corresponding value \( \text{lit}(g_i) \in c \) before time \( t-d_i \). Since any of the above cases makes \( \chi_{f,t} \) false, the condition can be formally translated to

\[
\neg \chi_{f,t} = \bigwedge_{g_i \in F_I(f)} \neg \chi_{g_i,t-d_i} \lor \bigvee_{c \in C} (\neg \chi_{g_i,t-d_i} \land \text{lit}(g_i)),
\]

whose negation equals Equation (5).

Note that, for simple gates (with controlling values, in other words, with one-literal controlling cubes), Equation (5) reduces to Equation (4).

With the key observation that \( \chi_{f,t} \) is recursively defined in Equation (4) with the appearance only in the positive phase without any negation, implication suffices to express the TCF constraints. The advantage of using implication, instead of equation, is that we can apply Plaisted-Greenbaum encoding [12], instead of Tseitin encoding, in converting TCFs to CNF formulas. Specifically, Equation (5) with the equality sign \( \equiv \) being replaced by the implication sign \( \rightarrow \) can be directly translated into the CNF formula

\[
(\neg \chi_{f,t} \lor \bigvee_{g_i \in F_I(f)} \chi_{g_i,t-d_i}) \land (\neg \chi_{f,t} \lor \bigvee_{c \in C} (\chi_{g_i,t-d_i} \land \neg \text{lit}(g_i))),
\]

which consists of \( |C| + 1 \) clauses without introducing any extra variable. Hence, unlike prior methods, building TCF circuits is unnessecitated

Note that, in converting the entire recursive definition of \( \chi_{f,t} \), Tseitin encoding is still needed for parts of the original circuit that are relevant to the literals \( \text{lit}(g_i) \) in individual TCFs (since these literals may appear in both positive and negative phases). Nevertheless the conversion with Tseitin encoding is applied once on the original circuit and is shared by all individual TCFs.

3.2 TCF with 0/1-Specificity

3.2.1 Prior Formulation

Prior work [6] intended to improve [14] by exploiting early TCF to simplify TCF circuits. The following equations were proposed.

\[
\chi_{f,t}^{1.1} = \chi_{f=1,t} \lor \chi_{f=0,t} = (f \land \neg \chi_{f=1,t} \lor \neg f \land \neg \chi_{f=0,t})
\]

\[
\chi_{f=1,t} = \bigwedge_{g_i \in F_I(f)} \chi_{g_i,t-d_i} \land \bigvee_{c \in C} (\chi_{g_i,t-d_i} \land \neg \text{lit}(g_i))
\]

\[
\chi_{f=0,t} = \bigwedge_{g_i \in F_I(f)} \chi_{g_i,t-d_i} \land \bigvee_{c \in C} (\chi_{g_i,t-d_i} \lor \text{lit}(g_i))
\]

where \( d_i \) and \( d_f \) are the corresponding rising and falling pin-to-pin delays from \( g_i \) to \( f \), respectively. In the above expressions, TCF \( \chi_{f,t} \) is obtained from two subcases \( \chi_{f=1,t} \) and \( \chi_{f=0,t} \), where \( \chi_{f=1,t} \) is satisfiable if \( f \) stabilizes to value \( v \) no earlier than time \( t \). Note that \( \chi_{f=0,t} \neq \neg \chi_{f=1,t} \), but rather \( \chi_{f=0,t} = (f \land \neg v) \land \neg \chi_{f=1,t} \).

The advantages of separating \( \chi_{f=1,t} \) and \( \chi_{f=0,t} \) from \( \chi_{f,t} \) are two-fold: First, it allows distinction between rising and falling delays and thus permits more accurate timing analysis. Second, since Equation (8) in circuit representation consists of a single gate, no internal variable needs to be introduced in conversion to CNF. The resultant CNF formula is easier to solve.

The disadvantages, on the other hand, are also two-fold: First, such separation doubles the TCF formula size. Second, since the formulation works for simple gates only, timing analysis of circuits with complex gates is approximate. In fact, Equation (8) can be generalized for complex gates [10] with

\[
\chi_{f=1,t} = \bigwedge_{g_i \in F_I(f)} \chi_{g_i,v,t-d_{r_i}} \land \bigvee_{c \in C} (\chi_{g_i,v,t-d_{r_i}} \land \neg \text{lit}(g_i)),
\]

\[
\chi_{f=0,t} = \bigwedge_{g_i \in F_I(f)} \chi_{g_i,v,t-d_{f_i}} \lor \bigvee_{c \in C} (\chi_{g_i,v,t-d_{f_i}} \land \text{lit}(g_i)),
\]

where \( C_1 \) and \( C_0 \) are the sets of all prime implicants of \( f \) and \( \neg f \), respectively, and \( v = 0 \) if \( \text{lit}(g_i) = \neg g_i \) and \( v = 1 \) if \( \text{lit}(g_i) = g_i \). When translated to CNF, Equation (9) is more complicated than Equation (8) however. Note that Plaisted-Greenbaum encoding is not applicable here due to the negations in Equation (7).

3.2.2 Our Formulation

The aforementioned disadvantages can be overcome as follows.

PROPOSITION 2. Given a circuit, let \( f \) be a node with the set \( C_1 \) and \( C_0 \) of all prime implicants of \( f \) and \( \neg f \), respectively. Then 0/1-specified TCF can be expressed as

\[
\chi_{f=1,t} = f \land \bigwedge_{g_i \in F_I(f)} (\chi_{g_i,v,t-d_{r_i}} \land \neg \text{lit}(g_i)) \land \bigvee_{c \in C} (\chi_{g_i,v,t-d_{r_i}} \lor \text{lit}(g_i))
\]

\[
\chi_{f=0,t} = \neg f \land \bigwedge_{g_i \in F_I(f)} (\chi_{g_i,v,t-d_{f_i}} \lor \text{lit}(g_i)) \land \bigvee_{c \in C} (\chi_{g_i,v,t-d_{f_i}} \land \neg \text{lit}(g_i)),
\]

where \( v = 0 \) if \( \text{lit}(g_i) = \neg g_i \) and \( v = 1 \) if \( \text{lit}(g_i) = g_i \).

PROOF. If \( \chi_{f=1,t} \) is satisfied, it means that \( f \) values to true no earlier than time \( t \). That is, for every cube in \( C_1 \), it is either not satisfied, or satisfied with at least one controlling literal values to true no earlier than time \( t-d \). Similarly, one can prove the case of \( \chi_{f=0,t} \).

Since all the TCFs appear in \( \chi_{f,t} = \chi_{f=1,t} \lor \chi_{f=0,t} \) and in Equation (10) without any negation, again Plaisted-Greenbaum encoding applies for CNF conversion.

3.3 Comparison on TCF Formulas

Table 1 compares our formulations with those of [14], [6], and [10]. For a \( k \)-input simple gate, the number of extra variables and the number of clauses corresponding to the TCF

<table>
<thead>
<tr>
<th>Table 1: TCF Comparison</th>
<th>( \chi_{f=1,t} )</th>
<th>( \chi_{f=0,t} )</th>
<th>( \chi_{f,t} )</th>
<th>( \chi_{f=1,t} \lor \chi_{f=0,t} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \chi_{f=1,t} )</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>( \chi_{f=0,t} )</td>
<td>4 * 1</td>
<td>1 * 1</td>
<td>1 * 1</td>
<td>5 * 1</td>
</tr>
<tr>
<td>( \chi_{f,t} )</td>
<td>4 * 1</td>
<td>1 * 1</td>
<td>1 * 1</td>
<td>5 * 1</td>
</tr>
<tr>
<td>( \chi_{f=1,t} \lor \chi_{f=0,t} )</td>
<td>8 * 1</td>
<td>2 * 1</td>
<td>2 * 1</td>
<td>10 * 1</td>
</tr>
</tbody>
</table>

Table 1 compares our formulations with those of [14], [6], and [10]. For a \( k \)-input simple gate, the number of extra variables and the number of clauses corresponding to the TCF
3.4 TCF Equivalence Reduction

Given a circuit with a node \( f \), its TCFs \( \chi_f^A \) for all \( t \) can be partitioned into equivalence classes. This equivalence relation can be exploited to simplify the recursive TCF construction. In [10], TCF equivalence based on arrival-time information is introduced. Assume that the set \( A \) of all possible arrival times of node \( f \) is sorted in an ascending order as \( \{a_1, a_2, \ldots, a_m\} \) for \( a_{i-1} < a_i \). Then \( \chi_{f,a_i} = \chi_{f,a_{i-1}} \) if \( a_{i-1} < t \leq a_i \). That is, two temporal conditions \( t_1 \) and \( t_2 \) of \( f \) are equivalent if they have the same next larger or equal arrival time in \( A \).

For practical implementation, we propose a table look-up approach to TCF equivalence reduction with three improvements over prior works [10, 6]. First, for TCFs with 0/1-specificity, the set of arrival times of a node \( f \) is further distinguished into two sets \( A_1 \) and \( A_0 \) for those resulting in \( f = 1 \) and \( f = 0 \), respectively. This distinction reduces the number of arrival times and thus TCF equivalence classes.

Second, under boundary conditions, a TCF is substituted with a constant 0 or 1 for further reduction (constant 1 is not applicable for prior works). Specifically, Figure 1 depicts the equivalence intervals of the TCFs of node \( f \) with extended boundary conditions. If \( t \) is larger than the maximum arrival time \( a_m \) of a node \( f \), then \( \chi_f^A \) is unsatisfiable since \( f \) always stabilizes before \( t \). In this case, \( \chi_f^A, \chi_f^{A_1} \), and \( \chi_f^{A_0} \) are all equal Boolean constant 0. On the contrary, if \( t \) is no larger than the minimum arrival time \( a_1 \), then \( \chi_f^A \) is a tautology (but \( \chi_f^{A_1} \) and \( \chi_f^{A_0} \) are not necessarily tautologies). That is, \( \chi_f^A \) equals constant 1, and furthermore \( \chi_f^{A_1} \) and \( \chi_f^{A_0} \) can be simplified to \( f \oplus \neg v \) by \( \chi_f^{A_1} = (f \oplus \neg v) \land \chi_f^A \). Observe that, in Equation (10), \( \chi_{g_i,t-d_i} \) and \( \neg \lit(g_i) \) are always present together in a clause with \( \neg \lit(g_i) = g_i \oplus v \). When \( \chi_{g_i,t-d_i} = 1 \), since \( \chi_{g_i,v,t-d_i} = g_i \oplus v \), this clause must be satisfied due to \( \chi_{g_i,v,t-d_i} \lor \neg \lit(g_i) = ((g_i \oplus v) \lor (g_i \oplus t)) = 1 \). Therefore, whenever \( \chi_{g_i,t-d_i} = 1 \), substituting constant 1 for \( \chi_{g_i,v,t-d_i} \) and \( \chi_{g_i=0,t-d_i} \) is safe without altering the satisfiability of \( \chi_f^A \). As a result, our TCFs without and with 0/1-specificity can be simplified with such constant substitution.

Third, our TCF equivalence reduction is applied to all nodes including PIs and POs. Because of the aforementioned first improvement, any TCF of a PI is either constant 1 or constant 0 because any PI has only one arrival time. On the other hand, since the arrival times at POs are the only candidate circuit delays, this information is exploited to save unnecessary checking. More precisely, only PO arrival times are checked for circuit delay by Formula (1): once some candidate delay is falsified, this delay and other larger delays are removed from the arrival-time lists of all POs. For example, assume two POs \( p_1 \) and \( p_2 \) have arrival-time lists \{4, 5, 7\} and \{6, 7\}, respectively. If \( \chi_{p_1,6} \lor \chi_{p_2,7} \) is unsatisfiable, we remove 7 from the two lists. Then we check \( \chi_{p_1,8} \lor \chi_{p_2,6} = (0 \lor \chi_{p_2,6}) \). Note that this removal is crucial. If 7 were not removed from the list of \( p_1 \), then \( \chi_{p_1,6} \) would equal \( \chi_{p_1,7} \) instead of 0 and \( \chi_{p_1,7} \) would be built again.

4. ALGORITHMS

The overall algorithms of circuit delay computation and critical region identification are presented in this section.

4.1 Delay Computation

Figure 2 sketches a procedure for delay computation without rise/fall time separation. It can be easily extended under a similar framework to the computation with rise/fall time separation, which is omitted for brevity. To avoid confusion between a TCF and its output variable, in the pseudo code \( \chi^f \) represents the output variable of TCF \( \chi^f \).

While the code is self-explanatory, it should be noted that different delay search strategies can be applied depending on how functions GetDelayList, GetNextDelay, and UpdateDelayList are implemented. For instance, linear or binary search can be deployed with or without adaptive step-size adjustment. Counterintuitively empirical experience suggests that linear search in general works much better than binary search. Investigation reveals that, although linear search requires more SAT solving iterations than binary search, it allows the second improvement technique of Section 3.4 more applicable and thus making the CNF formula at each iteration easier to solve.

Upon termination (line 14 of ComputeDelay), Formula (1) must be satisfiable for \( D = lowerDelay \). That is, there exists a PI assignment to sensitize some true path achieving this delay value. By applying the assignment values to PIs, we can simulate and trace one true critical path based on the exact sensitization criterion [2].

4.2 Critical Region Identification

Our delay computation algorithm can be applied to identify true timing critical regions for delay optimization. Given a target required time of a circuit, topological timing critical regions (with small slack) can be identified by conventional STA analysis. Topological timing critical regions over-approximate functional true critical regions. The approximation can be very crude, and in this case many false critical gates and paths can be trimmed away. The true critical regions can be pinpointed by removing false arrival times with the third improvement technique of Section 3.4. Note that the TCFs of non-critical gates equal constant 0 due to the boundary condition \( t > a_m \) of TCF equivalence reduction. Effectively the computation considers only the timing critical sub-circuit, which can be much smaller than the entire circuit.

5. EXPERIMENTAL RESULTS

Our methods, named “Swift” for Equation (5) and “Swift-0/1” for Equation (10), were implemented in the C++ language using MiniSat version 2.2.0 [5] as the underlying SAT solver. All experiments were conducted on a Linux machine with a Xeon 3.4 GHz CPU and 32 GB RAM. Large ISCAS, ITC, and other industrial benchmark circuits were selected for experiments. For the sake of comparison with prior work [6], which handles only simple gate types, all circuits are technology mapped using only buffers, inverters, AND-gates, OR-gates, NAND-gates, and NOR-gates. It should be noted, however, that our computation is not restricted to these simple gate types and can be generally applicable to general complex gates.
ComputeDelay(C) //compute maximum true-path delay of circuit C
begin
1. L := GetDelayList(C); 
2. (lowerDelay, upperDelay) := MinMaxTopologicalDelay(C); 
3. do
4. D := GetNextDelay(L); 
5. for every PO p
6. LowerDelay := 
7. if LowerDelay < 0
8. if IsSat(Φ)
9. lowerDelay := D;
10. else
11. upperDelay := D;
12. UpdateDelayList(C, L, lowerDelay, upperDelay);
13. while L non-empty;
14. return lowerDelay and its corresponding true path;
end

BuildTcf(f[t,i]) := derive χf,t in CNF
begin
1. t := GetNextLargerOrEqualArrivalTime(f[t]);
2. if χf,t has been built
3. return 1;
4. if t > f.ao //largest arrival time of f
5. return (¬χf,t);
6. if t ≤ f.ai //smallest arrival time of f
7. return (χf,t);
8. if f has only one fanin g
9. return BuildTcf(gi, t − di) with χg,t − di, replaced by χf,t;
10. Φ := (¬χf,t) ∨ ∨ g∈Fl(t) (χg,t − di);
11. for each controlling cube c of f
12. Φ := Φ ∧ (¬χf,t) ∨ ∨ g∈Fl(t) (χg,t − di) ∨ ¬ldt(gi));
13. for each g, in Fl(f)
14. Φ := Φ ∧ BuildTcf(gi, t − di);
15. if gi’s circuit CNF has not been built
16. Φ := Φ ∧ BuildCktCnf(gi);
17. return Φ;
end

Figure 2: Algorithm: Delay Computation

5.1 Delay Computation

For circuit delay computation, prior method [6], using Equations (7) and (8), was re-implemented under the same setting (including the same linear delay search strategy in a descending order) as ours for fair comparison. (We did not compare with [14] and [10] as they are not as efficient as [6].) The comparison was performed under four delay models: the unit gate delay model, fanout delay model (by calculating a gate delay as 1 + 0.2 × fanout number), TSMC 0.18µm library model with combined rise/fall time (by calculating a gate delay as max{rise delay, fall delay}), and TSMC 0.18µm library model with separate rise/fall time. Table 2 shows the experimental results under the four delay models. Column 2 shows the gate count; Column 3 shows the longest topological delay and actual true-path delay; Column 4 shows the number of SAT solving iterations needed to identify the true-path delay; Columns 5 and 8 (respectively Columns 6 and 9) show the total number of variables excluding those in original circuits (respectively clauses) involved in the CNF formulas of all SAT solving iterations; Columns 7 and 10 show the total SAT solving time in seconds. (The reported runtime excludes preprocessing time as both our and their methods were processed in a similar way. The prior method may take slightly longer time because of converting circuits to CNF formulas.) Note that SWIFT is only applicable to the first three timing models (without separating rise and fall delays) because its TCF formulation has no 0/1-specificity, and thus SWIFT-0/1 is applied in the fourth timing model with separate rise and fall delays. The results suggest that SWIFT performs robustly and efficiently (with all runtimes within 0.06 seconds) under various delay models while the performance [6] is unpredictable (as exemplified by circuit leon3mp, which is solved in 12229.60 seconds under the unit delay model and 2.66 seconds under the fanout delay model) and is not as efficient. The efficiency of SWIFT stems from several factors. First, the numbers of variables and clauses encountered in SWIFT are about half of those in [6]. Second, replacing equivalence-based with implication-based TCF construction makes SAT solving easier. Third, the TCF without 0/1-specificity is more compact than that with 0/1-specificity. Fourth, perhaps most important, SWIFT yields more constant propagations due to equivalent TCF reduction. On the other hand, the results also suggest that SWIFT-0/1 outperforms [6] (by a factor of 3.09 measured by geometric mean). It is interesting to note that circuit netcard took SWIFT-0/1 long time to solve comparable to that of [6]. (Although the timing improvement is not remarkable in this case, SWIFT-0/1 offers the generality to handle complex gates, which is not available in [6].) Compared to SWIFT, SWIFT-0/1 does not enjoy as much variable and clause reductions, and constant propagations. The formulations of SWIFT-0/1 and [6] have their own strengths. For SWIFT-0/1, there are fewer variables and clauses, and constant propagation in equivalent TCF reduction is possible. For [6], because χf,t in Equation (8) depends only on its fanin TCFs but not on other variables, it makes CNF formulas simple. However the formulation is only applicable to simple gates. Table 2 also reveals that topological delay may be far pessimistic compared to true circuit delay, e.g., circuits b05 and b19 under the unit and fanout delay models. It suggests the importance of accurate functional timing analysis and its application on identifying true critical region for timing optimization.

5.2 Critical Region Identification

Table 3 evaluates the applicability of SWIFT on identifying timing critical regions under the unit delay model. For a circuit, its true delay is set to be the required time at its POs, and the gates and paths with non-positive slack values are declared critical. Column 2 shows the total number of gates of a circuit; Columns 3 and 4 (Columns 5 and 6) show the numbers of critical gates and paths, respectively, with respect to topological arrival times (functional true arrival times); Column 7 shows the runtime in identifying true critical regions. The results suggest that SWIFT effectively removed spurious critical gates and paths. As a matter of fact, true critical regions can be much smaller than topological critical regions. By taking circuit b17 as an example, SWIFT detected, in 0.61 seconds (the time spent in SAT solving), that only 79 out of its 1637 topological critical gates are true critical gates, and at least 5585733 out of its 5585965 topological critical paths are false critical paths. Pinpointing true critical regions efficiently can be beneficial to timing optimization.

6. CONCLUSIONS AND FUTURE WORK
This paper has shown that functional timing analysis can be made fast and general compared with state-of-the-art methods. Based on implication relation and other technical improvements, compact CNF encoding for TCFs without and with 0/1-specificity has been devised. Thereby the power of modern SAT solvers can be fully utilized. Experiments on large designs have demonstrated promising results on delay computation and critical region identification.

Acknowledgments

The authors acknowledge Yuji Kukimoto for referring us to [10]. This work was supported in the National Science Council under grants NSC 99-2221-E-002-214-MY3, 99-2923-E-002-005-MY3, and 100-2923-E-002-008.

7. REFERENCES