Verification of the Generic Architecture of a Memory Circuit Using Parametric Timed Automata ⋆

Remy Chevallier[1], Emmanuelle Encrenaz-Tiphene[2], Laurent Fribourg[2] and Weiwen Xu[2]

[1] STMicroelectronics, FTM, Central R&D, Crolles, France
[2] LSV - CNRS, ENS de Cachan, France

Abstract. Using a variant of Clariso-Cortadella's parametric method for verifying asynchronous circuits, we formally derive a set of linear constraints that ensure the correctness of some crucial timing behaviours of the architecture of SPSMALL memory. This allows us to check two different implementations of this architecture.

1 Introduction

In [10, 9], Clariso and Cortadella propose a technique for verifying the timings of asynchronous circuits. The approach infers a set of sufficient linear constraints relating the delays of the internal gates of the circuit to the external delays of the circuit specification that guarantee the correct behavior of the circuit. The method is based on the reachability analysis of a timed model of the circuit (with additional abstract interpretation techniques [11]). As pointed out in [10], such parametric constraint sets are very informative for the designer, as they identify sensitive parts of the circuits (e.g., "critical paths") and interrelations between various data of the specification. Moreover, many technology mappings can be tested immediately (by mere instantiation of the parameters).

We follow here a similar approach for formally verifying some generic properties of a commercial memory designed by STMicroelectronics, called SPSMALL. Such a memory can either read or write a data (depending on the value of an input signal WEN). For the sake of brievity, we focus here on the write operation ($WEN = 0$). In this case, the memory stores the value of the input signal D into an internal memory point (located at address A), and propagates it to the output port Q. The circuit is made of a dozen of elementary components. Each component c_i is associated with an interval $[l_i^{\uparrow}, u_i^{\uparrow}]$ (resp. $[l_i^{\downarrow}, u_i^{\downarrow}]$), which gives lower and upper bounds of the component traversal delay when the input is rising (resp. falling)[3]. Such a circuit is specified by the manufacturer according to

⋆ Partially supported by project MEDEA+ Blueberries

[3] This is a straightforward generalization of "bi-bounded delay" model (see [6]), taking into account the rising or falling nature of input signal.

several "external" parameters (such as periods of a cyclic clock CK, time of stabilization of signal D, ...). Our timing analysis method derives a set of sufficient linear constraints relating the external parameters to the internal gate delays that guarantee the correctness of the circuit's behavior. In particular, these constraints can be seen as sufficient conditions for certain paths of the circuit to be "critical" (i.e. those along which the propagation delay is the longest).

Using the model of parametric timed automata (see [3]) and tool HYTECH [14] for reachability analysis, we are able to generate a set of linear constraints that ensures that the correctness of some crucial timing behaviors of the memory: e.g., the result of a write or read command, transmits the value of input signal D to output port Q within one clock cycle. This method is applicable to several instances of SPSMALL memories implemented with different transistor technologies (corresponding to different sets of parameter values).

Comparison with Related Work. As pointed out above, our work is adapted from Clariso-Cortadella's method [10, 9]. However, [10, 9] focus on a particular form of linear constraints (linear inequalities with coefficients always equal to ± 1) and represent them as a particular form of convex polyhedra, called "octahedra" [9]. In contrast, we use here linear constraints and their classical form of convex polyhedra in their full generality [13]. Other differences with [10, 9] are:

- a different level of modelling: the components of the memory are represented here at the "latch" level instead of 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 ouput of one gate can be an input of another gate and the converse can be simultaneously true),

- the use of mere forward reachability analysis rather than techniques of fixpoint computation of abstract interpretation (using, e.g., widening operators),

- the use of a decompositional approach, which splits the global system into three smaller parts,

- the use of a *step-by-step refinement* of the reachability analysis process: we start with the most general form of constraints on parameters; we then refine them progressively, via *iterative* reachability analysis, detecting, at each run, some erroneous generated states until complete elimination.

Besides [10, 9], our work is along the lines of [16, 5, 17] where timed automata have been used extensively to model and check timing properties of asynchronous circuits (cf. [12]). Reachability analysis is there performed via tool KRONOS [18]. Our work is also a continuation of [4, 7] where the SPSMALL memory is modelled as a timed automaton and some of its timing properties are proven by reachability analysis (using tool UPPAAL [15]). The crucial difference here, with respect to these previous works, is that we use the model of *parametric* timed automata (performing reachability analysis with HYTECH [14]).

Plan of the paper. In Sect. 2, we present the general objectives of our verification process. In Sect. 3, we give a general description of our method. In Sect. 4, we explain how we apply it on SPSMALL memory after having split the model into 3 parts. Final remarks are given in Sect. 5.

2 Objectives of Verification

2.1 Timings of a Memory Circuit

A memory circuit aims at storing data at some addressed locations. It is associated with two operations: 'write' and 'read'. The memory circuit has several input ports and one output port (see Fig. 1). The signals driven by input ports are:

- CK, the signal of the periodic clock;
- D, the data to be stored;
- A, the address of the internal memory location;
- WEN, the nature (write/read) of the operation.

The signal driven by the output port is Q. The memory circuit makes use of some internal devices, called 'memory points', to store data. This is depicted on Fig. 1. The write operation ($WEN = 0$ when CK is rising) writes the value of D in the internal memory point selected by A, and propagates D on output port Q. The read operation ($WEN = 1$ when CK is rising) outputs on port Q a copy of the data stored in the memory point selected by A.

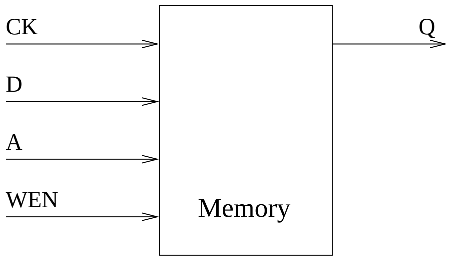

Fig. 1. Simplified interface of a memory.

A memory is embedded into a synchronous environment scheduled by a periodic signal, named 'clock' (CK). The period of a cycle, t_{cycle}, decomposes itself into a high period (t_{HI}) and a low period (t_{LO}). In order to be taken into account, each input signal I has to remain stable for a given amount of time, before the rising edge of the clock. This delay is called *setup* time for I, and denoted by t_{setup_I}[4]. A write operation requires a delay, denoted by $t_{CK \to Q}^{D,WEN}$[5] due to the time of traversal of the elementary components of the memory (See Fig. 2).

The specification states the maximum delay, denoted by t_{max}, needed by a write operation.

These values of the parameters of the specification ($t_{HI}, t_{LO}, t_{setup_D}, t_{setup_{WEN}}$, t_{max}) form the "external specification" or *datasheet* of the circuit provided by the

[4] There exists also a required delay of stability, called "hold" time, required *after* the rising edge of the signal, but we will not consider it in this paper.

[5] A similar delay exists for a read operation, but we will not consider it in this paper.

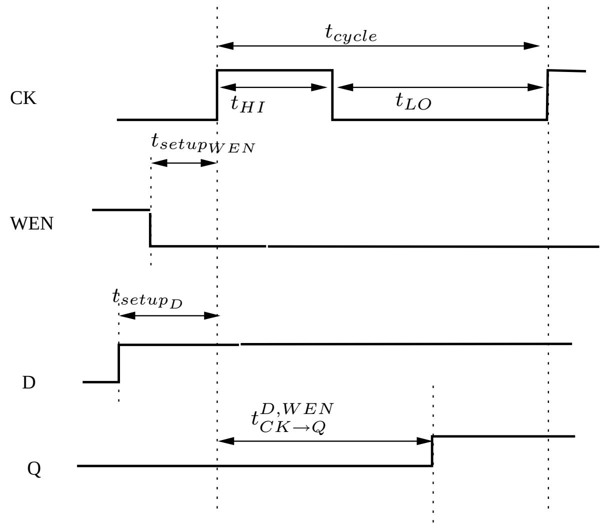

Fig. 2. A *write* operation corresponding to a rising edge of D ($D \uparrow$).

manufacturer to the customer. They are determined by *electrical simulation.* In order to perform such a simulation, each component of the memory is modelled at the transistor level as a set of differential equations, which represent the Kirshoff laws associated with the electric current traversing the component. These differential equations depend on the physical characteristics of the transistors and wires (capacitors, resistors, ...). The full memory is thus represented as a big system of differential equations. The values of the datasheet are computed by resolution of these differential equations, using, e.g., tool HSIM [1]. 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 traversal, 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 "response time" property, expressing an important aspect of the timing correctness of the memory's behavior.

The result of a *write* command is produced on output port Q within t_{max}. This will be expressed as: $t^{D,WEN}_{CK\to Q} \leq t_{max}$, where $t^{D,WEN}_{CK\to Q}$ represents the time (with respect to the beginning of clock cycle CK) after which signal Q reproduces the rising edge of D. Besides, our analysis will allow us to infer an optimal value for parameter t_{setup_D}.

Other properties regarding the writing into the internal memory point and read operation have been proven similarly, but will not be presented here, due to the lack of space.

Note that, as we focus here on the propagation of D through the whole memory (and disregard the writing in the internal memory point), the circuit parts involving the memory point and the address decoder are omitted here.

3 Method

Roughly speaking, the process consists to:

- construct a model of the memory under the form of a timed automaton,
- generate the set of reachable states (within two cycles),
- infer (by stepwise refinement) a set of timing constraints ensuring the response time property,
- verify that, for a given implementation, the response time property holds by checking the instantiated constraints.

3.1 General Scheme for Modelling a Circuit with Timed Automata

In contrast with electrical simulation, the verification process requires the modelling of the internal components of the memory, not from an electrical point of view (using differential equations), but at a *symbolic* abstract level. A circuit component is characterized by a Boolean function f (mapping its inputs to its outputs) and a propagation delay, given under the form of two intervals $[l^{\downarrow}, u^{\downarrow}]$ and $[l^{\uparrow}, u^{\uparrow}]$ (model of *bi-bounded inertial delays* distinguishing the propagation times of *rising* and *falling* edges [6]). The model of timed automata [3] is especially well-suited to represent asynchronous circuits (see, e.g., [5, 17]); however, these models use a bi-bounded delay model without distinguishing the propagation delays of rising and falling edges. This unique propagation delay interval is too coarse for our circuit, thus we introduce two interval delays, $[l^{\uparrow}, u^{\uparrow}]$ for the propagation of a rising edge, and $[l^{\downarrow}, u^{\downarrow}]$ for the propagation of a falling edge.

Roughly speaking, a timed automaton is a finite state automata 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 all t' $(0 \leq t' \leq t)$, $v + t'$ satisfies the invariant associated to ℓ. States can be expressed under the symbolic form of conjunctions of linear constraints. Such states are classically represented as convex polyhedra (see, e.g., [14]). Sets of states correspond to union of polyhedra. The *(forward) reachability analysis* consists in generating iteratively the "successors" of sets of states via the system transitions, using elementary manipulation of polyhedra.

In order to model the circuit, each component is represented as a timed automaton combining its functionality f and delay intervals $[l^{\downarrow}, u^{\downarrow}]$ and $[l^{\uparrow}, u^{\uparrow}]$ (see [16, 17] for a model with a unique interval). Input signals CK, D and WEN are themselves represented as timed automata. The global circuit is modelled as a composition of timed automata where synchronization is used to model the transmission of an internal signal between two components. A central clock s is used to measure the evolution of time. The states generated by reachability analysis correspond to linear constraints which relate the values of input and output signals, depending on timing values of the datasheet, the values of the internal delays, as time s evolves.

Let us note incidentally that our verification process still relies on results obtained by simulation, as it makes use of values of $[l^{\downarrow}, u^{\downarrow}]$ and $[l^{\uparrow}, u^{\uparrow}]$ for internal delays. However simulation is performed here at a lower scale (component), and is hopefully more reliable than for giving end-to-end values.

3.2 Modelling SPSMALL

Level of Modelling. A memory can be modelled at many different levels of complexity, e.g., in a increasing order: at the functional block level, at the "latch" level, at the gate level, or at the transistor level.
For the SPSMALL memory, the model can thus be implemented using: 3 main components, at the block level (cf [4]), 30 components at the latch level, 70 components at the gate level, or 200 components at the transistor level.
There is a tradeoff in finding the appropriate level of modelling. The lower the level of modelling is, the more faithful to the reality the model is, but the more difficult the verification process is. In particular, the task of finding by simulation the delays of each component becomes impracticable at the lowest levels (gate, transistors).

In this work, we chose to represent the memory at the latch level. The advantage is to limit the number of components (around 30) at a reasonable size, and to have a "schematics" describing the architecture of the memory at this level, which closely corresponds to (VHDL) code automatically produced with commercial tool TLL [2]. The interesting portion of this schematics for the property on which we focus here, is described in Fig. 3. It contains 12 components: 7 "wire" components, which transmits an input; one "not" component, which transmits the negation of its input ; one "or" component which computes the logical "or" of its two inputs[6]; and two "latches", $latch_D$ and $latch_{WEN}$. Two timing intervals are associated to each component, one representing the propagation delay of a rising edge (e.g. $[l_0^{\uparrow}, u_0^{\uparrow}]$ for component $wire_0$), and the other

[6] The associated delay has been actually incorporated into the input wires, and will not appear in the following.

representing the propagation of a falling edge (e.g. $[l_0^\downarrow, u_0^\downarrow]$ for component $wire_0$). The precise behavior of these components is explained below.

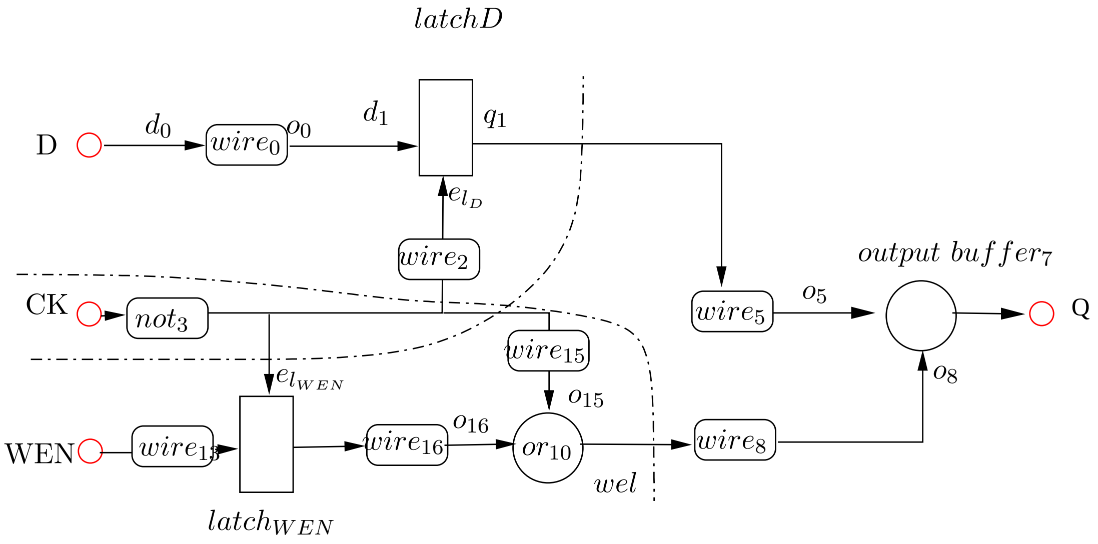

Fig. 3. Schematics'excerpt of SPSMALL.

Modelling SPSMALL as a timed automaton. The SPSMALL memory is modelled as a timed automaton, resulting from the composition of all the timed automata corresponding to the input signals and the components. More precisely:

- Each input signal of the memory (synchronous signal CK, input signal D, read/write command WEN) is represented as a timed automaton.
- Each component of the memory (latch, wire, ...) is also represented as a timed automaton,
- The transmission of an internal signal between two components is modelled by synchronizing the two corresponding timed automata via a shared discrete transition.
- The emission of a signal at the output port Q of the memory corresponds to the updating of an internal variable, denoted by q, of the model.

We will focus on two main kinds of components: wires and latches (the "not","or" or "output buffer" components are similar). The simplest kind of component is a "wire" component, which transmits an input signal after a certain delay: the wire component is enabled $l^\uparrow$ (pico)secs after the rising edge of the input signal, and the output signal is fired after a delay lying in $[l^{\uparrow}, u^{\uparrow}]$ (parameters $l^{\uparrow}$ and $u^{\uparrow}$ are used to propagate a falling edge). We adopt the "inertial delay" interpretation (see [6]): changes that do not persist for l time are filtered out. Such a wire component is naturally modelled as the timed automaton depicted in Fig. 4. It makes use of 1 internal clock c and 5 locations. The symbol $d\uparrow$ (resp. $d\downarrow$) corresponds to a rising edge (resp. falling edge) of input internal signal d, and similarly for symbol $o\uparrow$ (resp. $o\downarrow$) with output signal o. Each edge corresponds to a discrete transition labelled with the name of input signals ($d\uparrow$ or $d\downarrow$) or output signals ($o\uparrow$ or $o\downarrow$). The medium-level locations are associated with invariants $c \leq u^{\uparrow}$ or $c \leq u^{\downarrow}$. The guard associated with edges outgoing downwards from these locations is $c \geq l^{\uparrow}$ or $c \geq l^{\downarrow}$.

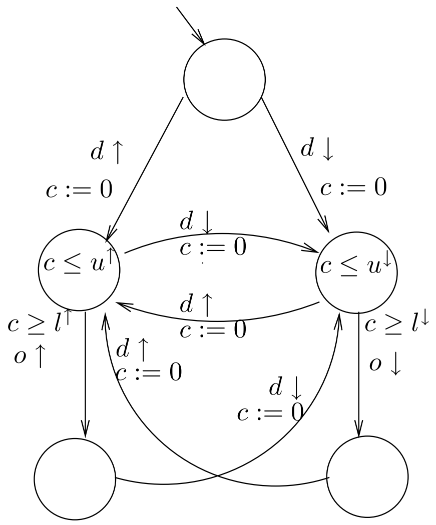

Fig. 4. Timed automaton of a wire component with delays $[l^{\uparrow}, u^{\uparrow}]$ and $[l^{\downarrow}, u^{\downarrow}]$ to propagate an edge from d to o.

A latch is a component that can store one bit of data. It has two inputs d (*data to be latched*) and e (*enable*), and one ouput q. It propagates the input data d to its output o (with a delay in $[l^{\uparrow}, u^{\uparrow}]$ or $[l^{\downarrow}, u^{\downarrow}]$) as long as e is high. While e is low, q keeps its value (even if d changes). A latch corresponds to the timed automaton depicted on Fig. 5 (using the same conventions as for the wire automaton). There are 6 locations and 1 clock c. The name of the locations $e_i d_j$ reflects the values i and j of e and d respectively; besides $e_1 d_j B$ indicates that an output $q\uparrow$ (resp. $q\downarrow$) has already been output if $j = 1$ (resp. $j = 0$).

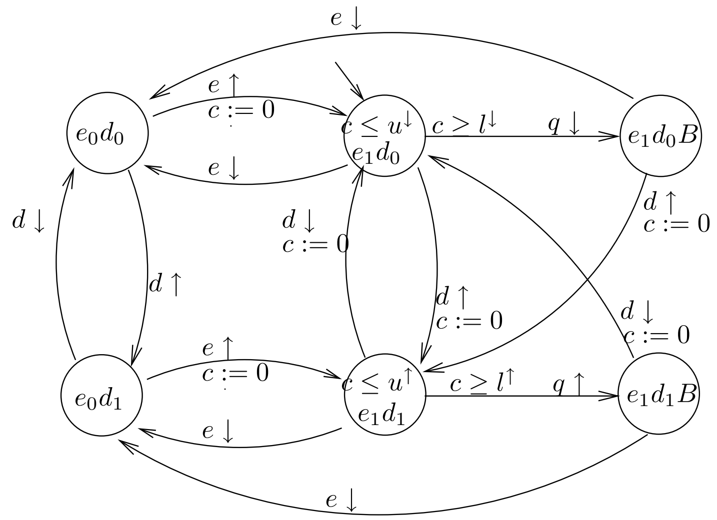

Fig. 5. Timed automaton of a latch component with delays $[l^\uparrow, u^\uparrow]$ and $[l^\downarrow, u^\downarrow]$ to propagate an edge from d to q.

3.3 Reachability Analysis

In order to verify property: $t_{CK\to Q}^{D,WEN} \le t_{max}$, we model the behavior of the memory along two cycles:

- a 1st cycle where the values of D and WEN are set t_{setup_D} and $t_{setup_{WEN}}$ time before the 2nd rising edge of CK (corresponding to the write operation); input signal WEN is modelled as a falling edge, followed by a low level (selection of a write command), and input signal D is modelled as a rising edge, followed by a high level (in keeping with the stabilization requirement $setup_D$).
- a 2nd cycle where the write operation is performed (the D value is propagated on Q port).

Accordingly, the observation of the generated states is done along two cycles (see Fig. 6).

In the next subsection, we will explain how the verification method applies without instantiation of the parameters. Let us first explain how the method works given a specific implementation of the memory: in the rest of this section, we assume all the parameters to be instantiated with the values of the datasheet and those given by simulation.

A central clock s (initialized to 0, and never reset) is used to measure the evolution of time, during 2 cycles. We also use a flag q (initialized to 0), which stores the fact that the rising edge of input signal D has reached Q port. The value of s when flag q is set to 1, corresponds to the sought value $t_{CK\to Q}^{D,WEN}$.

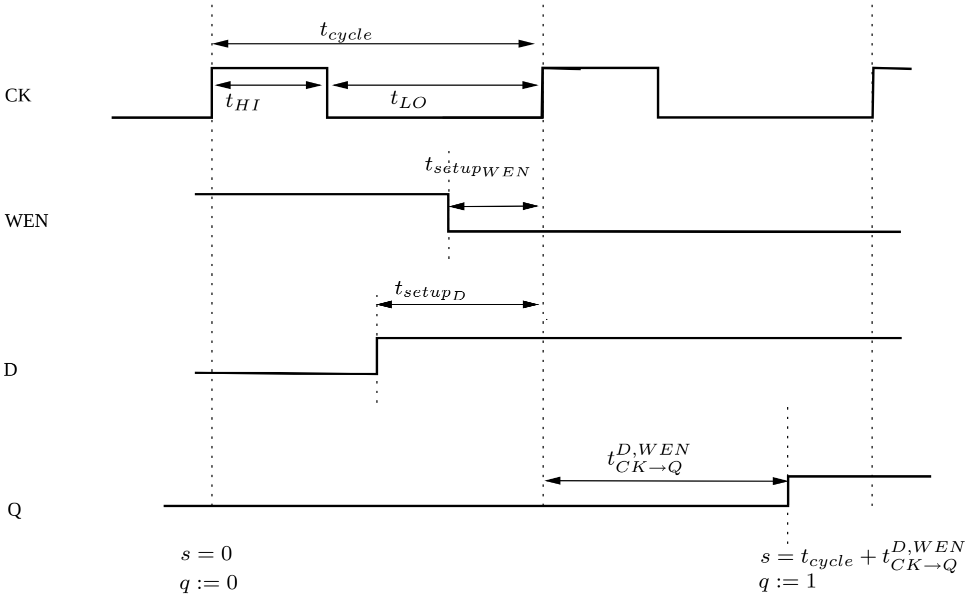

Fig. 6. The *write* operation used in our experiment.

Starting from $s = 0 \wedge q = 0$, the set of reachable states is iteratively computed until, either:

- the switch of flag q occurs before 2 cycles, or
- 2 cycles are run out without any switch of q.

This yields a set of final states, denoted by $Post^{2t_{cycle}}$, which can be decomposed into:

- *good* states, i.e, states, for which the switch of q has occurred ($q = 1 \wedge s = t_{cycle} + t^{D,WEN}_{CK\to Q} \leq 2t_{cycle}$).
- *bad* states, i.e., states, for which 2 cycles has run out without any switch of q ($q = 0 \wedge s = 2t_{cycle}$).

The property holds iff:

- all the final states of $Post^{2t_{cycle}}$ are good (no bad state), and
- for each final state, the value $t^{D,WEN}_{CK\to Q}$ of s is at most equal to t_{cycle} (i.e.: $t^{D,WEN}_{CK\to Q} \leq t_{cycle}$).

3.4 Timing Constraints Extraction by Stepwise Refinement

Let us now explain how to perform the reachability analysis at a *generic* level, without setting the parameters to some specific values of a given implementation. We denote by *Assumption*, the set of symbolic constraints, relating the parameters t_{HI}, t_{LO}, t_{setup_D} and $\{[l_i, u_i]\}$ together. At the beginning of the process, $Assumption$ is $True$, which means that we start with the most general parameters (l_i is just assumed to be less than or equal to u_i). Given an initial set of states characterized by $Assumption$, we perform reachability analysis on 2 cycles. We thus generate a set of constraints, denoted by $Post^{2t_{cycle}}(Assumption)$. The first run of $Post^{2t_{cycle}}(Assumption)$, with $Assumption = True$, usually contains bad (final) states. The *refinement* process consists to eliminate such bad states, by restricting the possible values of the parameters, as follows:

1. select a bad state of $Post^{2t_{cycle}}(Assumption)$;
2. detect a "suspect" constraint;[7]
3. add the negation of this subconstraint to $Assumption$.
4. Recompute $Post^{2t_{cycle}}(Assumption)$ after having reset s and q to 0.

And so on, perform iteratively 1-2-3-4 until no bad state is generated.

At each run, $Assumption$ is a conjunction of linear constraints with an increasing number of conjuncts. Accordingly, $Post^{2t_{cycle}}(Assumption)$ decreases at each run. At the end, the refinement process outcomes two formulas, $Assumption$ and $Final \equiv Post^{2t_{cycle}}(Assumption)$:

- $Assumption$ is a conjunctive constraint binding parameters $t_{HI}, t_{LO}, t_{setup_D}$, $t_{setup_{WEN}}$, $\{[l_i, u_i]\}$.
- $Final$ is a set of conjunctive constraints relating $t^{D,WEN}_{CK\rightarrow Q}$ to $t_{HI}, t_{LO}, t_{setup_D}$, $t_{setup_{WEN}}$, $\{[l_i, u_i]\}$.

Moreover, by construction, we have:

- $Final$ is the set of states reachable from $Assumption$ for which signal D has reached Q before 2 cycles (i.e., such that $q = 1 \wedge s = t_{cycle} + t^{D,WEN}_{CK\rightarrow Q} \leq 2t_{cycle}$).
- the propagation of D on Q *always* occurs *before* 2 cycles. ($Final$ contains *all* the states reachable from $Assumption$ such that $q = 1$).[8]

This set of constraints can be used for different purposes:

1. The validation of a specific implementation of SPSMALL. It consists in checking that:
   (a) the specific values of the datasheet ($t_{HI}, t_{LO}, t_{setup_D}$, $t_{setup_{WEN}}$) and those of the internal delays $\{[l_i, u_i]\}$ are *compatible*, i.e: they satisfy $Assumption$ all together;
   (b) ensure that $t^{D,WEN}_{CK\rightarrow Q} \leq t_{max}$.

2. The optimisation of some parameters of the memory: it can be an external parameter (such as t_{setup} of an input signal) or an internal timing (such as $[l_i, u_i]$ of an internal component). For instance, we can find the minimal value of t_{setup_D} such that all constraints remain satisfiable.

[7] In our context, a *suspect* constraint is a constraint violated by the values of the parameters of an SPSMALL available instance, *viz.* $SP1$ (see Sect. 4.3).

[8] This comes from the fact that all the bad states have been eliminated.

4 Verification of SPSMALL

We now apply the above method for deriving the set $Assumption \cup Final$ of constraints associated with memory SPSMALL, and checking the correctness of two of its instances $SP1$, a "high-speed" implementation, and $SP2$, a "low-power" implementation.

4.1 Decomposition

In practice, we cannot apply directly the method described above, because we cannot perform reachability analysis with HYTECH due to the high number (34) of parameters. The model is therefore decomposed into three parts (see the dashed lines on Fig. 3):

- The 1st part represents the D's edge propagation through $latch_D$: The input signals are D and CK. The output is the output of $latch_D$, denoted by q_1. The goal is to compute the constraints on lower and upper bounds on $t^{D}_{CK \to q_1}$.
- The 2nd part represents the propagation of the WEN edge through $latch_{WEN}$ (among other components). The input signals are WEN and CK. The output is denoted by wel. The goal is to compute the constraints on the lower and upper bounds of $t^{WEN}_{CK \to wel}$.
- The 3rd part represents the propagation of q_1's edge through Q. The input signals are q_1 and wel. The output is Q. The goal is to compute the constraints on the lower and upper bounds of $t^{q_1,wel}_{CK \to Q}$. Using the bounds $t^{WEN}_{CK \to wel}$ and $t^{D}_{CK \to q_1}$ found for q_1 and wel respectively in the two first parts, this will allow us finally to determine the constraints on $t^{D,WEN}_{CK \to Q}$.

4.2 Generic Constraints

We analyze separately each part, thus obtaining constraints binding intermediate input and output parameters (see [8]). The "suspect" constraints discarded during the refinement process (see Sect. 3.4) are those incompatible with the values of the delay intervals $[l^{\uparrow}, u^{\uparrow}]$ and $[l^{\downarrow}, u^{\downarrow}]$ of instance $SP1$ (see Sect. 4.3). By recombination of these separate sets of constraints, we obtain constraints relating the inputs and outputs of the whole memory, that are given below. For conciseness, we consider only the case of a rising edge of D ($D\uparrow$):

$Assumption$:

$$t_{setup_D} + u^{\downarrow}_2 + u^{\downarrow}_3 < l^{\uparrow}_0 + t_{LO} \wedge t_{HI} + t_{LO} < l^{\downarrow}_2 + l^{\downarrow}_3 + t_{setup_D} \wedge u^{\downarrow}_2 + u^{\downarrow}_3 + u^{\uparrow}_1 < t_{LO}$$
$$\wedge\ u^{\downarrow}_3 + t_{setup_{WEN}} < t_{LO} + u^{\downarrow}_{13} \wedge u^{\downarrow}_{13} + u^{\downarrow}_{14} < t_{setup_{WEN}} + l^{\downarrow}_3 \wedge u^{\downarrow}_{14} < t_{HI}$$
$$\wedge\ u^{\downarrow}_{13} + u^{\downarrow}_{14} + u^{\downarrow}_{16} < t_{setup_{WEN}} + l^{\downarrow}_3 + u^{\downarrow}_{15} \wedge t_{setup_D} + u^{\downarrow}_3 + u^{\downarrow}_{15} \leq l^{\uparrow}_5 + l^{\uparrow}_0 + l^{\uparrow}_1$$
$$\wedge\ u^{\uparrow}_5 + u^{\uparrow}_0 + u^{\uparrow}_1 \leq l^{\downarrow}_8 + l^{\downarrow}_3 + l^{\downarrow}_{15} + t_{setup_D}$$

Final:
$l_3^{\downarrow} + l_{15}^{\downarrow} + l_8^{\downarrow} + l_7^{\uparrow} \leq t_{CK\rightarrow Q}^{D,WEN} \leq u_3^{\downarrow} + u_{15}^{\downarrow} + u_8^{\downarrow} + u_7^{\uparrow}$

The constraints are symmetrical in the case of a falling edge of D ($D\downarrow$): more precisely, each $l_i^{\uparrow}$ (resp. $u_i^{\uparrow}$) should be changed into $l_i^{\downarrow}$ (resp. $u_i^{\downarrow}$).

From *Final* (and its symmetrical counterpart for $D^{\downarrow}$), we infer the following constraint, guaranteeing property $t_{CK\rightarrow Q}^{D,WEN} \leq t_{max}$:

$$\boxed{u_3^{\downarrow} + u_{15}^{\downarrow} + u_8^{\downarrow} + \max\{u_7^{\uparrow}, u_7^{\downarrow}\} \leq t_{max}.} \qquad (*)$$

Constraints of *Assumption* (and its symmetrical counterpart for $D^{\downarrow}$) can be used to determine lower and upper bounds for t_{setup_D}:

$$\max\{u_0^{\uparrow}+u_1^{\uparrow}+u_5^{\uparrow}-l_8^{\downarrow}-l_3^{\downarrow}-l_{15}^{\downarrow}, u_0^{\downarrow}+u_1^{\downarrow}+u_5^{\downarrow}-l_8^{\downarrow}-l_3^{\downarrow}-l_{15}^{\downarrow}, t_{HI}+t_{LO}-l_2^{\downarrow}-l_3^{\downarrow}\} \leq t_{setup_D}$$
$$\wedge\ t_{setup_D} \leq \min\{l_0^{\uparrow}+l_1^{\uparrow}+l_5^{\uparrow}-u_3^{\downarrow}-u_{15}^{\downarrow}, l_0^{\downarrow}+l_1^{\downarrow}+l_5^{\downarrow}-u_3^{\downarrow}-u_{15}^{\downarrow}, t_{LO}+l_0^{\uparrow}-u_2^{\downarrow}-u_3^{\downarrow}\}$$

We thus infer an *optimal* value of t_{setup_D}, denoted by $t_{setup_D}^{opt}$, which corresponds to its lower bound[9]:

$$\boxed{t_{setup_D}^{opt} = \max\{u_0^{\uparrow}+u_1^{\uparrow}+u_5^{\uparrow}-l_8^{\downarrow}-l_3^{\downarrow}-l_{15}^{\downarrow}, u_0^{\downarrow}+u_1^{\downarrow}+u_5^{\downarrow}-l_8^{\downarrow}-l_3^{\downarrow}-l_{15}^{\downarrow}, t_{HI}+t_{LO}-l_2^{\downarrow}-l_3^{\downarrow}\}}$$

A similar expression can be obtained for the optimal value of $t_{setup_{WEN}}$.

4.3 Application to Instance $SP1$

The above sets of constraints now allow us to give a formal justification of the correctness of the instance $SP1$. The values of the datasheet are (in tens of picoseconds):
$t_{HI} = 36, t_{LO} = 74, t_{setup_D} = 108, t_{setup_{WEN}} = 48, t_{max} = 56$.
The internal delays are (in tens of picoseconds):

	$[l_i^{\uparrow}, u_i^{\uparrow}]$	$[l_i^{\downarrow}, u_i^{\downarrow}]$
(l_0, u_0)	$(95, 95)$	$(66, 66)$
(l_1, u_1)	$(14, 14)$	$(18, 18)$
(l_2, u_2)	$(23, 30)$	$(23, 30)$
(l_3, u_3)	$(5, 5)$	$(2, 2)$
(l_5, u_5)	$(22, 22)$	$(45, 45)$
(l_7, u_7)	$(21, 21)$	$(20, 20)$
(l_8, u_8)	$(0, 0)$	$(22, 22)$
(l_{13}, u_{13})	$(11, 11)$	$(8, 8)$
(l_{14}, u_{14})	$(21, 22)$	$(21, 22)$
(l_{15}, u_{15})	$(14, 14)$	$(11, 11)$
(l_{16}, u_{16})	$(24, 24)$	$(0, 0)$

[9] Actually, we checked that $t_{setup_D}^{opt}$ also satisfies constraints, coming from other parts of the circuit, which are not limitative in the case of instances $SP1$ and $SP2$.

We check that all the constraints of $Assumption$ (and those of the symmetrical counterpart for $D\downarrow$) are satisfied. We also check constraint $(*)$. This shows that $SP1$ satisfies $t^{D,WEN}_{CK\to Q} \leq t_{max}$. Furthermore, we find the value 96 for $t^{opt}_{setup_D}$, which matches with the optimal value found by simulation by the designer.

4.4 Application to Instance $SP2$

As mentioned above, the values of the datasheet and internal delays of $SP1$ have been used in the refinement process in order to derive the appropriate set of generic constraints. Therefore, the derived set of constraints $Assumption \cup Final$ of Sect. 4.2 has not been produced independently from $SP1$, and the correctness of $SP1$ has been checked *a posteriori*. The constraints are however available now once for all, and can be reused to check immediately any other instance of SPSMALL. This is what has been done with instance $SP2$. The values of the datasheet are (in tens of picoseconds):

$t_{HI} = 72, t_{LO} = 170, t_{setup_D} = 241, t_{setup_{WEN}} = 109, t_{max} = 142$.

The internal delays are (in tens of picoseconds):

	$[l_i^{\uparrow}, u_i^{\uparrow}]$	$[l_i^{\downarrow}, u_i^{\downarrow}]$
(l_0, u_0)	$(197, 197)$	$(140, 140)$
(l_1, u_1)	$(60, 60)$	$(58, 58)$
(l_2, u_2)	$(66, 66)$	$(43, 43)$
(l_3, u_3)	$(8, 8)$	$(4, 4)$
(l_5, u_5)	$(61, 61)$	$(63, 63)$
(l_7, u_7)	$(47, 47)$	$(52, 52)$
(l_8, u_8)	$(0, 0)$	$(42, 42)$
(l_{13}, u_{13})	$(23, 23)$	$(23, 23)$
(l_{14}, u_{14})	$(35, 35)$	$(36, 36)$
(l_{15}, u_{15})	$(56, 56)$	$(43, 43)$
(l_{16}, u_{16})	$(24, 24)$	$(0, 0)$

As in the case of $SP1$, we check that $SP2$ satisfies $t^{D,WEN}_{CK\to Q} \leq t_{max}$. Besides, we find the value 229 for $t^{opt}_{setup_D}$, which matches with the optimal value found by electrical simulation by the designer.

5 Final Remarks

We have shown in this paper how to apply parametrized methods to verify timed properties of the generic architecture of a memory. We have thus found certain sufficient conditions (under the form of linear inequalities between parameters) that ensure that the response time lies between certain bounds, and check this property on an instance $SP1$ of the memory. These linear inequalities have been also used to derive the optimal values of setup timings of input signals ($viz.$, setup timing for D). This analysis can be immediately applied to the verification of other instances of the SPSMALL memory, as examplified here on instance $SP2$.

Our method requires from the user a certain knowledge of the circuit, especially in the stepwise refinement process when taking the refutation of a suspect constraint. By negating such constraints, we focus on a certain class of the circuit, disregarding other possible circuit implementations. In the future, we plan to improve this phase of constraint selection, in order to make the method more complete and more automatic.

References

1. HSIM Simulator Description. In `http://www.synopsys.com/products/`.
2. TLL Transistor Abstraction Tool description. In `http://www.transeda.com/products/`.
3. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science 126*, pages 183–235, 1994.
4. M. Baclet and R. Chevallier. Timed verification of the SPSMALL memory. In *1th International Conference Memory Technology and Development*, pages 1–2, Giens France, 2005.
5. M. Bozga, H. Jianmin, O. Maler, and S. Yovine. Verification of asynchronous circuits using timed automata. In *TPTS'02, ENTCS vol 65*, 2002.
6. J.A. Brzozowski and C-J.H. Seger. *Asynchronous Circuits*. Springer, 1994.
7. R. Chevallier, E. Encrenaz-Tiphène, L. Fribourg, and W. Xu. Timing analysis of an embedded memory: SPSMALL. In *10th WSEAS International Conference on Circuits, Athens, Greece*, 2006.
8. R. Chevallier, E. Encrenaz-Tiphène, L. Fribourg, and W. Xu. Verification of the Generic Architecture of a Memory Circuit Using Parametric Timed Automata. Technical Report LSV-06-??, Laboratory Specification and Verification, 2006.
9. R. Clariso and J. Cortadella. The octahedron abstract domain. In *Proc. 11th Static Analysis Symposium (SAS), LNCS 3148, Springer, pp. 312-327*, 2004.
10. R. Clariso and J. Cortadella. Verification of timed circuits with symbolic delays. In *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 628-633*, 2004.
11. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL, pp. 238-252*, 1977.
12. D. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems, LNCS 407, Springer*, 1989.
13. N. Halbwachs. Delay analysis in synchronous programs. In *CAV'93, LNCS 697, Springer, pp. 333-346*, 1993.
14. T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. A User Guide to HYTECH. In *TACAS'95, LNCS 1019, Springer, pp.41-71*, 1995.
15. K. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1:134–152, 1997.
16. O. Maler and A. Pnueli. Timing analysis of asynchronous circuits using timed automata. In *CHARME'95, LNCS 987, Springer, pp.189-205*, 1995.
17. R. Ben Salah, M. Bozga, and O. Maler. On timing analysis of combinational circuits. In *FORMATS'03, LNCS 2791, Springer, pp.204-219*, 2003.
18. S.Yovine. KRONOS: A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, 1:123–133, 1997.