Timing Analysis Using Propositional Satisfiability

Luís Guerra e Silva, João P. Marques Silva, Luís Miguel Silveira and Karem A. Sakallah*

Cadence European Laboratories/INESC
Instituto Superior Técnico
R. Alves Redol, 9
1000 Lisboa, Portugal

*Electrical Engineering and Computer Science Dept.
Advanced Computer Architecture Lab.
University of Michigan
Ann Arbor, MI 48109-2122

Abstract

The existence of false paths represents a significant and computationally complex problem in the timing analysis of combinational and sequential circuits. In this paper we describe Propositional Satisfiability based algorithms for timing analysis, which introduce significant performance improvements over existing procedures. In particular, we address the problems of circuit delay computation and path delay validation, describing algorithms and providing experimental results for both problems.

1 Introduction

Recent years have seen an ever increasing need for more accurate delay estimation methodologies in digital circuits, in particular due to the decisive role that delay estimation plays in determining limiting operating clock frequencies. A key problem associated with circuit delay estimation is the existence of false paths, which cause straightforward and efficient topological path analysis procedures to yield potentially conservative circuit delay estimates. In contrast with topological delay estimation, solving the false path problem is computationally hard, being an NP-complete problem [7]. Research work on false paths has been extensive and several promising modeling and algorithmic approaches have been proposed [1, 2, 4, 5, 7-9, 10, 12].

The purpose of this paper is to propose improvements to Propositional Satisfiability based algorithms for Timing Analysis. In particular we address the problems of circuit delay computation and path delay validation. We identify drawbacks in existing algorithms and propose solutions to these drawbacks. Comprehensive experimental evaluation of the proposed algorithms indicates significant performance gains over existing a procedures.

The paper is organized as follows. We start by introducing a few definitions related with combinational circuits and the false path problem and with the propositional satisfiability problem. In Section 3 we present the SAT-based model for path sensitization and illustrate the application of the model on a simple example. Afterwards, we describe algorithms for circuit delay computation and path delay validation. Section 5 includes preliminary experimental results for both algorithms. Finally, we conclude the paper with ideas and directions for future research work on timing analysis of combinational circuits.

2 Definitions

In the following we shall assume a combinational circuit *C*, with *PI* primary inputs, *PO* primary outputs, composed of simple gates (AND, NAND, OR, NOR, NOT), where for a circuit node *f*, *c*(*f*) denotes the controlling logic value of *f* and *nc*(*f*) denotes the non-controlling logic value of *f*. For each circuit node *f*, *FI*(*f*) denotes the fanin nodes of *f* and *FO*(*f*) denotes the fanout nodes of *f*. The delay between the fanin node *g* of a circuit node *f* and *f* is denoted by *d*(*g*,*f*). A *complete path* (or simply a path) in a circuit is a sequence of nodes connecting a primary input to a primary output. A *partial path* denotes a connected sequence of nodes within a path.

With respect to Propositional Satisfiability (SAT), the following definitions apply. A conjunctive normal form (CNF) formula φ on n binary variables $x_1, \ldots, x_n$ is the conjunction (AND) of m clauses $\omega_1, \ldots, \omega_m$ each of which is the disjunction (OR) of one or more literals, where a literal is the occurrence of a variable or its complement. A CNF formula φ denotes a unique n-variable Boolean function $f(x_1, \ldots, x_n)$ and each of its clauses corresponds to an implicate of f. The satisfiability problem is concerned with finding an assignment to the arguments of $f(x_1, \ldots, x_n)$ that makes the function equal to 1 or proving that the function is equal to the constant 0.

3 Path Sensitization Model

In this section we detail a SAT-based model that is used for solving the problems of circuit delay computation and path delay validation.

The conditions under which signals propagate from the primary inputs to the primary outputs in a combinational circuit are generally referred to as path sensitization conditions. Path sensitization conditions depend on the model of operation assumed for the circuit, in particular the different forms of stimuli on the primary inputs, and the waveform model assumed at each node in the circuit. Even though detailed and precise models can be considered, we shall restrict ourselves to floating mode operation [4], under which all nodes are assumed to undergo a single known transition, from an initial unknown value to a final *stable* known value. Most criteria defined under floating mode operation are conservative (e.g. viability [7] and the *exact* criterion under floating mode operation [4]), thus overestimating the actual circuit delay in some situations. Nevertheless, as shown in [7], viability and floating mode sensitization are *robust*, thus providing upper bounds on the circuit delay under the bounded gate delay model (i.e. assuming each gate delay is within some interval $[0, d_{max}]$) [7-9].

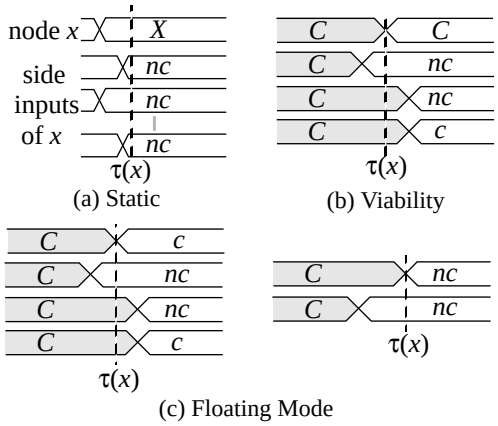

Figure 1: A characterization of path sensitization criteria

A characterization of different sensitization criteria for floating-mode operation for simple gates, under the assumption of single path sensitization, is illustrated in Figure 1 (which is adapted from [12]), and identifies logical and temporal constraints on the side inputs to each node x in a path. $\tau(x)$ denotes the propagation delay of a signal transition to node x along a given path. The side inputs values can either be *controlling* (*c*) or *non-controlling* (*nc*). Symbol *C* indicates that a given circuit node value is unknown and may experience changes in time. A more detailed description of each criterion can be found in [4, 7, 12].

The conditions associated with each path sensitization criterion can easily be captured using Propositional Satisfiability [8, 12]. Basically, the objective is to define conditions under which a given circuit node can stabilize at a given time instant.

Definition 1. We define the Boolean function $\chi^{f,\,t}(c)$ such that $\chi^{f,\,t}(c) = 1$ if and only if circuit node f stabilizes at a time greater than or equal to t when input vector c is applied to the primary inputs.

As a result, for a given circuit delay Δ and considering the set of primary outputs *PO*, we have the condition,

$$\sum_{g \in PO} \chi^{g,\,\Delta}(c) = 1 \qquad (1)$$

for some input vector c. If there exists at least one path with delay Δ that is sensitizable under the path sensitization model assumed, then condition (1) must be satisfiable. Moreover, different sensitization criteria can be captured with different definitions of $\chi^{f,\,t}(c)$ [12].

Given the interpretation of viability for simple gates in Figure 1-b and considering a straightforward generalization for multiple paths with the same path delay values, we have the following conditions for a given circuit node f to stabilize at a time no earlier than a given delay t for some input vector c:

1. At least one fanin node g of f, with delay $d(g, f)$ between g and f, must stabilize at a time no earlier than $t - d(g, f)$. This condition permits the existence of multiply sensitized partial paths.
2. Furthermore, either a fanin node assumes a non-controlling value or it stabilizes at a time no earlier than $t - d(g, f)$, thus being passive regarding propagating a signal transition from g to f. Formally, we have,

$$\chi^{f,\,t}(c) = \sum_{g \in FI(f)} \chi^{g,\,t - d(g,\,f)}(c) \cdot \prod_{h \in FI(f)} (\chi^{h,\,t - d(h,\,f)}(c) + (h = nc(f))) \qquad (2)$$

which is basically equivalent to the condition originally proposed in [8]. Furthermore, observe that each function $\chi^{f,\,t}(c)$ can be viewed as a node in a combinational circuit. Given T. Larrabee's well-known mapping [6] from circuits into CNF formulas, and from conditions (1) and (2), it is straightforward to generate a CNF formula for capturing the sensitization conditions for all paths with delay no smaller than a given threshold delay Δ. It can easily be concluded that the CNF formula size is polynomial in the number of $\chi^{f,\,t}(c)$ functions considered [8, 12]. Finally, we note that other sensitization criteria can also be captured using SAT [12].

As an example, let us consider the circuit in Figure 2. Assuming a unit-delay model (each gate has delay 1), the longest topological path delay of the circuit is 3. In order to decide whether 3 is the critical delay in the circuit, we create the associated instance of SAT specified by conditions (1) and (2).

We start by noting that only two complete paths, x_1-x_5-x_7-x_9 and x_2-x_6-x_7-x_9 have topological delay equal to 3. Hence, by applying expression (2) to all the nodes contained in these two paths we obtain φ_{ta}. From this formula the equivalent circuit, representing $\varphi_{fc} \cup \varphi_{ta}$, can be derived, as shown in Figure 3. For this particular example it can easily be concluded that the condition $\chi^{x_9,\,3} = 1$ is unsatisfiable, and so the critical delay cannot be 3. Clearly this process is iterated until a satisfiable set of conditions is found for a given circuit delay value.

The *critical path delay* of a circuit, Δ_{CD}, is defined as the *largest* delay value of a path in a circuit along which a signal transition is able to propagate, under a

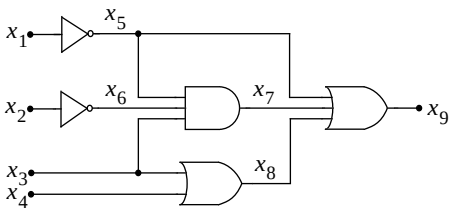

Figure 2: An example circuit

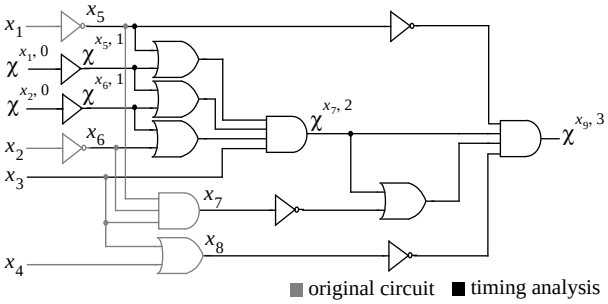

Figure 3: Circuit representing the sensitization conditions

```
CircuitDelayComputation (C)
{
    (o, Δ_TD) = LongestTopologicalPathDelay(C);
    do {
        φ_fc = FunctionalClauses(C, o);
        φ_ta = TimingClauses(C, o, Δ_TD);
        status = SATSolve(φ_fc ∪ φ_ta);
        (o, Δ_TD) = GetNextDelay(C, o, Δ_TD);
    } while (status==UNSAT and Δ_TD>0)
    return Δ_TD ; // returned value is Δ_CD
}
```

Figure 4: The circuit delay computation algorithm

chosen path sensitization model, from the primary input to the primary output, for a given primary input vector. Basically the critical path delay corresponds to the largest value of Δ_{TD} for which $\chi^{f,\Delta_{TD}}(c)$ is satisfiable.

4 Timing Analysis Algorithms

In this section we describe two algorithms used in the timing analysis of digital circuits. We start with circuit delay computation, for the identification of the critical path delay, and then describe a path delay validation algorithm, with potential application in early prototype and design validation.

4.1 Circuit Delay Computation

The computation of the critical path delay is defined as the circuit delay computation problem. Our proposed algorithm for solving this problem is shown in Figure 4, and it follows the ideas described in [12].

Starting from the longest topological path delay, and for each target delay Δ_{TD}, a CNF formula $\varphi_{fc} \cup \varphi_{ta}$ is created, which includes the clauses created by the functions `FunctionalClauses` and `TimingClauses`. These two functions capture, respectively, the circuit's function and the path sensitization constraints. Procedure `SATSolve` is then used to evaluate the satisfiability of the CNF formula. If it is satisfiable then the critical delay of the circuit has been identified, otherwise we proceed to the next primary output path delay pair (which is determined by `GetNextDelay`, using one of the strategies suggested in the next section).

4.2 Delay Stepping Strategies

A key procedure in the circuit delay computation algorithm is the stepping of target path delays. In general delay stepping plays a crucial role in the overall efficiency of the algorithm, since it determines the number of iterations of the algorithm to be executed.

The most simple delay stepping strategy is to change the target delay by the *least* delay unit at each iteration of the algorithm. Consequently, Δ_{TD} is continuously decreased by 1 delay *fraction* (that corresponds to the smallest delay variation possible, given a predefined precision). Although the computation of Δ_{TD} is immediate, this kind of strategy can result in an enormous number of iterations, specially for circuits with a large number of false paths, where the critical delay is much smaller than the topological delay, or whenever precise delay models are assumed.

Moreover, it is clear that not all path delays are possible on each primary output. Hence, instead of decreasing Δ_{TD} in a fractional step basis we can analyze the circuit topology and get the next topological delay present at the specified primary output. This is called the *next delay stepping* strategy, because the delay step is determined by the next topological delay present at a given primary output. For circuits with few path delays, this strategy can yield a reduction in the number of iterations, with a small increase in the time necessary to compute Δ_{TD}. However, for circuits with a large number of paths, where there exists an almost continuous delay distribution, this strategy rapidly becomes as inefficient has the previous one, thus taking significantly more time to compute Δ_{CD}.

To overcome the limitations of the two previous strategies, we can use *dynamic stepping*. In this strategy the delay step is dynamically adjusted, according to certain criteria, but is independent of the circuit topology (no next delay computation is performed). For a given primary output, we first perform timing analysis with $\Delta_{TD} = LTP$. If this instance is unsatisfiable, then we round Δ_{TD} to the nearest multiple of 0.5. Afterwards, we iteratively decrease Δ_{TD} by 0.5 until we find a satisfiable instance. Now Δ_{TD} is a tight upper bound (i.e error < 0.5) of the critical delay. Next, we decrease Δ_{TD} by 0.1 until we get an unsatisfiable instance. The delay of the last satisfiable instance is the critical delay with an accuracy of one decimal place. To determine the second decimal place we use the exact same procedure, but we multiply the delay steps by 0.1. Hence, the delays we obtain have a precision of two decimal places. Clearly, this procedure can easily be extended to compute delays with any required precision. Note that, in order to use dynamic stepping with the algorithm shown in Figure 4, we must associate search state data with each PO, that will allow us to check each PO search state (direction of the search and decimal places already identified). This search state data is used and updated by `GetNextDelay` and is also used in the stopping condition of the main loop.

4.3 Path Delay Validation

Often the circuit designer is only interested in checking whether signal delays at some output meet certain timing constraints, usually imposed by the system where a design is integrated. The purpose of the path delay validation problem is to check whether, for a specific circuit output, a given delay d_{max} is an upper bound on the largest propagation delay to that output.

The procedure shown in Figure 5 evaluates whether $\Delta_{CD} < d_{max}$, given an output o of a circuit C. If this constraint is satisfied it returns `True`, otherwise it returns `False`. If $LTP \le d_{max}$ then clearly $\Delta_{CD} \le d_{max}$, since $\Delta_{CD} \le LTP$. When $LTP > d_{max}$ we must perform path sensitization for d_{max}. If the result is satisfiable then $\Delta_{CD} \ge d_{max}$, otherwise $\Delta_{CD} < d_{max}$.

5 Experimental Results

The algorithms described in the previous sections were build on top of a SAT solver [11], and run on the ISCAS'85 benchmark circuits [3]. These circuits were mapped using the standard-cell library ECPD07 (ES2/ Atmel) and the parasitics of the interconnect were extracted. The gate and interconnect delays (with a precision of two decimal places) were obtained combining

```
PathDelayValidation (C, o, d_max)
{
    LTP = LongestTopologicalPathDelay(C, o);
    if (LTP > d_max) {
        φ_fc = FunctionalClauses(C, o);
        φ_ta = TimingClauses(C, o, d_max);
        status = SATSolve(φ_fc ∪ φ_ta);
        if (status==SAT)
            return False;
    }
    return True;
}
```

Figure 5: The path delay validation algorithm

Circuit	LTP	Δ	Fraction delay		Next delay		Dynamic	
			Iter	CPU	Iter	CPU	Iter	CPU
C432	20.20	19.90	31	2.98	26	8.17	30	4.71
C499	16.67	16.64	4	0.13	2	0.08	21	0.91
C880	18.59	18.59	1	0.05	1	0.06	1	0.05
C1355	22.38	21.97	82	9.68	31	8.65	23	10.59
C1908	32.44	29.68	1079	892.84	667	2420.83	52	33.89
C2670	40.31	38.62	220	141.22	133	1363.06	15	36.91
C3540	45.19	43.10	515	2496.68	—	—	17	196.43
C5315	58.57	57.36	129	34.82	49	300.51	16	9.44
C6288	73.82	73.06	140	5091.82	—	—	10	789.57
C7552	38.57	36.39	325	232.99	143	1846.51	33	37.12
CSA.16.4	36.00	20.10	7489	282.65	283	13.33	170	7.00
CSA.32.4	72.84	28.54	46680	8859.46	9178	3339.78	976	327.71
CBP.12.2	22.65	13.94	3417	400.62	1018	186.52	97	25.87
CBP.16.4	25.84	16.52	4418	304.33	608	60.28	105	10.26
CLA.16	21.68	21.65	4	0.26	4	0.26	6	0.73

Table 1: Delay stepping results for delay computation

Circuit	LTP	Δ	$0.8LTP$	CPU	$0.9LTP$	CPU
C432	20.20	19.90	16.16	3.21	18.18	4.32
C499	16.67	16.64	13.31	0.07	14.98	0.06
C880	18.59	18.59	14.87	0.34	16.73	0.26
C1355	22.38	21.97	17.90	6.43	20.13	3.72
C1908	32.44	29.68	24.06	47.06	27.06	5.69
C2670	40.31	38.62	32.25	182.04	36.28	33.16
C3540	45.19	43.10	36.15	—	40.67	143.38
C5315	58.57	57.36	46.86	141.18	52.71	17.11
C6288	73.82	73.06	59.06	—	66.44	—
C7552	38.57	36.39	29.71	53.32	33.43	13.09
CSA.16.4	36.00	20.10	27.33	0.04	30.74	0.03
CSA.32.4	72.84	28.54	55.92	0.24	62.91	0.12
CBP.12.2	22.65	13.94	17.58	0.18	19.77	0.08
CBP.16.4	25.84	16.52	20.12	0.11	22.64	0.06
CLA.16	21.68	21.65	17.34	0.16	19.51	0.20

Table 2: Statistics for path delay validation using viability

the information provided by the extraction and the IC databook, in a simple delay model, described in [12]. The results for circuit delay computation are shown in Table 1, whereas results for path delay validation on a single primary output are shown in Table 2.

As can be concluded from Table 1, the selection of the delay stepping strategy significantly affects the overall efficiency of the algorithm. From the results it is clear that dynamic stepping leads to significantly better running times.

For the path validation algorithm, and depending on the required upper bound (d_{max}) we need, at most, to solve one SAT instance for each circuit, which will allow us to check whether $\Delta_{CD} < d_{max}$, even if $LTP > d_{max}$. In Table 2 we have results for this algorithm, using two sample values for d_{max}, $0.7LTP$ and $0.9LTP$, respectively. As can be concluded, path delay validation is in general easier to solve than circuit delay computation, and so potentially preferable by circuit designers in the early stages of a design.

6 Conclusions

In this paper we present satisfiability-based timing analysis algorithms for circuit delay computation and for path delay validation. For the circuit delay computation algorithm the critical delay of a circuit is computed by iteratively solving a sequence of instances of propositional satisfiability. Three delay stepping strategies were considered: fractional delay stepping, next delay stepping and dynamic stepping. The dynamic stepping provides by far the most efficient and robust results. For the path delay validation algorithm, an upper bound on the critical delay of a circuit is tested, by performing, at most, one satisfiability check. This algorithm allows straightforward validation of timing constraints by circuit designers.

Future research work mainly consists of improving the efficiency of the underlying SAT engine by taking into account structural information about the circuit.

References

[1] P. Ashar, S. Malik and S. Rothweiler, "Functional Timing Analysis using ATPG," in *Proceedings of the European Design Automation Conference*, 1993.

[2] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo and F. Somenzi, "Algebraic Decision Diagrams and Their Applications," in *Proceedings of the International Conference on Computer-Aided Design*, November 1993.

[3] F. Brglez and H. Fujiwara, "A Neutral List of 10 Combinational Benchmark Circuits and a Target Translator in FORTRAN," in *Proceedings of the International Symposium on Circuits and Systems(ISCAS)*, 1985.

[4] H.-C. Chen and D. H. Du, "Path Sensitization in Critical Path Problem," *IEEE Transactions on Computer-Aided Design*, vol. 12, no. 2, pp. 196-207, February 1993.

[5] S. Devadas, K. Keutzer and S. Malik, "Computation of Floating-Mode Delay in Combinational Circuits: Practice and Implementation," *IEEE Transactions on Computer-Aided Design*, vol. 12, no. 12, pp. 1924-1936, December 1993.

[6] T. Larrabee, *Efficient Generation of Test Patterns Using Boolean Satisfiability*, Ph.D. Dissertation, Computer Science Dept., Stanford University, STAN-CS-90-1302, February 1990.

[7] P. C. McGeer and R. K. Brayton, *Integrating Functional and Temporal Domains in Logic Design: The False Path Problem and its Implications*, Kluwer Academic Publishers, 1991.

[8] P. McGeer, A. Saldanha, P. R. Stephan, R. K. Brayton and A. L. Sangiovanni-Vincentelli, "Timing Analysis and Delay-Test Generation Using Path Recursive Functions," in *Proceedings of the International Conference on Computer-Aided Design*, November 1991.

[9] P. McGeer, A. Saldanha, R. K. Brayton and A. L. Sangiovanni-Vincentelli, "Delay Models and Exact Timing Analysis," in *Logic Synthesis and Optimization*, T. Sasao (Ed.), 1993.

[10] J. P. M. Silva and K. A. Sakallah, "Efficient and Robust Test-Generation Based Timing Analysis," in *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, May 1994.

[11] J. P. M. Silva and K. A. Sakallah, "GRASP—A New Search Algorithm for Satisfiability," in *Proceedings of the International Conference on Computer-Aided Design*, November 1996.

[12] L. G. Silva, J. P. M. Silva, L. M. Silveira and K. A. Sakallah, "Realistic Delay Modeling in Satisfiability-Based Timing Analysis," to appear in the *Proceedings of the International Symposium on Circuits and Systems (ISCAS)*, May 1998.