Unifying Functional and Parametric Timing Verification

Luis Guerra e Silva
INESC-ID / IST / TU Lisbon
Lisbon, Portugal
lgs@inesc-id.pt

ABSTRACT

This paper proposes a unified modeling framework for timing verification of IC designs that, through an elegant SMT-based formulation, seamlessly integrates functional timing analysis and parametric delay modeling. Such framework enables accurate timing verification by simultaneously ignoring false paths and accounting for process variability. By casting the timing verification problem as a general SMT instance it is possible to benefit from the continuous advances in performance and robustness of modern SMT engines. The proposed framework is validated for a representative set of benchmarks, using Microsoft's Z3 SMT solver.

Categories and Subject Descriptors

B.7.2 [**Hardware**]: Integrated Circuits—*Design Aids*

General Terms

Algorithms, Design, Theory, Verification

Keywords

Timing Verification, False Paths

1. INTRODUCTION

Timing verification is concerned with predicting, prior to fabrication, the timing-critical paths of an IC design, and assessing whether it will be able to operate at its target clock frequency. Such task is becoming increasingly challenging, given the sheer size and complexity of modern IC designs and the uncertainty introduced by nanometric fabrication technologies, extremely sensitive to process variations.

The need to adequately model process variability has motivated the introduction of parametric delay models, where cell and interconnect delays are no longer given by fixed real numbers, but instead by affine functions of the parameters of the fabrication process. Several compatible parametric static timing analysis (PSTA) techniques that make use of such models have likewise been proposed [20, 18, 10].

GLSVLSI'12, May 3–4, 2012, Salt Lake City, Utah, USA.

While modern timing verification tools incorporate sophisticated, variability-aware, parametric delay modeling techniques, they only consider the topology of the circuit, still lacking the ability to account for its logic behaviour. Unfortunately, for reasons hard to assess, high-level synthesis systems are prone to generate circuits with many *false paths* [3], i.e. paths that cannot be exercised by any input pattern [13]. Topology-based timing verification tools, the standard in industrial design flows, can often produce conservative timing estimates by considering such paths as critical, even though they cannot be exercised in real circuit operation. This can lead to wasteful overdesign, which constitutes a serious problem in the competitive market of microelectronics, where continuously increasing performance and complexity must be packed into the same die area.

The inability of industrial timing verification tools to identify false paths was slightly mitigated by allowing the designer to tag known false paths, to be ignored during timing verification. However, it is impossible to manually identify and tag all the false paths of any useful design block which, most often, may contain thousands of cells. Therefore, the integration of automated and systematic false path identification capabilities into modern timing verification tools is badly needed for enabling performance optimization to be guided in a computationally efficient manner.

Due to the existence of false paths, the problem of computing the delay of a circuit can no longer be solved in linear time, being instead an NP-complete [11] problem. During the 90s, the research work on false paths was extensive and, among others, several promising modeling and algorithmic approaches have been proposed [1, 4, 7, 16, 15, 22]. The added complexity that entails solving the false path problem and the existence of comfortable design margins, that could accomodate conservative timing estimates, were the main reasons why, despite all the research work conducted on the topic, automated false path detection techniques never found their way into industrial timing verification tools.

Recent years have seen a renewed interest on false path identification techniques, with most contributions [19, 23, 5] targeting practical aspects of their integration into deterministic timing verification. Variability-aware timing verification has also been addressed [14, 12, 21]. However, the difficulty of combining, in a single formulation, logic constraints and parametric delay models, led to *ad hoc* solutions. False paths were ignored either by performing Monte Carlo electrical simulation, for several parameter settings, or through a pre-processing step, that would perform functional analysis assuming worst-case parameter settings.

Indeed, the integration of functional analysis and parametric delay modeling is a complex task, since it requires a computational framework capable of simultaneously manipulating Boolean and linear numeric constraints which, until recently, was not readily available. However, the advent of Satisfiability Modulo Theories (SMT) [2], which generalizes Boolean Satisfiability (SAT) by adding several first-order theories such as equality, arithmetic, quantifiers, etc, has opened the possibility of formulating problems that combine Boolean constraints with other types of constraints, namely linear constraints on real values.

Leveraging on the contemporary SMT technology, we propose a unified modeling framework for timing verification of IC designs that, through an elegant SMT-based formulation, seamlessly integrates functional timing analysis and parametric delay modeling. Such framework achieves accurate timing verification by simultaneously ignoring false paths and accounting for process variability.

To the best of our knowledge, this is the first work to cast the timing verification problem as an SMT instance. This approach has several advantages. Firstly, the expressive power of SMT formulas enables the representation of complex relations between timing quantities and/or process parameters, which can be used to model a wealth of timing verification problems. Secondly, the increased support of SMT solvers for non-linear arithmetic enables easy extension to support future non-linear delay models. Lastly, by casting the timing verification problem as an SMT instance, it is possible to benefit from the continuous advances in performance and robustness of modern SMT engines.

The remainder of this paper is organized as follows. Section 2 introduces the parametric timing modeling, underlying the work presented in this paper, as well as the essential aspects of SMT, necessary to understand the formulations discussed in upcoming sections. Section 3 presents an overview of the proposed modeling framework. Sections 4 and 5 detail functional and timing modeling for cells and interconnect, respectively. The experimental results are presented and discussed in Section 6. Finally, Section 7 presents brief concluding remarks and foresees future work.

2. BACKGROUND

2.1 Parametric Timing Modeling

The timing information of a circuit is modeled by a *timing graph* $G = (V, E)$, where *vertices*, $v \in V$, correspond to pins in the circuit, and directed *edges*, $e \in E$, correspond to pin-to-pin delays in cells or interconnect. The *primary inputs*, $u \in PI(G)$, are vertices with no incoming edges. All vertices with no outgoing edges are *primary outputs*, $w \in PO(G)$, but there may also be primary outputs with outgoing edges. A *complete path* is a sequence of edges, connecting a primary input to a primary output. A *path* is a sequence of edges connecting any two vertices

Edges are annotated with the corresponding *delays*. At most four delays can be annotated on each edge, depending on the input and output rise/fall transitions: d^{FF}, d^{FR}, d^{RF} and d^{RR}. Since it is out of the scope of this paper to discuss the delay computation procedure, in the following, we will assume that the timing information of any circuit is already made available in the form of an annotated timing graph. Output pins of cells are also annotated with the corresponding logic function.

This work assumes a PSTA model [20, 18, 10], where delays are described by affine functions of process and operational parameter variations, corresponding to a first-order linearization of every delay, d, around a nominal point, λ_0, in the parameter space. Considering the parameter space to have size p, and representing d as a function of the incremental parameter variation vector, $\Delta\lambda = \lambda - \lambda_0$, around a nominal value λ_0, we obtain

$$d(\Delta\lambda) = d_0 + \sum_{i=1}^{p} d_i \Delta\lambda_i \quad (1)$$

where $d_0 = d(\lambda_0)$ is the nominal value of d and d_i is the sensitivity of d to parameter λ_i, $i = 1, 2, \ldots, p$, computed at the nominal point λ_0. Parameter variations are assumed to lie within a given range, $\Delta\lambda_i \in \left[\Delta\lambda_i^{min}, \Delta\lambda_i^{max}\right]$,

2.2 Modes of Operation

The primary task in any timing verification run is arrival time computation. The *arrival time*, represented by at, is a conservative estimate of the earliest or the latest time instant that a signal transition can reach a given circuit point, when traveling from an input (or the output of a sequential element). The meaning of the arrival time values depends on whether we assume the early or the late mode of operation. In *early mode*, we are concerned with computing the earliest time instant that a signal transition can reach a given circuit point. Conversely, in *late mode* we are concerned with computing the latest time instant that a signal transition can reach a given circuit point. Both early and late mode analyses are of practical interest. Setup constraints are verified through late mode analysis, while hold constraints are verified through early mode analysis. In the following, and without loss of generality, we assume the late mode.

Even though functional analysis in static timing verification does not require any specific input excitation, some assumptions on the variation of logic values of circuit nodes must be made. Two possibilities have been considered and extensively studied in the literature: transition mode and floating mode. In the *transition mode* of operation [8], circuit nodes are assumed to switch from a known initial logic value to a known final logic value (e.g. $1 \rightarrow 0$). In the *floating mode* of operation [4], circuit nodes are assumed to switch from an unkown initial logic value to a known final logic value (e.g. $? \rightarrow 0$). In this work we assume the floating mode of operation. Even though the transition mode provides more accurate timing estimates, it has a significantly higher computational cost.

2.3 Satisfiability Modulo Theories

This subsection is not meant to be a comprehensive introduction to SMT, as it would probably exceed the length of the entire paper (see [2] for that). It is solely intended to informally introduce a few basic concepts necessary for the reader to understand the use of SMT throughout the paper.

Satisfiability is one of the quintessential problems in computer science, which consists of determining whether a given formula, enconding one or more constraints, has a solution. SAT is the best known of the constraint satisfaction problems, where the formula is built using logical connectives, over the Boolean variables.

An SMT instance is a formula in first-order logic [17] where function and predicate symbols can be interpreted resorting to a variety of underlying theories. SMT can be

seen as a generalization of SAT, but where some of the Boolean variables are replaced by predicates. A *predicate* is a Boolean-valued function, $P : X \rightarrow \{0, 1\}$, designated by predicate on X. A predicate can be seen as a condition that evaluates to either 1 (*true*) or 0 (*false*), depending on the values of its variables. For instance, $a > 3.14$ is a predicate that evaluates to 1 if the real-valued variable a assumes a value larger than 3.14 and evaluates to 0 otherwise. Predicates are classified according to the theory they belong to. For example, linear inequalities over real variables are evaluated using the rules of the theory of linear real arithmetic.

The SMT problem is concerned with determining a set of variable assignments that make the corresponding formula *true*, or prove that no such assignments exist and the formula is always *false*. An example SMT formula, on a real-valued variable a and two Boolean-valued variables b and c is,

$$(a > 3.14) \wedge (\neg b \vee c)$$

A *model* for this problem, i.e. a set of satisfying variable assignments, is $a = 3.15$, $b = 0$ and $c = 1$.

SMT formulas are built by combining logical connectives, such as $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\oplus$ (exclusive disjunction), $\Rightarrow$ (implication) and $\Leftrightarrow$ (equivalence), with Boolean variables and predicates. The structure of the predicates depends on their underlying theory. For example, predicates under the theory of linear real arithmetic, can be built by combining real-valued variables and constants with comparison operators ($<$, $=$, $>$, $\leq$, $\geq$, $\neq$) and arithmetic operators ($+$, $-$, $*$, $/$), among others.

3. MODELING FRAMEWORK

First and foremost, we should clearly state the general problem to be addressed by our modeling framework. *Given the timing graph for a combinational circuit, as described in Section 2, we want to determine the largest arrival time at its primary output vertices.* This problem, which is often designated by *circuit delay computation*, constitutes the cornerstone of any timing verification procedure.

The problem variables are, in first instance, the values that need to be computed, i.e. the arrival times at the primary output vertices. However, such arrival times are dependent on the arrival times at the intermediate vertices, as well as on the delays between them. Delays, on the other hand, are dependent on the Boolean values assumed by the vertices and on the values of the process parameter variations. Through this simple dependency analysis we are able to conclude that the variables of our problem should be:

- a Boolean-valued variable b_v, for every vertex v, corresponding to the final logic (Boolean) value assumed by vertex v after the transition;
- a real-valued variable at_v, for every vertex v, corresponding to the arrival time at vertex v;
- a real-valued variable $\Delta\lambda_i$, for every parameter of order i, corresponding to the parameter variation.

Formulating the timing verification problem as an SMT instance, requires capturing both functional and timing constraints, for all circuit elements, into an SMT formula, φ. While the expressive power of SMT admits a wealth of different formulations, in the following we shall assume that such SMT formula is a conjunction of other partial formulas, capturing functional (φ^b) and timing (φ^t) constraints:

$$\varphi = \left(\bigwedge_i \varphi_i^b\right) \wedge \left(\bigwedge_j \varphi_j^t\right) \tag{2}$$

Therefore, for the whole formula to be satisfied, all the partial formulas must also be satisfied. φ can be progressively built by traversing the timing graph in a levelized breadth-first fashion, and augmenting it with the partial formulas that capture the relations between boolean values and arrival times of input and output vertices of cells and interconnect, as detailed in Sections 4 and 5.

The SMT solver is not an optimization engine, since it is only able to find *a* solution that satisfies the formula, not the *best* solution according to some given criteria. However, we need to determine the *largest* arrival time at the primary output vertices, not just some valid arrival time value. This is the typical case where an optimization problem must be cast into a sequence of decision problems.

The topological arrival time is cheap to compute (linear time), yet it assumes that all paths can be exercised. Since some paths may be false, the topological arrival time is actually an upper bound to the true arrival time (assuming late mode). Therefore, we can start by checking whether the topological arrival time is the true arrival time or not. If it is not, then we can check that for a slightly smaller value, and continue iterating until we reach the true arrival time. On each iteration, we must check whether the arrival time at some primary output can be not smaller than the given *required arrival time*, rat. This "question" can be asserted into the SMT formula by adding,

$$\varphi_{PO}^t = \bigvee_{w \in PO(G)} (at_w \geq rat) \tag{3}$$

Once all the functional and timing constraints are built into the SMT formula, as detailed in Sections 4 and 5, other "questions" may be asked, by adding proper assertions and subsequently running the SMT solver. For example, this same formulation can be used to check setup times, or arrival times for specific parameter variation settings (corners). Moreover, it can also be used for generating input test patterns capable of producing specific arrival time values at particular points in the circuit (ATPG). For any problem instance, the answer will always be a set of assignments for the logic values, arrival times and process parameter variations, that satisfy the problem constraints, or the proof that no such assignments exist.

4. CELL MODELING

This section details how digital cells are modeled in our SMT-based timing verification framework, both at functional and timing level.

4.1 Functional Constraints

The logic function of a given output pin, for any given combinational cell in the library, is described in a particular field of the Liberty (`.lib`) file, in terms of one- or two-operand elementary logic operations: `&` (AND), `|` (OR), `!` (NOT) and `^` (XOR). Exemplifying, for the single output pin of a 3-input `AOI21` cell we would have the following formula: "`!(A | (B1 & B2))`". This representation enables arbitrarily complex functions to be described in terms of elementary

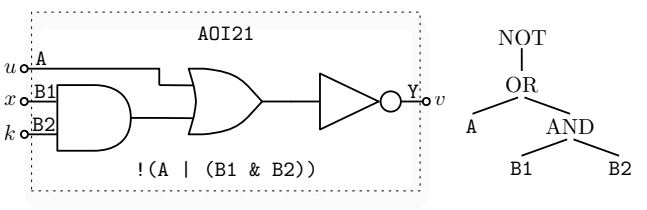

Figure 1: Parsing tree for logic function of `AOI21` cell.

logic operations. A trivial parser for this type of logic expressions was developed. Such parser maps each expression into a parsing tree, as illustrated in Figure 1, where internal nodes are logic operations and leaf nodes represent input pins of the cell. As we will see, the parsing tree is a convenient representation that can be easily traversed for generating all the required functional and timing constraints.

Modeling the functional behaviour of a combinational cell amounts to adding to the SMT formula, φ, all the constraints that assign to the logic value variables of every output pin their corresponding value, in terms of the logic value variables of the input pins. Since the parsing tree of each output pin already contains its logic function factored into elementary logic operations, we can trivially generate all the necessary constraints by traversing the tree in a bottom-up fashion and generating the constraints for each operation, as summarized below:

$$\varphi^b_{AND} = \{b_v = b_u \wedge b_x\} \tag{4}$$

$$\varphi^b_{OR} = \{b_v = b_u \vee b_x\} \tag{5}$$

$$\varphi^b_{NOT} = \{b_v = \neg b_u\} \tag{6}$$

$$\varphi^b_{XOR} = \{b_v = b_u \oplus b_x\} \tag{7}$$

where v is assumed to be the output pin and u and x the input pins. This procedure is illustrated, for the `AOI21` cell, in the left tree of Figure 2. Since only the root and the leaves of the parsing tree actually correspond to vertices in the circuit, with associated logic value variables, we must add variables to represent the logic values in the intermediate nodes of the parsing tree, that we designate by $b_{i1,2,\ldots}$.

4.2 Timing Constraints

The timing constraints for any given cell must enable the computation of the arrival times at its output pins from the logic values and arrival times at its input pins, and from the input/output pin delays.

We start by adding an artificial arrival time variable, for each cell input/output combination, representing what would be the arrival time at the output if its transition was triggered by the corresponding input. The value of this variable will be computed by adding the arrival time value at the input pin to the proper delay value, chosen according to the logic value variables of the input and output pins. Assuming the input and output pin vertices to be u and v, respectively,

$$at_{u,v} = \begin{cases} at_u + d^{FF}_{u,v} & \text{if } \neg b_u \wedge \neg b_v \\ at_u + d^{FR}_{u,v} & \text{if } \neg b_u \wedge b_v \\ at_u + d^{RF}_{u,v} & \text{if } b_u \wedge \neg b_v \\ at_u + d^{RR}_{u,v} & \text{if } b_u \wedge b_v \end{cases} \tag{8}$$

Eqn. (8) can be added to the SMT formula, φ, by nested application of the $ite(t_1, t_2, t_3)$ (if-then-else) operator, supported by most SMT engines, whose result is t_2, when t_1 is true, and t_3 otherwise. Therefore, we obtain,

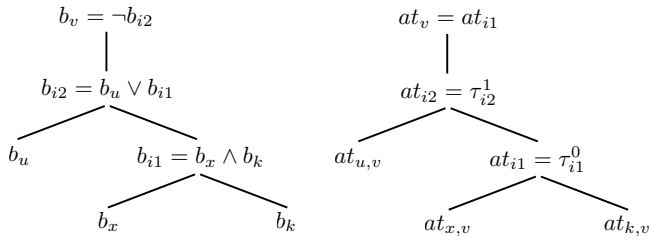

Figure 2: Functional and timing constraints.

$$\begin{aligned} \varphi^t_{ATD} = \{\, at_{u,v} = at_u + ite(\neg b_u \wedge \neg b_v,\ d^{FF}_{u,v},\\ ite(\neg b_u \wedge b_v,\ d^{FR}_{u,v},\\ ite(b_u \wedge \neg b_v,\ d^{RF}_{u,v},\\ d^{RR}_{u,v}))) \,\} \end{aligned} \tag{9}$$

Computing the true arrival time at any given cell output pin involves considering its logic function as well as the logic values, arrival times and delays at the relevant input pins. For the two-input AND and OR cells, the arrival time at the output pin is computed considering whether each input pin assumes a controlling value or not. For the AND cell the controlling value is $c = 0$ and for the OR cell the controlling value is $c = 1$. The arrival time for the output pin vertex v of an AND/OR cell, assuming the controlling value to be c and the input pin vertices to be u and x, is given by

$$\tau^c_v = \begin{cases} \min(at_{u,v}, at_{x,v}) & \text{if } b_u = c \wedge b_x = c \\ at_{u,v} & \text{if } b_u = c \wedge b_x = \neg c \\ at_{x,v} & \text{if } b_u = \neg c \wedge b_x = c \\ \max(at_{u,v}, at_{x,v}) & \text{if } b_u = \neg c \wedge b_x = \neg c \end{cases} \tag{10}$$

Eqn. (10) can be added to the SMT formula using nested *ite* operations, as in the case of Eqn. (8). However, a more efficient encoding would be to use implications, resulting in

$$\begin{aligned} \varphi^t_{AND/OR} = &\{b_u = c \Rightarrow at_v \leq at_{u,v}\} \wedge \\ &\{b_x = c \Rightarrow at_v \leq at_{x,v}\} \wedge \\ &\{b_u = \neg c \wedge b_x = \neg c \Rightarrow at_v \leq at_{u,v} \vee at_v \leq at_{x,v}\} \end{aligned} \tag{11}$$

For the XOR cell, no single input pin can independently determine the logic value at the output pin, therefore,

$$\varphi^t_{XOR} = \{at_v \leq at_{u,v} \vee at_v \leq at_{x,v}\} \tag{12}$$

For the NOT cell, assuming input pin vertex u and output pin vertex v, we obtain,

$$\varphi^t_{NOT} = \{at_v = at_{u,v}\} \tag{13}$$

The timing constraints for complex cells can now be easily generated. Since cell delays have already been incorporated into the artificial arrival time variables given by Eqn. (8), we can plug such variables into the corresponding input pin leaves of the parsing tree and, by traversing it in a bottom-up fashion and applying Eqns. (11), (12) or (13), according to the logic function of each internal node, all the required timing constraints are generated. New arrival time variables $at_{i1,2,\ldots}$ must be added for the internal nodes of the parsing tree. This procedure is illustrated, for the `AOI21` cell, in the right tree of Figure 2.

5. WIRE MODELING

This section details how the interconnect (wires) between digital cells are modeled in our SMT-based timing verification framework, both at functional and timing level.

5.1 Functional Constraints

In a digital circuit, the role of signal wires is to carry digital signals with minimal voltage degradation, such that the voltage on each endpoint corresponds to the same logic value. The logic function of wires can thus be thought as of identity. Therefore, for every wire edge $\langle u, v \rangle$, a constraint enforcing the equality between the logic values at vertices u and v must be added to the SMT formula, φ:

$$\varphi^{b}_{WIRE} = \{b_v = b_u\} \tag{14}$$

Most often this equality does not need to be explicitly stated in the form of a constraint. It is sufficient to define a single logic variable b_{uv} that will be used in place of both b_u and b_v. This approach reduces the number of variables, thus contributing to improve the performance of the SMT engine.

5.2 Timing Constraints

Timing constraints for wires are quite easy to generate. Since both ends of a wire must assume the same logic value, to compute the arrival time at the output pin of the wire we only have to select the rise/rise or fall/fall delay corresponding to that logic value and add it to the arrival time of the input pin. Assuming the input pin vertex to be u and the output pin vertex to be v, we obtain,

$$\varphi^{t}_{WIRE} = \{\ at_v = at_u + ite(b_v,\ d^{RR}_{u,v},\ d^{FF}_{u,v})\ \} \tag{15}$$

6. EXPERIMENTAL RESULTS

The modeling framework described in the previous sections was coded in C++. We have used Microsoft's Z3 Theorem Prover v3.2 [6] as our SMT solver, given its performance, robustness, and the availability of a C++ API. Z3 is one of the most reputed SMT solvers, and is the workhorse behind several of Microsoft's software verification tools. For benchmark circuits we have used the traditional ISCAS'85 combinational suite. All the results presented in this section were obtiained on a machine with an Intel Core i7 @ 3.07GHz and 12GB of available RAM. For all the runs a single processor was used.

6.1 Validation

The first stage of our experimental procedure was dedicated to validate that the proposed framework was actually ignoring the false paths, and therefore was able to compute the true arrival time values of any given circuit. Therefore, we evaluated our implementation with a few benchmark circuits, for which correct results are published in [16, 9]. Such benchmarks assume the unit delay model. The results are reported in Table 1, where column "TD/RD" presents the topological vs. the real (true) delay computed by the proposed framework, column "SMT" presents the CPU time in seconds taken by such computation and column "SAT" presents the CPU time in seconds taken by the SAT approach described in [9]. For all approaches, the computed delays were the same, which validates our proposed framework. CPU times in our case are larger, particularly because the ones reported in column "SAT" were obtained for a significantly slower machine. Nevertheless, this was to be expected, since our formulation is significantly more complex, as necessary for handling process variability.

Design	TD/RD	SMT	SAT
c432	17/17	0.08	0.03
c499	11/11	0.22	0.02
c880	24/24	0.09	0.04
c1355	24/24	0.75	0.12
c1908	40/37	1.34	0.26
c2670	32/30	0.74	2.83
c3540	47/46	1.30	0.54
c5315	49/47	1.63	1.27
c6288	124/123	853.02	11.19
c7552	43/42	1.23	0.17
cbp.12.2	40/23	0.67	1.53
cbp.16.4	44/27	1.01	1.03
cla.16	34/34	0.06	0.04
tau92ex1	27/24	0.79	0.63
mult-csa	78/78	299.60	5.90

Table 1: Results for unit delays.

6.2 Evaluation

For evaluating the proposed framework in the context of process variability, we have synthesized and mapped the benchmark circuits to the Nangate OCL $45nm$ technology. As process parameters, we have considered the widths and thicknesses of the 10 metal routing layers, resulting in a total of 20 parameters. Variational delay computation was subsequently performed, and the resulting affine delay formulas were annotated into the corresponding timing graphs. Table 2 presents a brief characterization of the resulting benchmark circuits, where "#PI" and "#PO" columns report the number of primary inputs and outputs, "#C" and "#N" columns report the number of combinational cells and nets, and "#V" and "#E" columns report the number of vertices and edges in the corresponding timing graph.

Table 3 reports the experimental results for true arrival time computation, using the proposed timing verification framework. Column "%OPT" reports the percentual reduction of the true arrival time, computed with the aid of functional analysis, over the topological arrival time. When they are the same, a value of 0 is reported. Columns "Formula", "Solve" and "Total" report CPU times in seconds for formula generation, SMT solve and total, respectively.

Analyzing the results presented in Table 3 we conclude that for some benchmarks it is possible to obtain non-negligible improvements on arrival time estimation, with a fair computational cost, given the complexity of the problem. The cost of formula generation seems to be related to circuit size and the improvement on arrival time estimation. The latter implies, in general, more iterations (see Eqn. 3), which can impact the cost of both formula generation and SMT solve. We believe that the results presented in Table 3 compare favorably to the results presented in Table 1 since, in a variability context, the problem is much harder, and the increase in CPU time is nonetheless limited.

While the performance of the proposed modeling framework does not yet make it adequate for the characterization of large digital blocks, it can be used for characterizing small critical blocks. Nevertheless, we believe that this approach has enormous potential, since our implementation is rather simplistic and can be much improved, which should enable great savings in terms of CPU time. Moreover, the problem can be easily partitioned for parallelization in modern multicore machines.

Design	#PI	#PO	#C	#N	#V	#E
c432	37	7	88	124	356	457
c499	41	32	170	211	595	736
c880	60	26	169	232	697	910
c1355	41	32	170	211	595	736
c1908	33	25	202	235	708	921
c2670	157	63	278	511	1204	1474
c3540	50	22	469	520	1841	2622
c5315	178	123	597	781	2551	3429
c6288	32	32	1005	1470	3556	5006
c7552	208	107	764	986	2820	3593

Table 2: Benchmark characterization.

Design	%OPT	Formula	Solve	Total
c432	4.25%	0.73	7.94	8.67
c499	0.03%	1.15	0.23	1.38
c880	0%	0.84	<0.01	0.84
c1355	0.24%	1.22	0.19	1.41
c1908	5.49%	1.70	8.49	10.19
c2670	0%	1.16	<0.01	1.16
c3540	3.19%	13.21	162.07	175.28
c5315	0%	8.54	<0.01	8.54
c6288	0.88%	302.33	1717.48	2019.81
c7552	0%	9.67	<0.01	9.67

Table 3: Results for variability-aware delays.

7. CONCLUSIONS AND FUTURE WORK

This paper proposes an SMT-based timing verification framework that enables accurate computation of timing estimates by integrating functional and variation-aware timing constraints, that characterize modern digital IC designs and associated fabrication technologies. While experimental evidence shows that the performance of the proposed framework is not yet adequate for application to large digital blocks, we still believe that it constitutes a significant advancement of the state-of-the-art, as it enables better accuracy in timing estimates, and provides a general variability-aware SMT formulation, that can benefit from the continuous advances in SMT engines.

Due to space restrictions, several relevant implementation details were left out of this paper, as well as the application of the proposed framework to sequential circuits. We intend to publish them in a more comprehensive journal paper, together with several strategies for significantly improving the performance of formula generation and SMT solve.

8. ACKNOWLEDGMENTS

I would like to thank João Marques-Silva for introducing me to SMT and for preliminary discussions on efficiency issues. This work was supported by FCT (INESC-ID multiannual funding) through the PIDDAC Program funds.

9. REFERENCES

[1] P. Ashar, S. Malik, and S. Rothweiler. Functional Timing Analysis using ATPG. In *Proceeding of The European Design Automation Conference*, 1993.

[2] C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability Modulo Theories. In A. Biere, M. J. H. Heule, H. van Maaren, and T. Walshy, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 26, pages 825–885. IOS Press, February 2009.

[3] R. Bergamaschi. The Effects of False Paths in High-Level Synthesis. In *Proceedings of ICCAD*, November 1991.

[4] H.-C. Chen and D. H. C. Chu. Path Sensitization in Critical Path Problems. *IEEE Transactions on CAD*, 12(2):196–207, February 1993.

[5] O. Coudert. An Efficient Algorithm to Verify Generalized False Paths. In *Proceedings of DAC*, pages 188–193, June 2010.

[6] L. de Moura and N. Bjorner. Z3: An Efficient SMT Solver. In *Proceedings of TACAS*, pages 337–340, Budapest, Hungary, March-April 2008.

[7] S. Devadas, K. Keutzer, and S. Malik. Computation of Floating-Mode Delay in Combinational Circuits: Practice and Implementation. *IEEE Transactions on CAD*, 12(12):1924–1936, December 1993.

[8] S. Devadas, K. Keutzer, S. Malik, and A. Wang. Certified Timing Verification and the Transition Delay of a Logic Circuit. In *Proceedings of DAC*, pages 549–555, June 1992.

[9] L. G. e Silva, J. Marques-Silva, L. M. Silveira, and K. Sakallah. Realistic Delay Modeling in Satisfiability-Based Timing Analysis. In *Proceedings of ISCAS*, Monterrey, CA, USA, May-June 1998.

[10] L. G. e Silva, J. Phillips, and L. M. Silveira. Effective Corner-Based Techniques for Variation-Aware IC Timing Verification. *IEEE Transactions on CAD*, 29(1):157–162, 2010.

[11] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, 1979.

[12] R. Garg, N. Jayakumar, and S. Khatri. On the Improvement of Statistical Timing Analysis. In *Proceedings of ICCD*, pages 37–42, 2006.

[13] V. Hrapčenko. Depth and Delay in a Network. *Soviet Math. Dokl.*, 19(4):1006–1009, 1978.

[14] J.-J. Liou, A. Krstic, L.-C. Wang, and K.-T. Cheng. False-Path-Aware Statistical Timing Analysis and Efficient Path Selection for Delay Testing and Timing Validation. In *Proceedings of DAC*, pages 566–569, New Orleans, LA, June 2002.

[15] J. Marques-Silva and K. A. Sakallah. Efficient and Robust Test-Generation Based Timing Analysis. In *Proceedings of ISCAS*, pages 303–306, 1994.

[16] P. McGeer, A. Saldanha, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli. Timing Analysis and Delay-Fault Test Generation Using Path Recursive Functions. In *Proceedings of ICCAD*, November 1991.

[17] E. Mendelson. *Introduction to Mathematical Logic*. Chapman & Hall / CRC, 1997.

[18] S. Onaissi, K. Heloue, and F. Najm. A Linear-Time Approach for Static Timing Analysis Covering All Process Corners. *IEEE Transactions on CAD*, 27(7):1291–1304, 2008.

[19] D. Tadesse, D. Sheffield, E. Lenge, R. I. Bahar, and J. Grodsteint. Accurate Timing Analysis using SAT and Pattern-Dependent Delay Models. In *Proceedings of DATE*, pages 1–6, 2007.

[20] C. Visweswariah, K. Ravindran, K. Kalafala, S. Walker, S. Narayan, D. Beece, J. Piaget, N. Venkateswaran, and J. Hemmett. First-Order Incremental Block-Based Statistical Timing Analysis. *IEEE Transactions on CAD*, 25(10):2170–2180, 2006.

[21] L. Xie, A. Davoodi, K. Saluja, and A. Sinkar. False Path Aware Timing Yield Estimation under Variability. In *Proceedings of the IEEE VLSI Test Symposium (VTS)*, pages 161–166, 2009.

[22] H. Yalcin and J. P. Hayes. Hierarchical Timing Analysis using Conditional Delays. In *Proceedings of ICCAD*, November 1995.

[23] J. Zeng, M. S. Abadir, J. Bhadra, and J. A. Abraham. Full Chip False Timing Path Identification: Applications to the PowerPC Microprocessors. In *Proceedings of DATE*, pages 1–5, Apr 2010.