Functional Timing Analysis Made Fast and General

Yi-Ting Chung[1] and Jie-Hong R. Jiang[1,2]
[1]Graduate Institute of Electronics Engineering; [2]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.

2. PRELIMINARIES

A *literal* is a Boolean variable or its negation. A *clause* (*cube*) is a disjunction (conjunction) of literals. A propositional formula is in *conjunctive normal form* (CNF) if it is written as a conjunction of clauses. The satisfiability (SAT) problem asks whether there exists a satisfying assignment to the set of variables that makes a CNF formula true. The reader is referred to [9, 8, 5] for modern SAT solving techniques, and to [15, 12] for circuit-to-CNF conversion.

2.1 Circuit Model

A (combinational) circuit $C(N, E)$ consists of nodes (or gates) N, (directed) edges $E \subseteq N \times N$. Two disjoint subsets of N are distinguished as primary inputs (PIs) and primary outputs (POs). Each node is associated with two attributes: function and delay. We assume the function can be arbitrary, from simple gate types, such as buffer, inverter, NAND, NOR, etc., to complex function units, such as XOR, multiplexer, AOI, etc. In the sequel, we sometimes do not distinguish a node from its function and its output variable when it is clear from the context. We assume the gate delay can vary from pin to pin and vary between rise and fall time. Without loss of generality, interconnect delays are assumed to be integrated into the gate delays under this timing model.

For a node f in a circuit, we let $FI(f)$ and $FO(f)$ denote the fanin and fanout nodes of f, respectively. For $g \in FI(f)$, we say g is of *controlling value*, denoted $v_c \in \mathbb{B} = \{0, 1\}$ (respectively, *non-controlling value*, denoted $v_n \in \mathbb{B}$) of f if the output value of f can (respectively, cannot) be completely determined by g with v_c (respectively, v_n) 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 $ab \vee c$ are $\{ab, c, \neg a\neg c, \neg b\neg c\}$, where cubes ab and c make $f = 1$ and cubes $\neg a\neg c$ and $\neg b\neg c$ make $f = 0$. In addition, $\neg a$ is a controlling literal of cube $\neg a\neg c$.

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 stablize 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 stablize *no earlier* than time $t \geq 0$ is characterized by a (no-early) TCF, denoted $\chi^{f,t}$. In other words, a PI assignment satisfying $\chi^{f,t}$ makes the output value of f remains unknown (under the floating-mode assumption) until time t. When the stablization value of f is specific to value 0 (respectively 1), the corresponding 0/1-specified TCF is denoted as $\chi^{f=0,t}$ (respectively $\chi^{f=1,t}$). Likewise one can define an early TCF, denoted $\chi^{f,t-}$, characterizing the set of PI assignments that make the output value of f stablize *earlier* than time $t \geq 0$. Note that $\chi^{f,t_1} \rightarrow \chi^{f,t_2}$ for $t_1 \geq t_2$, and $\chi^{f,t} = \neg\chi^{f,t-}$.

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

$$\bigvee_{p \in PO} \chi^{p,D} \tag{1}$$

$$= \bigvee_{p \in PO} (\chi^{p=1,D} \vee \chi^{p=0,D}) \tag{2}$$

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

$$\chi^{f,t} = \bigvee_{g_i \in FI(f)} \chi^{g_i,t-d_i} \wedge \{ \bigwedge_{g_j \in FI(f)} (g_j = v_{n_j}) \vee (g_i = v_{c_i}) \wedge \bigwedge_{g_j \in FI(f)} (\chi^{g_j,t-d_j} \vee (g_j = v_{n_j}))\}, \tag{3}$$

$$\chi^{f,t} = \bigvee_{g_i \in FI(f)} \chi^{g_i,t-d_i} \wedge \bigwedge_{g_i \in FI(f)} (\chi^{g_i,t-d_i} \vee (g_i = v_{n_i})) \tag{4}$$

respectively, where d_i is the pin-to-pin delay from g_i to f and v_{c_i} and v_{n_i} are the controlling and non-controlling values of g_i.[1] Equations (3) and (4) were considered in [14] as exact and approximative circuit delay computation, respectively.

The recursive definition of $\chi^{f,t}$ 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*

$$\chi^{f,t} = \bigvee_{g_i \in FI(f)} \chi^{g_i,t-d_i} \wedge \bigwedge_{c\in C} \bigvee_{lit(g_i)\in c} (\chi^{g_i,t-d_i} \vee \neg lit(g_i)), \quad (5)$$

where d_i is the pin-to-pin delay from g_i to f and $lit(g_i)$ 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 $g_i \in FI(f)$ is determined before time $(t - d_i)$. Second, every constituent input g_i of some controlling cube c is determined to its corresponding value $lit(g_i) \in c$ before time $(t - d_i)$. Since any of the above cases makes $\chi^{f,t}$ false, the condition can be formally translated to

$$\neg\chi^{f,t} = \bigwedge_{g_i\in FI(f)} \neg\chi^{g_i,t-d_i} \vee \bigvee_{c\in C} \bigwedge_{lit(g_i)\in c} (\neg\chi^{g_i,t-d_i} \wedge lit(g_i))),$$

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 "=" being replaced by the implication sign "→" can be directly translated into the CNF formula

$$(\neg\chi^{f,t} \vee \bigvee_{g_i\in FI(f)} \chi^{g_i,t-d_i}) \bigwedge_{c\in C} (\neg\chi^{f,t} \vee \bigvee_{lit(g_i)\in c} (\chi^{g_i,t-d_i} \vee \neg lit(g_i))), \quad (6)$$

which consists of $|C| + 1$ clauses without introducing any extra variable. Hence, unlike prior methods, building TCF circuits is unnecessitated.

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 $lit(g_i)$ 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.

$$\begin{aligned} \chi^{f,t} &= \chi^{f=1,t} \vee \chi^{f=0,t} \\ &= (f \wedge \neg\chi^{f=1,t-}) \vee (\neg f \wedge \neg\chi^{f=0,t-}) \quad (7) \end{aligned}$$

$$\chi^{f=1,t-} = \begin{cases} \bigwedge_{g_i\in FI(f)} \chi^{g_i=1,(t-d_{r_i})-}, & \text{for AND-gate } f \\ \bigvee_{g_i\in FI(f)} \chi^{g_i=1,(t-d_{r_i})-}, & \text{for OR-gate } f \end{cases}$$

$$\chi^{f=0,t-} = \begin{cases} \bigvee_{g_i\in FI(f)} \chi^{g_i=0,(t-d_{f_i})-}, & \text{for AND-gate } f \\ \bigwedge_{g_i\in FI(f)} \chi^{g_i=0,(t-d_{f_i})-}, & \text{for OR-gate } f \end{cases} \quad (8)$$

where d_{r_i} and d_{f_i} 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 $\chi^{f=v,t}$ is satisfiable if f stablizes to value v no earlier than time t. Note that $\chi^{f=v,t} \neq \neg\chi^{f=v,t-}$, but rather $\chi^{f=v,t} = (f \oplus \neg v) \wedge \neg\chi^{f=v,t-}$.

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 approximative. In fact, Equation (8) can be generalized for complex gates [10] with

$$\begin{aligned} \chi^{f=1,t-} &= \bigvee_{c\in C_1} \bigwedge_{lit(g_i)\in c} \chi^{g_i=v,(t-d_{r_i})-} \text{ and} \\ \chi^{f=0,t-} &= \bigvee_{c\in C_0} \bigwedge_{lit(g_i)\in c} \chi^{g_i=v,(t-d_{f_i})-}, \quad (9) \end{aligned}$$

where C_1 and C_0 are the sets of all prime implicants of f and $\neg f$, respectively, and $v = 0$ if $lit(g_i) = \neg g_i$ and $v = 1$ if $lit(g_i) = g_i$. 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*

$$\begin{aligned} \chi^{f=1,t} &= f \wedge \bigwedge_{c\in C_1} \bigvee_{lit(g_i)\in c} (\chi^{g_i=v,t-d_{r_i}} \vee \neg lit(g_i)) \text{ and} \\ \chi^{f=0,t} &= \neg f \wedge \bigwedge_{c\in C_0} \bigvee_{lit(g_i)\in c} (\chi^{g_i=v,t-d_{f_i}} \vee \neg lit(g_i)), \quad (10) \end{aligned}$$

where $v = 0$ if $lit(g_i) = \neg g_i$ and $v = 1$ if $lit(g_i) = g_i$.

PROOF. If $\chi^{f=1,t}$ is satisfied, it means that f valuates 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 valuates 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 $\chi^{f,t} = \chi^{f=1,t} \vee \chi^{f=0,t}$ and in Equation (10) without any negation, again Plaisted-Greenbaum encoding applies for CNF conversion.

3.3 Comparison on TCF Formulas

Table 1: TCF Comparison

	TCF			PO			Generality	
	Eq	#Vr	#Cl	Eq	#Vr	#Cl	CG	RF
[14]	(3)	$k+1$	$5k+3$	(1)	0	1	No	No
	(4)	$4k+1$	$k^2+13k+2$	(1)	0	1	No	No
[6]	(8)	0	$2k+2$	(7)	$2m$	$9m$	No	Yes
[10]	(9)	$k+1$	$4k+4$	(7)	$2m$	$9m$	Yes	Yes
Our	(5)	0	$k+1$	(1)	0	1	Yes	No
	(10)	0	$k+3$	(2)	0	1	Yes	Yes

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

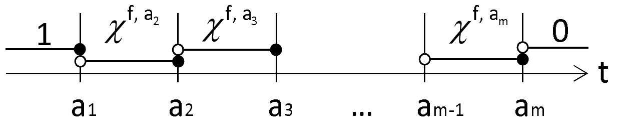

Figure 1: Equivalence intervals of $\chi^{f,t}$.

equations in Column 2 are shown in Columns 3 and 4, respectively. For a circuit with m POs, the number of extra variables and the number of clauses corresponding to the PO equations in Column 5 are shown in Columns 6 and 7, respectively. The generality for each formulation in supporting complex gate types and supporting rise/fall delays are summarized in Columns 8 and 9, respectively.

3.4 TCF Equivalence Reduction

Given a circuit with a node f, its TCFs $\chi^{f,t}$ 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 are sorted in an ascending order as $\{a_1, a_2, \ldots, a_m\}$ for $a_{i-1} < a_i$. Then $\chi^{f=v,t-} = \chi^{f=v,a_i-}$ 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 lookup 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 $\chi^{f,t}$ is unsatisfiable since f always stabilizes before t. In this case, $\chi^{f,t}$, $\chi^{f=1,t}$ and $\chi^{f=0,t}$ all equal Boolean constant 0. On the contrary, if t is no larger than the minimum arrival time a_1, then $\chi^{f,t}$ is a tautology (but $\chi^{f=1,t}$ and $\chi^{f=0,t}$ are not necessarily tautologies). That is, $\chi^{f,t}$ equals constant 1, and furthermore $\chi^{f=v,t}$ can be simplified to $f \oplus \neg v$ by $\chi^{f=v,t} = (f \oplus \neg v) \wedge \chi^{f,t}$. Observe that, in Equation (10), $\chi^{g_i=v,t-d_i}$ 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 $\chi^{g_i=v,t-d_i} = g_i \oplus \neg v$, this clause must be satisfied due to $(\chi^{g_i=v,t-d_i} \vee \neg lit(g_i)) = ((g_i \oplus \neg v) \vee (g_i \oplus v)) = 1$. Therefore, whenever $\chi^{g_i,t-d_i} = 1$, substituting constant 1 for $\chi^{g_i=1,t-d_i}$ and $\chi^{g_i=0,t-d_i}$ is safe without altering the satisfiability of $\chi^{f,t}$. 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 $(\chi^{p_1,7} \vee \chi^{p_2,7})$ is unsatisfiable, we remove 7 from the two lists. Then we check $(\chi^{p_1,6} \vee \chi^{p_2,6}) = (0 \vee \chi^{p_2,6})$. 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 $x^{f,t}$ represents the output variable of TCF $\chi^{f,t}$.

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 slacks) 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 2.20 [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
    $L := GetDelayList(C)$;
    (lowerDelay, upperDelay) := $MinMaxTopologicalDelay(C)$;
    do
        $D := GetNextDelay(L)$;
        $\Phi := (\bigvee_{p \in PO} x^{p,D})$;
        for every PO $p$
            $\Phi := \Phi \wedge BuildTcf(p, D)$;
        if $IsSat(\Phi)$
            lowerDelay := $D$;
        else
            upperDelay := $D$;
        $UpdateDelayList(C, L,$ lowerDelay, upperDelay$)$;
    while $L$ non-empty;
    return lowerDelay and its corresponding true path;
end

*BuildTcf(f,t)* //derive $\chi^{f,t}$ in CNF
begin
    $t := GetNextLargerOrEqualArrivalTime(f)$;
    if $\chi^{f,t}$ has been built
        return 1;
    if $t > f.a_m$ //largest arrival time of $f$
        return $(\neg x^{f,t})$;
    if $t \leq f.a_1$ //smallest arrival time of $f$
        return $(x^{f,t})$;
    if $f$ has only one fanin $g_i$
        return $BuildTcf(g_i, t - d_i)$ with $x^{g_i, t-d_i}$ replaced by $x^{f,t}$;
    $\Phi := (\neg x^{f,t} \vee \bigvee_{g_i \in FI(f)} x^{g_i, t-d_i})$;
    for each controlling cube $c$ of $f$
        $\Phi := \Phi \wedge (\neg x^{f,t} \vee \bigvee_{lit(g_i) \in c} (x^{g_i, t-d_i} \vee \neg lit(g_i)))$;
    for each $g_i \in FI(f)$
        $\Phi := \Phi \wedge BuildTcf(g_i, t - d_i)$;
        if $g_i$'s circuit CNF has not been built
            $\Phi := \Phi \wedge BuildCktCnf(g_i)$;
    return $\Phi$;
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 $1 + 0.2 \times$ fanout number), TSMC $0.18\mu m$ library model with combined rise/fall time (by calculating a gate delay as max{rise delay, fall delay}, and TSMC $0.18\mu m$ 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 prior and our methods were preprocessed 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 3.06 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 importantly, SWIFT yields more constant propagations due to equivalent TCF reduction.

Table 3: Critical Region Identification

Circuit	#G	Topological		Functional		Time
		#G	#Path	#G	#Path	(s)
b05	1022	322	7435427	186	8669	0.04
b17	33741	1637	5585965	79	232	0.61
b18	117941	1101	77585298	465	20839024	18.67
c3540	1741	270	1054	91	100	0.02
c5315	25585	213	832	76	60	0.02
c7552	3827	304	97	61	8	0.01
i10	2724	452	127483	338	3071	0.08
s15850	11067	408	73984	389	22016	0.16
s38417	2608	230	476	112	10	0.06

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 $\chi^{f=v,t}$ 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

Table 2: Circuit Delay Computation

Unit Delay

Circuit	#Gate	Delay	#SAT	[6]			Swift		
				#Var	#Clause	Time (s)	#Var	#Clause	Time (s)
b05	1022	54→42	8	15672	51104	0.03	7786	25030	0.01
b18	117941	164→159	5	18746	58335	1.72	9221	28300	0.29
b19	237959	168→158	7	84174	262549	11.50	41597	128850	0.76
c6288	2480	124→123	3	4248	12747	0.19	2118	6324	0.08
leon2	1119384	42→42	1	514	1703	0.21	250	829	< 0.01
leon3	1272597	44→44	1	354	1171	0.06	173	560	< 0.01
leon3mp	824294	40→38	3	427522	1387725	12229.60	155786	519905	0.79
netcard	983683	29→29	1	144	503	0.09	70	226	< 0.01
ray	235526	178→178	1	10338	35173	2.01	5051	17205	0.08
s35932	19876	29→26	4	100608	301828	6.95	49152	138244	0.06
uoft_raytracer	218671	178→178	1	11476	39015	1.08	5618	19109	0.02

Fanout Delay

Circuit	#Gate	Delay	#SAT	[6]			Swift		
				#Var	#Clause	Time (s)	#Var	#Clause	Time (s)
b05	1022	80.6→64.0	44	291364	945194	0.54	145404	469541	0.10
b18	117941	242.8→238.0	12	20140	62692	1.23	9861	30677	0.15
b19	237959	244.4→234.4	25	528358	1652131	77.60	262406	820125	3.06
c6288	2480	176.4→174.8	3	3222	9669	0.12	1606	4798	0.03
leon2	1119384	2070.0→2070.0	1	41544	149437	241.61	14628	55764	0.03
leon3	1272597	6854.4→6854.4	1	198674	599655	1172.05	66569	201522	0.11
leon3mp	824294	3093.2→3093.2	1	2224	7419	2.66	744	2604	< 0.01
netcard	983683	16390.2→16390.2	1	393228	1179685	0.76	131078	393231	0.27
ray	235526	383.6→383.6	1	796	2561	0.13	334	1087	0.01
s35932	19876	42.8→39.0	4	86784	260356	8.92	42240	122692	0.12
uoft_raytracer	218671	383.6→383.6	1	796	2561	0.06	334	1087	0.01

TSMC 0.18μm Cell Library with Combined Rise/Fall Time

Circuit	#Gate	Delay	#SAT	[6]			Swift		
				#Var	#Clause	Time (s)	#Var	#Clause	Time (s)
b05	1022	5.67→4.45	62	484714	1571927	0.89	241959	782062	0.13
b18	117941	13.49→13.43	3	2326	7415	0.16	1083	3457	0.02
b19	237959	14.09→13.80	15	114446	352033	1.53	56743	174462	0.25
c6288	2480	13.95→13.79	5	6916	20753	2.12	3450	10315	0.05
leon2	1119384	13.16→13.16	1	9356	28285	38.36	3142	9532	0.10
leon3	1272597	14.28→14.16	8	62946	190674	169.67	24924	75932	0.12
leon3mp	824294	14.58→14.27	17	169690	511361	876.46	57133	172517	0.19
netcard	983683	8.61→8.42	3	13180	39583	88.08	4404	13227	0.10
ray	235526	31.84→31.75	5	10310	34683	0.82	4999	16866	0.02
s35932	19876	2.83→2.64	5	85888	257669	6.06	41760	121125	0.05
uoft_raytracer	218671	31.98→31.98	1	804	2711	0.11	386	1306	0.02

TSMC 0.18μm Cell Library with Separate Rise/Fall Time

Circuit	#Gate	Delay	#SAT	[6]			Swift		
				#Var	#Clause	Time (s)	#Var	#Clause	Time (s)
b05	1022	4.67→3.52	59	433885	1407065	0.83	433148	1132193	0.72
b18	117941	11.08→10.98	7	9278	28949	0.58	9134	23404	0.48
b19	237959	11.67→11.42	14	135078	415139	1.24	134374	340148	0.92
c6288	2480	10.13→10.02	5	4068	12206	0.59	4060	10126	0.79
leon2	1119384	11.07→11.07	1	4678	14143	35.55	3142	7987	0.97
leon3	1272597	12.81→12.68	9	26375	79963	130.69	18137	46295	3.07
leon3mp	824294	12.39→12.03	4	95668	324471	603.09	64020	197810	50.22
netcard	983683	7.67→6.87	27	2915186	9076031	92964.40	2334866	6173237	90456.2
ray	235526	26.77→26.43	18	55045	183570	0.58	54186	140414	0.37
s35932	19876	2.45→2.35	5	40768	122308	2.91	39616	98148	1.67
uoft_raytracer	218671	26.40→25.82	31	213908	743738	7.02	213179	567656	2.11

This paper has shown that functional timing analysis can be made fast and general compared with sate-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 part by 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

[1] P. Ashar and S. Malik. Functional timing analysis using ATPG. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 14(8): 1025-1030, Aug. 1995.

[2] H.-C. Chen and D. Du. Path sensitization in critical path problem. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 12(2): 196-207, Feb. 1993.

[3] O. Coudert. An efficient algorithm to verify generalized false paths. In *Proc. Design Automation Conf.*, 2010.

[4] S. Devadas, K. Keutzer, and S. Malik. Computation of floating mode delay in combinational circuits: Theory and algorithms. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 12(12): 1913-1923, Dec. 1993.

[5] N. Eén and N. Sörensson. An extensible SAT-solver. In *Proc. SAT*, pp. 502-518, 2003.

[6] Y.-M. Kuo, Y.-L. Chang, and S.-C. Chang. Efficient Boolean characteristic function for timied automatic test pattern generation. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 28(3): 417-425, March 2009.

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

[8] M. Moskewicz, C. Madigan, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proc. DAC*, pp. 530-535, 2001.

[9] J. Marques-Silva and K. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. on Computers*, vol. 48, no. 5, pp. 506-521, May 1999.

[10] P. McGeer, A. Saldanha, R. Brayton, and A. Sangiovanni-Vincentelli. Delay models and exact timing analysis. In *Logic Synthesis and Optimization*, Kluwer Academic Publishers, pp. 167-189, 1993.

[11] P. C. McGeer, A. Saldanha, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vicentelli. Timing analysis and delay-fault test generation using path-recursive functions. In *Proc. Int. Conf. on Computer-Aided Design*, pages 180-183, 1991.

[12] D. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *J. Symbolic Computation*, 2:293-304, 1986.

[13] S. Roy, P. P. Chakrabarti, and P. Dasgupta. Event propagation for accurate circuit delay calculation using SAT. *ACM Trans. Design Autom. Electron. Syst.*, 12(3), Aug. 2007.

[14] L. Silva, J. Marques-Silva, L. Silveira, and K. Sakallah. Satisfiability models and algorithms for circuit delay computation. *ACM Trans. on Design Automation of Electronic Systems*, 7(1): 137-158, Jan. 2002.

[15] G. Tseitin. On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic*, pp. 466-483, 1970.