Formal Verification of Timed VHDL Programs

A. Bara[1] P. Bazargan-Sabet[1] R. Chevallier[2]
E. Encrenaz[1] D. Ledu[1] P. Renault[1]
1 : LIP6, Université Pierre et Marie Curie (UPMC), CNRS, France
2 : STMicroelectronics, TR&D, Central CAD, Crolles1, France
This work is partially supported by French Research Agency ANR, project ARFU-VALMEM.

***Abstract*— The verification of timed digital circuits is an important issue. These circuits are composed by logical gates, each of them being associated with propagation delays. The analysis of such circuits is necessary to identify critical path and adjust the clock period of the circuit or to determine the stability period of input/ouput signals. These circuits are represented by a functional model described in VHDL and a timing model associating propagation delays to each functional block. This model is translated into timed automata formalism upon which classical simulation or model checking verification can be performed.**

This method rises two problems: 1) Propagation delays associated to a gate depend on the transistor assembly and the manufacturer's technology. How do we associate propagation delays to a logical gate ? 2) How to automatically translate a VHDL functional description, combined with propagation delays, into timed automata ? This paper addresses these two problems. It presents a method automating the verification of VHDL descriptions, augmented with interval bounded propagation delays, obtained by electrical simulation of the transistor model of the gates.

I. Introduction

Timing verification has been a critical factor in design flow because of the complexity of modern chip. At transistor level, the two main methods used to verify the timing characteristics of a digital circuit are : electrical simulation [1] and static timing analysis [10]. Electrical simulation gives very accurate results. But, due to the complexity of the transistors models and the number of transistor elements, electrical simulations are very slow. Another drawback is that electrical simulations are based on simulation and patterns are necessary. So, the exhaustiveness of electrical simulation depends on the number and the quality of the patterns used, and the result is not associated with an uncertainty margin. Static timing analysis methods (STA) give the longest path in a reasonable execution time. But, these methods do not take into account the correlation between signals. False paths have to be deleted from the list of longest paths provided by STA tools, and must be found by the circuit designer manually. This step can take a lot of time.

The alternative is to work at register-transfert level (RTL) using functional abstraction method [12]. Functional abstraction method computes the functional description of the block at RTL level from transistor level net list. The simulation performance is dramatically improved but this method is limited because it does not check the internal timings of the block. So, the RTL description of a digital circuit cannot be used directly for timing verification.

To go through these limitations, we propose to combine two different views of the behavior of a circuit :

- a functional view, described in RTL-synthesizable VHDL. This view is directly written by the designer (standard-cell approach), or has been abstracted from the transistor description (full-custom design)
- a timed view, associating to each circuit's element (either combinatorial or sequential) a set of propagation delays. These delays have been obtained by electrical simulation of the transistor model of each circuit's element.

These two views are merged into the timed automata formalism [2], which is devoted to the modeling and analysis of discrete events systems whose functionality highly depends on delays of actions. The obtained model can be used to perform timed simulations at RTL abstraction level, which speeds up simulation times.

Furthermore, timed automata formalism can be used to formally prove some complex timed properties, such as the maximal response time of the system, or the stability period of input and output signals *for a set of environment behaviors*, varying the input sequences and the delays between events. It can also give information on the margin of error of timing characteristics and extract some timed parametric constraints.

The subsequent sections describe our verification flow. In the second section an overview of the functional and timed extraction engine is developed. Its output is translated automatically into the formal model by a tool presented in section 3. The section 4 concentrates on timed formal verification applied to several circuits.

II. From Transistor Level to Timing Extraction

The problem of functional abstraction has been addressed since several years. Two main approaches have been employed.

The pattern matching technique [15] consists of identifying a known transistor structure inside the circuit's description seen as a transistor network. Each transistor structure is tied up to a gate with a given functional and timing behavior.

The alternative approach relies on a formal abstraction method. It consists of cutting the circuit into Channel Connected Components (CCC) [8]. Each CCC can be seen as a stand alone gate. The conduction conditions of paths that

connect the output of each CCC to Vdd or to Vss are then analyzed resulting to the boolean expression of the gate.

Our approach is based on this later technique and does not require a preliminary knowledge of the circuit. Although the abstraction process may be more complex to set up and more time consuming, from the timing analysis point of view this method confers a great advantage to the abstracted description. It ensures that the abstraction's result is an interconnection of CCCs, each CCC being electrically isolated from the other components. Thus, the timing characteristics of each CCC can be evaluated independently. This step is called the timing abstraction.

The timing abstraction involves a timing model and a timing characterization method.

A. Timing model

The timing model defines how the behavior of a gate is seen from the timing point of view. The paradigm of State Transition Graph (STG) can be used as a general model of the timing behavior. The concept is similar to [13] where a State Transition Graph for Power Estimation is presented. Here, each gate is abstracted as a set of states (graph vertices). The transition between two states (a graph's edge) that exhibits a modification of the gate's output state may be characterized from the timing point of view. For our concerns, the timing characteristics are reduced to the propagation delays from an input to the output.

Various kinds of STGs with different level of complexity and accuracy can be considered.

The simplest STG model is the **general inverter model**. The STG of each gate has only two states regardless of the number of its inputs. Each state represents an electrical level of the gate's output. The transition from one state to the other is characterized by the propagation delay to the rising or the falling edge of the output. Obviously, the input that produces the transition of the output does not appear in this model and the propagation delay should summarize all the different situations that may result to a rising or a falling transition of the output. This can be done by attributing to a graph edge the maximal, the minimal or the average delay of the different transitions or an interval of propagation delay. In all the cases, the ignorance of the source of transition affects the precision of the global timing verification step. The maximal delay through the overall circuit may be overestimated whereas the minimal delay tends to be underestimated.

The **input-output** STG, represents an *N*-input gate as a set of *N* independent graphs. Each graph describes the timing behavior of the output regarding the transition of a given input. This model incorporates the knowledge of the source of transition and offers a higher accuracy. Even though, the configuration of the other inputs during the transition is ignored. The lack of this information may introduce some error in the global timing verification due to the functional correlation of signals within the circuit. Figure 1 illustrates this situation. Assuming that the worst configuration for the transition of *A* to *D* is *B=1* and *C=0*, the correlation between

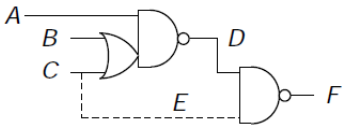

Fig. 1. Example of signal correlation

C and *E* inhibits this configuration. As a result, the overall propagation delay from *A* to *F* will be overestimated.

A more complex STG overcomes this inconvenient. In this model, each state represents a configuration of inputs. The transition between two states that produces a change on the output is characterized. In counter part, the complexity grows as 2^N for an *N*-input gate.

The **complete** STG is even more accurate with a higher complexity. It takes into account not only the configuration of inputs but also the state of the gate's internal nodes that may be charged or discharged.

In practice, the timing model and the number of propagation delays have a direct impact on the complexity of the timed automata. Hence, to reduce the expansion of these automata, we consider an intermediate STG where a state is coded as a configuration of inputs, but where multiple input transitions are excluded. Then, a functional analysis of signal correlations is applied to reduce the STG to those transitions that may effectively be produced in the circuit.

B. Timing characterization method

Regardless of the type of STG, a method should be defined to compute the timing characteristics associated with each edge. Obviously, an accurate evaluation of the propagation delays requires the knowledge of the gate's structure in terms of transistors. The wire that connects two gates has also a significant effect on the delay. These two informations should be preserved through the functional abstraction process. Two types of evaluation methods may be considered.

A first approach consists in setting up a direct expression of the propagation delay. This expression is derived from the resolution of the set of differential equations that denote the charge or the discharge of the gate's output through a specific path to Vdd or to Vss. Although this approach seems very attractive in terms of evaluation time, it shows a poor accuracy. In fact, the formal resolution of the differential equations implies a drastic simplification of the gate's structure, of the wire's description as well as of the transistor's model.

The second approach relies on the classic electrical simulation. The propagation delays can be extracted from a SPICE simulation of the gate as a stand alone circuit. The results tend to exhibit a high accuracy and the simulation time remains reasonable. Nevertheless, slight differences can be observed compared to the delays extracted from a simulation of the whole circuit. The differences come from 3 factors.

- The transition slope of the gate's input is of the mere importance. It should be as close as possible to the output slope of the gate connected to the input.

- The absence of the gate connected to the output make some coupling capacitance being neglected.
- The absence of the power grid and the IR-drop phenomena tends to underestimate the delays.

The experience shows that error introduced by these factors is less than 5% compared to a global simulation of the circuit.

As a result, the functional and timing abstraction step produces a VHDL description enclosing the functional aspect and a separate file describing the STG of each gate and the propagation delays.

III. Automatic translation of Timed VHDL into Timed Automata

This section introduces the timed automata formalism, presents the VHDL subset and timing model we consider, describes the translation algorithm and comment our modeling choices.

A. Basic model of Timed Automata

Roughly speaking, a timed automaton [2] is a finite state automaton enriched with *(symbolic) clocks* that evolve at the same uniform rate, and can be reset to zero. A *state* is a pair (ℓ, v) where ℓ is a *location* (or "control state"), and v a clock valuation. Each location is associated with a conjunction of linear constraints over clocks, called *invariant*. A state (ℓ, v) has a *discrete transition*, labeled e, to (ℓ', v') if v satisfies a constraint, called *guard*, associated to e, and v' is obtained from v by resetting certain clocks to 0. The state (ℓ, v) has a *time transition* of duration t to (ℓ, v') if $v' = v + t$ and for all t' $(0 \leq t' \leq t)$, $v + t'$ satisfies the invariant associated to ℓ.

The composition of two timed automata is obtained by synchronizing the actions labeling two (or more) transitions on emission of a signal q and simultaneous reception(s) of the same signal.

The network of timed automata may be analyzed through model-checking techniques : tools UPPAAL [11] or KRONOS [18] perform automatic verification of timed properties, expressed in TCTL; tool IMITATOR [3] extracts constraints on timing parameters ensuring a correct functioning.

B. VHDL subset and Timing model

The VHDL programs we consider are Data Flow descriptions, made of concurrent assignments representing combinatorial blocks and processes representing sequential elements (either latches, buffers or memory points). Each concurrent statement is responsible for the assignment of one signal, representing the output of the corresponding combinatorial or sequential block. The timing information are external to the VHDL description : two timing intervals are assigned to each block, representing the propagation of an edge from an input to the output; $\delta^{\uparrow}$ stands for the propagation of a rising edge on the output and $\delta^{\downarrow}$ for a falling one.

The timing part of the system is given in a external table stating, for each output of a combinatorial or sequential block, for each (valid) input configuration and input edge inducing an output edge, the direction of the edge of the output signal (either rising or falling), and the propagation delay between the input and output edges. From this table, one can easily extract intervals bounding the propagation delay of each signal's edge.

C. Translation algorithm

The timed analysis of combinatorial circuits with timed automata has been proposed by [14]. The generic gate model proposed by these authors is the basis of our translation method : a model for a combinatorial gate whose propagation delays are enclosed into a unique timed interval, whose delay is inertial, emulates the propagation of every transaction (in the VHDL sense) along each signal. With this model, asynchronous circuits composed of several gates are represented as a product of timed automata, each transaction occurring on a signal being propagated for the evaluation of subsequent signals. We propose the following extensions and provide an automated tool generating directly, from a VHDL description and a set of propagation delays, an input description for several model-checking tools, such as UPPAAL, HyTech, IMITATOR:

• Propagation of VHDL transactions is costly : each time a signal is written (even if the value written is similar to the current one), this induces the re-computation of values to be written to subsequent signals. This extra-computation increases the complexity of the region's graph summing-up the timed behaviors of the system, and it does not add useful information for the functional properties we are interested in. In our model, *signal's edges* are identified and propagated to subsequent part of the circuits. Edge detection imposes the use of extra variables to memorize the value of the signal *before* and *after* a writing action, however this simplifies further analysis.

• The circuits we consider have been manually designed and are tuned for timing performance. In this context, a unique inertial delay bound into one interval is a too coarse approximation. We adopt the *inertial and bi-bounded delay model*: the delay distribution is not uniform but concentrates around two picks: one corresponding to the *falling edge* and the other one to the *rising edge*. Management of two delays induces a greater complexity of each timed automaton, but restricts the non-determinism of the model since actions' firings are more constrained (moreover, for a given signal, the two intervals are generally thin but distant from each other).

• We consider combinatorial *and sequential blocks*. In previous works ([14],[7]), sequential elements are not considered or are supposed to be described at gate level ([9]) while in this paper we represent its macroscopical behavior. The transient behaviors relating to stabilization phases of sequential elements are abstracted into the delay propagation. The correct estimation of these stabilization delays is the most difficult part of timing extraction as it concerns conflicting gates.

The translation algorithm we propose is described below :

1) Identify and create global variables : for each VHDL block (either combinatorial or sequential), assigning a signal s, create :
   - clock x_s,
   - boolean variable v_s,

- synchronization labels $s^{\uparrow}$ and $s^{\downarrow}$,
- delays bounds $l_s^{\uparrow}$, $u_s^{\uparrow}$, $l_s^{\downarrow}$,$u_s^{\downarrow}$.

2) For each concurrent assignment, assigning signal s, create $TA(s)$ (cf. subsection III-C.1).
3) For each sequential process, assigning signal s, create $NTA(s)$ (cf. subsection III-C.2).
4) Instantiate delay parameters (cf. subsection III-C.3).
5) Insert a timed automaton for the environment of the circuit.

We obtain a network of timed automata whose timed traces emulate the signal edges propagations along the combinatorial and sequential elements of the circuit. The set of timed traces is complete : each potential execution of the circuit is represented by a timed trace in the timed automata network. The set of timed trace over-approximates the set of executions of the circuit : due to interval approximation, some timed traces may not correspond to real executions in the circuit. This case is reduced by using two intervals, distinguishing rising and falling edges, and being very thin.

1) Representation of a combinatorial block: Each combinatorial block, assigning a signal s is represented by a unique timed automaton $TA(s)$, composed of three states : *stable(s)*, *rising(s)* and *falling(s)*. In *stable(s)*, the value of s conforms to the input configuration (even if this latest changes). In *rising(s)* (resp. *falling(s)*), a new input configuration will induce a rising (resp. falling) edge on s, after a delay $\delta^{\uparrow}$ in-between two bounds $l_s^{\uparrow}$ and $u_s^{\uparrow}$ (resp. $\delta^{\downarrow}$ in-between two bounds $l_s^{\downarrow}$ and $u_s^{\downarrow}$). Transitions between *rising(s)* and *falling(s)* may occur in case of two successive edges on input configurations inducing two opposite edges on the output, occurring before the output for the first edge has stabilized. With this representation, only significant edges are computed and passed through subsequent gates, inducing the minimal computations. Fig. 2 presents the timed automaton associated with the combinatorial statement assigning signal `s1 <= i1 and i2`. For a sake of readability, synchronization labels have been omitted and transitions are designed by letters. In the Figure, transition labeled with $< b >$ (resp $< b' >$) refers to the occurrence of an edge on one signal among `i1`, `i2`, inducing a rising (resp falling) edge on `s1` when delay $\delta_{s1}^{\uparrow}$ (resp. $\delta_{s1}^{\downarrow}$) will have elapsed; the combinatorial block enters into a computation state, waiting for delay propagation elapsing (represented as state invariant : $x_{s_1} \leq u_{s1}^{\uparrow}$). Transition labeled with $< a >$ (resp. $< a' >$) refers to the production of the rising (resp. falling) edge of `s1` once the propagation delay has elapsed (represented as transition timed guard : $x_{s_1} \geq l_{s1}^{\uparrow}$). Transition $< c >$ (resp. $< c' >$) refers to the occurrence of an edge on an input signal, while the gate is into a computation state, if the input edge induces a value change of the output. Other transitions refer to the occurrence of an input edge which does not induce a change on the output `s1`.

2) Representation of a sequential block: A sequential block can be viewed as a collection of n exclusive guarded commands, whose variables are input and internal signals in a set e, and assigning a value to signal s. For $i \in [1..n]$, guarded command i is modeled by : $[g_i] : s \leftarrow f_i^s(e)$, where

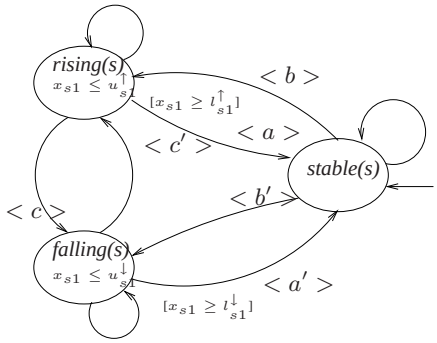

Fig. 2. Timed automaton associated with s1 assignment.

$f_i^s(e)$ represents the boolean function (whose support is e) assigned to s. All guarded commands are evaluated as soon as an edge on an element of e occurs : the guard evaluation is instantaneous, and if one is selected, say the i^{th} one, the corresponding command is performed. This latest consists in evaluating $f_i^s(e)$, and if its evaluation differs from the current value of s (stored in variable v_s), an edge (either rising $s^{\uparrow}$ or falling $s^{\downarrow}$) will be produced.

This computation is represented by a product of $n+1$ automata connected with n auxiliary variables g_i^s representing the truth value of the corresponding guard. The system is composed of n automata $TA(g_i^s)$, evaluation each guard g_i^s, plus one automaton $TA(s)$ assigning s.

The instantaneous evaluation of each guard g_i^s is represented by a distinct automaton $TA(g_i^s)$, each of them being activated on each edge on each signal in e. It is composed of two states, corresponding to the instantaneous value of the guard, either true or false. The guard is evaluated each time an edge occurs on one of its variable. Evaluation transitions are urgent (they do not let time elapse). For a given evaluation, among these n automata, only one may assign its variable g_i^s to true.

The assignment of signal s is represented by another automaton $TA(s)$ (represented on Fig. 3; for readability, all synchronization labels and transition guards have been omitted). Two states correspond to a stable output : state *stable(s)* corresponds to a case when at least one guard condition evaluates to true (the register is open and any change on its input may induce a delayed writing on its output), while state *close(s)* corresponds to the case when no condition evaluates to true and the output is locked. As for combinatorial blocks, all assignments producing a rising edge are merged into a unique state *rising(s)*, while all assignments producing a falling edge are merged into state *falling(s)*. This merge reduces the size of the automaton, but induces the merging of the timing intervals associated with the assignments. The upper bounds of timing interval are represented as state invariants, and one has to take the maximum of upper bounds of $\delta_s^{\uparrow}(i)$ as the common state invariants for *rising(s)* (and the same for state *falling(s)*). In Fig. 3, transitions $< a >$ to $< c' >$ have a similar meaning as on Fig. 2, when the process corresponds to an open register (the output signal is stable and exactly one guard is true).

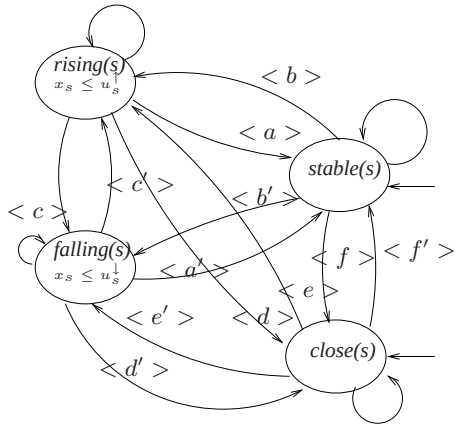

Fig. 3. Timed automata associated with s sequential assignment.

Transitions $< e >, < e' >, < f' >$ refer to the production of output when the process was closed (no guard was true) and an input edge opened it (one guard, say g_i^s becomes true). In this case, an evaluation of the output value is necessary: variable v_s will be assigned to the evaluation of f_i^s after a delay $\delta_s^{\uparrow}$ (or $\delta_s^{\downarrow}$, depending on whether f_i^s evaluates to 1 or 0); if this assignment changes the value of v_s, an edge (either $s^{\uparrow}$ or $s^{\downarrow}$) will be produced and will propagate as a triggering event for subsequent automata. Transitions $< d >, < d' >, < f >$ refer to the closing of the register.

3) Addition of timing constraints: In this model, for each internal or output signal s, delays $l_s^{\uparrow}, u_s^{\uparrow}$, $l_s^{\downarrow}$ and $u_s^{\downarrow}$ may be either set to real values or left as parameters, depending on the subsequent analysis to be performed. Each timed automaton is deterministic in actions but not in delays (except if $l_s^{\uparrow} = u_s^{\uparrow}$, and $l_s^{\downarrow} = u_s^{\downarrow}$, which is a common approximation). However, even in this case, the overall system may not be deterministic, due to the simultaneous occurrence of concurrent signals.

IV. Timed verification of combinatorial and sequential circuits

In this section, we present the timed verification we performed on a set of digital circuits. Some of them are described in the asynchronous design litterature and in these cases the functionality and timing were given by means of a logical circuit whose gates are associated with delays. Others circuits are transistor level descriptions of commercial products developed by STMicrolectronics: we had to abstract their functional behavior and extract the propagation delays of each abstracted block, as described in Sec. II.

Table I presents some of the circuits we analyzed, a more complete description can be found in [5]. The three first ones are combinatorial circuits while the two last ones are sequential. The columns of the table contain from left to right: circuit's name and reference paper (if any), number of combinatorial statements and process in the VHDL code, number of timed automata, clocks, variables, location and transitions in the zone graph, translation time from VHDL to UPPAAL, model-checking time with UPPAAL.

For each circuit, timed properties are expressed in timed logic TCTL and checked using model-checker UPPAAL. Timed properties refer to the instant of occurrence of certain signal's edges, considering other signals' edges fulfill some assumptions. For instance, for circuit spsmall-3x2, one timed property checked is :

$$AG((t \geq 0 \wedge t < 270 \Rightarrow q = 0) \wedge (t > 278 \Rightarrow q = 1))$$

This property states that for any execution trace of the model (i.e. for any propagation delay occurring between the definite bounds for each gate), the output signal q will eventually rise from 0 to 1 between 270 and 278 time units and remains stable afterwards. The verification of this property gives a guaranteed interval for the response time of the circuit, assuming the input signals conform to the timing constraints induced by the environment automaton. This property was checked with various environments, fixing the clock shape and period, and the setup timings of input signals. In Env1, the setup timings are set to a fixed value $(t_{setup_D} = 108, t_{setup_A} = 58, t_{setup_W} = 48)$. The property is satisfied within 10 mn and requires 100 MB of memory. In Env2, the setup timings are free within bounded intervals $(t_{setup_D} \in [81, 108], t_{setup_A} \in [33, 58], t_{setup_W} \in [32, 48])$. The property is also satisfied within 25 mn and requires 800 MB of memory. For this circuit, functional abstraction was performed in 14 s and timing extraction in about 1 hour. These results show the complementarity of timed model checking results versus electrical simulation: due to the dense time model and delays modeled within bounded intervals, we are able to compute safe uncertainty margins for input signals and find the largest timed bounds of some critical gates of the circuits.

In our experiments, once the functional and timed model is set, the translation into timed automata is straightforward. The distinction of propagation delays for rising and falling edges of each signal, and the accurate modeling of edge generation induce a complexity in automata but drastically restricts the non-determinism of the whole system and prunes extra non necessary computation. This explains the velocity of the model-checking analysis. As a comparison, previous attempts of timed model-checking of circuits were successful for smaller circuits: in [7], the authors analyze asynchronous circuits up to 30 gates. They propose a significant improvement for combinatorial circuits restricted to a unique edge on each input signal in [16], and analyze circuits up to 90 gates. Other authors ([9] propose to model sequential circuits at gate level, and perform parametric analysis to extract critical paths symbolically. This approach gives very rich results but is limited to very small portions of circuits (15 gates). With our modeling choices, we are able to analyze sequential circuits containing up to 100 combinatorial gates and 15 latches, supporting an environment with several edges on each input signal. This approach is helpful to analyze limited portions of circuit and is a complement to electrical simulation.

circuit's name	#comb / process	#TA	#clocks	#var	VHDL2TA time	#locations	#transitions	MC time
D flip-flop [9] [4]	5 / 0	6	5	7	0.1	10	10	< 0.1s
half [7]	7 / 0	9	7	9	0.1s	33	53	< 0.1s
sbuf-send-ctl [6]	14 / 0	17	14	17	0.1s	80	108	< 0.2s
spsmall-blueb-lsv2 [17]	25 / 6	32	31	35	0.3s	248	338	< 0.3s
spsmall-3x2	62 / 30	117	93	117	1mn	-	-	10mn

TABLE I

FUNCTIONAL AND TIMING ANALYSIS OF SEVERAL CIRCUITS.

V. CONCLUSION

We presented a methodology to perform accurate timing analysis of logical-gate level description of circuits. It combines timing evaluation of each gate and then timed and functional analysis of the whole set of gates. This analysis is achieved by translating VHDL description into timed automata formalism. We developed an efficient algorithm to generate a network of timed automata representing the functional and timed behavior of the circuit. The model obtained is rather efficient since we may analyze circuits containing one hundred gates. Three factors explain this good result :

- accurate timing extraction : this is mainly due to the fact that the delays associated to each block may be extracted in a independent way from the other gates. In order to refine some critical interval bounds, signal correlation analysis has been performed.
- timing non-determinism is restricted : associating two distinct intervals for the propagation of each signal edge is currently used in the electronic design community, but was not adopted in the timed model-checking analysis ([14]). Even if it doubles the number of delays to be considered, it highly restricts the timing non-determinism which is a critical factor to analyze medium sized circuits.
- propagation of real edge (value change) instead of "potential change" (value writing) : this reduces the evaluation of downstream automata, hence reduces the size of the reachability graph to be analyzed. The counterpart of this is the introduction of new variables to determine whether a computation will induce a change on a signal or not.

With these choices, the timed model proposed is quite complex, but grows linearly with the size of the VHDL description, nevertheless the reachability graph it induces is as small as possible. Thanks to our modeling choices, experimentations have shown that small-sized circuits may be analyzed with efficient timed automata model-checkers such as UPPAAL and results are complementary with those produced by electrical simulation or static analysis tools.

This approach offers a new opportunity to designers to automatically analyze functionality and timing of their circuits by bridging the gap between circuits'designers models written in VHDL and efficient timed model-checking tools. It can also be used to speed up the accurate timed simulation of complex circuits moving from transistor level description and electrical simulation to register transfer level and simulation. The transistor level model of each gate is necessary to set appropriate delays, but once the delay of each gate is extracted, the simulation of the whole system is performed at logical level. In this approach, the most difficult task remains the determination of accurate timings, particularly those of sequential elements involving conflicting gates. More work has to be performed to enhance this step.

REFERENCES

[1] John Wiley A. Vladimirescu and Sons. The Spice Book. 1994.

[2] Rajeev Alur and David Dill. A Theory of Timed Automata. *Theor. Comp. Sci.*, 126(2):183–235, 1994.

[3] É. André. Imitator: A tool for synthesizing constraints on timing bounds of timed automata. *In Martin Leucker and Carroll Morgan (eds.), ICTAC'09, LNCS*, 2009.

[4] É. André, T. Chatain, E. Encrenaz, and L. Fribourg. An inverse method for parametric timed automata. In *Int. Work. on Reachability Problems (RP'08)*. Elsevier, 2008.

[5] A. Bara. VHDL2TA: A Tool for Automatic Translation of VHDL Programs plus Timings into Timed Automata. *ANR-VALMEM Technical Report, 2009*, http://www.lsv.ens-cachan.fr/ encrenaz/valmem/vhdl2hytech.

[6] Peter A. Beerel, Cheng-Ta Hsieh, and Suhrid A. Wadekar. Estimation of energy consumption in speed-independent control circuits. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 15(6):672–680, 1996.

[7] M. Bozga, H. Jianmin, O. Maler, and S. Yovine. Verification of asynchronous circuits using timed automata. In *TPTS'02, ENTCS*, volume 65, 2002.

[8] R. E. Bryant. Boolean analysis of mos circuits. In *IEEE Transactions on Computer Aided Design*, 1987.

[9] R. Clarisó and J. Cortadella. Verification of timed circuits with symbolic delays. In *Proc. ASP-DAC*, pages 628–633, 2004.

[10] K. Dioury, A. Greiner, and M-M Louerat. Accurate static timing analysis for deep submicronic CMOS circuits. In *IFIP International Conference on Very Large Scale Integration (VLSI'1997)*, pages 439–450, Gramado, Brasil, August 1997.

[11] K. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1:134–152, 1997.

[12] A. Lester, P. Bazargan-Sabet, and Greiner A. Yagle, a Second generation Functional Abstractor for CMOS VLSI circuits. In *10th International Conference on Microelectronics*, pages 265–268, Monastir, Tunisia, December 1998.

[13] J. Lin, T. Liu, and W. Shen. A cell-based power estimation in CMOS combinational circuits. In *Proc. IEEE/ACM Int. Conf. on Computer-Aided Design*, pages 304–309, 1994.

[14] O. Maler and A. Pnueli. Timing analysis of asynchronous circuits using timed automata. In *Int. conf. on Correct Hardware Design and Verification Methods (CHARME)*, volume 987, pages 189–205. Springer, 1995.

[15] W. Nebel. Automatic extraction of rt-level description from integrated circuit layout data. In *Doctoral Thesis of Kaiserlauten University*, 1986.

[16] R. Ben Salah, M. Bozga, and O. Maler. On Timing Analysis of Combinatorial Circuits. In *FORMAT'03, LNCS*, pages 204–219, 2003.

[17] W. Xu. Timing analysis of SPSMALL. *Internal report LSV*, June 2006.

[18] S. Yovine. Kronos: A Verification Tool for Real-Time Systems. *International Journal on Software Tools for Technology Transfer*, 1:123–134, 1997.