Task T4 - Livrables D4.2 and D4.3 - established at $T_0 + 48$
Experiments of Prototype Tools on Case Studies, Comparison of obtained results and Conclusion

É. André, A. Bara, P. Bazargan-Sabet, R. Chevallier, D. Ledu, E. Encrenaz, L. Fribourg, P. Renault

LIP6 – Université Pierre et Marie Curie
LSV – ENS de Cachan
STMicroelectronics

This document is a merge of livrables D4.2 ”Experiments of Prototype Tools on Cases Studies” and D4.3 ”Comparison of obtained results and Conclusion” in the initial proposition. It concludes the VALMEM project by applying the set of tools developed during the project to (some of) the case studies defined initially. The obtained results are discussed and compared with standard methodologies outcomes. In the initial proposition, these two documents were separated since we planned to apply our flow to several case studies (SPSMALL memory, SPREG memory, including self-timed logics). At the end of the project, we were able to apply it to the SPSMALL memory, but the particular mechanisms of SPREG have not been introduced into the preliminary stages of our flow. Hence we found preferable to merge the two documents into a unique one.

This document is structured as follows: the first part recalls our methodological flow. Parts 2 to 6 describe the application of this flow to SPSMALL memory. The architecture and specificities of SPSMALL are recalled in section 2; abstraction and timing extraction are applied in section 3; formal analysis performed by timed-model checking and parametric timed model-checking are reported into sections 4 and 5. Encountered and remaining difficulties are commented in section 6. Comparisons and conclusions are drawn in section 7.

1 Analysis Flow of full custom memory proposed in the VALMEM project

The Functional and Timing analysis problem we concentrate on can be expressed in the following way.
Given :

- a full-custom memory circuit described at transistor level, for a given technology,
- a specification provided by the manufacturer, describing (1) the conditions to be met by the environment (called nominal conditions), (2) the guaranteed performances of the memory (namely the *access timings*) assuming these nominal conditions are met.

Determine :

– the correctness of the access timings given in the specification,
– the extremal stability periods of environment signals still guarantying the functionality and the access timings of the memory.

1.1 Analysis

Our analysis flow is composed of three major steps which are depicted in Fig. 1:

– **Functional Abstraction and Temporal Extraction** : the transistor netlist is functionally abstracted and once the functional blocks (either elementary gates or combination of several gates) have been identified, the delay propagation inside each block is computed through electrical simulation.
– **Modeling as a product of timed automata** : each functional block is modeled as a timed automaton propagating – after an appropriate delay– *edges* on output signals whenever *edges* occur on input signals. Internal delays are fixed according to the values computed in the previous step, while delays referring to environment signals are left unbound (they act as *parameters*).
– **Model-checking** : The timing analysis is based on the traversal of the state-space of the product of timed automata; the good behaviors of the systems are characterized as a set of states to reach in a unavoidable manner. Two kinds of analysis are performed : one may *verify* that, under the nominal conditions given in the specification, the access timing (also given by the specification) are met [1]; this is done by classical model-checking techniques provided in tool Uppaal [2] or Kronos [3], however, the size of the description is much bigger than in the case of [1]. Alternatively, one may compute a constraint binding delay parameters, ensuring a good behavior (e.g. the preservation of the critical path) [4]: a guided (parametric) state-space traversal of the product of timed automata is performed and constraints binding parameters, ensuring the convergence into these "good regions of states" are generated on the fly. This is performed through an original algorithm making use of tool IMITATOR-2 [5].

To automate this flow, several prototype tools have been developed during the project:

– MYGALE : functional abstraction from transistor netlist to untimed VHDL
– TIMEX : script for timing extraction
– VHDL2TA : generation of product of timed automata from untimed VHDL plus STG of each functional block
– IMITATOR-2 : extraction of linear constraints of timed parameters, ensuring correct functioning of the system.

2 Architecture of SPSMALL

The SPSMALL is a SRAM compiler. Indeed, the designers are not developing one memory by customer but several memory compilers in order to address

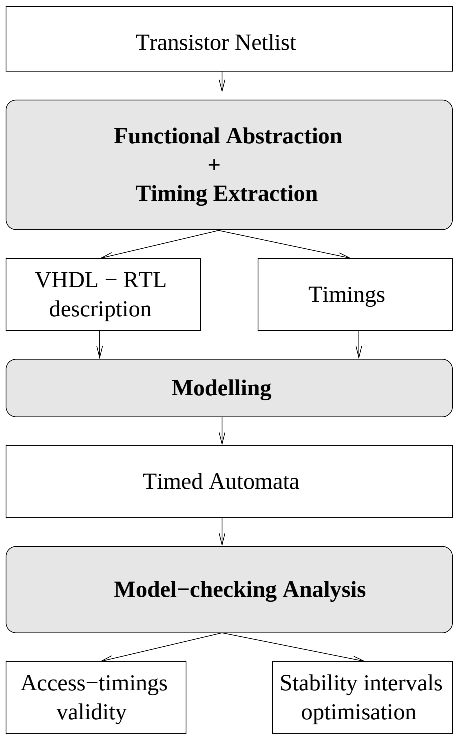


Fig. 1. Analysis flow.

the various needs of each customer. The SPSMALL is a key generator in ST offer: it is answering to the designer need for small and robust memory with a competitive area and performances.

2.1 Global overview

SPSMALL generates single port memories: Only one read or one write can be performed during a cycle. A write cycle is performed when WEN (write enable active low) is equal to zero and a read when this signal is set at high. Moreover, these memories are synchronous: the clock signal is called CK. Two others command signals are available: CSN is used to enable the block when it is set at low, and OEN which puts the output of the memory at 'z' state when it is set high. The data are provided through the vector D, and the address is defined with the vector A. The architecture of this SRAM is divided into 3 main pieces (cf. Fig. 2, namely the Input Saver, the Memory Array and the Output Manager:

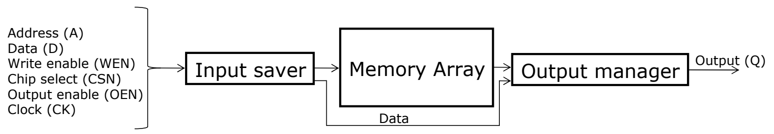


Fig. 2. Global Architecture of SPSMALL Memory.

2.2 Input saver

The input saver is saving the control and the data signals. In addition, it is generating internal clock signals to enable the read/write process in the memory array. The data is also provided to the output manager for a potential direct copy to the output. This feature could be very useful for silicon testing with ATPG.

2.3 Memory array

The memory array block is embedding the banks of the memory. This memory family does not use sense-amplifier. As a consequence, real 'one' and real 'zero' is generated at the output of this block during the read cycle.

2.4 Output manager

The output manager is a kind of big multiplexor. If the output enable (OEN) is disabled (OEN at high), the output stay at high impedance state called also 'z' state. Else, if the memory is in read cycle, the data saved inside the memory array is copied at the output of the memory (cf. Fig. 3). If the memory is in write cycle, the input data are saved inside the SRAM and copied to the output Q (cf. Fig. 4).

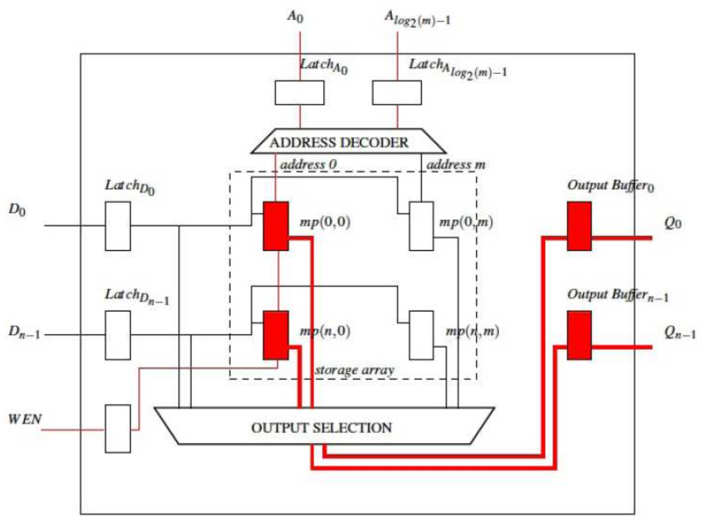


Fig. 3. Architecture and data transfers during a read cycle.

2.5 Design strategies

Strategy used in digital flow .

In a full digital and synchronous flow, the functions are spread on several clock cycles. Indeed, each operation is computed and saved on a rising edge sensitive register called flip flop. The operations are based on pure standard-cells based functions called logic cone. When a function is computed, the data is saved in the next flip-flop and reused during the next cycle to compute the next function. Each flip flop are driven by the same clock signal.

As a consequence, if the time to compute the function is kept, the final value saved on the next flip-flop is independent to the various delays of each standard-cell. Thus, the timing closure and the functional verification can be performed independently. In this case, the formal methods can be used without timing to verify the functionality. The only reference to the timing is the clock cycles spent during the verification.

Full custom flow .

In a full custom flow design, like the SRAM studied in the VALMEM program, this flow cannot be applied. Indeed, in order to save time, the memory registers are replaced by latches (saving the data at logic one or logic zero). It implies that the different functions are not independent: The strategy used on

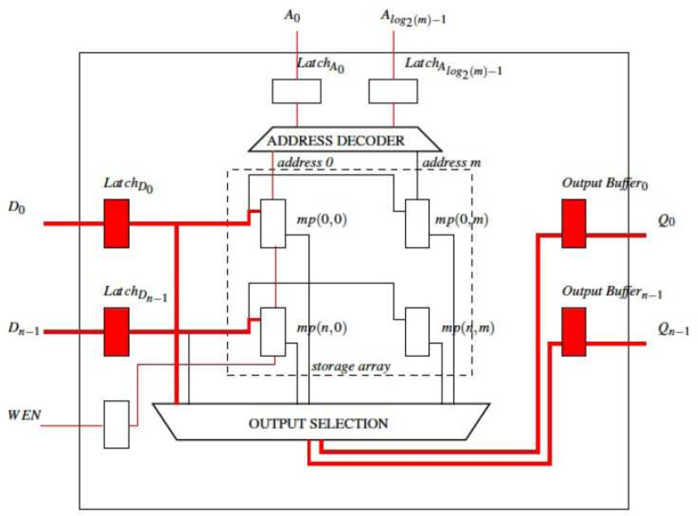


Fig. 4. Architecture and data transfers during a write cycle.

full digital flow cannot be applied. Therefore, the formal methods must involve the timing/delay of functions in addition to the clock cycle. To address this huge challenge, an abstraction tool has been built to bring the right level of abstraction in order to enable the usage of formal engine with timed automaton. The main constrains are: " No logic loop: the latches must be detected " No in-out pins: just in pins and out pins

The main idea is to compute the delay of each function on one side and map each boundaries of timing in each function. In this strategy, the timing analysis is performed on one side, and the functional abstraction independently on the other side. Unfortunately, we prove during this project that this strategy cannot be used in all cases.

Indeed, if the memory point used in the 6T SRAM is studied, we conclude that there is no read and no write ports, but 2 differentials in/out (Bit Line True and Bit Line False) ports.

The abstraction is then generated with: " One input port, based on the logic controlling WLi+1 and the data for the write cycle. " The output is modeling the read cycle.

The timing of each function cannot be mapped directly to the functional model, because it depends to the type of the cycle checked. As a consequence, when a read cycle is performed, the timing constraints will not be the same as when a write is performed. To conclude the timing computation cannot be

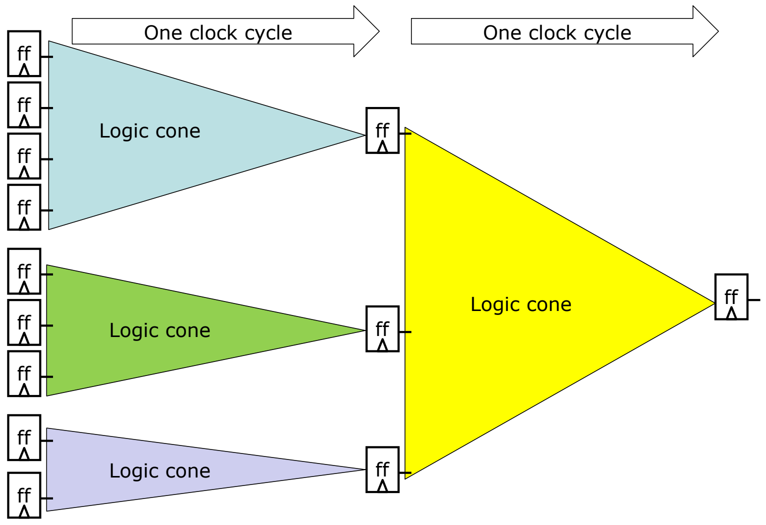


Fig. 5. Two-phases functioning of the memory.

independent to the functional abstraction and cannot be plugged directly: some loop and inter-actions between functional abstraction and timing computations are mandatory.

3 Functional and Timing Abstraction

Functional and timing abstractions aim at filling the gap between the description issued from the designers and the description accepted by the formal verification tool based on timed automata.

The functional abstraction builds a network of Boolean functions from the transistor level description. The resulted network is then processed by the timing abstraction which characterizes the propagation delays through the Boolean functions.

3.1 Functional Abstraction

The functional abstraction is the first step in the process of abstracting a circuit described as a network of transistors into a timed-RTL (Register Transfer Level) description.

In the framework of Valmem project, the circuit represents the so called critical cut of an embedded memory. The critical cut has the same topology

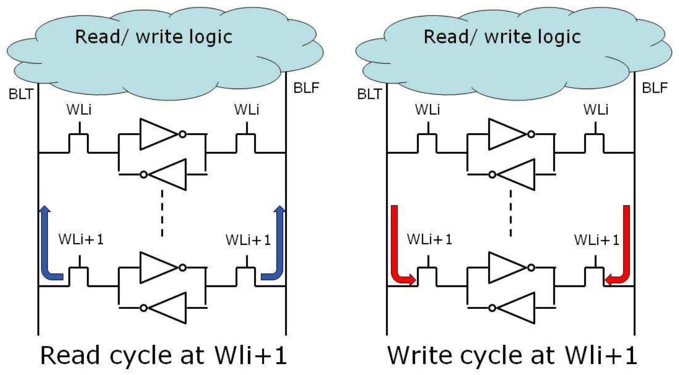


Fig. 6. Use of the same transistors in different modes.

as the real memory except that the memory array contains only those memory points that participate to the critical path of the circuit.

A preliminary step consists in obtaining a netlist of transistors, capacitors and resistors from the physical layout of the circuit. This is accomplished by a layout extraction tool.

Then, the function abstraction provides a gate level description from this transistor level description. This task may be achieved by using two different approaches.

The first approach is based on pattern matching technique. The structure of each gate that may be found in the circuit is described as an interconnection of transistors. These descriptions constitute a collection of patterns. In this collection, the structure of each gate, in terms of transistors, may be seen as a graph. The abstraction process tends to identify, inside the circuit, the structures that are similar to the structure of a gate of the collection. In other terms, considering the circuit as a graph, this first technique consists in matching the graph of a gate with a sub-graph of the circuit. When the matching succeeds, the sub-graph is replaced by its abstracted view, i.e. the Boolean expression of the gate.

An alternative approach relies on a formal method. It aims at identifying the function implemented by a group of transistors. The signals connected to the transistors' grid are considered. These signals should be the output of a gate. Starting from such a signal, the abstraction process traces all the paths that connect the signal to V_{dd} or to V_{ss}. Each path is a series of channel connected transistors and is called a branch. The collection of branches represents the structure of a potential gate.

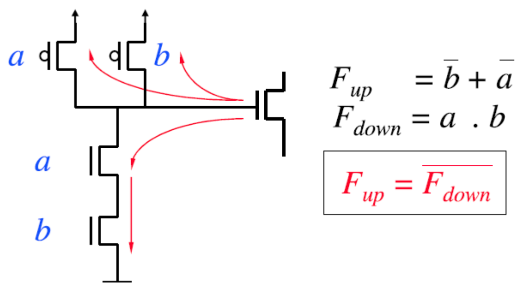

a
b
a
b
$F_{up} = \overline{b} + \overline{a}$
$F_{down} = a \, . \, b$
$F_{up} = \overline{F_{down}}$

Fig. 7. Functional abstraction of a regular gate using formal method

The Boolean expression of the gate is then obtained from the conditions that makes the branches being conducting. For each gate, two Boolean expressions may be considered. F_{up} is the condition that makes the signal being connected to V_{dd}. F_{down} is the condition that makes the signal being connected to V_{ss}. As long as F_{up} and F_{down} are complementary the gate is a dual gate and its Boolean function is F_{up}. However, the structure of some gates may be non dual. This happen when F_{up} and F_{down} are not complementary requiring a deeper analysis to identify the exact Boolean function of the gate.

In practice, a functional abstraction tool combines both techniques. Traditionally, regular gates are recognized by the formal approach. The analysis of non regular and complex gates such as precharged signals, memory points or analogue devices is harder and seems to be unreachable by the formal approach. Thus, these parts are abstracted by pattern matching.

Nevertheless, proceeding to the functional abstraction in such a way is far to be consistent. Most of the time, the part of the circuit that includes regular structures is build using standard cell library. Complex and non regular structures are implemented using a full custom design approach.

Therefore, considering the designers' point of view, the functional abstraction of regular gates should be achieved by the pattern matching since the standard cell library provides the structure of the gates used to build the circuit. On the contrary, the part of the circuit designed using the full custom approach should

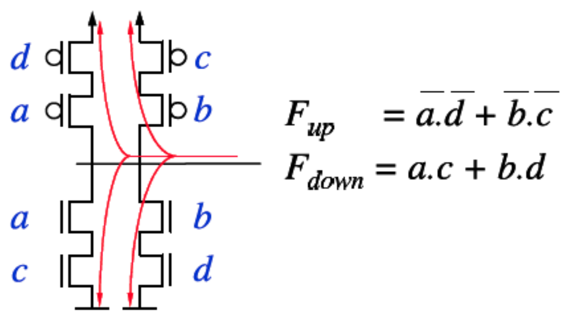

d
a
c
b
a
c
b
d
$F_{up} = \overline{a}.\overline{d} + \overline{b}.\overline{c}$
$F_{down} = a.c + b.d$

Fig. 8. Functional abstraction of a conflictual gate using the formal approach

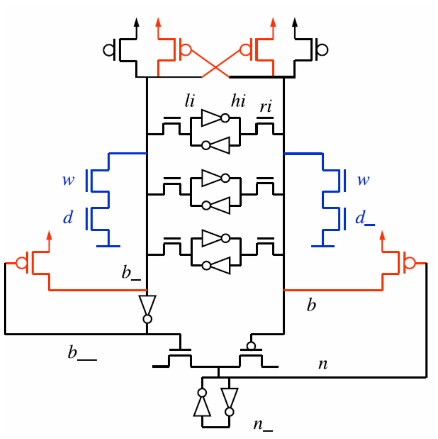
be abstracted through the formal approach since the scheme of the gates is not available. Yet, traditional functional abstraction tools work on the opposite way.

In the framework of this project, the challenge was to extend the application of formal abstraction methods to complex and non regular structures to avoid an overhead of work for the designers. An other advantage of this approach is that the resulted abstracted network represents the circuit as an interconnection of CCCs (Channel Connected Components). Each CCC is electrically isolated from the other CCCs and can be characterized separately from the timing point of view.

The main obstacle to this extension is the analysis of conflictual gates. At first glance, the F_{up} and F_{down} of complex gates are not complementary. A conflict condition arises when both F_{up} and F_{down} may be activated for a given combination of the inputs. Then, the problem is to prove, by exploring the gate's environment that, due to the spatial correlation of signals, the conflict conditions cannot be realized. The solution resides in an efficient algorithm able to identify the spatial correlation of signals inside the circuit.

An other problem comes from structures presenting a circular dependency. A circular dependency is detected when the input of a potential gate depends through a series of other gates on its own output. Most of the time, a circular dependency denotes the presence of a memory point. In such a situation, the extraction of the gate's Boolean function needing the exploration of its environment will eventually require the knowledge of Boolean function of the gate. The solution resides in a formal resolution of the circular dependencies represented as a system of Boolean equations.

To overcome these difficulties, we have developed two original algorithms. A first technique, based on graph coloration, aims at the identification of sources



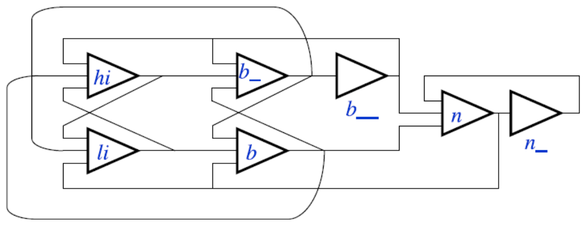


Fig. 9. Spsmall : a column of bits and the circular dependencies of the gates

of correlation inside the circuits. The second algorithm targets the resolution of systems of Boolean equations and relies of the calculation of Boolean derivatives.

The two methods extend the scope of formal functional abstraction to a new class of complex and non regular gates. As a consequence, the full abstraction of Spsmall has been made reachable using the formal approach.

Even if the formal abstraction method proposed within the framework of this project is general and does not rely on the specific structures of Spsmall's gates, it is not yet able to cover the abstraction of all classes of memories. For instance, the formal abstraction of self timed memories requires another type of analysis namely, the identification of timing correlation between signals.

3.2 Timing Abstraction

The problem of extracting timing information such as propagation delays is a classic issue in timing verification. In our proposed approach, the description abstracted from the transistor level is analyzed to obtain the local timing characteristics of each abstracted gate.

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

The timing model defines defines how the behavior of a gate is seen form 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 [**?**] 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 kind 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.

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 step due to the functional correlation of signals inside the circuit. Figure 10 illustrates this situation. Assuming that the worst configuration for the transition of A to D is B=*1* and C=*0*, the correlation between C and E inhibits this configuration. As a result, the overall propagation delay from A to F will be overestimated.

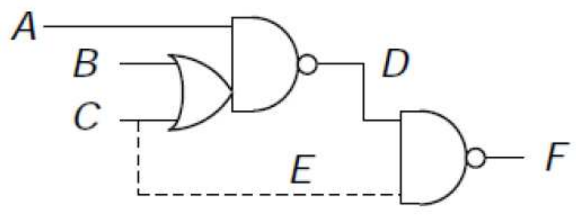


Fig. 10. Example of signal correlation

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.

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 remove from the STG those transitions that may not be produced in the circuit.

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 V_{dd} or to V_{ss}. 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.

In the framework of Valmem project an alternative approach is used. It 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 result from a combinaison of 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 the to underestimate the delays.

Considering the narrowness of the timing marigin within the specifications of Spsmall, the application of the above approaches to this case presents at least two difficulties.

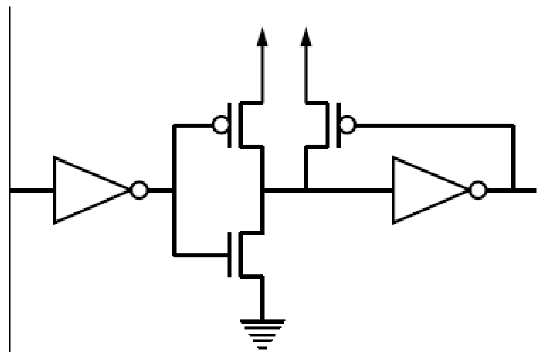


Fig. 11. Example of timing correlation producing multiple input transitions

First, the environment of some abstracted gates in the circuit is such that several inputs may switch in the same time. However, to reduce the complexity of timed automata, the type of STG upon which the timing abstraction is build does not consider multiple input transitions. Figure 11, extracted from the abstracted view of Spsmall, shows an example of such situation.

The second problem resides in the characterization of precharged signals. For a precharged signal, the rising edge of the output can only be produced by a transition of the clock signal. Simetrically, the falling transition of the output may only be produced during the evaluation phase of the clock, on the transition of another input signal. Therefore, the spatial correlation analysis implemented in the timing abstraction is not enough to eliminate the non functional input configurations from the STG. To address this problem, a more powerfull representation of precharged signals is required. This representation should likely be based on finite state machine formalism.

These difficulties combined to the small margin of the Spsmall's specifications make the timing characterization, resulted from the abstraction, not fit within the specified margin.

4 Transformation into Timed Automata and Analysis of the Instanciated Timed Model

VHDL2TA Tool performs the automatic translation into a network of timed automata from a functional description given in VHDL and a set of timing delays associated with each functional block of the VHDL description. In the Timed Automata model, the set of timings of each blocks are abstracted into two delay intervals, one corresponding to the propagation of a rising edge on the output of the block, and one corresponding to the propagation of a falling edge. The output format is either compatible with model checkers UPPAAL or HyTech, or with the timing constraint synthesis tool IMITATOR-2. The timed model and the translation principles have been described in [6].

4.1 Translation in Timed Automata

The VHDL file describes the functionality of the SPSMALL 3×2 memory, automatically abstracted by LIP6 . The VHDL description contains

- 62 concurrent assignments, representing combinatorial blocks and
- 30 processes, representing either latchs, memory points and output buffers.

Associated with each block (either combinatorial or sequential), an external file describes the set of propagation for all configurations.

Tool VHDL2TA generates a network of timed automata, instantiates the timing parameters with the bounds of delay intervals computed from the timing files, and adds an environment automaton mimicking the scenarii to be evaluated. The UPPAAL representation of the memory is composed of :

- 92 timed automata for logical gates and latches, plus 24 automata with urgent transitions evaluating complex boolean guards, plus 1 automaton for the environment.

– 93 clocks, $(92 + 24)$ boolean variables, (92×4) delay parameters.

The generation produces 18087 lines of code for the UPPAAL description and is generated within 1 mn.

4.2 Analysis by Model-Checking with UPPAAL

Timed properties have been evaluated on the Instantiated Timed Automata model through the use of model checker UPPAAL. Here we present a particular environment corresponding to a common use of the memory (cf. Fig. 12). Is is composed of a *read* operation at address 0, followed by a *write* operation at address 1. As SPSMALL is a write-through memory, the data written is present at the output after delay $t_{CK \rightarrow Q}$, producing a rising edge of output signal Q_0.

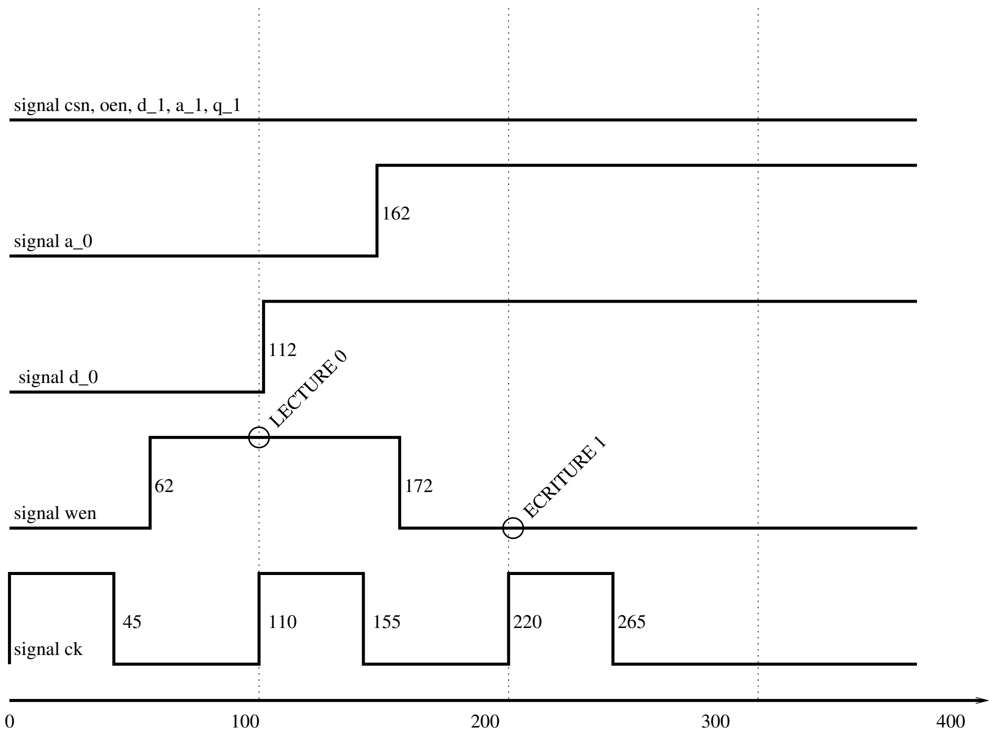


Fig. 12. Environment for the verification of SPSMALL Memory.

We consider the delay given in the specification to determine the instants of occurrence of edges for signals of the environment:

– Clock shape : thi 36 tlo 74;

– D rising edge : d_0 : 112 up; (this means $t^{D}_{setup} = 108$)
– WEN rising and falling edges : wen : 62 up, 172 down; (this means $t^{WEN}_{setup} = 48$)
– Address rising edge : a_0 : 162 up; (this means $t^{A}_{setup} = 58$).

The maximal response time of the memory is given by the specification : taaw = 56. (Timings of the specification have been established by STMicroelectronics designers by electrical simulations).

With UPPAAL, we proved the following TCTL property:
$\mathrm{AG}((t \geq 0 \wedge t < 253) \implies q\_0 = 0) \wedge (t > 304 \implies q\_0 = 1))$

The property is successfully evaluated in 10 mn and requires 200 MB of memory. The satisfaction of this property implies that $t_{CK \to Q} \in [33, 84]$. This interval encompasses the response time given by the specification, however it is too large: the timed automata model built is a too coarse approximation of the memory.

This is mainly due to the range of delay intervals of four identified signals : $b\_0, b\_1, b0$ and $b1$. For these signals, the interval range obtained by the electrical simulation of the logical block disconnected from its environment is too wide. We have to restrict the delay intervals by eliminating input configurations that will never occur during the functioning of the memory.

The reduction is performed in two steps :

– an automated one which eliminates input configurations that are not compatible with the VHDL description, or correspond to multiple transitions.
– a manual one which restricts the input configurations to exactly those in use for each operation (either read or write).

The application of these reductions and its impact on the range of delay intervals is given in Table 1.

signal name	initial delays	automatic reduction	manual reduction
b0	$\delta^{\downarrow} = [0,38]$,$\delta^{\uparrow} = [1,39]$	$\delta^{\downarrow} = [5,38]$,$\delta^{\uparrow} = [1,37]$	$\delta^{\downarrow} = [17,19]$,$\delta^{\uparrow} = [1,37]$
b1	$\delta^{\downarrow} = [0,37]$,$\delta^{\uparrow} = [0,39]$	$\delta^{\downarrow} = [0,37]$,$\delta^{\uparrow} = [0,39]$	$\delta^{\downarrow} = [17,19]$,$\delta^{\uparrow} = [0,39]$
b_0	$\delta^{\downarrow} = [0,39]$,$\delta^{\uparrow} = [1,39]$	$\delta^{\downarrow} = [5,39]$,$\delta^{\uparrow} = [1,37]$	$\delta^{\downarrow} = [17,19]$,$\delta^{\uparrow} = [1,37]$
b_1	$\delta^{\downarrow} = [0,39]$,$\delta^{\uparrow} = [1,39]$	$\delta^{\downarrow} = [5,39]$,$\delta^{\uparrow} = [1,37]$	$\delta^{\downarrow} = [17,19]$,$\delta^{\uparrow} = [1,37]$

Tableau 1. Reduction of delay intervals of signals $b\_0, b\_1, b0$ and $b1$.

With the automatic reduction of delay intervals, we were able to prove that $t_{CK \to Q} \in [38, 78]$. Applying the manual reduction further, we were able to prove that $t_{CK \to Q} \in [50, 58]$ (and even $[51, 57]$ when delays associated with simultaneous transitions are eliminated). In both cases, the property is verified within 10 mn. Although the result is not as accurate as the one obtained by electrical simulation, it is formally established for all ranges of delays occurring within the defined intervals.

One can evaluate the performances of UPPAAL to compute optimal setup timings for the model of the memory, still guaranteing a response time bound in [50, 58]. This is performed by changing the environment, and bounding the occurrence's instants of edges of input signals D, WEN and A to an interval instead of a punctual value. Precisely, the new environment stipulates that $t^{D}_{setup} \in [81, 108]$, $t^{WEN}_{setup} = [32, 48]$ and $t^{A}_{setup} = [33, 58]$. With this new environment, we were able to prove that the response-time property is bound by [50, 58]. The proof took 23mn and 700 MB of memory.

5 Parametric Analysis

We synthesize here constraints guaranteeing the proper behavior of the memory, first by using the *inverse method* [7], then using the *behavioral cartography* [8].

5.1 Inverse Method

We first apply here the inverse method algorithm *IM* described in [7] and implemented in IMITATOR II [5] to two models of the SPSMALL memory.

Manually Abstracted Model

Description. We consider here a model manually abstracted, close to the model considered in [9]. We recall the model considered in [9] in Figure 13 under the form of an AFTG. This model was abstracted in order to consider that only one bit is stored. As a consequence, D becomes a 1-bit signal. Furthermore, we consider only the portion of the circuit relevant to the *write* operation.

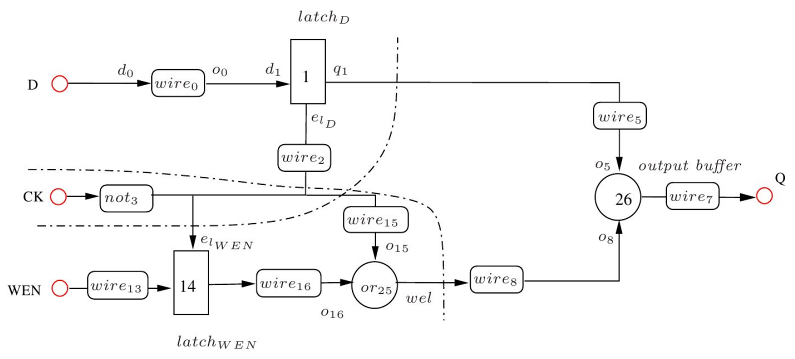


Fig. 13. Abstract model of the SPSMALL memory (write operation)

Although the model we consider here is close to the model considered in [9], a major difference with the model of [9] though is that delays are not only associated to latches and wires anymore, but to latches, wires and *gates*, depending on the components. This model has been designed partially automatically from the VHDL code, using abstractions. This VHDL source code (available in [10]) was itself manually written.

This model, depicted in Figure 14, results in 9 components. Components $delay_D$ and $delay_{WEN}$ are delays (i.e., the logical functionality is the identity), components NOT_1, NOT_2 and NOT_3 are "NOT" gates, WEL is an "OR" gate, and components $delay_{WEN}$, $latch_D$ and net_{27} are latches. A further difference with the model considered in [9] is that several components have been grouped together in order to avoid the state-space explosion problem[1]. For example, several delays associated to wires have been incorporated into the previous elements: this is the case, e.g., of component $wire_5$ from Figure 13, the delay of which has been incorporated into the element $latch_D$, resulting in only one component ($latch_D$) in our model depicted in Figure 14.

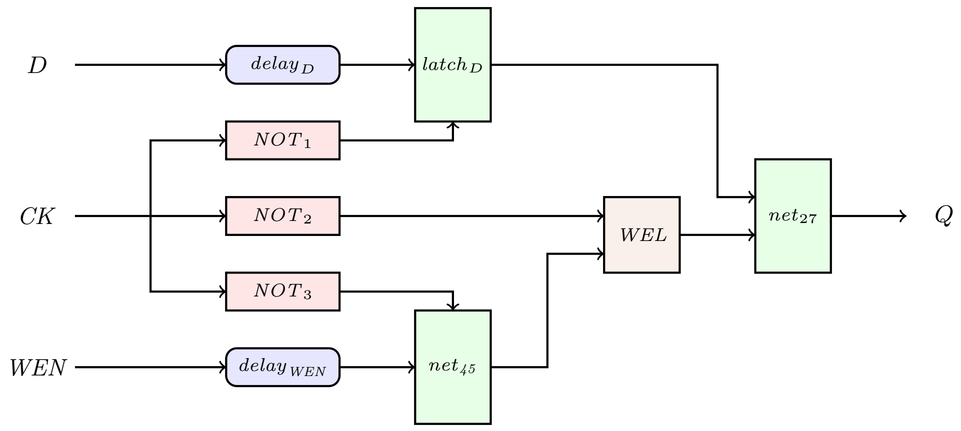


Fig. 14. PTAs modeling the write operation of SPSMALL

Each of the components depicted in Figure 14 (wires, gates, latches) is modeled using a PTA. The translation of the gates into PTAs has been performed automatically using a preliminary version of VHDL2TA. The other components were manually written, and so was the composition of all components together. The environment is also modeled using a PTA. This results in a model containing 10 automata, 10 clocks and 26 parameters corresponding to the traversal delays of the components and the environment. Contrary to [9], the PTAs modeling the gates are actually *complete*, in the sense that all possible configurations

[1] This model was actually first designed to be analyzed using HYTECH, which can hardly accept more than 10 components modeled by PTAs in parallel. However, analyzing this model using IMITATOR II is performed easily in a couple of seconds.

and transitions are modeled, not only the configurations that will be met for a precise environment, as it was the case in [9]. This is in particular due to the automatic generation of the PTAs.

Implementation SP1. We give below the set of parameter valuations (say, π_1) coming from the implementation SP1 and adapted to this first model (timings are given in tens of pico-seconds).

$$\begin{array}{rlrlrl}
\mathit{d\_up\_q\_0} &= 21 & \mathit{d\_dn\_q\_0} &= 20 & \mathit{d\_up\_net27} &= 0 \\
\mathit{d\_dn\_net27} &= 0 & \mathit{d\_up\_d\_inta} &= 22 & \mathit{d\_dn\_d\_inta} &= 45 \\
\mathit{d\_up\_wela} &= 0 & \mathit{d\_dn\_wela} &= 22 & \mathit{d\_up\_net45a} &= 5 \\
\mathit{d\_dn\_net45a} &= 4 & \mathit{d\_up\_net13a} &= 19 & \mathit{d\_dn\_net13a} &= 13 \\
\mathit{d\_up\_net45} &= 21 & \mathit{d\_dn\_net45} &= 22 & \mathit{d\_up\_d\_int} &= 14 \\
\mathit{d\_dn\_d\_int} &= 18 & \mathit{d\_up\_en\_latchd} &= 28 & \mathit{d\_dn\_en\_latchd} &= 32 \\
\mathit{d\_up\_en\_latchwen} &= 5 & \mathit{d\_dn\_en\_latchwen} &= 4 & \mathit{d\_up\_wen\_h} &= 11 \\
\mathit{d\_dn\_wen\_h} &= 8 & \mathit{d\_up\_d\_h} &= 95 & \mathit{d\_dn\_d\_h} &= 66 \\
T_{HI} &= 45 & T_{LO} &= 65 & t^{D}_{setup} &= 108 \\
t^{WEN}_{setup} &= 48 & & & &
\end{array}$$

Constraint synthesized by the inverse method. We apply the inverse method algorithm *IM* described in [7] and implemented in IMITATOR II [5] to this model and the reference valuation π_1 (corresponding to the SP1 implementation). The following constraint K_1 is synthesized after 32 iterations (31 reachable states with 30 transitions) in less than 2 seconds:

$$\begin{array}{l}
\quad T_{HI} + \mathit{d\_up\_net13a} > \mathit{d\_dn\_net13a} + \mathit{d\_dn\_wela} + \mathit{d\_up\_net27} + \mathit{d\_up\_q\_0} \\
\wedge\ T_{LO} > \mathit{d\_up\_en\_latchd} + \mathit{d\_up\_d\_int} + \mathit{d\_up\_d\_inta} \\
\wedge\ t^{D}_{setup} + \mathit{d\_dn\_en\_latchd} > \mathit{d\_up\_d\_h} + \mathit{d\_up\_d\_int} + \mathit{d\_up\_d\_inta} \\
\wedge\ t^{WEN}_{setup} + \mathit{d\_up\_d\_h} > t^{D}_{setup} + \mathit{d\_dn\_wen\_h} + \mathit{d\_dn\_net45} + \mathit{d\_dn\_net45a} + \mathit{d\_up\_wela} \\
\wedge\ T_{LO} + \mathit{d\_dn\_wen\_h} > t^{WEN}_{setup} + \mathit{d\_up\_net13a} + \mathit{d\_up\_wela} \\
\wedge\ T_{HI} > \mathit{d\_dn\_net13a} + \mathit{d\_dn\_wela} \\
\wedge\ T_{LO} > t^{WEN}_{setup} + \mathit{d\_up\_en\_latchwen} \\
\wedge\ t^{D}_{setup} > \mathit{d\_up\_d\_h} \\
\wedge\ t^{D}_{setup} \geq T_{LO} \\
\wedge\ T_{LO} + T_{HI} \geq t^{D}_{setup} \\
\wedge\ \mathit{d\_dn\_en\_latchwen} \geq 0 \\
\wedge\ \mathit{d\_up\_en\_latchwen} \geq 0 \\
\wedge\ t^{WEN}_{setup} + \mathit{d\_up\_en\_latchd} > T_{LO} + \mathit{d\_dn\_wen\_h} \\
\wedge\ \mathit{d\_dn\_net13a} > \mathit{d\_dn\_en\_latchwen} \\
\wedge\ t^{WEN}_{setup} + \mathit{d\_up\_net13a} > T_{LO} \\
\wedge\ \mathit{d\_up\_en\_latchwen} + \mathit{d\_up\_net45} + \mathit{d\_up\_net45a} > \mathit{d\_up\_en\_latchd} \\
\wedge\ \mathit{d\_dn\_net13a} + \mathit{d\_dn\_wela} > \mathit{d\_dn\_en\_latchd} \\
\wedge\ \mathit{d\_up\_wela} \geq 0 \\
\wedge\ t^{D}_{setup} + \mathit{d\_up\_en\_latchd} + \mathit{d\_dn\_d\_int} + \mathit{d\_dn\_d\_inta} > T_{LO} + \mathit{d\_up\_d\_h} \\
\wedge\ \mathit{d\_up\_en\_latchd} + \mathit{d\_up\_d\_int} + \mathit{d\_up\_d\_inta} > \mathit{d\_up\_en\_latchwen} + \mathit{d\_dn\_net45} + \mathit{d\_dn\_net45a} \\
\wedge\ \mathit{d\_up\_d\_h} + \mathit{d\_up\_d\_int} + \mathit{d\_up\_d\_inta} > t^{D}_{setup} + \mathit{d\_dn\_net13a} \\
\wedge\ \mathit{d\_dn\_net13a} + \mathit{d\_dn\_wela} + \mathit{d\_up\_net27} + \mathit{d\_up\_q\_0} > T_{HI} + \mathit{d\_up\_en\_latchwen}
\end{array}$$

Interpretation. The main advantage of the constraint synthesized by IMITATOR II is that it allows to show the link between the internal timing delays and the external values of the environment. Indeed, the timing parameters corresponding to the environment are *constrained* by the internal traversal delays of the gates, wires and latches. Despite the complex form of the constraint synthesized, it is possible to give an interpretation for some of the inequalities.

First of all, some inequalities are actually synthesized because of the environment that we consider. Inequalities such as $t^D_{setup} \geq T_{LO}$ or $T_{LO} + T_{HI} \geq t^D_{setup}$ come from the way we modeled the environment, and are bound by the *model* more than the system.

Moreover, other inequalities can be interpreted as a guarantee on the *order* of the events. Recall that our inverse method guarantees the same trace sets and, as a consequence, the same ordering of events. For example, the inequality $t^{WEN}_{setup} + d\_up\_en\_latchd > T_{LO} + d\_dn\_wen\_h$ implies that the (timed) path through wire $delay_{WEN}$ is greater than the path through gate NOT_3. In other words, the upper input of latch net_{45} must change before its left input.

Optimization. By replacing within K_1 every parameter except t^D_{setup} and t^{WEN}_{setup} by its valuation as defined in π_1, one gets the following constraint on t^D_{setup} and t^{WEN}_{setup}:

$$46 < t^{WEN}_{setup} < 54 \quad \wedge \quad 99 < t^D_{setup} \leq 110 \quad \wedge \quad t^D_{setup} < t^{WEN}_{setup} + 61$$

It is then interesting to *minimize* those setup timings. Indeed, if one minimizes the setup duration of the input signals without changing the overall behavior of the system, then this means that the memory can be inserted in a faster environment where the input signals change faster. One can thus minimize t^D_{setup} and t^{WEN}_{setup} according to K_1 as follows:

$$t^{WEN}_{setup} = 47 \quad \wedge \quad t^D_{setup} = 100$$

By comparison with the original parameter valuation π_1 (viz., $t^D_{setup} = 108$ and $t^{WEN}_{setup} = 48$), this results in a decreasing of the setup timing of signal D (resp. WEN) of 7.4 % (resp. 2.1 %).

In [9], the authors compute a minimum value of 95 for t^D_{setup}, and a minimum value of 29 for t^{WEN}_{setup}. As a consequence, our values may still be improved. Improving those values for this model will be the purpose of Section 5.2.

Automatically Generated Model This second version of the SPSMALL memory is a more complete model of the memory, representing not only the portion of the memory corresponding to the *write* operation, but the complete architecture. As in the previous section, this model was abstracted in order to consider that only one bit is stored. As a consequence, D becomes a 1-bit signal. We give in Figure 15 the schematics from [9] depicting the wires, gates and latches under the form of an Abstract Functional and Timing Graph, and corresponding to the complete architecture of SPSMALL.

A further major difference with the manual model described in the previous section is that the PTAs are here fully automatically generated. Recall that, in

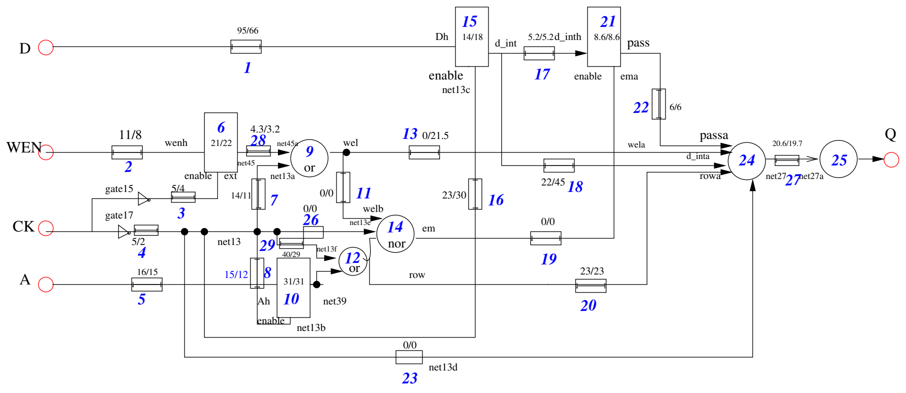


Fig. 15. Abstract model of the SPSMALL memory

the previous section, the PTAs were written in a partially manual way, and the model was then simplified by grouping together several automata. Here, we first manually wrote the VHDL code corresponding to the different elements of the memory (which is much quicker and less error-prone than describing the PTAs), and then automatically synthesized the PTAs using the tool VHDL2TA [11]. This leads to more parameters, including a slightly richer environment, involving explicitly signal A, characterized by its setup value, viz., t^{A}_{setup}. This technique results in a model containing 28 automata, 28 clocks, 32 discrete variables and 62 parameters.

Due to the high number of parameters and the complexity of the model, we do not give here the set of parameter valuations coming from the datasheet of SP1 and adapted to this second model, but it can be found in [10, 12]. We only give below the set of parameter valuations (say, π'_1) corresponding to the three input timings we are interested in optimize (timings are given in tens of pico-seconds):

$$t^{D}_{setup} = 108 \qquad t^{WEN}_{setup} = 48 \qquad t^{A}_{setup} = 58$$

Applying IMITATOR II to this model and this reference valuation π'_1, one synthesizes a constraint K'_1, projected below onto t^{A}_{setup}, t^{D}_{setup} and t^{WEN}_{setup}. The interested reader may refer to [12] for the complete valuation and the complete constraint on the whole set of parameters.

$$t^{D}_{setup} = 108 \quad \wedge \quad t^{WEN}_{setup} = 48 \quad \wedge \quad 56 < t^{A}_{setup} < 60$$

This constraint is an “interesting” (though unfortunate) example of constraint for which the output parameter domain is (almost) reduced to a single point. Thus, it is not possible to optimize values of t^{D}_{setup} and t^{WEN}_{setup} according to this constraint. Nevertheless, the cartography algorithm introduced in [8] will allow us to overcome this shortcoming, and synthesize a dense set of parameters allowing us to minimize those input timing parameters (see Section 5.2).

Larger Models Two other versions of the SPSMALL memory have been considered. The first one is actually the full SPSMALL memory with 1 memory point of 2 bits. The model described as a network of PTAs has been automatically generated from the VHDL code using VHDL2TA. The VHDL code itself was also automatically generated from the transistor netlist given by ST-Microelectronics. This chain of analysis has been performed in the framework of the VALMEM project. Unfortunately, because of the high size of this model (101 automata, 101 clocks, 200 parameters, 130 discrete variables, which result in more than 6000 lines of code described in the IMITATOR II syntax), IMITATOR II does not succeed to synthesize a constraint after several hours.

The second version corresponds to a larger version of the SPSMALL memory, with 3 memory points of 2 bits. Due to the even larger size of this model (more than 130 automata), IMITATOR II does not succeed to synthesize a constraint either.

Improving IMITATOR II so that it can synthesize constraints for such large systems is the subject of future work. It is also interesting to note that non-parametric analyses of these two models have been successfully performed using the UPPAAL model checker [13], allowing to verify several properties.

5.2 Behavioral Cartography

We apply here the behavioral cartography as described in [8] and implemented in IMITATOR II [5].

We will consider here two versions of the memory: the manually abstracted model, and the automatically generated model.

Manually Abstracted Model We first consider here the model manually abstracted, described in Section 5.1. We are interested in minimizing the values of the setup timing parameters, viz., t^{D}_{setup} and t^{WEN}_{setup}, so that they still verify the following good property mentioned in [9]: "the response time of the memory must be smaller than 56" (recall that units are given in tens of *ps*). This response time corresponds to the value $T_{CK \rightarrow Q}$, and represents the time between the second rise of input signal CK and the rise of the output signal Q. Note that this good property does not strictly speaking correspond to a property on traces. As a consequence, we make use of an *observer* (as in [14] and [9]), i.e., an additional PTA which waits for the rise of Q and, depending on the time of this action, goes into a good location or into a bad location. Locations are observable within traces, thus this property is now a property on traces.

We perform a behavioral cartography of the SPSMALL memory, for the following V_0:

$$t^{D}_{setup} \in [65; 110] \quad \wedge \quad t^{WEN}_{setup} \in [0; 66].$$

The other parameters are instantiated like in π_1. We give in Figure 16 the cartography of the SPSMALL memory, as automatically output by IMITATOR II [5]. The dashed rectangle corresponds to V_0. The red zone above t^{WEN}_{setup} is infinite, and corresponds to a bad behavior.

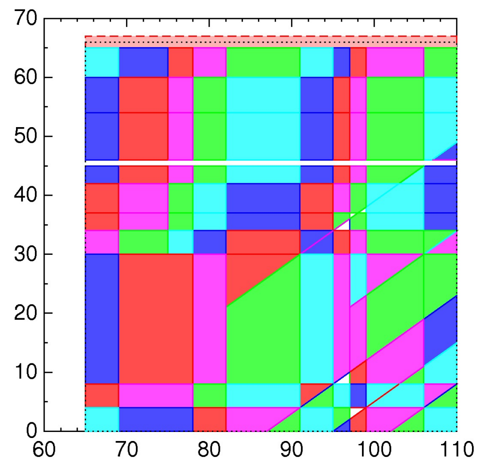


Fig. 16. Cartography of the SPSMALL memory

Recall that each different colored zone corresponds to a different behavior[2]. Note that the cartography actually contains a few holes, i.e., zones (depicted in white) covered by no tile. We manually "filled" those zones by calling again the inverse method on one point in each zone, which allowed us to cover the whole rectangle V_0.

We then partition the tiles into good and bad. This partition is depicted under a graphical form in Figure 17, where the light red (resp. dark blue) zone corresponds to the bad (resp. good) values of the parameters. After partitioning the tiles into good and bad, one is able to infer the following constraint corresponding to the set of parameters for which the memory circuit behaves well:

$$99 < t^{D}_{setup} \leq 110 \quad \wedge \quad 30 < t^{WEN}_{setup} \leq 65$$

This constraint corresponds to the maximal constraint solving the good parameters problem for the SPSMALL memory within V_0, because the whole rectangle has been covered by the tiles.

Due to the way we modeled the system (in particular the environment), values such that $t^{D}_{setup} < 65$ or $t^{D}_{setup} > 110$ do not correspond to any proper behavior. As a consequence, the constraint synthesized corresponds to the maximal constraint for the whole parameter space of this model.

One can thus minimize t^{D}_{setup} and t^{WEN}_{setup} according to the cartography as follows:

$$t^{D}_{setup} = 100 \quad \wedge \quad t^{WEN}_{setup} = 31$$

[2] Recall that this cartography has been automatically output by IMITATOR II which can only represent a few colors (due to the use of an external plot tool). As a consequence, different zones depicted using the same color do not necessarily have the same trace set.

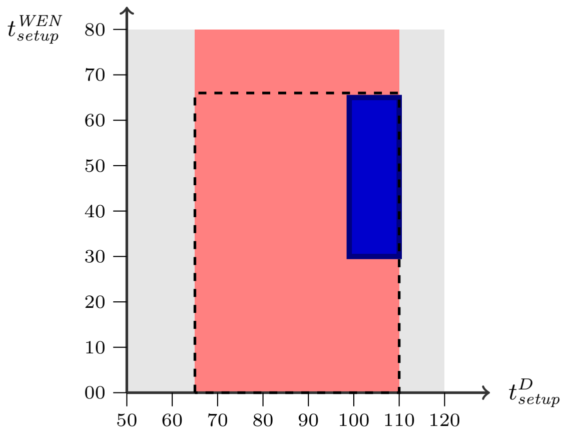


Fig. 17. Cartography of the SPSMALL memory (after partition)

By comparison with the original datasheet π_1 (viz., $t^{D}_{setup} = 108$ and $t^{WEN}_{setup} = 48$), this results in a decreasing of the setup timing of signal D of 7.4 %, and a decreasing of the setup timing of signal WEN of 35.4 %.

Comparison with Other Methods. In [9], the authors synthesize a minimum for these setup timings, by iteratively decreasing the setup timings until the system does not behave well anymore, i.e., until the response time is not guaranteed anymore. When compared to our approach, the approach of [9] has the following limitation: they test only the *integer* points, and do not have any guarantee for the dense set of parameters between two integer points. In [9], a minimum value of 95 is given for t^{D}_{setup}. However, our approach indicates that the value of 95 corresponds to a bad behavior, and therefore shows a discrepancy between our respective models. A minimum value of 29 is given for t^{WEN}_{setup}, which is slightly smaller as ours. Again, this indicates a discrepancy between our respective models.

Automatically Generated Model We now consider the model automatically generated, described in Section 5.1. As in the previous section, we are interested in minimizing the values of the setup timing parameters, viz., t^{D}_{setup} and t^{WEN}_{setup}, so that they still verify the following good property mentioned in [9]: "the response time of the memory must be smaller than 56" (recall that units are given in dozens of *ps*). Again, we make use of an *observer* in order to transform this property into a property on traces.

We perform a behavioral cartography of the SPSMALL memory, for the following V_0:

$$t^{D}_{setup} \in [89; 98] \quad \wedge \quad t^{WEN}_{setup} \in [25; 34].$$

Due to the complexity of this model, note that the rectangle V_0 is not as large as for the manual model. We give in Figure 18 the cartography of the SPS-

MALL memory, as automatically output by IMITATOR II. The dashed rectangle corresponds to V_0.

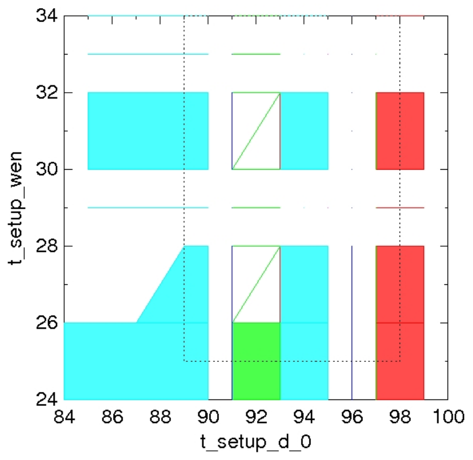


Fig. 18. Cartography of the SPSMALL memory (generated model)

Recall that each different colored zone corresponds to a different behavior. This cartography, though interesting, contains many holes, i.e., zones (depicted in white) covered by no tile.

We then chose to launch again the analysis using a tighter grid, viz., by calling the inverse method on points multiple of 1/3 instead of integer points. This corresponds to the algorithm BC' sketched in [8]. The reason for the choice of 1/3 is that, with such a step, one is sure to cover any tile delimited by integer points. This is not the case of a step of 1 (or even 1/2), because tiles delimited by integer points may *exclude* those integer points in the case of strict inequalities.

This second cartography of the SPSMALL, with step 1/3, is given in Figure 19. This cartography is this time successful in the sense that the whole bounded parameter domain V_0 is covered by the tiles. Furthermore, a significant part of the parametric space outside V_0 is also covered.

We then partition the tiles into good and bad. This partition is depicted under a graphical form in Figure 20, where the light red (resp. dark blue) zone corresponds to the bad (resp. good) values of the parameters. From this partition, one is able to infer the following constraint corresponding to the set of parameters within V_0 for which the memory circuit behaves well:

$$96 \leq t^{D}_{setup} \leq 98 \quad \wedge \quad 29 \leq t^{WEN}_{setup} \leq 34$$

This constraint corresponds to the maximal constraint solving the good parameters problem for the SPSMALL memory within V_0, because the whole rectangle has been covered by the tiles. Also note that the cartography gives further information outside V_0.

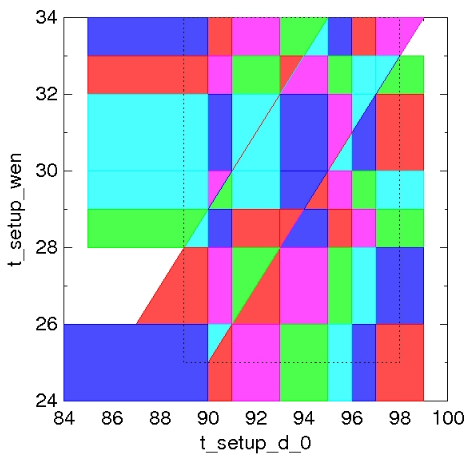


Fig. 19. Cartography of the SPSMALL memory (full coverage)

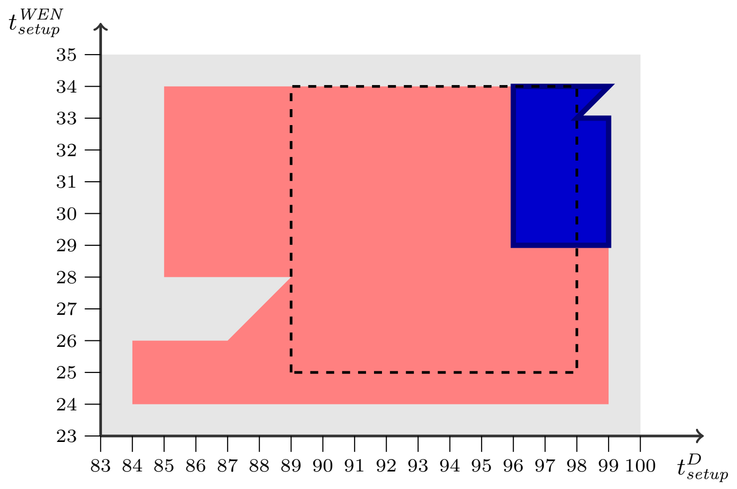


Fig. 20. Cartography of the generated model of the SPSMALL memory (after partition)

One can thus minimize t^{D}_{setup} and t^{WEN}_{setup} according to the cartography as follows:

$$t^{D}_{setup} = 96 \quad \wedge \quad t^{WEN}_{setup} = 29$$

By comparison with the original valuation for t^{D}_{setup} and t^{WEN}_{setup} (viz., $t^{D}_{setup} = 108$ and $t^{WEN}_{setup} = 48$), this results in a decreasing of the setup timing of signal D of 11.1 %, and a decreasing of the setup timing of signal WEN of 39.6 %. Such an important decreasing of some of the values of the environment show the interest of the cartography algorithm for the optimization of timing parameters.

Comparison with Other Methods. Recall that, in [9], the authors also synthesize a minimum for these setup timings, by iteratively decreasing the setup timings until the system does not behave well anymore. In [9], a minimum value of 95 is given for t^{D}_{setup}. However, our approach indicates that the value of 95 corresponds to a bad behavior, and therefore shows a slight discrepancy between our respective models. Also observe that the authors of [9] find a minimum value of 29 for t^{WEN}_{setup}, which is exactly the same as ours. This shows the interest of our method, which computes a constraint allowing to retrieve fully automatically the (manually computed) results from [9], with the advantages that we considered the full model of the memory (not only the write operation), that we give relations between the parameters (under the form of a constraint), and above all that we now give conditions of correctness on the *dense* space of parameters.

Due to the high size of this model (viz., an NPTA composed of 28 PTA containing 28 clocks, 32 discrete variables and 62 parameters) and to the practical interest of the constraint output, this case study can be considered as an extremely interesting application of IMITATOR II.

Remarque 1. In [9], values corresponding to simulation are given. Simulation is a technique based on an exact virtual version of the memory. It is usually extremely costly to perform (and is suitable for only one environment) but its results can be considered as exact for this particular case. For this case study, a simulation has been performed using the entire system (i.e., without cutting away some parts of the memory), for some (punctual) values of the input timings. For this environment and those values of the parameters, according to [9], the minimum possible value computed by simulation for t^{WEN}_{setup} is 36, and the minimum possible value for t^{D}_{setup} is 95. For t^{D}_{setup}, this means that the value we compute is suitable, because it is greater than the minimum possible value. Moreover, it is almost the optimal value, since our method allows to minimize t^{D}_{setup} to 96, whereas the minimum value is 95. For t^{WEN}_{setup}, however, our value is strictly smaller than the value computed using the simulation, which represents a minimum. This indicates that (at least) one delay assigned to a gate of our model (which has been automatically computed in the framework of the VALMEM project) is too approximative.

6 Subsisting Problems

The previous experiment shows that two main problems remain in our methodological flow, in order to perform a fully automated formal post-validation of the timing specification of the SPSMALL memory:

6.1 Accuracy of the formal model

The timed automata model obtained from the functional abstraction + timing extraction steps is an abstraction of the spice model: it encompasses all the timed behaviors the spice model may produce, but it also contains some extra behaviors. The property we verify have to be satisfied for all behaviors produced by the formal model, hence there may exist some behaviors in the formal model, not existing in the spice model, that invalidate a too precise property. This explains the dispersion of 5% of the response time we analyze, while the specification allows a 1% dispersion.

These extra-behaviors are mainly due to two phenomena:

1. The collapse of sets of punctual delays for rising and falling edges into two dense-time intervals. This corresponds to a basic choice of our methodology, which addresses two important features. On the one hand, the delays associated to each gate are extracted by electrical simulation *for particular conditions (external temperature, input slope)*. An important work has been done to reduce the uncertainty induced by the input slope, but other varying parameters remain. Hence the punctual delay given by TIMEX are known with a 5% margin. This uncertainty is captured into the dense-time delay interval model adopted in the formalism of timed automata. On the other hand, the reduction of punctual delays into two timed intervals is adequate for timed automata: in a time automaton representing a gate or a process, one location is associated to each interval. A multiplication of delays would induce wide automata, this model would have presented less behaviors than the 2 interval model, but the former would not have been tractable by standard model-checking tools.
2. The consideration of non-functional input configurations during the timing extraction (in case of pre-charged signals). The timing extraction is performed considering each gate or functional block without considering the environment in which it evolves. This disconnection has a small impact on the timings being extracted for each input configuration, but it has a great impact when one has to filter the input configurations and transitions that have to be considered. Ideally, for a gate or a block, all input configurations and for each configuration, all single transitions have to be considered. This is what was applied to extract the timings of combinatorial blocks, leading to two thin delay intervals (dispersion is less than 10% in each interval). For sequential blocks however, the dispersion of delay in each interval is much more important (up to 4000 %) and among all the configurations evaluated, only a small fraction is significative. We had to reduce the timing intervals of these components. Some reduction have been automated but others, based on reasoning of pre-charged signals, needs to analyze the dynamics of signals in order to filter some non-functional configurations.

6.2 Scale to industrial size circuits

The automatic analysis with timed model-checkers as UPPAAL is successful for circuits of the size of SPSMALL (about 100 gates with rising and falling intervals for each gate), but the analysis would fail for bigger circuits, or in case of less constrained environments. The automatic extraction of timing constrains is applicable to small asynchronous circuits (up to 20 or 30 gates and 60 parameters), but the extraction procedure is too expansive to be applied to bigger systems.

6.3 Other limitations

Other limitations prevented us from applying our flow to memory SPREG, initially defined in the proposal of the project. These are mainly due to the particular self-timed logics this circuit contains, which is not taken into consideration in the functional abstraction tool.

7 Comparison of our results with existing flows

The methodology we developed gives several types of results:

- an abstract functional view and related timed sets. These models can be used to build a timed VHDL model and perform VHDL event-driven timed simulation. In this case, environment and internals delays are punctual. The VHDL model growths fast (as it contains all punctual delays), but the simulation performed at logical level is quicker than the one performed at electrical level.
- formal proof of timed properties for pre-defined scenarios including varying delays. These proofs cannot be performed with the electrical simulator, which evaluates one precise delay for each action, and cannot reason with delay variation.
- extractions of linear constraints between timed parameters. These constraints may be used to perform delay optimizations, but also to visualize the timing domains of internal and environmental delays guaranteeing good behaviors and to understand the relationship between delays. These aspects are not achievable with electrical simulation tools.

These results are automatically obtained thanks to the the set of tools developed during the VALMEM project. This set of results types is interesting since it gives the opportunity to the designer to have a better understanding and confidence of its circuit during the different steps of its conception. However, the accuracy of the formal model is not as high as the spice model is, hence the results we provided are not accurate enough to validate the datasheet of our case study: our results are guaranteed within a margin of 5% while the datasheet requires a lower margin (around 1%). The tools developed are not restricted to the analysis of memory circuits and the rich results they provide can be of great help in the development of complex systems combining concurrency and timing features. Successful examples of use of tools VHDL2TA and IMITATOR-2 can be found in [6] and [5] in the context of asynchronous gate-level circuits or high-level concurrent and timed protocols.

References

1. R. Chevallier, E. Encrenaz-Tiphène, L. Fribourg, and W. Xu. Timing analysis of an embedded memory: SPSMALL. In *10th WSEAS Int. Conf. on Circuits and Systems, WSEAS Trans. on Circuits and Systems, vol 5(7)*, pages 973–978, 07 2006.

2. K. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1:134–152, 1997.
3. S. Yovine. Kronos: A Verification Tool for Real-Time Systems. *International Journal on Software Tools for Technology Transfer*, 1:123–134, 1997.
4. R. Chevallier, E. Encrenaz-Tiphène, L. Fribourg, and W. Xu. Verification of the generic architecture of a memory circuit using parametric timed automata. In *Int. Conf. on Formal Modelling and Analysis of Timed Systems (FORMATS)*, volume 4202 of *LNCS*, pages 113–127. Springer, 2006.
5. Étienne André. IMITATOR II: A tool for solving the good parameters problem in timed automata. In Yu-Fang Chen and Ahmed Rezine, editors, *INFINITY'10*, volume 39 of *Electronic Proceedings in Theoretical Computer Science*, pages 91–99, 2010.
6. A. Bara, P. Bazargan-Sabet, R. Chevallier, E. Encrenaz, D. Ledu, and P. Renault. Formal verification of timed vhdl. In *International Forum on Design Languages (FDL)*, Southampton, U.K., sept. 2010. ECSI.
7. É. André, T. Chatain, E. Encrenaz, and L. Fribourg. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science*, 20(5):819–836, 2009.
8. É. André and L. Fribourg. Behavioral cartography of timed automata. In *RP'10*, volume 6227 of *LNCS*, pages 76–90. Springer, 2010.
9. R. Chevallier, E. Encrenaz-Tiphène, L. Fribourg, and W. Xu. Timing analysis of an embedded memory: SPSMALL. *WSEAS Transactions on Circuits and Systems*, 5(7):973–978, July 2006.
10. VHDL2TA Web page. http://www.lsv.ens-cachan.fr/ encrenaz/valmem/vhdl2hytech/.
11. Abdelrezzak Bara. Vhdl2ta: A tool for automatic translation of vhdl programs plus timings into timed automata. Research report, LIP6, 2009. ANR-VALMEM Technical Report.
12. Étienne André. Synthesizing parametric constraints on various case studies using IMITATOR II. Research Report LSV-10-21, Laboratoire Spécification et Vérification, ENS Cachan, France, November 2010.
13. K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, 1997.
14. 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.