Proceedings of the 10th WSEAS International Conference on CIRCUITS, Vouliagmeni, Athens, Greece, July 10-12, 2006 (pp302-307)

Timing Analysis of an Embedded Memory: SPSMALL *

[1]Remy Chevallier, [2]Emmanuelle Encrenaz-Tiphène, [2]Laurent Fribourg and [2]Weiwen Xu
[1]STMicroelectronics, FTM, Central R&D, Crolles, France
[2]LSV - CNRS, ENS de Cachan, France

Abstract: This paper proposes a high-level formalism, called Abstract Functional and Timing Graph (AFTG), for describing a memory architecture, which combines logical functionality and timing. After translation of the AFTG into the form a timed automaton, we are able to compute the response times of the modeled memory, and check their consistency with the values specified in the datasheet. We also address the problem of finding optimal values of setup timings.

Keywords: Formal Verification, Embedded Memory, Timing Analysis, Timed Automata

1 Introduction

The purpose of this paper is to verify the timing consistency of an embedded memory, named SPSMALL: given a transistor-level description of the memory and its interface specification (a *datasheet* that specifies the timing constraints to be satisfied by the memory), formally prove that the implementation satisfies the specification.

There are two main methods to compute the timings of the datasheet: electrical simulation and static timing analysis. The timing verification of memory is generally performed by electrical simulation (e.g. by tool HSIM [1]) at transistor level. However, as currents and voltages are simulated at transistor level, electrical simulation cannot be exhaustively applied to large circuits. Designers, in practice, have to apply simulation to restricted parts of the circuit, thus compromising the soundness of the whole process. An alternative to electrical simulation is "static timing analysis", e.g., with tool HiTAS [15]. This method constructs a graph of dependencies between signals, and computes the timing associated with the longest path of the graph. However, this analysis ignores the functionality of the circuit components, thus often yielding an over-approximation of the set of longest paths.

In order to formally verify that the implementation performances meet the values specified by the datasheet, some abstract structures have been proposed (timed-event structures [18, 6], timed symbolic simulation [10] and timed automata [3, 7, 14, 13, 11]). These structures combine functional and timing information, and have been successfully applied to the timing verification of hardware components, (e.g., [19] starting from a VHDL description, or [7] for speed-independent asynchronous circuits).

We will adopt here a similar approach based on timed model checking. We will first construct a model of the memory, starting from its transistor level description. This model (*Abstract Functional and Timing Graph*, in short AFTG) will incorporate the functionality and timings of the memory architecture. It can be easily translated into a product of timed automata, which is successfully exploited with timed model checking tools (e.g. UPPAAL [16]). We thus check that some end-to-end timing values of signal propagation ("response times") match the values of the datasheet. We also discuss the problem of computing optimal values of setup timings of input signals. We will focus on two distinct implementations of the SPSMALL memory, called $SP1$ and $SP2$.

This paper is organized as follows. Section 2 describes the behavior of the memory, and the properties related to its response times. Section 3 presents the principles of construction of the AFTG. In Section 4, we explain how to translate this AFTG into a product of timed automata, then give experimental results obtained with model checker UPPAAL (Section 5). We conclude in Section 6.

2 Memory Specification

The memory SPSMALL can be implemented using different transistor technologies. We consider here two implementations: $SP1$, which corresponds to a "high speed" technology, and $SP2$, which corresponds to a "low power" technology. The architecture is common to both implementations, but the propagation delays are specific to each technology.

2.1 Functionalities and Interface

The functionality of a memory circuit is to store and supply data at a given location, which involves two operations: *write* and *read*. In order to achieve such semantics, the memory has to be supplied with data (D), address (A) and control information (WEN) to indicate the type of operations (write or read) at the input port. The resulting data is produced at the output port Q. In practice, the memory is embedded into the synchronous environment scheduled by a periodic signal 'clock' (CK). The clock period t_{cycle} is made up of a high level period (T_H) and a low level period (T_L). This is depicted on Fig.1

*Partially supported by project MEDEA+ Blueberries

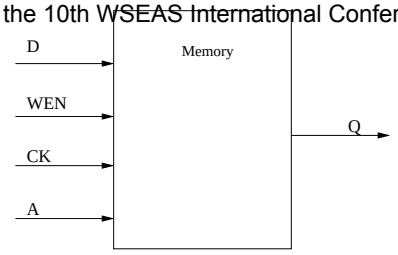

Figure 1: Basic interface of a memory

The value of signals occurring at input ports determines the memory operation: either *read* or *write* a data at an addressed location. In order to to be taken into account, each input signal I must remain stable *before* and *after* the rising edge of the clock. These delays are called $setup$ and $hold$ times for I, and denoted by t^I_{setup} and t^I_{hold} respectively.

A read (resp. write) operation requires a delay, denoted by $t^{read}_{CK\to Q}$ (resp. $t^{write}_{CK\to Q}$) due to the time of traversal of the elementary components of the memory. The specification states the maximum delay, denoted by t^{read}_{max} (resp. t^{write}_{max}) in the case of a read (resp. write) operation.

The set of timings $\{t^{read}_{max}, t^{write}_{max}, t_H, t_L, t^I_{setup}, t^I_{hold}\}$ constitutes the *interface specification* shown in Fig.2.

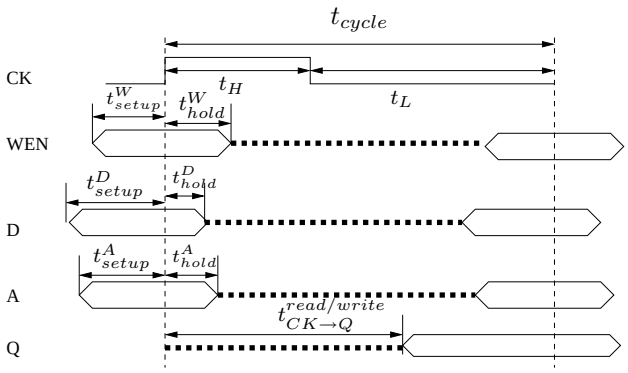

Figure 2: Interface Specification

This set of timings heavily depends on the transistor technology upon which the memory circuit relies. The values of the interface specification are usually determined by electrical simulation at transistor level. Actually, such a simulation process is much too long to be performed in a complete manner. Sensitive portions of the circuit, which are supposed to contain the longest paths of propagation, are therefore identified by hand. Electrical simulations are performed only for such limited portions of circuit, which are assumed to contain the critical paths. Such an assumption of 'criticality' is risky: it is very difficult to identify by hand relevant sensitive portions of the circuit (especially when the complexity of the circuit increases). The need for formal methods to *verify* the timings of the datasheet is therefore widely recognized.

2.2 Verified property

We will focus in this paper on the following property, expressing an important aspect of the timing correctness of the behavior of the memory.

The result of a *write* (resp. *read*) operation is produced on output port Q within t^{write}_{max} (resp. t^{read}_{max}). This will be expressed as: $t^{write}_{CK\to Q} \leq t^{write}_{max}$ (and similarly for the read operation). Besides, our analysis will allow us to find the optimal value for parameters t^I_{setup}, for any input signal I.

3 Abstract Functional and Timing Graph

We describe here an abstract level of representation of circuits, called "Abstract Functional and Timing Graph" (AFTG).

3.1 Level of modeling

In general, there are several levels to model circuits: transistor level, gate level, latch level and block level. We adopt the 'latch level' instead of the traditional 'gate level'. At this level of representation, the flow of input signals traverses the circuit in a linear manner (without loops), while the flow is cyclic at the gate level (the output of one gate can be an input of another gate and the converse can be simultaneously true). Note that this level is more precise than the block level used in [4, 5].

The circuit is then modeled in a two-steps construction: first, we build a *functional graph* representing the functionality of the memory at an abstract level (similar to the notion of "circuit graph" of [9]), then we add some timing information to this graph, representing the propagation delays through the components associated with the graph vertices.

3.2 Functional Graph

The functional graph associated with the memory then relies on the same principles as those described in [9]. The wires, latches and (blocks of) logical gates are represented as vertices (labelled by numbers). These vertices are interconnected by arcs labelled by signal names. Wire vertices have indegree and outdegree 1. Input (resp. output) vertices have indegree (resp. outdegree) 0. Wire vertices are drawn as "narrow" rectangles, functional vertices as circles and latches as "large" rectangles.

More precisely, the functional graph is constructed according to the following process:

- Model reduction:
  - reduction of a m words memory array to a one word array (the propagation delay is extracted from a m address decoder).
  - reduction of a n bits word to a one bit word (the taken propagation delay corresponds to the longest path from the address decoder to the memory location).

- Abstraction of the reduced model:
  - isolation of latches (as they correspond to breakage flow in the circuit),
  - insertion of logical gates between latches (as they shape the edges of the output signal).

In practice, the functional graph is constructed by manually extracting the functional components from VHDL code [12] (which is itself generated from transistor netlist by the tool TLL [2]).

3.3 Graph with Timings

In a second step, vertices are associated with delays: each vertex i is associated with two delays t_i^1 and t_i^0, which represent the time for a rising and falling edges to traverse i respectively. (We distinguish here the propagation of a rising edge from that of a falling edge, because the confusion of the two values would yield a too coarse approximation). We adopt the *inertial delay* interpretation (see [9]): given a vertex i, values on inputs of i that do not persist for t_i^1 (or t_i^0) time are filtered out.

In practice, once given the functional graph, associated timings are obtained by electrical simulation performed with tool HSIM [1].

The graph can be essentially decomposed into two different kinds of subcomponents: gates (with their input wires), and latches (with their input wires).

3.3.1 Gate component

A gate component can then be depicted as in Fig. 3. (For the sake of brevity, we consider the case of two inputs only.) It is composed of a functional vertex n preceded by two input wires k and ℓ. Signal i' corresponds to the input signal i delayed by the traversal of k. The value of the delay is represented by t_k^1/t_k^0, where t_k^1 (resp. t_k^0) corresponds to the rising (resp. falling) edge of i. The delay of traversal of the functional gate n is considered as null. (In other words, the propagation delay from an input wire to the output of the gate is totally assigned to the input wire, cf [17, 20].) The vertex n of the logical gate has a functionality f_n. The value of the output q is set to $f_n(i^{'}, j^{'})$, whenever one of its inputs $\{i^{'}, j^{'}\}$ changes.

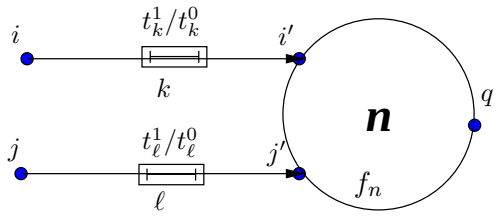

Figure 3: Graph of a gate component

3.3.2 Latch component

A latch component can then be depicted as in Fig. 4. It is composed of a latch vertex n preceded by two wires, denoted by k and ℓ respectively, where k propagates the data input (d'), and ℓ the enable input ($enable'$). Wires k and ℓ have their own inertial delays. The latch vertex n has a delay t_n^1/t_n^0, corresponding to the propagation of the data from d' to q when the latch is open ($enable' = 1$).

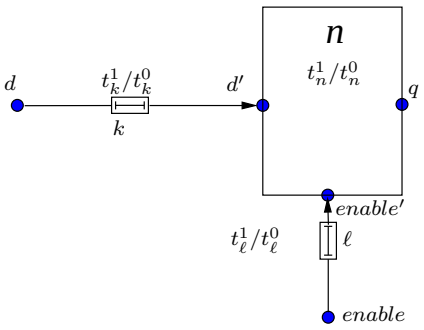

Figure 4: Graph of a latch component

As soon as the latch is open ($enable' = 1$), the data on $d^{'}$ is copied to q after t_n^1/t_n^0 according to the value of $d^{'}$. When the latch is closed, q remains stable no matter how the input $d^{'}$ changes.

3.3.3 AFTG of $SP1$ and $SP2$

By connecting the different subgraphs of gates and latches of the memory together, we obtain a global AFTG capturing the abstract architecture of SPSMALL. The inertial delays are instantiated with two different sets of values, depending on the considered implementation $SP1$ (see Fig. 5) or $SP2$ (see Fig. 6).

4 Verification Using Timed Automata

Once given an abstract representation like an AFTG, it is easy to translate it under the form of a "timed automaton" [3], and to compute the values of relevant response times using a model checker tool (see [4, 5]).

4.1 Timed Automata

The model of timed automata [3] is especially well-suited to represent asynchronous circuits: see, e.g., [8, 20]. Roughly speaking, a timed automaton 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*, labelled 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

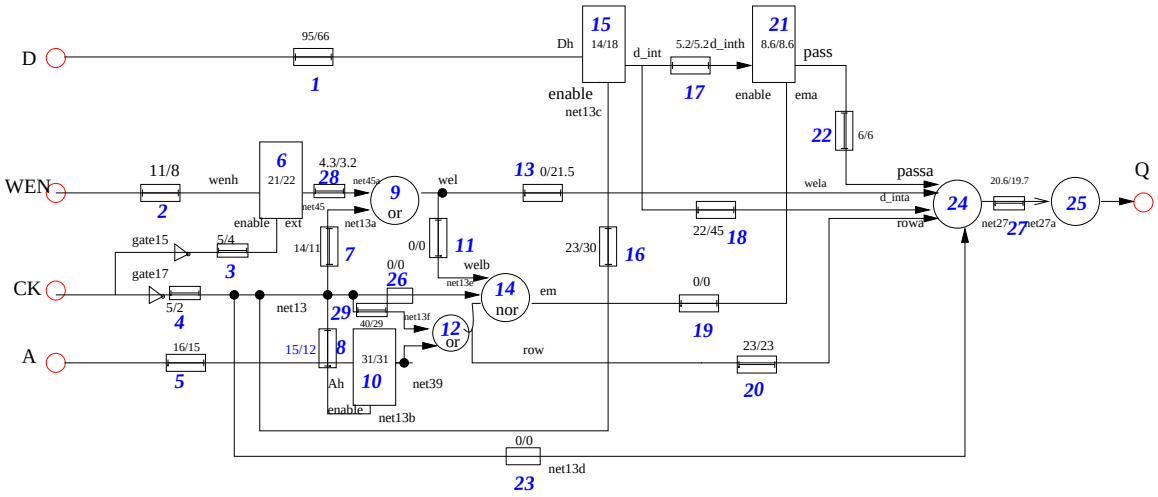

Figure 5: AFTG of SP1

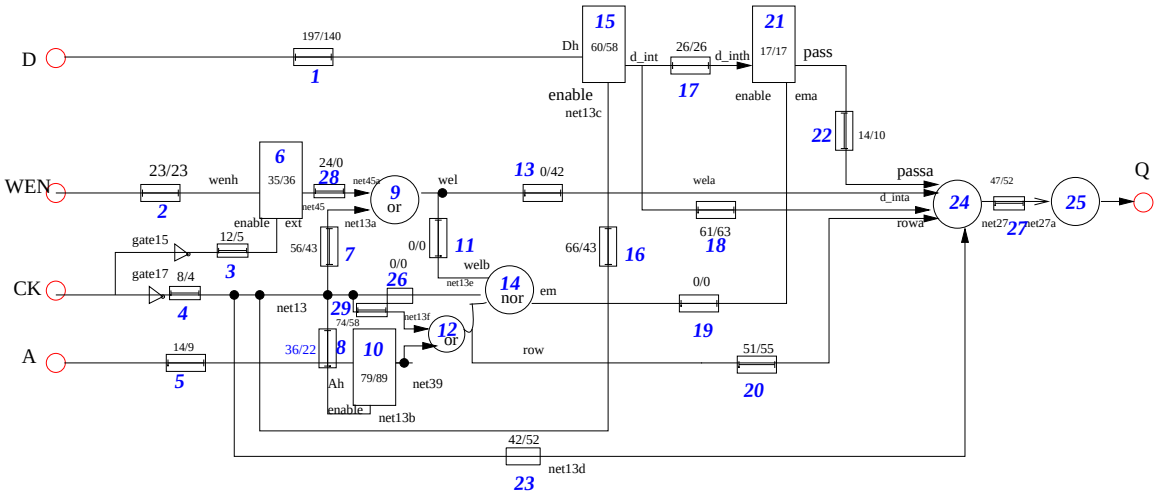

Figure 6: AFTG of SP2

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 labelling two transitions on emission of a signal q and simultaneous reception of the same signal.

In order to model the AFTG, each of its components is represented as a timed automaton combining its functionality and propagation delays. Input signals CK, D, A and WEN are themselves represented as timed automata. The AFTG is then modeled as a composition of timed automata where synchronization is used to model the transmission of an internal signal between two components. A central clock, which is never reset to 0, is used to measure the evolution of time.

4.2 Computation of Response Times

Using the model of timed automata, we are able to compute the maximal response time $t^{read}_{CK\to Q}$. This is done by using an additional "observer" timed automaton (see [5]). We are then able to check immediately the property $t^{read}_{CK\to Q} \leq t^{read}_{max}$, where t^{read}_{max} is the value specified by the datasheet. The same method applies for computing $t^{write}_{CK\to Q}$, and checking the property $t^{write}_{CK\to Q} \leq t^{write}_{max}$.

4.3 Optimal Values of Setup Timings

Intuitively, the response times are related to the "critical path" of the circuit. Such a critical path involves only some specific input signals, which allows us to safely reduce the setup timings of the other input signals (as long as this reduction does not modify the critical path).

Once we have computed the response times $t^{read}_{CK\to Q}$ and $t^{write}_{CK\to Q}$, one can try to find the optimal values of setup timings of input signals which preserve these response times. This is done by decrementing iteratively the setup timing values until this entails a modification in one of the response times.

5 Experimental Results

Experiments are performed using model checker UPPAAL [16]. Each property is verified within 5 minutes on a 1GHz PowerPC G4 with 512 MB of memory.

5.1 Computed Response Times

Table 1 gives the values of $t^{read}_{CK\to Q}$ (resp. $t^{write}_{CK\to Q}$) together with the value t^{read}_{max} (resp. t^{write}_{max}) of the datasheet for $SP1$. We immediately check the property: $t^{read}_{CK\to Q} \leq t^{read}_{max}$ (resp. $t^{write}_{CK\to Q} \leq t^{write}_{max}$).

Similarly, the computed response time for $SP2$ and the related datasheet values are given in table 2. We also check immediately: $t^{read}_{CK\to Q} \leq t^{read}_{max}$ (resp. $t^{write}_{CK\to Q} \leq t^{write}_{max}$).

computed response time	value of the datasheet
$t^{read}_{CK\to Q} = 74$	$t^{read}_{max} == 77$
$t^{write}_{CK\to Q} = 56$	$t^{write}_{max} == 56$

Table 1: Response times for SP1 (time unit =10 ps)

computed response time	value of the datasheet
$t^{read}_{CK\to Q} = 169$	$t^{read}_{max} == 169$
$t^{write}_{CK\to Q} = 142$	$t^{write}_{max} == 142$

Table 2: Response times for $SP2$ (time unit = 10 ps)

5.2 Optimal Values of Setup Timings

Table 3 gives three columns of values of setup timings of input signals t^I_{setup} for input signal I, in the case of $SP1$. The 1st column presents the optimal values computed with UPPAAL according to the method described in Sec. 4.3. The 2nd column gives the corresponding optimal values found by the designer using electrical simulation (with tool HSIM). The 3rd column gives the nominal values specified in the datasheet.

setup parameter	optimal value obtained by computation	optimal value obtained by simulation	value of the datasheet
t^D_{setup}	95	95	108
t^{WEN}_{setup}	29	36	48
t^A_{setup}	31	30	58

Table 3: Optimal setup timings for $SP1$ (time unit = 10 ps)

Analogous results for $SP2$ are given in table 4.

Almost all the optimal values computed by model checking and by electrical simulation are very close to each other. However, one computed value is significantly different from the simulated one: for $SP1$, the computed value t^{WEN}_{setup} is 29, which is much less than the value 36 obtained by simulation. This may indicate that (at least) one delay assigned to a vertex of the AFTG of $SP1$ (which has been computed by electrical simulation) is too approximative. Such a dependency on the precision of the simulation measures points out a limitation of our approach.

6 Conclusion

As a recapitulation, the main steps of our work are the following:

- construction of an abstract model (AFTG) for the SPSMALL memory,
- computation of the response times by translation of the AFTG model into a timed automaton, and model

setup parameter	optimal value obtained by computation	optimal value obtained by simulation	value of the datasheet
t^{D}_{setup}	229	229	241
t^{WEN}_{setup}	55	55	109
t^{A}_{setup}	73	74	110

Table 4: Optimal setup timings for $SP2$ (time unit = 10 ps)

checking using UPPAAL,

- verification of the consistency of the computed response times with the values of the datasheet,
- discussion of the problem of finding the optimal setup timings of the input signals.

This analysis provides us with satisfactory answers concerning the response timings. However, our method heavily relies on the accuracy of the measures done for evaluating the propagation delays through the internal components of the memory.

References

[1] HSIM simulator description. http://www.synopsys.com/products/mixedsignal/nsd /hsimplus_ds.html.

[2] TLL transistor abstraction tool description. http://www.transeda.com/products/datasheets/improve-tll.pdf.

[3] R. Alur and D.L. Dill. Automata-theoretic verification of real-time systems. *In Formal Methods for Real-Time Computing, Trends in Software Series*, pages 55–82, 1996.

[4] M. Baclet and R. Chevallier. Timed verification of the SPSMALL memory. In *Proceedings of the 1st International Conference on Memory Technology and Design (ICMTD'05)*, pages 89–92, Giens, France, May 2005.

[5] M. Baclet and R. Chevallier. Timed Verification of the SPSMALL Memory. Technical Report LSV-05-01, Laboratory Specification and Verification, 2005.

[6] W. Belluomini and C.-J. Myers. Timed circuit verification using TEL structures. *IEEE Transactions on CAD*, 20(1):129–146, January 2001.

[7] M. Bozga, J.Hu, O. Maler, and S. Yovine. Verification of asynchronous circuits using timed automata. In *Proceedings of TPTS'02 Workshop*. Elsevier, April 2002.

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

[9] J.A. Brzozowski and C-J.H. Seger. *Asynchronous Circuits*. Springer, 1994.

[10] Supratik Chakraborty, David L. Dill, and Kenneth Y. Yun. Min-max timing analysis and an application to asynchronous circuits. *Proceedings of the IEEE*, 87(2):332–346, 1999.

[11] C.-L. Chen, T. Lin, , and H.-C. Yen. Modelling and analysis of asynchronous circuits and timing diagrams using parametric timed automata. In *Modelling, Identification, and Control*. ACTA press, 2004.

[12] R. Chevallier. Study of a SPSMALL memory. Technical report, STMicroelectronics, 2004.

[13] R. Clarisó and J. Cortadella. Verification of concurrent systems with parametric delays using octahedra. In *Proc. 5th International Conference on Application of Concurrency to System Design (ACSD'05)*. IEEE Computer Society Press, June 2005.

[14] Robert Clarisó and Jordi Cortadella. Verification of timed circuits with symbolic delays. In Masaharu Imai, editor, *ASP-DAC*, pages 628–633. IEEE, 2004.

[15] K. Dioury, A. Lester, A. Debreil, G. Avot, A. Greiner, and M. Louerat. Hierarchical static timing analysis at bull with HiTAS. In *Proc. of the Design, Automation and Test in Europe User Forum*, pages 55–60, March 2000.

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

[17] O. Maler and A. Pnueli. Timing analysis of asynchronous circuits using timed automata. In *CHARME'95, LNCS 987, Springer, pp.189-205*, 1995.

[18] M. Pena, J. Cortadella, A. Kondratyev, and E. Pastor. Formal verification of safety properties in timed circuits. In *Proc. Int. Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 2–11, April 2000.

[19] Jürgen Ruf and Thomas Kropf. Analyzing real-time systems. In *DATE*, pages 243–. IEEE Computer Society, 2000.

[20] R. Ben Salah, M. Bozga, , and O. Maler. On timing analysis of combinational circuits. In *FORMATS'03, LNCS 2791, Springer, pp.204-219*, 2003.