M. Baclet and R. Chevallier

Using UPPAAL to verify an on–chip memory

Research Report LSV–04–10, May 2004

Laboratoire Spécification et Vérification

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

Ecole Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

http://www.lsv.ens–cachan.fr/Publis/
Research Report LSV–04–11, Lab. Spécification et Vérification, CNRS & ENS de Cachan, France, May 2004

Using UPPAAL to verify an on-chip memory*

Manuel BACLET[1] and Rémy CHEVALLIER[2]

[1] LSV - CNRS UMR 8643 - ENS de Cachan
61, Av. du Président Wilson - 94235 Cachan Cedex - France
baclet@lsv.ens-cachan.fr
[2] ST Microelectronics - D.A.I.S - Central R&D
850, rue Jean Monnet - 38921 Crolles cedex - FRANCE
remy.chevallier@st.com

Abstract. As the complexity of designed Systems on Chip (SoC) increases, due to the ever growing number of transistors that can be integrated on a chip, methods have been developed to validate a design before it is actually manufactured. Currently, the method that is the most used is based on software simulation, which does not ensure that all the behaviors are correct. In order to provide exhaustive verification, formal methods such as model-checking can be used. However those approaches are often confronted by the state explosion problem.
Among the different components embedded on SoCs, memories are of particular interest, since the SoCs require both large memory capacities (leading to a large number of transistors dedicated to the memories) and high performance level. The aim of the paper is to verify a very simple memory component with the real-time model checker Uppaal, taking into account the electrical propagation delays.

keywords: Real-time systems, Formal verification, Systems on Chip, Uppaal

1 Introduction

Timed automata have been proposed by R. Alur and D. L. Dill in [AD90]. They are a suitable model for describing and verifying real-time systems. They have been widely studied and efficient algorithms have been designed and implemented in real-time model-checker such as HyTech[1], Kronos[2] or Uppaal[3] and the verification of many real cases have been successfully carried out. For instance, Bang & Olufsen proposed an audio/video protocol so that several communication sources could use the same channel in order to communicate. The protocol was designed to detect collisions. However, in some cases, collisions happened and the engineers were not able to correct the error. The protocol was modeled with the model-checker Uppaal [HSLL97] and a scenario leading to the bug was exhibited. A new version of the protocol was proposed, which was proved correct by the same approach.

System on Chips The micro-electronics firms have two main focuses on the digital products: firstly the improvement of the product capabilities, and secondly the decrease of the product cost. In the cellular phone market for instance, the products have more and more functions but their prices have to remain stable for the customer. In order to conciliate these two antagonistic objectives, the only way is the increase of the system integration capability. One chip includes today more functionality than few chips of yesterday plugged together: a complete system, processors, peripherals and memories are put now in the same component. This component is called a System on Chip (SoC). The other big issue is that the improvement of the system capability induces the explosion of the data flow in the SoC and then, the explosion of the memory size needed in the component. The memories integrated in a SoC take then more and more surface on the silicon and the time spent by the system to read and write data in

* Partially supported by project MEDEA+ Blueberries
[1] http : //www − cad.eecs.berkeley.edu/ tah/HyTech/
[2] http : //www − verimag.imag.fr/TEMPORISE/kronos/
[3] http : //www.uppaal.com/

the memories is directly impacting the performances of the SoC. The timings of the memories must be then well computed and checked. The verification by timed automata is then a good solution in middle term in order to be sure that the right behavior of the memory is preserved even if the timings of the memories needed by the SoC are very aggressive.

A one-bit memory component In this paper, we propose to model and verify a very simple one-bit memory component, using the real-time model checker Uppaal. The main point is that we want to take into account the propagation delay of electrical signals through transistors. The description of the memory, provided by ST Microelectronics, details the functional behavior of the component and the electrical network that implements it. The memory can be divided into two main subcomponents: an address manager and a memory point. The address manager is to enable the memory point if the incoming address is valid. The component has one output pin and several input pins among which one is a clock signal: the component is synchronous. The electrical network consists in the connection of several *logical gates* (which are unsynchronized components that compute a boolean function) and *registers* (which are activated at each rising edge of the clock). Each component has a characteristic *propagation delay* which corresponds to the time needed to obtain the right output value if its inputs are kept constant. Those delays leads to the definition of *setup* and *hold* time for the inputs of the memory: for the signals to be correctly interpreted, they must keep constant values in some interval around the rising edges of the clock. We propose to model each component and each input signal by mean of a timed automaton. Those automaton will use messages to communicate with one another and electrical signals between them will be represented by shared variables. We point out the fact that even if the component is not realistic since it has only one memory point, our approach can be viewed as a general modeling framework for on-chip memories.

Related works Our approach is close to the modeling of asynchronous circuits as proposed in [BJMY02]. However, the memory component we study is a synchronous device and we are not only interested in the verification of timings but also in the verification of the function implemented by the component. Since the memory is synchronous, we will have to model both the rising edges of the clock that drives the system and the registers which are the only sub-components that take the clock signal into account.

Outline The memory component and the specifications it is intended to have are precisely detailed in section 2. In section 3, the models handled by Uppaal are briefly described and we explain how the different component of the memory are modeled in the Uppaal syntax. Finally, in section 4, the way the verification was carried out and our results are detailed.

2 Description of the memory

We begin this section by describing the functional behavior of the memory and then we give details about the actual implementation in terms of gates network. In chip design, electrical signals values can be 0, 1 or X (for undetermined value). This will be taken into account in our study for signals between components. Inputs signals will be supposed to have binary values (0 or 1) and boolean functions are extended in the following way:
Let f be a boolean function.

$$\begin{array}{rccc} f: & \{0,1\}^n & \longrightarrow & \{0,1\} \\ & (x_1,x_2,..,x_n) & \longmapsto & f(x_1,x_2,..,x_n) \end{array}$$

f is extended to a function $\overline{f}$ from $\{0,1,X\}^n$ into $\{0,1,X\}$ by setting:

$$\begin{array}{rccc} \overline{f}: & \{0,1,X\}^n & \longrightarrow & \{0,1,X\} \\ & (x_1,x_2,..,x_n) & \longmapsto & \begin{cases} f(x_1,x_2,..,x_n) \text{ if } \forall i, x_i \in \{0,1\} \\ X \text{ otherwise.} \end{cases} \end{array}$$

2.1 Functional description

The memory is a synchronous device: all operations (inputs, outputs) are synchronized with the rising edge of a unique clock. A clock is an electrical periodic signal with crenel shape. Rising edges are times where the signal goes from low level to high level. The environment performs an operation of reading or writing on the memory by setting special value on its input pins. The different signals involved in these operations are summarized in the following subsection.

Signals description

Pin name	direction	function
CK	IN	Clock - Active on rising edge
CSN	IN	Cell Select Not - Active low
WEN	IN	Write Enable Not - Active low
A	IN	Address Bus - One bit width
D	IN	Data Input Bits - One bit width
Q	OUT	Data Output Bits - One bit width

Address (A), input data (D), write enable not (WEN) and chip select not (CSN) are latched in at the rising edge of the clock (CK). If CSN is set high at the rising edge of CK, it inhibits the access to the memory for the current clock cycle, and the memory is held in stand-by mode. WEN precises the direction of a memory operation (read when set to 1 and write when set to 0). Output data are internally latched at the completion of each access, and are valid until the next positive edge of CK. During write operations, data written into the memory are copied back to the output data pins (write-through).

2.2 Timing diagrams

Timing description Two timings are defined for each input signals: the setup timing, $setup$ and the hold timing, $hold$. These timings are the delays before and after the rising edge of the clock for the signals to be stable to be correctly interpreted by the system, as can be seen on Fig. 1 and 2. This means that if a rising edge occurs at date t, the input signal must keep constant value in the interval $[t - setup, t + hold]$. Those definitions are motivated by the fact that the electrical signals are not instantaneous and hence delays of propagation along wires and through gates must be considered to obtain right behaviors.

Timings

Symbol	Value (ns)
$tcycle$	20
$tckh$	10
$tckt$	10
taa	0.340
$taaw$	0.340
tps	0.886
tph	0.397
tas	0.886
tah	0.397
tws	0.489
twh	0.397
tds	0.345
tdh	0.397

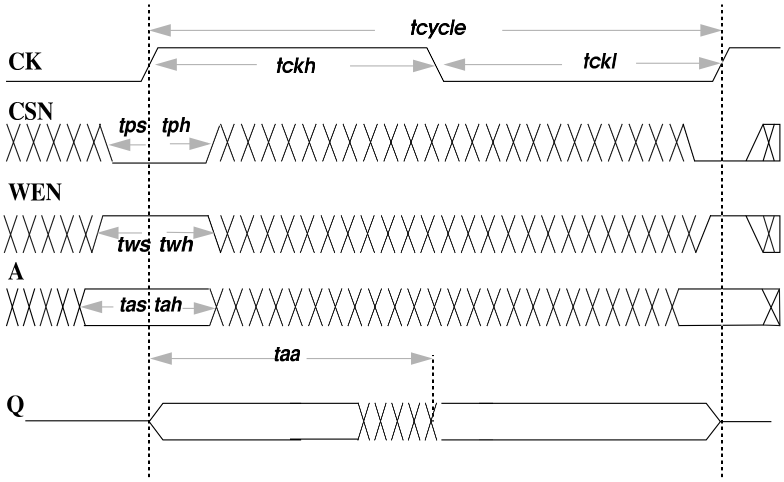


Fig. 1. A read cycle

Read cycle During a read cycle beginning at date t, at the address A, the environment sets CSN to 0 and WEN to 1 at the rising edge of the clock. The result is available on the output pin Q from date $t + taa$ until the next rising edge of the clock.

Write cycle During a write cycle at date t, at the address A, the environment sets CSN to 0 and WEN to 0 at the rising edge of the clock. The incoming data which has been stored in the memory point, is then available on the output pin Q from date $t + taaw$ until the next rising edge of the clock.

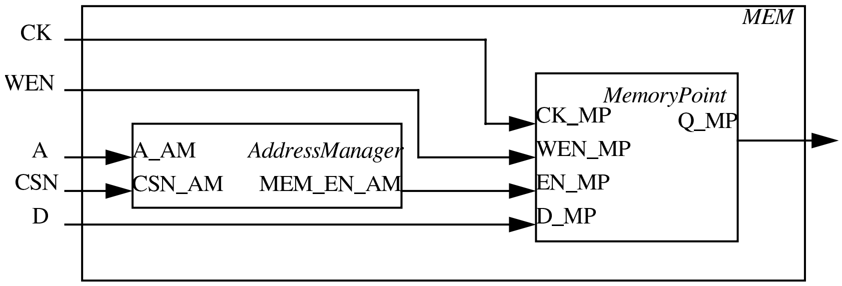
Definitions and restrictions

Words The number of words defines how many addresses are physically accessible. In this study, there is only one address bit and then two addresses are defined. However, only one address is valid as we consider only one memory point.

Bits The number of bits defines how many bits are contained in each address location. In this study, only one bit for the address is available.

2.3 Register Transfer Level description

We now describe the real implementation of the memory. This implementation is divided into two sub-entities called «MemoryPoint» and «AddressManager». The mapping of the system is like the following:



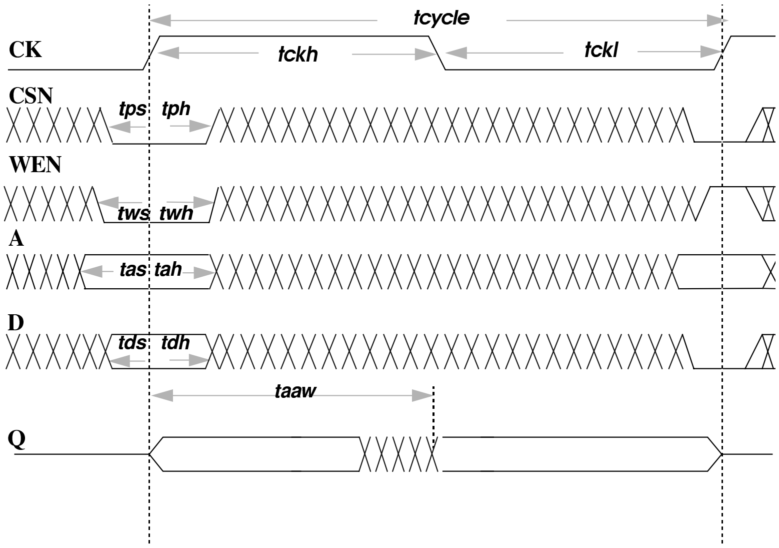
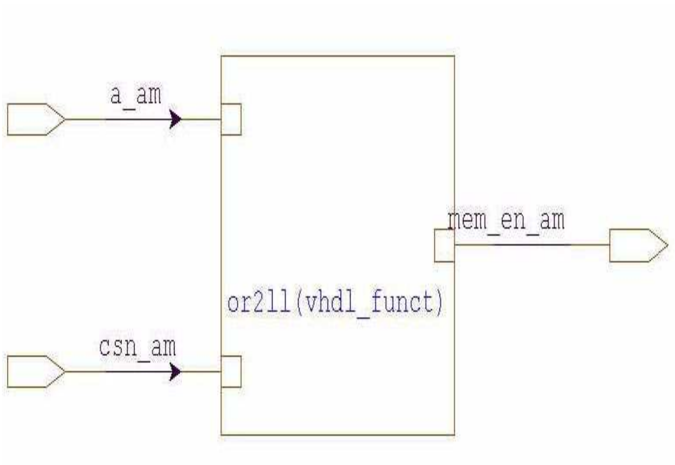


Fig. 2. A write cycle

AddressManager behavior The aim of this sub-entity is to enable or disable the memory point. Its output depends on the value of the address A and the value of the chip select not CSN port. If the Address is equal to 0 (A=0) and chip select not equal to 0 (CSN=0) then the Memory point is enable (MEM_EN_AM=0). In the other cases, the memory point is disable (MEM_EN_AM=1).

MemoryPoint behavior When the MemoryPoint is not selected (EN_MP=1) and when a rising edge of the clock occurs, the output Q_MP is 0.
When the MemoryPoint is selected (EN_MP=0) and when a rising edge of the clock occurs, if the WEN_MP is equal to 1 (WEN_MP=1), the MemoryPoint is in the read state, and the output Q_MP takes the value saved in the memory point. When the MemoryPoint is selected (EN_MP=0) and when a rising edge of the clock occurs, if the WEN_MP is equal to 0 (WEN_MP=0), the MemoryPoint is in written state, the output Q_MP takes the value of the data called D_MP, and the value of the data D_MP is saved into the memory point.

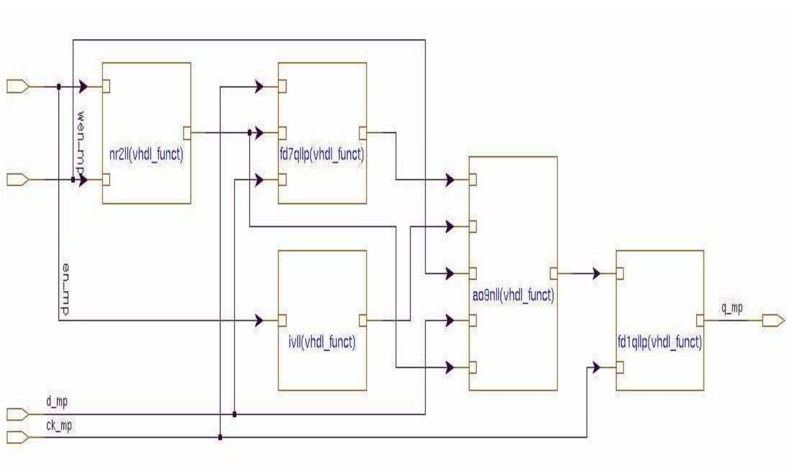
Address Manager



or2ll is a gate that computes the logical OR of the input signals:

$$mem\_en\_am = a\_am \vee csn\_an.$$

Memory Point



The output of the gate named *gate* will be denoted by $gate\_out$.

nr2ll is a logical gate that computes the logical NAND of its inputs:

$$nr2ll\_out = (\neg mem\_en\_am) \wedge (\neg wen\_mp).$$

ivll is a logical gate that computes the negation of its input:

$$ivll\_out = \neg en\_mp.$$

ao9nll is a logical gate that computes the boolean function:

$$ao9nll\_out = (fd7qllp\_out \wedge ivll\_out \wedge wen\_mp) \vee (d\_mp \wedge nr2ll\_out)$$

fd7qllp is a register. At each rising edge of the clock, if $nr2ll\_out = 1$ it stores d_mp in its internal memory else it keeps the previous value. After some delay the value stored is available on the output pin.

fd1qllp is a register. At each rising edge of the clock, it stores $ao9nll\_out$ in its internal memory and after some delay makes it available on its output pin.

Gates delays The following array gives the delays for the electrical signals to go through a gate:

Gate name	Delay
OR2LL	0.191ns
AO9NLL	0.345ns
NR2LL	0.144ns
IVLL	0.066ns
FD1QLLP	0.340ns
FD7QLLP	0.397ns

2.4 Specifications

The properties we want to check are the following:

i. When the environment performs a write operation on an invalid address, the output is set to zero after $taaw$ ps. (At a rising edge of the clock, if WEN is set to 0, CSN to 0 and A to 1, output Q is set to 0 after $taaw$ ps.)
ii. When the environment performs a write operation, the new data is effectively stored in the memory and available on the output pin $taaw$ ps after the rising edge of the clock. (At a rising edge of the clock, if WEN is set to 0, CSN to 0 and A to 0, the value of the input data D is saved into the memory point and output Q is set to the value of D after $taaw$ ps.)
iii. When the environment performs a read operation, the value stored in the memory is available on the output pin taa ps after the rising edge of the clock. (At a rising edge of the clock, if WEN is set to 1 and CSN to 0, Q is set to the value kept into memory after taa ps.)
iv. When the memory is not selected, the data stored in the component remains unchanged. (At a rising edge of the clock, is CSN is set to 1, output Q is set to 0 after $taaw$ ps and the value stored in the memory is kept.)

We now detail how the modeling was done, using the syntax of the real time model-checker Uppaal.

3 Modeling

We propose to model this memory by a network of timed automata described in the Uppaal syntax. These automata use shared variables to model the electrical signals. We recall that in our model, input signals have only two possible values: 0 and 1, the undetermined value X is not considered whereas it will be taken into account for the output of gates. The time unit is the picosecond. We now give a brief description of the temporized model-checker Uppaal.

3.1 Modeling with Uppaal

Descriptions of models handled by Uppaal can be found in many papers [BL96,LPY97]. Uppaal models are network of timed automata which are a variant of the original model proposed by R. Alur and D. L. Dill that is described in [AD94]. We briefly describe the characteristics of Uppaal.

In Uppaal, a model consists in a collection of timed automata which are synchronized via binary channel or broadcast channel. In the the case of the binary synchronization, two components synchronize through channels with a sender/receiver syntax. In the case of broadcast synchronization, one component (the sender) synchronizes with several components (the receivers). For instance, on a given channel c, a sender emits a signal denoted by c! and receivers synchronize with the sender via the corresponding reception denoted by c?. These two actions are called complementary actions. A timed automaton is a finite structure handling a finite set of variables. These variables are either clocks, which evolve with the same increase rate or bounded integer variables. A location in an automaton can be labeled with clock conditions, called invariants, which must be satisfied as long as the automaton remains in this state. A transition of the automaton can be decorated with three types of (optional) labels:

- a guard expressing conditions on the values of the variables, which must be satisfied for the transition to be fired,
- a synchronization label of the form c! or c?,
- a clock reset and an update of integer variables.

A (global) configuration of the model is a a couple (l, v) where l is a location vector (indicating the current state in each component of the model) and v is a valuation of the variables, that is a function assigning to each clock and integer variables a value. An execution in the network starts in initial locations of the different components with all the variables (clock and integer variables) set to zero. The semantics of the model is expressed by moves between configurations. Three types of moves can occur in a system: delay moves, internal moves and synchronized moves.

Delaying. As long as no invariant is violated, time may progress, without changing the location vector and the values of discrete variables. Clock values increase by the time elapsed.

Performing an internal action. If an internal action is enabled in a component (i.e. a transition without synchronization label), then the component can perform this internal action, without synchronizing with other component. The configuration is then changed following the same rules as for the synchronization described below.

Synchronizing. There are several methods of synchronization available in Uppaal. However, we will only describe the binary and broadcast synchronization since we will not use the other methods in the sequel.
In the binary case, two processes can synchronize on enabled transitions labeled with complementary actions. When it is the case, both transitions are fired at the same time and the update expression of the emitting transition is executed before the update expression of the receiving transition.
Broadcast channels allow one-to-many synchronizations. The intuitive idea is that an enabled transition emitting on a broadcast channel will synchronize with every enabled transition that can receive the message. This method differs from the binary synchronization since the enabled emitting transition can be fired even if no receiving transition is enabled. Like in the case of binary synchronization, all transitions are fired at the same time and the update expression of the emitting transition is executed first, then, the update expressions of the receiving transitions are executed in the order of the model description.

Another feature of Uppaal which will widely used in our modeling: locations can be decorated with a special flag called *committed* (in the figures, committed locations are marked with a c). This notion is used to model atomicity of transitions sequences in some given component. If the current location is committed, no time can elapse in this state and a transition must be fired immediately. In this way, two transitions separated by a committed location will be fired sequentially without delay.
Each input signal and each component will be modeled by a timed automaton. We now detailed this description.

3.2 Input signals

3.3 Clock signal

The automaton that represents the clock is quite simple. It is represented on Fig. 3. It has a clock c which can be read by the automata representing the input signals and the register gates. It sends a (broadcast) message $tick$ with period π representing the rising edge of the clock.

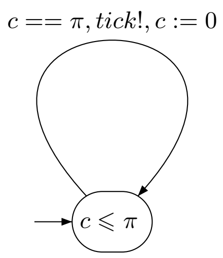


Fig. 3. Modeling of the clock signal

3.4 Binary input signals

Let in be an input signal with setup timing $setup$ and hold timing $hold$. Its model is represented on Fig. 4, where c is the clock of the automaton associated with the clock of the memory.

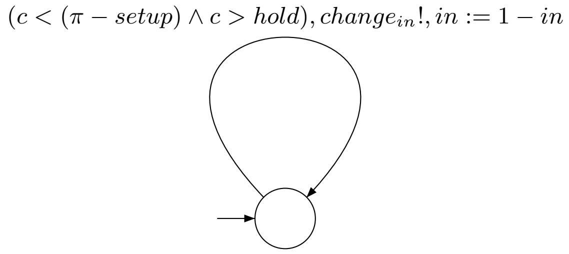


Fig. 4. Modeling of an input signal

The input signal can change its value ($in := 1-in$) at anytime except in an interval around the rising edge of the clock. When it is the case, it sends a broadcast message $change_{in}$ indicating that its value has changed.

Input signals CSN, WEN, D and A are modeled in a similar way.

3.5 Combinatorial components

Modeling a combinatorial component We now detail how we model a gate with inputs $in_1, in_2, ..., in_k$ and output out, that computes the boolean function f with delay δ.

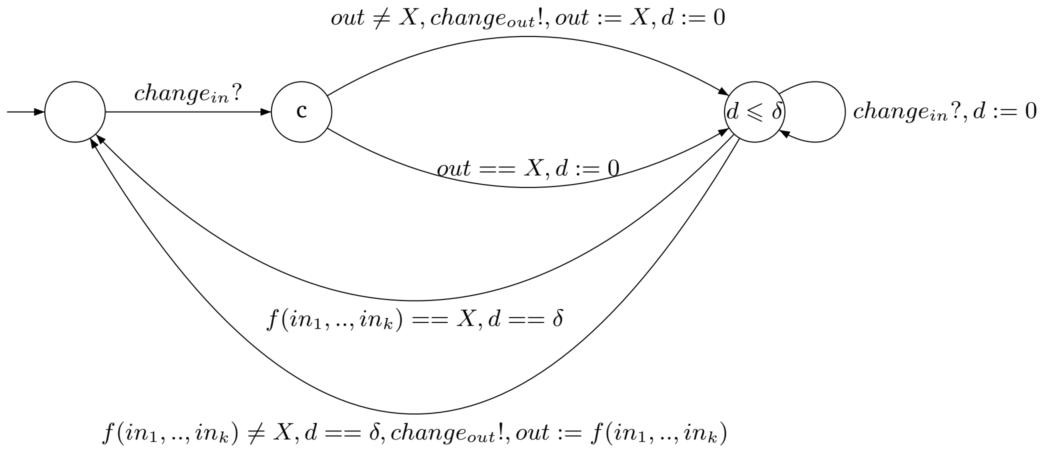


Fig. 5. Modeling of a combinatorial gate

The initial state is the state marked with an incoming arrow. When the automaton received a message $change_{in}$, indicating that the value on one input has changed, the current state becomes the committed state (marked with a c). This state must be left immediately and if the value of out is X then the system goes to the wait state while reseting the clock d. If the value of out is not X then a message $change_{out}$ is sent while setting out to X, reseting the clock d and going to the wait state. In the wait state, if a new message $change_{in}$ is received, the system stays in the wait state after reseting its clock. When a delay of δ has elapsed in the wait state. The system goes back to the initial state while updating its output in the following way:

- If the value of the boolean function on the inputs is X, then nothing changes.
- Else, the output is updated with the right value as a message $change_{out}$ is sent.

3.6 Synchronized components

Modeling a synchronized component We now detail how we model a register with inputs $in_1, in_2, ..., in_k$ and output out, that computes the boolean function f with delay δ. The register has an internal memory point which is represented by the variable mem. The automaton can read the value of the clock c from the automaton modeling the clock signal.

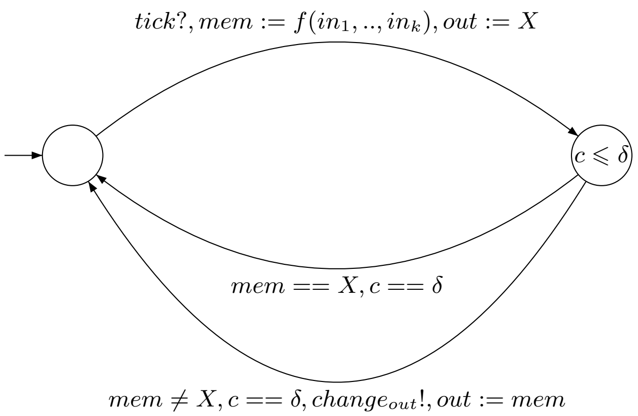


Fig. 6. Modeling of a synchronized gate

When this automaton receives a $tick$ message, it updates its internal memory mem with the result of the computation of $f(in_1, .., in_k)$, sets its output to X and enters a wait state. The system stays in this wait state until the value of the clock c from the automaton clock reaches value δ. Then it goes back to the initial state while updating its output and sending a message $change_{out}$ if necessary.

4 Model-checking and results

In this section, we describe how the specifications were encoded in the tool Uppaal and then, we detail the results of our experimentations.

4.1 Encoding specifications

As the formulae allowed in Uppaal are not expressive enough for our specifications, we use auxiliary automata, usually called *test automata* to describe the properties we want to check on our model.
We describe precisely how is modeled the first property. We recall that this property states that when a write operation is performed on an invalid address, the output is set to 0 $taaw$ ps after

the rising edge of the clock (†). The other properties are modeled in a similar way. Let $prop$ be the boolean formula $\neg wen \wedge \neg csn \wedge a$. The test automaton associated with property (†) is represented in Fig. 7: when, at a rising edge of the clock ($tick$ message), proposition $prop$ is satisfied, the current state becomes (or remains) the state with label $test$, otherwise, the current state is the initial state.

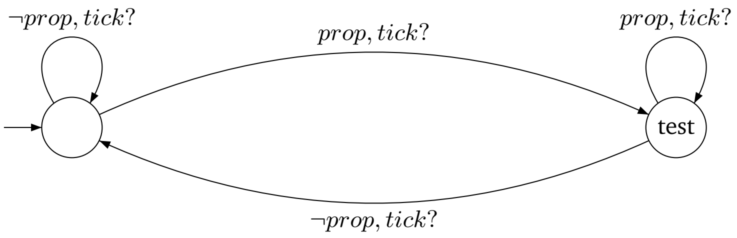


Fig. 7. Test automaton for a property

The logical formula associated to this automaton is:

$$A\Box\Big[(test \wedge (c > taaw)) \Rightarrow fd1qllp.out = 0\Big]$$

Intuitively, it says that «it is always true» ($A\Box$) that when the automaton is in state $test$ and when the clock c has value greater than $taaw$, then the value of the output ($fd1qllp.out$) is 0.

The other specifications are modeled in a similar way. We now detail the model-checking results.

4.2 Results

The model-checking was realized on a server with two 2.4 GHz Intel Xeon processors with 1 Go of memory. The testing of the four specifications of the memory and the lack of deadlock requires about 26Mo of memory and about 97 seconds of processor time. The results of the computation is that all properties are satisfied by the model. We extended the test by changing the values of setup and hold timings. We were able to ensure that the timings given in the description of the memory are optimal.

5 Conclusion

After the description of a memory component, we have explained how it was modeled with the model-checker Uppaal in order to verify if the implementation of the component was following its specifications. We were able to prove, using a small amount of resources, that the behaviors of the components are correct. Now, we plan to model a small but real memory component (three banks of two bits words) to see if complete modeling leads to a tractable model. If it is not the case, we will propose abstractions that allow to verify the properties on a smaller system while ensuring that the original system behave correctly. Another approach we want to explore is a way to reduce the number of clocks involved in the models, since several clock resets often happen at the same time.

References

[AD90] R. Alur and D. L. Dill. Automata for modeling real-time systems. In *Proc. 17th International Colloquium on Automata, Languages, and Programming (ICALP'90)*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990.

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[BBP03] B. Bérard, P. Bouyer, and A. Petit. Une analyse du protocole PGM avec UPPAAL. In *Actes 4ième Colloque sur la Modélisation des Systèmes réactifs (MSR'2003)*, pages 415–430. Hermès, 2003.

[BF99] B. Bérard and L. Fribourg. Automatic verification of a parametric real-time program: the ABR conformance protocol. In *Proc. 11th Int. Conf. Computer Aided Verification (CAV'99)*, volume 1633 of *Lecture Notes in Computer Science*, pages 96–107. Springer-Verlag, 1999.

[BJMY02] M. Bozga, H. Jianmin, O. Maler, and S. Yovine. Verification of asynchronous circuits using timed automata. In *Proc. 1st Workshop on the Theory and Practice of Timed Systems (TPTS'2002)*, volume 65 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2002.

[BL96] J. Bengtsson and F. Larsson. UPPAAL, a tool for automatic verification of real-time systems. Master's thesis, Department of Computer Science, Uppsala University (Sweden), 1996. ISSN 0283-0574.

[Bou02] P. Bouyer. A logical characterization of data languages. *Information Processing Letters*, 84(2):75–85, 2002.

[BPT03] P. Bouyer, A. Petit, and D. Thérien. An algebraic approach to data and timed languages. *Information and Computation*, 182(2):137–162, 2003.

[HSLL97] K. Havelund, A. Skou, K. G. Larsen, and K. Lund. Formal modelling and analysis of an audio/video protocol: An industrial case study using UPPAAL. In *Proc. 18th IEEE Real-Time Systems Symposium (RTSS'97)*, pages 2–13. IEEE Computer Society Press, 1997.

[LPY97] K.G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *Journal of Software Tools for Technology Transfer*, 1(1–2):134–152, 1997.

[SBB+01] Ph. Schnoebelen, B. Bérard, M. Bidoit, F. Laroussinie, and A. Petit. *Model-checking techniques and tools*. Springer, 2001.

[TY98] S. Tripakis and S. Yovine. Verification of the fast reservation protocol with delayed transmission using the tool KRONOS. In *Proc. 4th IEEE Real-Time Technology and Applications Symposium (RTAS'98)*. IEEE Computer Society Press, 1998.