# Why we should use Min Max DEVS for modeling and simulation of digital circuits

M Hamri, A Naamane, C Frydman, N Driouche

Journal Title
XX(X):1–12

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

# Why Using Min-Max DEVS for Modeling and Simulation Digital Circuits

**M. Hamri[1] A. Naamane[1] C. Frydman[1] N. Driouche[2]**

**Abstract**

The delay is a very important element in modeling hardware behavior, that realised in hardware description languages as ADLIB-SABLE, Verilog and VHDL, in the literature delay may be organized into four classes. The first class, mean values are used such a precise delay element in the simulation; we found it in VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language ) where a single value is utilised to characterize transport delays, DEVS proposes to use a precise delay element. Under the second class, the delay is represented by an interval min-max, causes that the delay value is precisely unknown and every value in the interval can represent such a possible value for the actual delay. The third class, a delay is expressed in the form of a stochastic distribution. The fuzzy models of delay constitute the last class.

In reality, it is very difficult, if not impossible to obtain a precise value of the delay; there are many reasons for that: temperature, voltage, variation in the manufacturing process and other environment parameters.

The Min-Max DEVS formalism allows an efficient design of the min-max delay by proposing a life time function definition based on time interval. Moreover, its simulation semantics allows the simulation of Min Max DEVS models with only one replication, allowing us to conclude on the min-max delay if it is too large or no to obtain exact simulations.

In this paper, we propose to highlight the Min Max DEVS through examples from digital circuits, after having recalled its basic definitions and simulation semantics. Then, we compare the obtained simulation results with those provided of the well known tool in the field of digital circuits Verilog, using the same examples.

**Keywords**

Logic Gates, DEVS, Modeling and Simulation, Verilog.

## Introduction

Digital circuits are electronic components that have been introduced massively in our daily life, there are some decades, i.e., from the invention of transistors. These components participate in the design of a lot of objects (calculator, vehicle, peace maker, cellular phone, etc.). The scientists have been investigate the field of digital circuits by proposing language to design these digital circuits, to analyze and synthesize them, to detect faults, to simulate them, etc.

Norbert Giambiasi was one of these active scientists that contribute to the field of digital networks and a pioneer (with his colleagues of Languedoc university) by proposing the concept of logic-temporal design and analysis of such networks. His works were recognized by the community as shown by his publications at the last of 70ś Giambiasi et al. (1979) and Giambiasi et al. (1980). Moreover, he has developed in collaboration with D. Muriach and A. Miara a simulation tool for design and verify digital networks called SILOG Muriach et al. (1979).

In the beginning of 2000, Norbert contributed to the well-known formalism called DEVS (Discrete EVent system Specification) developed by Zeigler (1976). First, he proposed the formalism GDEVS (Generalized DEVS) Giambiasi et al. (2000) in order to enhance to state trajectory function of dynamic systems by representing it with a polynomial function instead of a piece-wise constant function. The second contribution is the Min-Max DEVS Giambiasi and Gosh (2002) and Hamri et al. (2006) in which the time life function is defined with a temporal window (time interval) to represent efficiently the knowledge of designers on time, on one hand. On the other hand, the simulation semantics on which this formalism is based and that introduces the unknown state, distinguishes it from the existing discrete event formalisms. These specific characteristics make these two formalisms so rewarding for the literature of DEVS and opening serious issues to investigate.

Therefore, I discussed (Amine) with Norbert to start a research around the modeling and simulation of digital circuits using GDEVS. The proposition was accepted, and we carried out significant works that were published in Hamri et al. (2014), Hamri et al. (2015) and Driouche et al. (2016).

However, it remains to explore the Min Max DEVS formalism using real applications and that we did not explore enough in the past. We chose the digital circuits due to its

[1]Aix Marseille Université, CNRS, ENSAM, Université de Toulon, LSIS UMR 7296 13397, Marseille cedex 20, France
[2]Independant software researcher, Marseille, France

**Corresponding author:**
M. Hamri, Aix Marseille Université, CNRS, ENSAM, Université de Toulon, LSIS UMR 7296 13397, Marseille cedex 20, France.
Email: amine.hamri@lsis.org, amine.hamri@univ-amu.fr

ambiguous delay element that fit perfectly to be modeled and simulated using this formalism. The designers of such circuits express often the delay with a time interval. These points motivate the work as a tribute to our colleague Norbert and highlight his formalism the Min Max DEVS.

The paper is organized as follows: Section 1 recalls the Min Max DEVS, its simulation semantics and the formal property closure under coupling. Section 2 recalls in its turn definitions on logic gates and briefly the different kinds of delay used in digital circuits. In Section 3, we propose a set of models designing basic components in digital circuits using the Min Max DEVS formalism and they may be coupled. Note that we focus only on the inertial delay for a seek of simplicity. Then, in Section 4, we carry out a series of simulations on simple gates and we compare the results with those provided by Verilog ©(a leader tool to design and simulate digital circuits). As a result of this comparison, the advantages of the Min Max DEVS formalism are highlighted and new issues are expected. Finally, Section 5 concludes on this work by recalling the power of this formalism and the future works that we expect.

## Recall on Min Max DEVS Formalism

The Min Max DEVS formalism consists of two algebraic structures: one for modeling the indivisible behavior which allows defining outputs according to some inputs and taking into account the current state of the model and the second one for modeling the inner structure of the whole model by reusing some submodels from those modelled and saved in user libraries, by making a coupling over them.

A Min Max DEVS atomic is an augmented DEVS with the unique difference for the modeller is that the time life function for active state is specified with temporal windows. Thus a Min Max DEVS atomic model is:

$$M =< X, Y, S, \delta_{int}, \delta_{ext}, \lambda, D >$$

The time interval specified for a given active state allows us to the model to send out an output inside the equivalent temporal window, then to transit to a new state. Note that the elapsed time after such a transition is reset to 0 .

However a Min Max DEVS coupled model is identical to a DEVS coupled one, that we recall its structure:

$$MC =< X_{MC}, Y_{MC}, D_{MC}, M_{d|d \in D}, EIC, EOC, IC, Select >$$

In Giambiasi et al. (2008), we showed that the Min Max DEVS formalism is closed under coupling, that means each for Min-Max DEVS coupled exists its equivalent Min Max DEVS atomic. Thus, such a coupled model is valid and eligible for simulation.

### *Internal Semantics*

By looking to the user specification of a Min-Max DEVS that we call an external model, there is an infinity of possible DEVS models. Because of the temporal window that specifies the duration of an active state, there is an infinity of real possible values. That means an infinity number of replications to make a decision. This is impossible and only partial significant conclusions based on simulation results can be made by the user.

However, the Min Max DEVS formalism proposes an original semantics to make simulations that consists of the following assumptions :

1. All possible systems may be enclosed by two significant models : the first one is the faster model which represents the faster system in which for each active state $s$ its duration is defined by the minimum time of the equivalent state $s'$ in the user specification ($min(ta(s'))$); and the second one is the slower model which represents the slower system in which for each active its duration is defined by the maximum time of the equivalent state in the user specification ($min(ta(s'))$).
2. Each occurred event makes a single transition in the slower or faster model. If an event occurs for the faster model, then the next event should be oriented to the slower model. Such a rule guarantees that the gap between the slower and faster models is at most one transition and let the slower model some laps of time reaching the faster one. Otherwise, the modeled system may have more than two possible states for a given date.

From these two principal assumptions, rules are described in order to define how a Min-Max DEVS behavior evolves through time.

Let us consider a given user specification for a Min-Max DEVS :

$$EM =< X, Y, S, \delta_{int}, \delta_{ext}, \lambda, D >$$

We construct the internal model $IM$ that defines how a simulation should simulate a user specification $EM$, as follows:

$$EM =< XI, YI, SI, \delta I_{int}, \delta I_{ext}, \lambda I, DI >$$

The sets $XI, YI$ and $SI$ are defined from the sets $X, Y$ and $S$ respectively specified in the user specification $EM$ :

- $XI = X \times \{fast, slow\}$

– The event $(x, fast) \in XI$ is interpreted as an external event with the value $x \in X$ destinated to the faster model. This event represents the beginning of the temporal window in which the system may receive the event $x \in X$ that may occur at each time depending on the real delay. – The state $(s_i, s_j, a) \in SI$ such as $s_i, s_j \in S$ and $a \in A$ represents the current state of the internal model $IM$. $s_i$ and $s_j$ represent the state of the slower and faster model respectively. $a$ returns an information on the next transition to do by the internal model $IM$. If $s_i = s_j$ means that all possible real systems are in the same state $s_i$; otherwise some fast real systems are in state $s_j$ and the slow real systems remain in state $s_i$ waiting to reach the state $s_j$. The state $(\phi, \phi, passive)$ is a particular state of the internal model $IM$, it indicates that there are more than two possible states for the existing real systems and that the internal model $IM$ cannot keep.

The benefit of the internal model $IM$ is its ability to return information about the possible current state of the real systems ; whereas the external model $EM$ for some temporal windows transients to the unknown state, due to the interval defined by the min-max life time function $D()$ and that the modeler can not define with exact values.

Moreover these extended sets, the construction of the different functions $\delta I_{int}, \delta I_{ext}(), \lambda I()$ and $DI()$ is as follows :

- The Internal transition function $\delta I_{int}()$

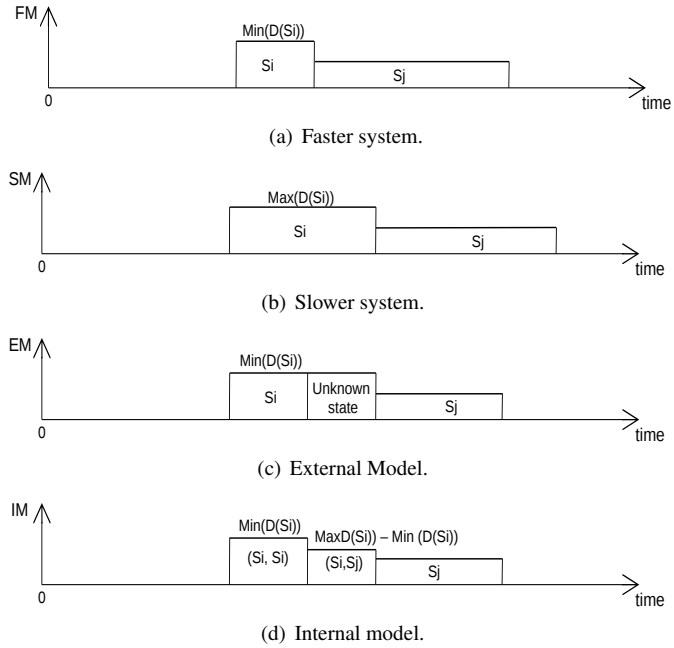
$\delta I_{int()} : SI = S \cup \{\phi\} \times S \cup \{\phi\} \times A \rightarrow S \cup \{\phi\} \times S \cup \{\phi\} \times A$

With the following rules:

$\delta I_{int}(s_i, s_i, autonomous) = (s_i, \delta_{int}(s_i), autonomous)$ with $DI(s_i, s_i, autonomous) = Min(D(s_i))$ The second element of the triplet $(s_i, s_i, autonomous)$ represents the current state of the faster model which is a candidate for an internal state change. For that, this internal transition is applied to it. However, the external model is in state $s_i$ and its duration is $Min(D(s_i))$. After this transition, the internal model $IM$ transients to $(s_i, s_j, autonomous)$ and waits that an internal state change occurs for the slower model. However, the external model $EM$ transients to the unknown state, because it is not sure that the real system makes or not a transition at this moment. The duration of this unknown state is $Max(D(s_i)) - Min(D(s_i))$, after this time the two models (slower and faster) had made the corresponding state change to $s_j$ with the condition that $Max(D(s_i)) - Min(D(s_i)) \leq Min(D(s_j))$. Otherwise, the internal model transients, in its turn, to the totally unknown state $(\phi, \phi, passive)$. Thus, the next possible internal transition is :

$$\delta I_{int}(s_i, s_j, autonomous) \begin{cases} (s_j, sj, passive) \text{ if } s_j \text{ is passive} \\ (s_j, s_j, autonomous) \\ \qquad \text{if } Max(D(s_i)) - Min(D(s_i)) \leq Min(D(s_j)) \\ (\phi, \phi, passive) \text{ otherwise} \end{cases}$$

In order to understand these rules, we have to consider that we model simultaneously the state change of all possible real systems. We consider the faster and slower models that represent respectively the faster and slower systems. The state $(s_i, s_j, autonomous)$ represents that some systems are in state $s_j$ until all slow systems reach this state (see Figure 1).




(a) Faster system.

(b) Slower system.

(c) External Model.

(d) Internal model.

**Figure 1.** Internal state change occurring in each model: FM, SM, EM and IM.

The Figure 1 shows how internal and external transitions are conducted for the states $s_i$ and $s_j$. The external model $EM$ transients to $s_i$ at time $t = tl + Min(D(s_i))$, and we have : $\delta I_{int}(s_i, s_i, autnomous) = (s_i, \delta_{int}(s_i), autonomous) = (s_i, s_j, autonomous)$.
at time $t = tl + Max(D(s_i)) - Min(D(s_i))$, we have :
$\delta I_{int}(s_i, s_j, autonomous) = (s_j, s_j, autonomous)$
because $DI(s_i, s_j, autonomous) = Max(D(s_i)) - Min(D(s_i))$ and $s_j$ is active.

- The output function $\lambda I()$

As in classical DEVS, the output event is emitted before firing an internal state change, so we have:
$\lambda I(s_i, s_i, A) = (\lambda(s_i), fast)$
$\lambda I(s_i, s_j, A) = (\lambda(s_i), slow)$

The interpretation of these output events for the internal model $IM$:

– The event $(\lambda(s_i), fast)$ occurs when the faster model transients to state $s_j$. At the same time, it sends out the event $\lambda(s_i)$.
– The event $(\lambda(s_i), slow)$ occurs when no external event occurs in the internal model $IM$ and the slower model transients to the state $s_j$. At the same time, it sends out the output event $(\lambda(s_i))$.
For the external model $EM$, the interpretation is different. The fast event, emitted by the faster model, starts a temporal window in which any real system may send out an output event. The slow event, emitted by the slower model, ends the temporal window. It informs that all existing systems have sent out their output event $\lambda(s_i)$.

- The external transition function $\delta I_{ext}()$

In order to define the external transition function $\delta I_{ext}()$ related to the internal model $IM$, we have to analyse

the different possible cases in which an external event $x$ may occur. Note that the internal model $IM$ represents all possible systems, so the fast external event $(x, fast)$ should be applied to the faster model in state $s_j$ and the slow external event $(x, slow)$ should be applied to the slower model, in the state $s_j$. Let us consider the different cases :

– First case: for an input $x$, the present state of the external model $EM$ is the state $s_i$ which is passive and the next state $s_j = \delta_{ext}(s_i, e, x)$ is passive too. Thus, the current state of the internal model is $(s_i, s_i, passive)$. At the occurrence of the fast event $(x, fast)$, we apply it to the faster model, as follows :

$$\delta I_{ext}((s_i, s_i, passive), e, (x, fast)) = (s_i, \delta_{ext}(s_i, e, x), fast) = (s_i, s_j, passive)$$

where $e$ is the elapsed time from the last transition occurred in the internal model $IM$.

$$DI(s_i, s_i, passive) = \infty \text{ and } DI(s_i, s_i, fast) = \infty.$$

After this transition, the external model transients to the unknown state, because some slow systems do not transient yet whereas the fast ones have transited. However, the internal model $IM$ transients to the state $(s_i, s_j, fast)$ which will wait for the next event $(x, slow)$ for the slower model. Once, this event occurs the internal model $IM$ applies the external transition $\delta I_{ext}()$ as follows:

$$\delta I_{ext}((s_i, s_j, fast), e, (x, slow)) = (s_j, s_j, passive)$$

Consequently, all systems have transited to the state $s_j$.
The temporal window that is defined by the occurrence of the fast and slow events, allows the occurrence of the external event $x$ at any time to be received by the external model $EM$ due to the imprecision on the lifetime function $D()$.

– Second case : $s_i$ is a passive state and $s_j$ is an active one. Firstly, we consider the occurrence of the fast external event $(x, fast)$ at time $t_i$ followed by the slow external event $(x, slow)$ at time $t_j$ such as $t_j - t_i \leq Min(D(s_j))$. Because the slow event $(x, slow)$ occurs before the internal state change of the faster model, we have:

$$\delta I_{ext}((s_i, s_j, autonomous), e, (x, slow)) = (s_j, s_j, autonomous) \text{ where}$$
$$DI(s_j, s_j, autonomous) = DI(s_j, s_j, autonomous) - e.$$

Now we consider the fact that $t_j - t_i \geq Min(D(s_j))$. From this analysis, for a present state $s_i$ of the external model $EM$ and a next active state $s_j = \delta I_{ext}(s_i, e, x)$, we define the following rule for the external transition function $\delta I_{ext}()$ of the internal model $IM$. So, we apply the fast event $(x, fast)$ to the faster model as follows :

$$\delta I_{ext}((s_i, s_i, passive), e, (x, fast)) = (s_i, \delta_{ext}(s_i, e, x), fast) = (s_i, s_j, fast)$$
$$DI(s_i, s_i, fast) = Min(D(s_i)) \text{ if } s_j \text{ is active}$$
$$\infty \text{ if } s_j \text{ is passive}$$

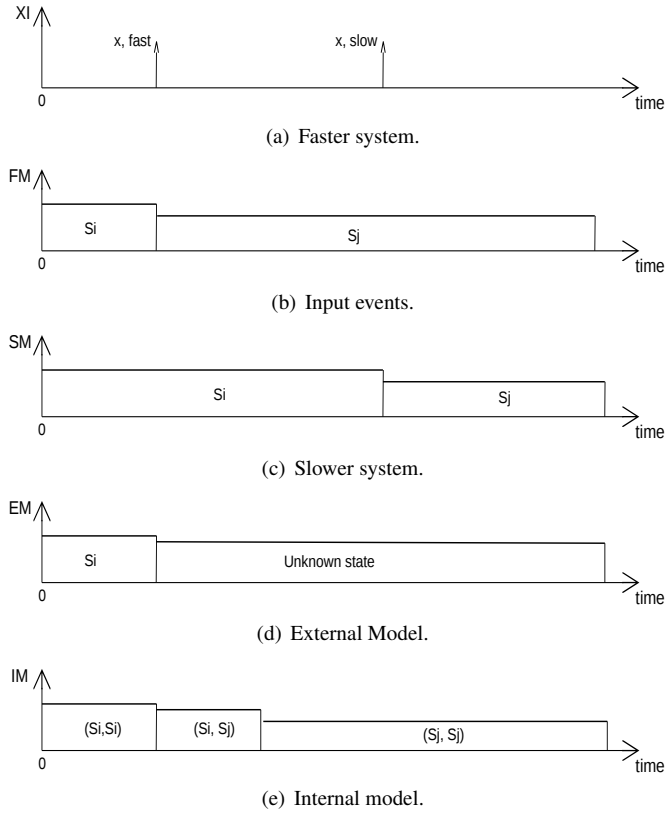



**Figure 2.** External state change from a passive state to a passive one, occurring in each model: FM, SM, EM and IM.

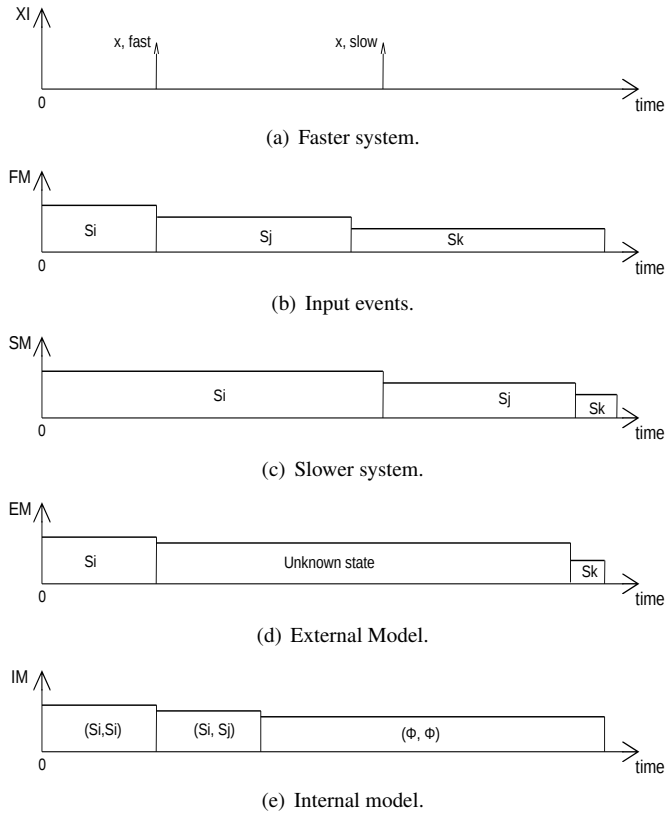
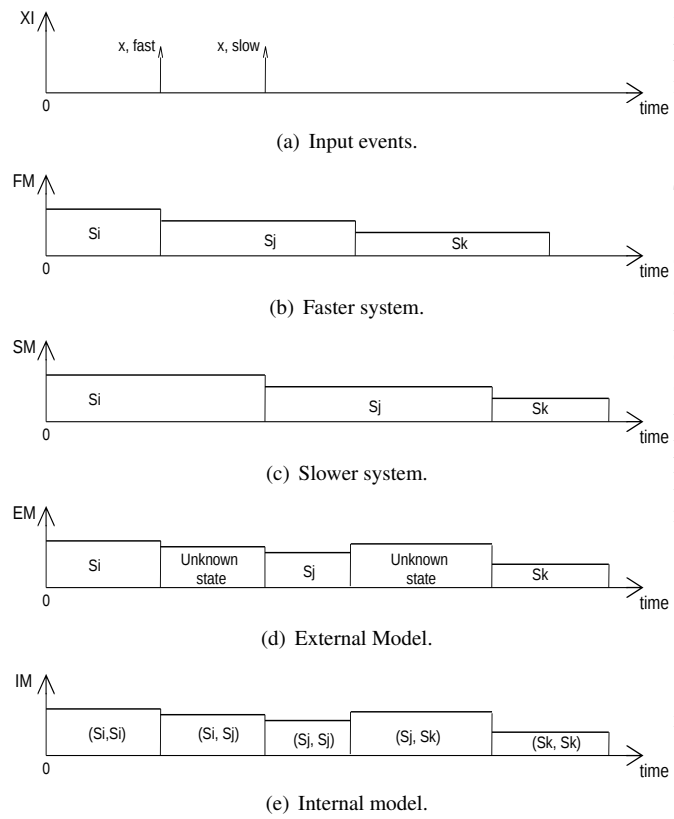



**Figure 3.** External state change from a passive state to an active one with an early internal state change occurring in each model: FM, SM, EM and IM.




(a) Input events.

(b) Faster system.

(c) Slower system.

(d) External Model.

(e) Internal model.

**Figure 4.** External state change from a passive state to an active one with a late internal state change, occurring in each model: FM, SM, EM and IM.

– Third case: the internal model $IM$ is in state $(s_i, s_j, autonomous)$. Then, the fast event $(x, fast)$ occurs before the internal transition of the slower system, and late, the slower event $(x, slow)$ arrives. Unfortunately, while the fast event $(x, fast)$ occurs, we have two successive transitions for the faster system whereas no transition occurred yet for the slower system. For that we choose the following rules:

$\delta I_{ext}((s_i, s_j, autonomous), e, (x, fast)) = (\phi, \phi, passive)$
$\delta I_{ext}((\phi, \phi, passive), e, (x, \{fast, slow\}) = (\phi, \phi, passive)$

Consequently, the internal model $IM$ transients from a known state to the totally unknown state $(\phi, \phi, passive)$.

– Fourth case: the internal model $IM$ receives two successive fast events. For a given state $(s_i, s_j, fast)$, while the second fast event $(x', fast)$ occurs, the internal model $IM$ transits to the totally unknown state because the slower model does not receive yet the first slower event $(x, slow)$. Thus, we obtain:
$\delta I_{ext}((s_i, s_j, fast), e, (x', fast)) = (\phi, \phi, passive)$

These rules that define the simulation semantics of a Min Max DEVS behavior in which the min-max life time function is specified for active states are deduced by enumerating different cases for which the possible fast possible systems pre-empt at more than one transition either internal or external the possible slow systems.

In the following, we will attempt to apply this semantics for the simulation of electronic circuits and compare it with a classical simulation based on fixed delays, after having recalled the closure under coupling property of this formalism.

## *Closure under coupling*

To allow hierarchical modeling of Min-Max DEVS specifications, we proved the formal property closure under coupling of the Min Max DEVS formalism Giambiasi et al. (2008). This proof shows that considering a Min-Max DEVS coupled model a Min Max DEVS atomic one equivalent exists. Therefore at the conceptual level, we can componentize a Min-Max DEVS coupled model and reuse it to define a new Min-Max DEVS one. Let us consider a Min-Max DEVS coupled model $MC$ that consists of Min-Max DEVS atomic ones. The equivalent Min-Max DEVS atomic model is defined as follows:

$$MC = (X, Y, S, \delta_{ext}, \delta_{int}, \lambda, D) \text{ where}$$

$X$ is the set of input events received by $MC$
$Y$ is the set of output events that are sent out by $MC$
In fact the interface of the coupled model MC defines the interface of the resultant model.

$S = \prod_{d=0}^{d=N-1} Q = ((s_0, e_0), \ldots, (s_d, e_d), \ldots) | d \in D$
$D : S \rightarrow \infty^+ \times \infty^+ D(S) =$
$[min(min(D(s_0), \ldots, D(s_d), \ldots, ),$
$min(max(D(s_0), \ldots, D(s_d), \ldots, ))]$

This means that an internal transition can be fired in an atomic model at early at the minimum of the minimal durations and not late than the minimum of the maximal durations of current states of atomic models. Thus, an internal transition can be activated in a temporal window defined with a min-max value. When the minimal duration becomes equal to zero, this means that the transition may be fired at now. Therefore the remaining time $\rho$ to fire a transition from the last transition is defined as follows:

$\rho = [min(D(s)) - e, max(D(s)) - e]$ if $min(D(s)) - e >= 0$
$\rho = [0, max(D(s)) - e]$ otherwise.
$e$ is the elapsed time in the state $s$. This means that the next expected transition will be fired in the temporal window $[\rho_{min}, \rho_{max}]$ since the last transition occurrence.

Let us consider the set of imminent atomic models from a current state $s$,

$$IMM(s) = d \in D | Min(\rho_d) = Min(D(s))$$

The set of imminent models is the set of atomic models that have the minimum of the set of temporal window beginning (to reduce the complexity we exclude external transitions). The corresponding models have an internal transition to fire. These transitions are candidates for the next expect internal transition of the equivalent model. To identify which one must be fired first, we call the function select to define the first atomic model to execute its output then its internal transition functions, we call it $d^* = select(IMM(s))$. Therefore we define the internal transition function of the resultant model as follows:

$\delta_{int} : S \rightarrow S$
$\delta_{int}(s) = s' = (\ldots, (s'_d, e'_d), \ldots)$
$(s'_d, e'_d) = (\delta_{int}((s_d), 0)$ if $d = d^*$

$(s'_d, e'_d) = (\delta_{ext}(s_d, e_d + e_{d^*}), Z_{d^*,d}(\lambda(s_{d^*}), 0))$ if $d^* \in I_d$ and $\lambda(s^*_d) \neq \phi$

The output function of the equivalent model $\lambda I()$ has the role to send out an external output event to make state changes in components outside $MC$. So there is an external output coupling between a subcomponent and $MC$. In other cases, a non-event $\phi$ may be sent out when a subcomponent belongs to an influence set of another subcomponent. This is to say when there is an internal coupling between two subcomponents. This event may be ignored due to the fact that there is no change to make. Consequently, the performances of the DEVS simulator are increased. Based on these definitions, we define the output function $\lambda$ as follows:

$\lambda(s) = Z_{d^*,MC}(\lambda(MC))$ if $d^* \in I_{MC}$

$\lambda(s) = \phi$

Now consider the external transition function $\delta I_{ext}()$ for the equivalent Min-Max DEVS. When the interface of $MC$ receives an external event $x$, it sends this event to the influenced models through the input-to-input function $Z_{i,j}$ to activate $\delta_{ext}$ then make the concerned state change. The other models are still in their current state and only their elapsed time is updated. So we define $\delta I_{ext}$ of the equivalent model as follows:

$\delta I_{ext} : Q \times X \rightarrow S$

$\delta I_{ext}((s, e), x) = s' = (\ldots, (s'_d, e'_d), \ldots)$

$(s'_d, e'_d) = (\delta_{ext}((s_d, e), x), 0) if MC \in I_d$

$(s_d, e_d) = (s_d, e_d + e)$ otherwise.

Based on these definitions, we conclude that the obtained Min-Max DEVS model is atomic. Consequently, the formalism allows hierarchical modeling and simulation.

## State of the art on logic circuits

Modeling an electronic circuit consists of reusing small chips and connect them to design the wished network. A small chip is a physical device implementing a boolean function that performs a logical operation on one or more logical inputs and produces a single logical output Hamri et al. (2014) Hamri et al. (2015) Driouche et al. (2016). In fact, logical circuits are the basis for the modern digital computer systems and digital electronic circuits. They are usually made from large assemblies of logic gates (chips) which are simple boolean functions.

A logic gate produces an output according to the received inputs with a certain delay. This delay justifies the use of a simulation technique to compute the output (otherwise the use of the boolean algebra is sufficient). Thus, we can design a logic gate by three blocks :

1. one block to define the boolean function,
2. a second block to define the delay, and
3. a third block to define the amplifier function.

The boolean function is defined by the well-known boolean algebra. This algebra employs boolean variables defined on $\{0, 1\}$ with a set of operators: $and, or$ and $not$. The operators $and$ and $or$ are binary and may be generalized to more than two variables. However, the operator $not$ is unary and can not be generalized.

$$and = min(a, b) \quad (1)$$

$$or = max(a, b) \quad (2)$$

$$not(a) = !a \quad (3)$$

In addition, from practical view, the unknown value $x$ is introduced to model the fact that the modeler does not have information about the state of a given logic gate. This situation happens often at the initialization of the simulation for which the modeler has not enough information about the initial state of a given logic gate and that he can not initialize. Like a flip-flop gate where the output is used to compute the next state and some simulation cycles are conducted to make this state known.The output of each operator is given on Table 1 by considering the state $x$.

**Table 1.** Truth tables of and, or and not with unknown input.

| a | b | and |
|---|---|---|
| $x$ | 1 | $x$ |
| 1 | $x$ | $x$ |
| $x$ | 0 | 0 |
| 0 | $x$ | 0 |

| a | b | or |
|---|---|---|
| $x$ | 1 | $x$ |
| 1 | $x$ | $x$ |
| $x$ | 0 | 0 |
| 0 | $x$ | 0 |

| a | not |
|---|---|
| $x$ | $\overline{x}$ |

Note that this unknown state $x$ and its use are different from the totally unknown state $(\phi, \phi, passive)$ that we use to define the Min-Max DEVS simulation semantics.

The delay functions, according to the literature may be grouped into four categories. The first category regroups models that use mean values to define a delay function (these mean values are considered as a fixed value). Under this category, we find the pure transport delay which has only one value, the rise-fall delay which has two values, one to define the needed time for the input transits from 0 to 1 and a second time for the input transits from 1 to 0, and the inertial delay with one or two values which has a pre-emption semantics (we use this kind of delay in the given examples). The second, third and fourth categories, due to the lack of knowledge or a lot of candidate values to define the delay function, they represent it by a set values. We fin the min-max delay function that defines all possible values for a given delay by an interval. Each value from this interval is a possible delay. Then, the probabilistic delay function where delay is represented by a stochastic variable with a distribution of probability. Finally, the fuzzy delay function that involves a fuzzification and defuzzification phases of the delay defined by a distribution of possibility.

Concerning the min-max delay which is related to our work, some tools leader in modeling and simulation of logic gate propose the use of the typical min and max delays to define such a function. However, its semantics is completely different from that defined by the Min Max DEVS. The modeler of such a delay, using Verilog for example, should indicate, at the beginning of the simulation, which value should be used for computing outputs of the designed logic gate.

The amplifier function is used to model a finite gain, supply rail clipping and slew rate limiting. The output signal computed according to input and delay functions is multiplied by a gain specified by the modeler. A detailed and mathematical description of this function is given in Hamri et al. (2014).

However, in this work, we ignore this function due to the fact we model inputs with only two discrete values 0 and 1. In this case, there is no need to amplify inputs. We

decide to model only the boolean and inertial delay functions by the Min-Max DEVS formalism in order to obtain the corresponding logic gates.

## Min-Max Modeling of logic circuits

### Basic logic gate

For a seek of simplicity, we ignore the amplifier function and we retain only the boolean and delay functions. Consequently, we model these two functions by the Min-Max DEVS atomic functions and for which the corresponding box may have one or two inports and one outport. Two state variables are used to store inputs from inports (or one state variable in case of the logic gate $not$) using the function $\delta_{ext}()$. While the logic gate emits its output through the function $\lambda()$, the function $\delta_{int}()$ is called to in order to put the logic gate in phase active, then wait for a new input. The last function $D()$ that will define the min-max delay when the logic gate is in an active phase.

The Min-Max DEVS atomic model of each boolean function *and*, *or* and *not*related to a min-max delay are given below:

$phase \in \{passive, active\}$ $a, b, v \in \{0, 1, x\}$

$\delta_{ext}(phase \times a \times b, e, in?v) : \begin{cases} a = v \text{ if } in = in_1 \\ b = v \text{ otherwise} \end{cases}$

$\delta_{int}(phase \times a \times b) : phase = passive$

$\lambda(phase \times a \times b) : out!min(a, b)$

$ta() : \begin{cases} [min, max] \text{ if } phase = active \\ \infty \text{ otherwise} \end{cases}$

$\delta_{ext}(phase \times a \times b, e, in?v) : \begin{cases} a = v \text{ if } in = in_1 \\ b = v \text{ otherwise} \end{cases}$

$\delta_{int}(phase \times a \times b) : phase = passive$

$\lambda(phase \times a \times b) : out!max(a, b)$

$ta() : \begin{cases} [min, max] \text{ if } phase = active \\ \infty \text{ otherwise} \end{cases}$

$\delta_{ext}(phase \times a, e, in?v) : a = v$

$\delta_{int}(phase \times a) : phase = passive$

$\lambda(phase \times a) : out!not(a)$

$ta() : \begin{cases} [min, max] \text{ if } phase = active \\ \infty \text{ otherwise} \end{cases}$

These logic gates, at this stage, can not be simulated directly. However, we should first, construct their internal models *IMs* according to the rules detailed previously. Then simulate them by any DEVS simulator. Without these internal models, it is impossible to make a useful simulation and conclude correctly on these logic gates that represent external models *EMs*.

Note that this formalism as DEVS facilitates the modeling of inertial delay thanks to its event filter mechanisms based on the transition and time life functions.

### Composite logic gate

Any composite logic gate is an assembly of basic and existing composite logic gates. They may be designed by the Min-Max DEVS coupled formalism. The key element is the function $select()$ that should be defined to handle simultaneous inputs. Otherwise, a logic gate may receive several inputs at time, which is absurd. In fact, a logic gate is a sequential model for which the simultaneous events should be ordered and sent correctly to the target logic gate without any conflict. At this subject, Norbert said always: simultaneous events can not occur for a physical system and we should take care on concurrent events and not simultaneous events; even the physician Albert Einstein did not define simultaneity.

Let us consider the composite gate formed by four gates in series, shown in Figure 5. It may have two or more subgates are candidates for an internal state change, the subgate that is more at right fires, first, its internal state change.

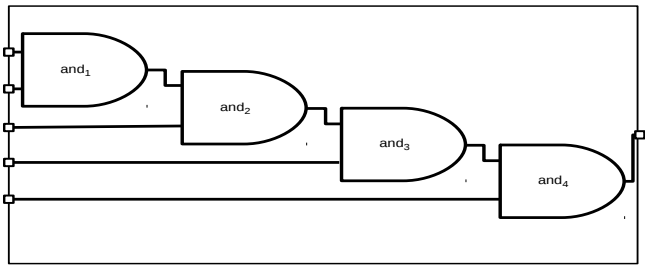



**Figure 5.** The composite gate $and^4$ in series.

Thus, the function $select()$ for this is composite gate is defined as follows:

$select() :< (and_4, 1), (and_3, 2), (and_2, 4), (and_1, 4) >$

Through this function, we associate for each subgate a weight. However, while subgates are in parallel (see Figure 6), their weights are identical and the function $select()$ returns one randomly.

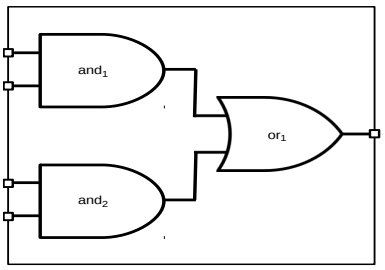



**Figure 6.** A composite gate with two subgates in parallel.

The function $select()$ is defined as follows:

$select() :< (or_1, 1), (and_1, 2), (and_2, 2) >$.

The subgates $and_1$ and $and_2$ have the same weight. When these two subgates are candidates for an internal state change, the function $select()$ identifies them and selects one randomly.

Note that this technique of affecting weights to subgates and choose the gate that has the priority can not return the right subgate to handle in case of sequential logic gates like flip-flop gate. However, we should identify all possible conflicting subgate scenarios and define for each scenario the imminent subgate.

Let us consider the RS-latch a composite gate (Figure 7).

There are 10 conflicting scenarios (computed by identifying all conflicting cases), for which the modeler should define the imminent subgate to handle first.

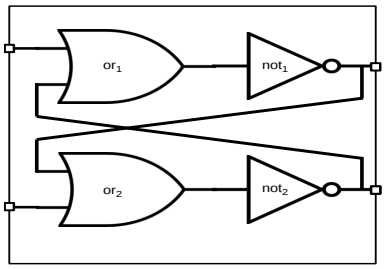



**Figure 7.** *RS-latch.*

$$select() \begin{cases} ... \\ < or_1, not_2 > \rightarrow or_1 \\ ... \\ < or_1, or_2, not_1 > \rightarrow not_1 \\ ... \end{cases}$$

For example, if the subgates $or_1$, $or_2$ and $not_1$ are candidate to throughput, the function $select()$ will choose the subgate $or_2$. A second example, when the subgates $or_1$ and $not_2$ are candidate, the subgate $or_1$ will be handled first. Whereas the technique of weighting subgates, will return the subgate $not_2$ which is incorrect.

## Designing Min Max DEVS simulation

In order to conduct Min Max DEVS simulations of logic circuits, we should first develop and implement the Min Max DEVS simulator. As we said previously, the Min Max DEVS simulation semantics may be simulated by any DEVS simulator. In this work, we choose the fwkDEVS simulator plug-in Hamri (2017) used successfully in many DEVS and GDEVS simulations Hamri et al. (2014) Hamri et al. (2015).

fwkDEVS is a simulator designed in object-oriented paradigm and implemented in Java 1.7. It distinguishes the simulation kernel from the modeling requirements. For designing DEVS atomic models, the user should extend the class `DEVSAtomic`; and for designing DEVS coupled models, he should extend the class `DEVSCoupled` and complete them with modeling requirements.

The definition of Min Max DEVS coupled is identical to that of DEVS. So, the class `DEVSCoupled` will be renamed only. The definition of Min Max DEVS atomic seems to that of DEVS atomic with the main difference, the lifetime of active states is expressed with time intervals, instead of a unique value. Moreover, the Min Max DEVS simulation semantics of the atomic models is completely different from DEVS simulation but consists in. For that, we supply the class `DEVSAtomic` with new responsibilities to be able to simulate correctly the Min Max DEVS models without modifying the simulation kernel.

In the class `DEVSAtomic`, we implement the Min Max DEVS simulation rules, i.e., the internal model $IM$ through the methods that are used by the modeler to design DEVS atomic models. Also, we define useful methods in order to design the Min Max DEVS external model $EM$. Note that this class has two references to store the state of the slow and fast systems on which transition functions will operate.

Once the Min Max DEVS atomic external model is designed, it may be simulated by the DEVS simulator or reused to design new Min Max DEVS coupled models.

The class diagram shown in Figure 8 summarizes the design of Min Max DEVS atomic and coupled models.

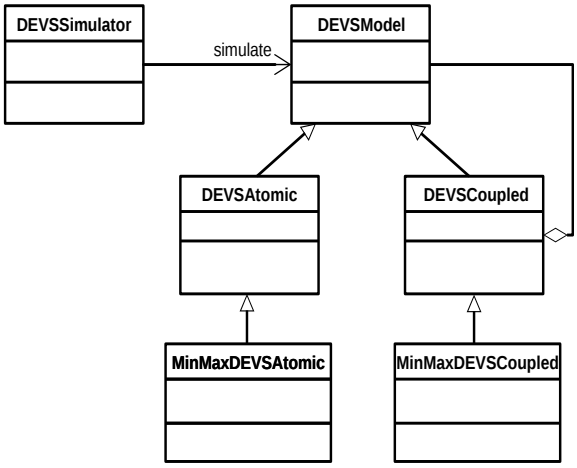



**Figure 8.** Class diagram showing design of Min Max DEVS models using a DEVS simulator.

Note that the designer of Min Max DEVS models should take care when he inserts the submodels to composite the whole model. These submodels should be typed as Min Max DEVS models and not as DEVS models. In order to avoid such a confusion, the Min Max DEVS coupled model contains only other Min Max DEVS models. For that, the designer may use an assertion to check whether the model is designed in Min Max DEVS or not.

Remains, the simulation process of Min Max DEVS models. The internal model $IM$ that is related to each atomic model will produce a behavior according to received events. The coupling specified by the coupled models will allow the simulator to dispatch events correctly to the concerned models. Note that events according to Min Max DEVS internal semantics possess two values, the event value responsible on firing transitions and the event type that identifies to which model the faster or slower the event is intended to. The design of such an event on the fwkDEVS simulator is easy to do, thanks to its object-oriented design of messages and that has been tested successfully to simulate GDEVS logic gates of first and second orders.

## Designing Min Max DEVS logic gates

The design of logic gates *and, or* and *not* using Min Max DEVS under fwkDEVS is easy to do. The designer extends the class `MinMaxDEVSAtomic` to design the corresponding gate, then he implements the different function $\delta_{ext}()$, $\delta_{int}()$, $\lambda()$ and $ta()$ through the methods `delat_extEM()`, `delat_intEM()`, `lambdaEM()` and `duration()` according to the external model specification of the concerned gate.

The method `init()` allows configuring the corresponding gate at initialization. It should be defined according to designer requirements: he affects concrete values to define the state of a given gate or he uses the undefined value $x$. Note that if the state of a given gate is unknown at initialization, it will be known after a certain number of simulation steps. Recall that the unknown state that we employ at initialization is due to a lack of information on the gate at this state which is completely different from the state $(\phi, \phi, passive)$ that occurs due to min-max delay.

Let us consider the external model of the gate *not*. Its specification seems to a DEVS specification with the slight difference, the method `getDuration()` is divided into two methods `getMinDuration()` and `getMaxDuration()`. Thus, the code of this gate is as follows:

```
@Override
public int deltaExtEM(int state, Object ev,
                      float e) {
// TODO Auto-generated method stub
  if((int)ev == 0) a = 1;
  else a = 0;
  active = true;
  return a;
}
@Override
public int deltaIntEM(int state) {
// TODO Auto-generated method stub
  active = false;
  return a;// return the current state
}
@Override
public float getMin(int state) {
// TODO Auto-generated method stub
  if(active == false){
      return Float.POSITIVE_INFINITY;
   }
  return delayMin;
}
@Override
public float getMax(int state) {
// TODO Auto-generated method stub
  if(active == false){
      return Float.POSITIVE_INFINITY;
  }
  return delayMax;
}
```

Note that the internal model of the gate *not* is not the responsibility of the designer. The class `Not` has its own methods inherited from the abstract class `MinMaxDEVSatomic` in order to make the internal model *IM* and to produce the correct behavior specified by the internal semantics.

The internal model is implemented through the DEVS methods in order to do not update and modify the designed links between the simulator and the model. For example, a piece of code of the method `delta_ext()` that designs the external function $\delta I_{ext}()$ of the internal model, is shown below:

```
public void deltaExt(Port p, Object ev,
                     float e){
   int value, type;
   value = ((int[])ev)[0];
   type = ((int[])ev)[1];

   if((a == indicator.passive)&&(type == 1)){
       ecr = deltaExtEM(ecr,value, e);
       ecl = ecl;
       a = indicator.fast;
       duration = getMin(ecr);
   }
...
}
```

Consequently, the internal model of the gate will be constructed on the fly according to the received events from the simulator and occurred transitions.

Once the basic gates *and*, *or* and *not* are completely designed, composite gates reusing Min Max DEVS gates may be designed like in DEVS. The code of the RS-latch gate declaring the subgates and the corresponding coupling over them are given below:

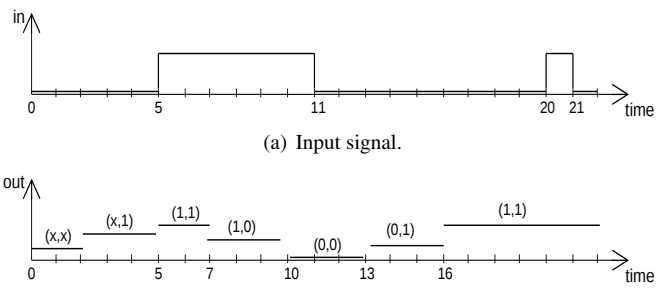
```
public class RS extends MinMaxDEVSCoupled {
   MinMaxOr or1, nor2;
   MinMaxNot not1, not2;
   Port in0, in1, out0, out1;
   public RS(){
     super();
     this.name = "rs";
     or1 = new MinMaxOr("or1");
     or2 = new MinMaxOr("or2");
     ...
     in0 = new Port(this, "in0");
     ...
     this.addInPort(in0);
     ...
}
```

Like in DEVS, the most critical element is the function *select()* that handles simultaneous events. It is the responsibility of the designer to implement correctly this method by weighting subgates or enumerating conflicting scenarios.

## Min Max DEVS vs. Verilog Simulation of Logic Gates

In order to execute Min Max DEVS simulations of the designed logic gates, we couple their input ports with generators models. These generators are configured by the user to send out events 0 or 1 at the hopeful time. Then, the simulator handles these events and produces the right outputs at right times, according to the behavior of simulated the logic gate. Recall that the current state of a Min Max DEVS model is described by a couple $(s_i, s_j)$ that shows the current state of slow and fast systems respectively. So a gate output represented with the couple $(x, y)$ is interpreted as follows: fast gates output $y$ and slow gates output $x$.

So, let us consider the simple gate *not* with min max delay designed in Min Max DEVS and a short simulation with two pulses. We define the min-max delay of this gate to $[2, 5]$ units of time(u.t). The simulation of this gate, as shown below, reproduces the first pulse by applying the *not* operator. However, the second pulse is cut, due to the fact that the second pulse is too short and the min-max delay too large. This is in respect to the inertial delay semantics.




(a) Input signal.

(b) State trajectory.

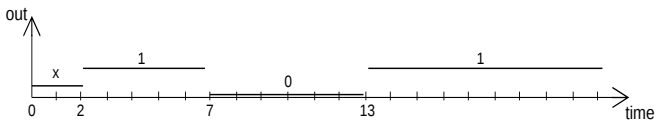
**Figure 9.** Min Max DEVS simulation of the gate *not*.

Note that the Min Max DEVS simulation produces events and their time occurrences. So, we are constraints to

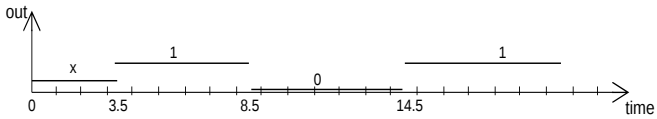
construct the trajectory of each observed variables in order to make a comparison with Verilog corresponding variables; or extract events from observed Verilog variables.

By reproducing such a simulation on Verilog © we should consider the min max delay with another delay semantics which is the min-typ-max delay to design a delay represented with a time interval. The min-typ-max delay of Verilog represents a delay interval with three different values: min, typical and max. Then, the designer runs each simulation separately by using the command `-iverilog -Tdelay` where `Tdelay` may be `Tmin`, `Ttyp` or `Tmax` to simulate one of the possible delay values: min, typical or max respectively. Consequently in the previous case, we have 2 u.t and 5 u.t to represent the min and max delays respectively; and we choose 3.5 u.t as a typical delay.

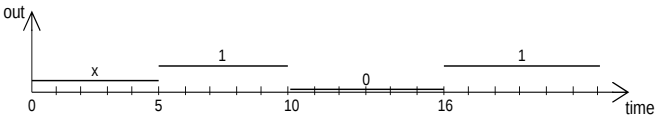
Thus, we obtain the following chronograms deduced from Verilog simulations of this example, shown on Figure 10.




(a) Simulation of the gate *not* with min delay.




(b) Simulation of the gate *not* with the typical delay.



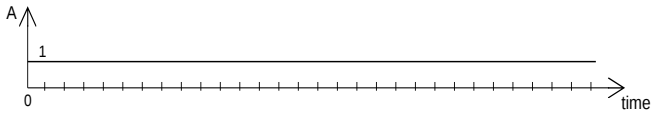

(c) Simulation of the gate *not* with the max delay.

**Figure 10.** Verilog simulation of the gate *not* with min-typs-max delay.

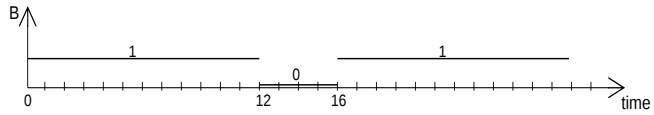
The chronogram shown on Figure 9 synthesizes and gives more information than the other chronograms traced from Verilog and that simulate the gate *not* with specific values of the delay. It shows that at the beginning when the state of the Min Max DEVS model is not yet setup, all possible gates are in the unknown state $x$, until the first output occurs for the faster gate (delay = min) at time $t = 2$ u.t. From this time, some gates have made a state change and had sent out the corresponding output; however other slow (late) gates are still in the unknown state $x$. At time $t = 5$ u.t, the slower gate with the max delay does its state changes to reach fast gates. So, all gates from this time, are transited from state $x$ to 1. These conclusions are confirmed by the Verilog simulations as shown on Figure 10.

Note that, for the moment, at initialization, the Min Max DEVS gates are set up only with the unknown state $x$ only, and we do not consider the other unknown state, the high impedance value $z$ that may occur like in Verilog which may create a slight divergence on the interpretation of simulation results. However, all these states are unknown ones which let us interpret them as identical states from this point view.

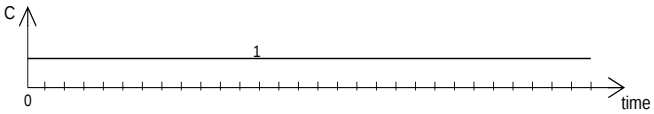
Now consider a simple network of two gates *and* in series. Then consider the three inputs $A$, $B$ and $C$ evolving as shown on Figures 11(a), 11(b) and 11(c) respectively.



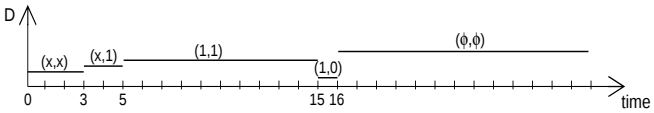

(a) Input $A$.



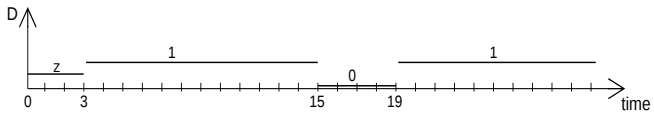

(b) Input $B$.



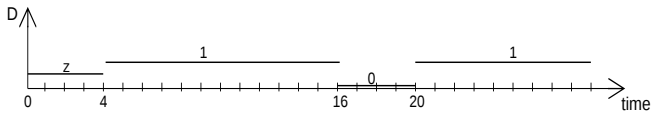

(c) Input $C$.



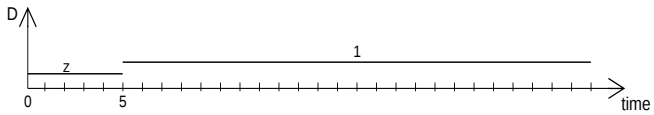

(d) Output $D$ with a Min Max DEVS simulation.



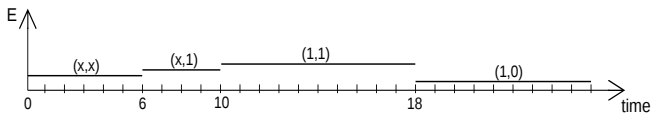

(e) Output $D$ with a Verilog simulation (delay = min).



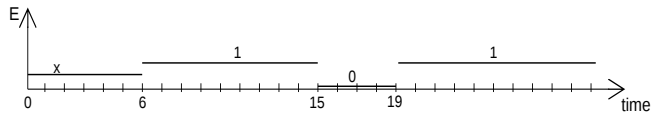

(f) Output $D$ with a Verilog simulation (delay = typical).



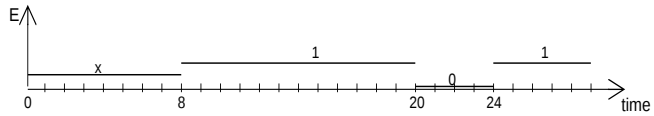

(g) Output $D$ with a Verilog simulation (delay = max).



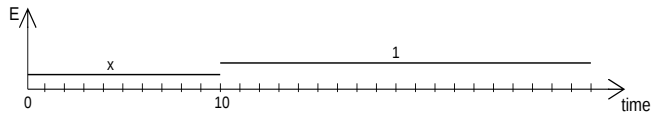

(h) Output $E$ with a Min Max DEVS simulation..




(i) Output $E$ with a Verilog simulation (delay = min).




(j) Output $E$ with a Verilog simulation (delay = typical).




(k) Output $E$ with a Verilog simulation (delay = max).

**Figure 11.** Simple network simulation : Min Max DEVS vs. Verilog.

The two outputs $D = A\ and\ B$ and $E = D\ and\ C$ are computed with two different simulations : Min Max DEVS and Verilog. The delay of each component is the time interval $[3, 5]$ u.t that we use such as in Min Max DEVS. However, in Verilog simulation, we represent this time interval with the three values $3 : 4 : 5$ u.t. The results are shown on Figure 11(d-k).

The Verilog simulation of the network is illustrated through three different simulations (Figures 11(e)–11(k)) that compute correctly the outputs $D$ and $E$ according to the chosen inputs and the selected delay. In the Verilog simulation with the min and typical delays (Figures 11(e), 11(f), 11(i) and 11(j)), the outputs $D$ and $E$ behaves likewise. They transient to the state $0$, then they come back to the state $1$ due to the state changes that occur in the input $B$ at times $t = 12$ and $16$ u.t. However, from the simulation with the max delay (Figure 11(g) and 11(k)), we remark that the state changes occurred on the input $B$ are inhibited on the outputs $D$ and $E$.

Now, consider the Min Max DEVS simulations (Figures 11(d) and 11(d)). We remark that the outputs $D$ and $E$ are computed correctly. However, the interpretation and conclusions on the simulation are completely different. For the output $D$, considering the time interval $[0, 3[$, we remark that this variable is not initialized for all possible gates (their state is $x$). At time $t = 3$ u.t which corresponds to the beginning of a temporal window, the first state change occurs in the faster gate which sends out the first output $1$.

From this time until time $t = 5$ u.t, the fast gates may send out the output $1$ and the others are not yet initialized. At time $t = 5$ u.t which corresponds to the end of the opened temporal window, the last late gate is initialized and output $1$. Then, look at the same output $D$ on the temporal window $[15, 16]$. Fast gates output $0$, others are still in the previous state $1$ and do not output yet. If we look at the Verilog simulation, this conclusion is confirmed. The output $D$ with min and typical delay output $1$ whereas the output $D$ with the max delay does not output and remains in the state $1$. However, from the time $t = 16$ u.t, the gate $D$ transits to the totally unknown state $(\phi, \phi, passive)$ and no output is possible from this state. So, we conclude that the chosen min max delay ($[3, 5]$ u.t) is too large in order to ovoid the totally unknown state $(\phi, \phi, passive)$ and to obtain a precise simulation.

Logically, if we reduce the min max delay to a time interval small than $[3, 5]$, we may avoid the totally unknown state and we may obtain a precise simulation.

Note that, for Verilog simulations continue normally. For example, from time $t = 24$ u.t, the output $D$ is in the state $1$, in the three simulations; may let us infer a wrong conclusion that this output in all gates behaves likewise if we do not take into account the previous state changes.

This comparison between Min Max DEVS and Verilog simulations reveals some points to enhance and questions to answer quickly. The given basic gates design only the inertial delay, so we should expand our basic gates to all delays of the literature (transport, rise-fall, stochastic and fuzzy delays) and compare them with Verilog and other simulators in the field. In addition, we should take into account when we redesign our basic gates to take into account the impedance value $z$ at initialization. More important which need more work, how to propagate the totally unknown state $(\phi, \phi, passive)$ in the network, on one hand. On the other hand, it is possible to use such a state for synthesizing a network of logic gates like in Verilog with the unknown state $x$. Thus, the road map is defined for the near future works.

## Conclusion

In this work, we used the Min Max DEVS formalism proposed by the Prof. and our colleague Norbert Giambiasi to model and simulate logic gates. We proceed, first by designing elementary gates based on the inertial delay due to the fact that this formalism defines naturally an event filter mechanisms thanks to its internal transition and time advance functions. So in our case short (parasite) pulses are automatically ignored. Then, the designer may define a network of logic gates eligible for Min Max DEVS simulation thanks to the closure under coupling property of this formalism.

Moreover, we designed and implemented the Min Max DEVS simulator, specific for the design and simulation of logic gates. This prototype allowed us to make a comparison with the Verilog simulator and conclude significant insights for the future works.

Recall that someone may believe that a Min Max DEVS simulation is less accurate comparing with a stochastic or fuzzy simulation, due to the fact that the simulation can not continue to give more results. In fact, the gate remains in unknown state definitely but when its state is known ($0$ or $1$) more information may be inferred like the possible states for all gates with different delay values.

In addition, in case of composite gates, we can deduce by simulation if a subgate switches on or off, or only some of them according to the min max delay of influence gate. Note that the similar gate simulators like VHDL©, Verilog©, etc. use simulations with three values (min:typ:max delay) to simulate a min-max delay. Such a simulation returns a behavior of the gate only for these three values and no information can be returned how the gate behaves for the other delay values.

Someone may ask the usefulness of such approach to model and simulate large circuits such used in industry. Firstly, we believe that if DEVS concepts are able to support the structure of large circuits (high number of components and complex coupling), the Min Max DEVS coupled formalism is able in its turn. Because this formalism uses the same concepts of DEVS. However, the Min Max DEVS simulation employs large event schedulers than those of DEVS simulation; this is due to the simulation semantics of Min Max DEVS atomic models. On the other hand, we may question the contribution the simulation results based on this formalism for such circuits because of reaching the unknown state during a simulation, quickly. A hypothesis that we hope highlight by conducting additional works in the future.

### References

N. Giambiasi, A. Miara, D. Muriach, Silog : A practical tool for logic digital networks simulation, in: 16th Design Automation Conference, San Diego, USA.

N. Giambiasi, A. Miara, D. Muriach, Methods for generalized deductive fault simulation, in: Proceedings of the 17th Design Automation Conference, DAC '80, Minneapolis, Minnesota, USA, June 23-25, 1980, pp. 386–392.

D. Muriach, N. Giambiasi, A. Miara, Simulators for design verification and diagnosis of logical networks, CAD (1979).

B. P. Zeigler, Theory of Modeling and Simulation, Fisrt edition - Academic Press, 1976.

N. Giambiasi, B. Escudé, S. Ghosh, GDEVS: A generalized discrete event specification for accurate modeling of dynamic systems, Simulation: Transactions of the Society for Modeling and Simulation International 17 (2000) 120–134.

N. Giambiasi, S. Gosh, Min max devs modeling and simulation, in: Proceedings of the 13th European Simulation Symposium, 2002, EMS.

M. Hamri, N. Giambiasi, C. Frydman, Min-max-devs modeling and simulation, Simulation Modelling Practice and Theory 14 (2006) 909–929.

M. Hamri, A. Naamane, N. Giambiasi, Generalized discrete event specifications of logic gates, in: IEEE 11th International Multi-Conference on Systems, Signals & Devices, SSD 2014, Castelldefels-Barcelona, Spain, February 11-14, 2014, pp. 1–6.

M. Hamri, N. Giambiasi, A. Naamane, Generalized discrete events for accurate modeling and simulation of logic gates, Concepts and Methodologies for Modeling and Simulation, A tribute to Tuncer Oren. Part III (2015) 257–272.

N. Driouche, M. Hamri, N. Giambiasi, Second order GDEVS abstraction of electronic circuits, in: Proceedings of the Summer Computer Simulation Conference, SummerSim 2016, Montreal, QC, Canada, July 24-27, 2016, p. 12.

N. Giambiasi, M. Hamri, C. Frydman, Simulation hiérarchique des modèles min max devs, in: Proceeding of the International Conference Modeling, Simulation and Performance, Mosim2008.

M. Hamri, fwkDEVS: A DEVS/GDEVS modeling and simulation framework, 2017. Http://www.lsis.org/hamria/fwkdevs.html.