THÈSE POUR OBTENIR LE GRADE DE DOCTEUR
DE L'UNIVERSITÉ DE MONTPELLIER

En Informatique

École doctorale : Information, Structures, Systèmes

Unité de recherche LIRMM

Vérification d'une méthodologie pour la conception de systèmes numériques critiques

Présenté par Vincent IAMPIETRO

Le Date de la soutenance

Sous la direction de David Delahaye
et David Andreu

Devant le jury composé de

[Nom Prénom], [Titre], [Labo]	[Statut jury]
[Nom Prénom], [Titre], [Labo]	[Statut jury]
[Nom Prénom], [Titre], [Labo]	[Statut jury]

UNIVERSITÉ
DE MONTPELLIER

Acknowledgements

The acknowledgments and the people to thank go here, don't forget to include your project advisor...

Contents

List of Figures

List of Tables

List of Abbreviations

SITPN	**S**ynchronously executed **I**nterpreted **T**ime **P**etri **N**et with priorities
VHDL	**V**ery high speed integrated circuit **H**ardware **D**escription **L**anguage
PCI	**P**lace **C**omponent **I**nstance
TCI	**T**ransition **C**omponent **I**nstance
GPL	**G**eneric **P**rogramming **L**anguage
HDL	**H**ardware **D**escription **L**anguage

For/Dedicated to/To my...

Chapter 1

Implementation of the HILECOP high-level models

In this chapter, we present the input formalism of our transformation function: Synchronously executed Interpreted Time Petri Nets with priorities (SITPNs). For the main part, the formalization of the SITPN structure and semantics is the result of a former thesis [4]. However, we contributed to the simplification of the definition of the SITPN structure and its semantics, and added complementary definitions for the purpose of the proof of behavior preservation. Our main contribution to this part lies in the implementation of the SITPN structure and semantics with the Coq proof assistant. This chapter is structured as follows: Section 1.1 is a reminder on the principles underlying the PN formalism and also gives an informal presentation of SITPNs; Section 1.2 lays out the formal definitions of the SITPN structure and semantics; Section 1.3 deals with the implementation of SITPNs with the Coq proof assistant.

1.1 Informal presentation of Synchronously executed Petri nets

Here, fundamentals on the Petri net formalism are outlined, and certain classes of Petri nets are described more precisely. Then, the specificities of the Petri nets used to design the behavior of electronic components in the HILECOP methodology are presented. For more information on the topic of Petri nets, the reader can refer to [2], [5], or [3].

1.1.1 Preliminary notions on Petri nets

Petri nets, invented by C. A. Petri [6], are used to model a broad range of dynamic systems: resource sharing between concurrent processes [2], behavior of agents in multi-agent systems [1], behavior of digital components [7]. A Petri net is a directed graph, composed of two types of node: place nodes (*circles*) and transition nodes (*squares* or *lines*). As shown in Figure 1.1, place nodes usually represent a part of the state of the modelled system, here the states of two computer processes and a semaphor; transition nodes usually refer to events triggering the system evolution (or state changing).

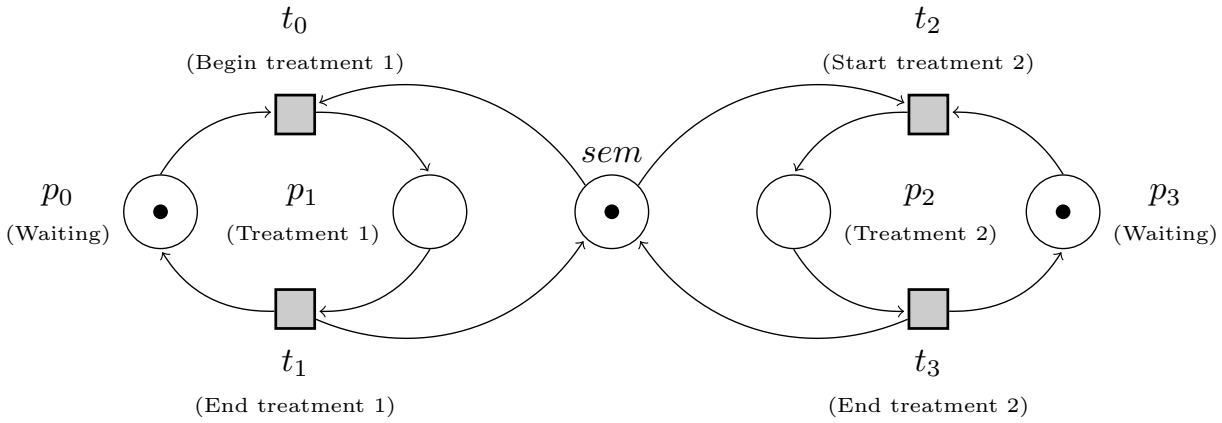


FIGURE 1.1: An example of Petri net - A semaphor to prevent the parallel execution of *Treatment 1* and *Treatment 2*.

Edges

In a Petri net, directed edges link together places and transitions. Places cannot be linked to other places, and the same stands for transitions. There are two kinds of edges, *pre* or *incoming* edges, going from a place to a transition, and *post* or *outcoming* edges, going from a transition to a place. Places linked to a transition t by incoming (resp. outcoming) edges will be referred to as the *input* (resp. *output*) of t. The same stands for a place p. For instance, in Figure 1.1, p_0 and *sem* are the input places of t_0, and p_1 is the output place of t_0; t_1 and t_3 are the input transitions of place *sem*, and t_0 and t_2 are the output transitions of *sem*. Some weight –a natural number– is associated to the edges of a Petri net. If no label appears on the edge then one is the default weight. Petri nets are said to be *generalized* when edge weights are possibly greater than one.

Marking

In Figure 1.1, places p_0, p_3 and *sem* are marked with tokens, represented by little black circles. This means that places p_0, p_3 and *Sem* are currently active. The distribution of tokens over places is called the *marking* of the net. The marking of a Petri net reflects the overall state of the modelled system at a certain moment in its activity cycle.

Transition firing

In a Petri net, the marking evolves based on a token consumption-production system. Transitions consume tokens from their input places, and produce tokens to their output places. This whole system is called *transition firing*. In order to be *firable*, a transition must be *sensitized* (or *enabled*), meaning that the number of tokens in each of its input places must be equal or greater than the weight of its incoming edges. For instance, in Figure 1.1, the transition t_0 is sensitized because the weight of the arc (p_0, t_0) is of one (default value), and place p_0 is marked with one token, and the same stands for the number of tokens in

place *sem* and the weight of the arc (*sem*, t_0). As a counter example, transition t_3 is not sensitized because there is no tokens in its input place p_2. Depending on the class of PNs that is considered, other parameters affect the *firability* of transitions (see interpreted Petri nets, time Petri nets and Section 1.1.2). When a sensitized transition is fired, tokens are retrieved from its input places (as much tokens as the weight of the arcs) and produced in its output places (as much tokens as the weight of the arcs). This process represents the occurence of an event –denoted by the transition– triggering the evolution of the system from one state to another. Figure 1.2 shows the state of the PN of Figure 1.1 after the firing of the transition t_0.

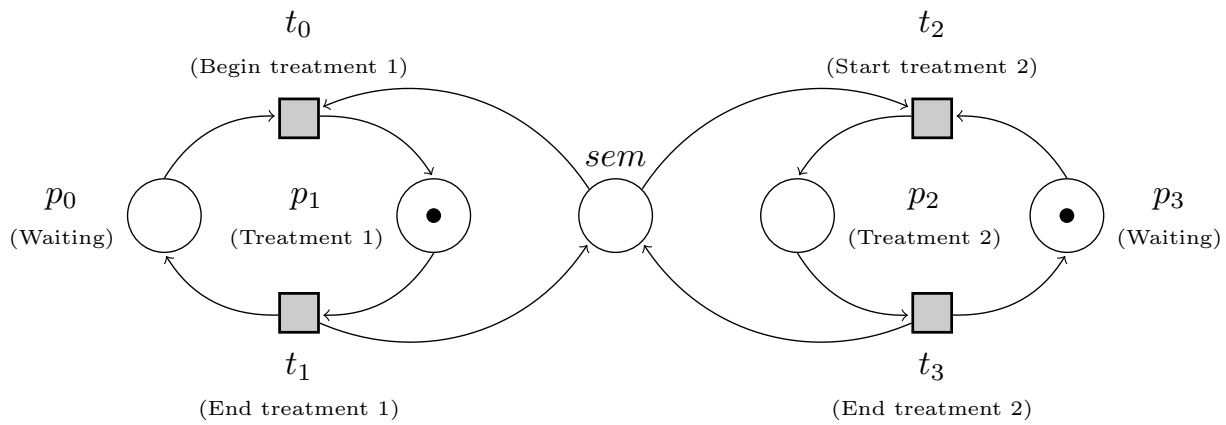


FIGURE 1.2: The PN of Figure 1.1 after the firing of transition t_0.

In Figure 1.2, the tokens in the input places of t_0, i.e. places p_0 and *sem* have been consumed, and one token has been produced in the output place p_1. The current marking indicates that the task "Treatment 1" is being performed (place p_1 is active).

In Figure 1.1, transition t_0 and t_2 are enabled at the same time. However, the *standard* semantics of Petri nets is such that only one transition can be fired in that case. Either t_0 consumes the token in place *sem* or t_2 does, but never both. Thus, the transition firing process in the standard PN semantics is an undeterministic process. From the marking of Figure 1.1, two marking are reachable: the marking resulting of the firing of transition t_0 and the one resulting of the firing of transition t_2. Also, the transition firing process is asynchronous. As soon as a transition is enabled, the transition firing process can be triggered.

Extended Petri nets

The class of *extended* Petri nets introduces the inhibitor and test edges. As shown in Figure 1.3, test arc tips are black circles and inhibitor arc tips are white circles. Inhibitor and test edges are incoming edges, always coming from a place toward a transition.

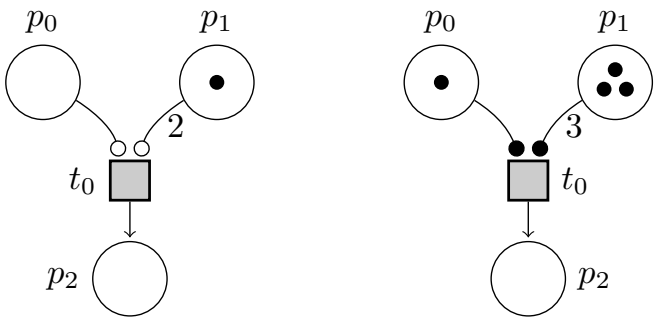


FIGURE 1.3: Two examples of extended Petri nets; on the left side, a PN with inhibitor arcs; on the right side, a PN with test arcs.

The particularity of the inhibitor and test edges is that they are not consuming tokens in input places after the firing of a transition. Indeed, they are just testing the number of tokens in incoming places to determine if the transition is enabled. Inhibitor arcs ensure that the number of tokens in input places is strictly lower than their weights; test arcs ensure that the number of tokens in pre-places is equal or greater than their weights. Therefore, on the left side of Figure 1.3, transition t_0 is sensitized because there is strictly less than one token in place p_0 and strictly less than two tokens in place p_1. On the right side of Figure 1.3, transition t_0 is sensitized because there is at least one token in place p_0 and three tokens in place p_1.

Interpreted Petri nets

Interpreted Petri nets (IPN) [2] are intended to describe the interaction between a system and its outside environment. Interpretation introduces three new concepts:

- Continuous actions, associated to the places of a Petri net. Actions associated to a place p are activated as long as p is marked. For instance, when modelling a controller with a IPN, actions can correspond to the setting of a electric signal controlling some actuator (e.g, maintaining a LED on).
- Functions (or discrete actions), associated to the transitions of a Petri net. When a transition t is fired, all functions associated to t are executed. Functions can be any kind of discrete operations –variable incrementation, for instance– manipulating both internal variable and external signal values.
- Conditions, associated to the transitions of a Petri net. Conditions are boolean expressions receiving their values from the environment of the PN. In an IPN, a transition is firable only if all its associated conditions are `true` (or `false` in the case where an inverse condition is associated).

Figure 1.4 illustrates the use of actions, functions and conditions in an interpreted Petri net.

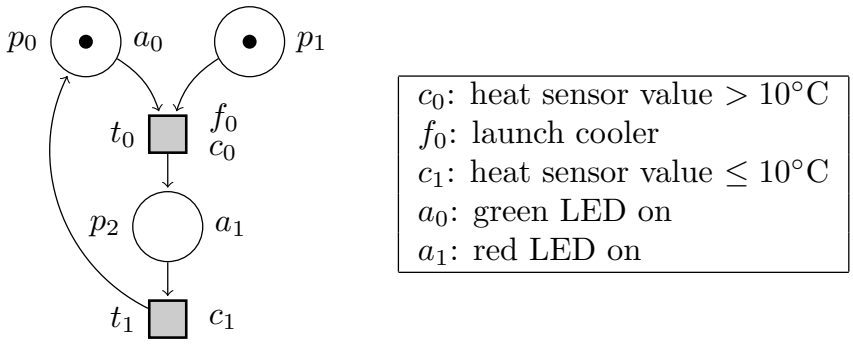


FIGURE 1.4: An example of Interpreted Petri net; on the left side, the interpreted Petri net; on the right side, examples of tests associated to conditions and operations associated to actions and functions.

In Figure 1.4, the action a_0 is activated as place p_0 is marked by one token. Also, function f_0 will be executed at the firing of t_0, that is if condition c_0 is `true` and t_0 is sensitized. On the right side of Figure 1.4, we associate a semantics to conditions, actions and functions in terms of concrete tests or operations. However, when considering the semantics of IPNs, what is of interest to us is the value of conditions and the execution state of actions and functions. We are not interested in interpreting the Boolean expressions associated to conditions but only to retrieve their value; likewise, we are only interested in the fact that a given action/function is activated/executed but not in what is its effect on the environment.

Time Petri nets

In a time Petri net (TPN), time intervals are associated to transitions. The goal of associating a time interval to a transition is to constrain the firing of this transition to a certain time window. As shown in Figure 1.5, time intervals are of the form $[a, b]$, where $a \in \mathbb{N}^*$ and $b \in \mathbb{N}^* \sqcup \{\infty\}$. Time intervals can also be defined with real numbers but in this thesis we not interested in this kind of time intervals. In Figure 1.5, time counters are represented in red between diamond brackets. The current value of time counters is part of the state of the TPN, along with its current marking, whereas time intervals are part of the static structure of the TPN.

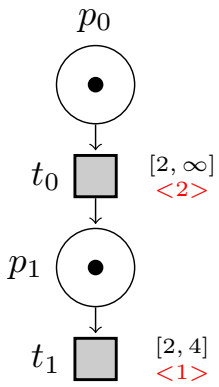


FIGURE 1.5: An example of time Petri net.

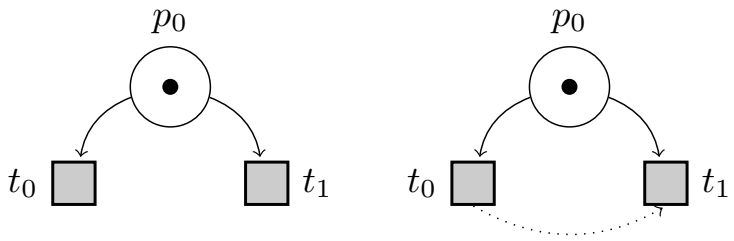
For each sensitized transition associated with a time interval, time counters are incremented at a certain time step, previously defined by the modeller. For instance, in the case of SITPNs, i.e. Petri nets used in the HILECOP methodology, the reference time step for the incrementation of time counters is the clock cycle.

When a transition associated with a time interval is fired or disabled, a reset order is sent to the transition to set its time counter to zero. The value of reset orders (Boolean values) is also a part of the TPN state. In time Petri nets, a transition is firable only if its time counter value is within its time interval. For instance, in Figure 1.5, only transition t_0 is firable.

There are multiple possible firing policy for TPNs. Here, we will only consider the *imperative* firing policy: as soon as a time counter reaches the lower bound of a time interval, the associated transition must be fired.

Petri nets with priorities

Two transitions are in structural conflict if they have a common input place connected through a *basic* arc (i.e. neither inhibitor nor test). When two transitions in structural conflict are firable at the same time, then, the conflict becomes *effective*. A Petri net with priorities, it is possible to specify a firing priority in the case where the conflict between two transitions becomes effective. In that case, the transition with the highest firing priority will always be fired first. Figure 1.6 illustrates the application of a priority relation to solve the effective conflict between two transitions.



(a) Without priorities. (b) With priorities.

FIGURE 1.6: An example of transitions in structural and effective conflict. In subfigure (b), the dotted arrow represents the priority relation between t_0 and $t1$. The transition with the highest firing priority is at the source of the arrow; here, transition t_0.

1.1.2 Particularities of SITPNs

Here, we will informally present the specificities of the Petri nets describing the internal behavior of the HILECOP high-level model components. These Petri nets are called: Synchronously executed, extended, generalized, Interpreted, Time Petri Nets with priorities or SITPNs. SITPNs are a combination of multiple classes of PNs, namely: extended PNs, generalized PNs, interpreted PNs, time PNs and PNs with priorities. These classes were presented in the above section. We will now talk about another aspect of SITPNs that constitutes the originality of the formalism compared to the standard PN semantics: its synchronous execution.

The class of interpreted Petri nets increases the expressiveness of the HILECOP high-level models. However, to ensure the safe execution of functions after the synthesis of the designed circuit on a FPGA card, the whole system must be synchronized with a clock signal [4]. As a consequence, a clock signal also regulates the evolution of SITPNS (i.e. it is a part of their semantics). The evolution of an SITPN is *synchronized* with two clock events: the rising edge and the falling edge of the signal. Figure 1.7 depicts the process of state evolution, following the clock signal.

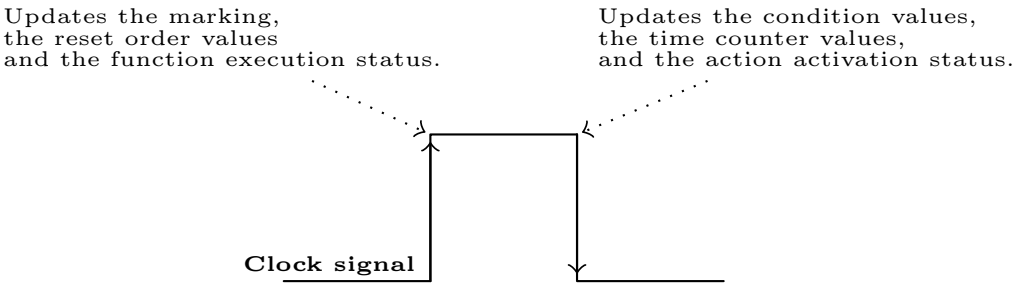


FIGURE 1.7: Evolution of an SITPN synchronized with a clock signal.

Considering the different classes of PNs that define SITPNs, the state of an SITPN is characterized by its marking, the value of time counters, the reset orders assigned to time

counters, the execution/activation status of actions/functions (Boolean values), and the value of conditions (also Boolean). As shown in figure 1.7, the state evolution process of an SITPN is divided in two parts is divided in two steps. At the rising of the clock signal, the marking is updated, i.e. transitions are fired, reset orders are sent to all transitions that have been fired or disabled by the firing process, and all functions associated to fired transitions are executed. Then, on the falling edge of the clock signal, fresh condition values are provided by the environment, the time counter values are either incremented, reset or values are stalling (see the following remark on locked time counters), and all action associated with marked places are activated. Figure 1.8 gives an example of the evolution of the state of a given SITPN through one clock cycle. The aim of this figure and the explanation that follows is to give some hints to the reader about the semantics of SITPNs before giving its formal definition in Section 1.2.4.

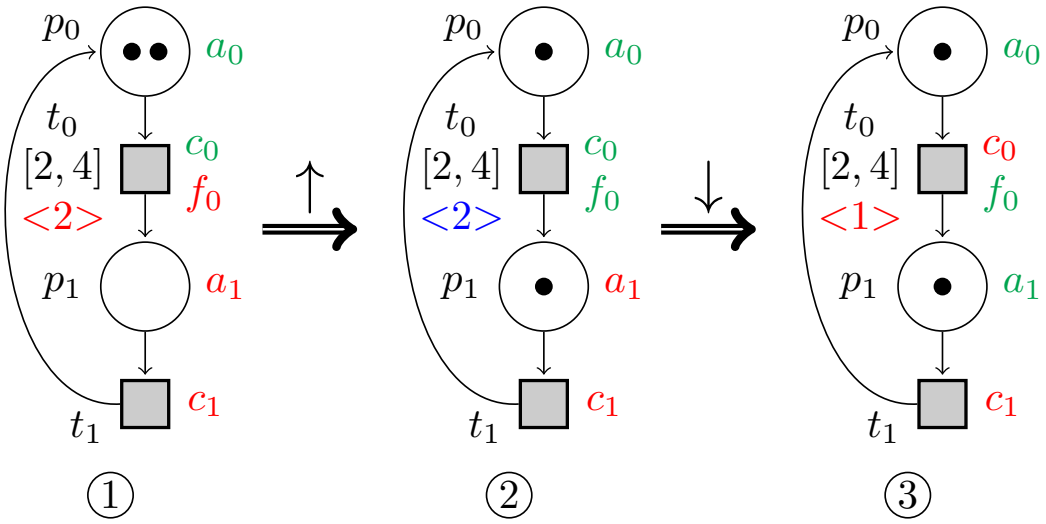


FIGURE 1.8: Evolution of an SITPN over one clock cycle. Conditions appear in green when their value is `true` and in red otherwise; actions and functions appear in green when they are activated/executed and in red otherwise; time counters appear in red and between diamond brackets; time counters appear in blue when they receive reset orders.

From Step 1 to Step 2, the rising edge of the clock signal triggers the SITPN state evolution. Here, transition t_0 is fired. At Step 1, transition t_0 gathers all the necessary conditions to trigger the firing process, namely: t_0 is enabled by the current marking, condition c_0 is `true` (appears in green), t_0's time counter value is within the time interval. As a consequence, one token is consumed in place p_0 and one token is produced in place p_1, function f_0 is executed at Step 2 (appears in green) and a reset order is sent to the time counter of t_0 (appears in blue). From Step 2 to Step 3, the activation activation status are updated; a_0 stays activated place p_0 is marked; a_1 becomes newly activated as p_1 is marked. The time counter values are updated; t_0's time counter is set to zero as the transition previously received a reset order. However, as t_0 is still sensitized by the new marking, its time counter is incremented. Thus, the resulting time counter value at Step 3

is one (i.e. result of reset plus incrementation). Also, the condition values are retrieved from the environment. As a consequence, condition c_0 takes the value `false`.

A remark on priorities

The semantics of synchronous execution is that all transitions are fired at the same time. In Figure 1.9, transitions t_0 and t_1 are both sensitized by place p_0, and consequently are both fired at the same time. The system acts as if two tokens were available in place p_0, one for the firing of t_0 and another for the firing of t_1.

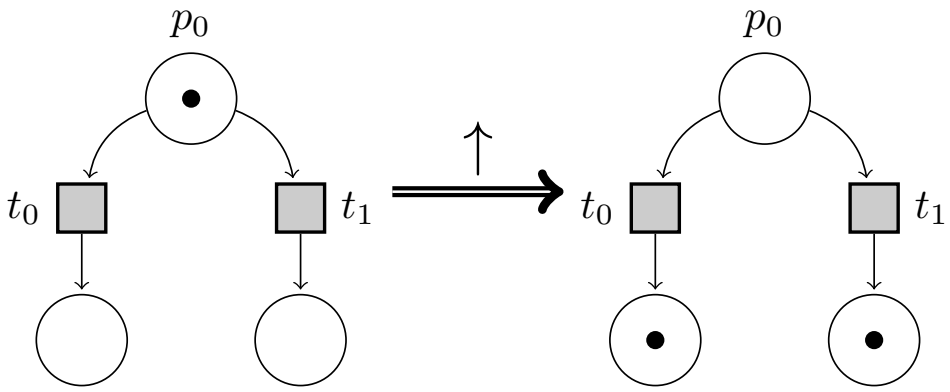


FIGURE 1.9: Double consumption of token in a SITPN. On the left side, the current marking before the firing of t_0 and t_1; on the right side, the marking resulting of the firing of t_0 and t_1. The arrow indicates the occurence of a rising edge that triggers the firing process.

In the context of an SITPN, a branching like the one of Figure 1.8, normally interpreted as an disjunctive branching, takes the semantics of a conjunctive branching when no priority are prescribed between the conflicting transitions. To avoid the phenomenon of "double consumption" of tokens, we enforce the resolution of the structural conflict by means of mutual exclusion or through the application of priorities. This policy about the resolution of structural conflict is part of the definition of a well-defined SITPN presented in Section 1.2.6, and is mandatory to produce safe models of digital systems.

When a structural conflict between transitions is solved with priorities, the firing process follows a slightly different mechanism. As illustrated in Figure 1.10, to determine which transitions of t_0, t_1 and t_2 must be fired, a *residual marking* is computed by following the priority order. For each transition of the group t_0, t_1 and t_2, the residual marking represents the remnant of tokens in p_0 after the firing of transitions with a higher firing priority. Thus, in the semantics of SITPNs, we add an extra condition to the firing of a transition: to be fired, a transition must be enabled by the current marking, must have all its conditions valuated to `true`, must have its time counter within its time interval and must be enabled by the residual marking. The computation of the residual marking only applies the consumption phase of the firing process, i.e. no tokens are generated.

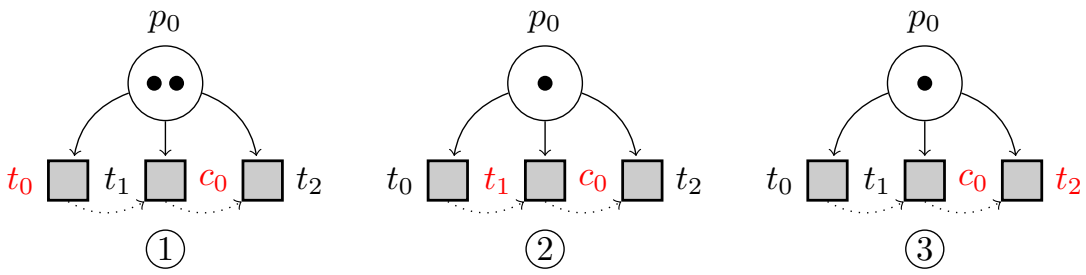


FIGURE 1.10: Computation of the residual marking for a group of conflicting transitions. At ① (resp. ② and ③), the residual marking for transition t_0 (resp. t_1 and t_2). Condition c_0 is in red to indicate that it is currently valuated to `false`.

In Figure 1.10, the residual marking for t_0 corresponds to the marking obtained after the firing of all transitions with a higher priority than t_0. As t_0 is the transition with the highest firing priority, the residual marking for t_0 is equal to the current marking. Transition t_0 gathers all the conditions to be firable and is enabled by the residual marking; thus, t_0 is fired. The residual marking for t_1 is the marking obtained after the firing of t_0, i.e. the only transition with a higher priority. As illustrated at ②, t_1 is enabled by the residual marking but it does not gather all the conditions to be firable; indeed, condition c_0 associated to t_0 is `false`. Thus, t_0 is not fired. The residual marking for t_2 is obtained after the firing of t_0 only. Indeed, transition t_1 has a higher firing priority than t_2, however, it is not part of the set of fired transitions and thus it is not taken into account in the computation of the residual marking for t_2. Thus, the residual marking at ③ enables transition t_2, and as t_2 gathers all the conditions to be firable, then t_2 is fired.

Locked time counters

SITPNs inherit from both the properties of time PNs and interpreted PNs. As a consequence, the phenomenon of *locked* time counters is a consequence of this combination. As illustrated in Figure 1.11, the value of a time counter can overreach the upper bound of the associated time interval.

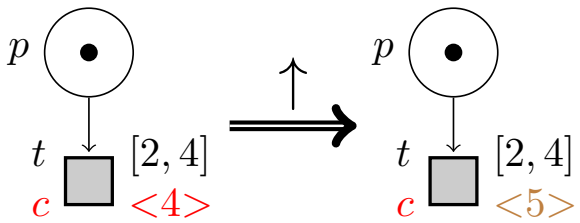


FIGURE 1.11: An example of locked time counter. Condition c is equal `false` and thus appears in red.

This situation can only arise if a condition hinders the firing of a given transition while the considered transition is still enabled by the marking. As a consequence, the time

counter will be incremented at every clock cycle until the upper bound of the time interval is overreached. Then, at this point, the time counter is said to be *locked* and its value will no more evolve. In Figure 1.11, condition c is valuated to `false`, thus, transition t can not be fired but is still sensitized by the marking. As a consequence, on the next falling edge, the time counter of transition t is incremented and overreaches the upper bound of interval $[2, 4]$ and thus becomes locked.

1.2 Formalization of the SITPN structure and semantics

Hoping that the reader has now a fair understanding of the concepts underlying the SITPNs and their evolution dynamics, we now give the formal definition of the SITPN structure and of its execution semantics. We also introduce the concept of a *well-defined* SITPN at the end of the section.

1.2.1 SITPN structure and well-definition

An SITPN is formally defined as follows:

Definition 1 (SITPN)**.** *A synchronously executed, extended, generalized, interpreted, and time Petri net with priorities is a tuple* $<P, T, pre, post, M_0, \succ, \mathcal{A}, \mathcal{C}, \mathcal{F}, \mathbb{A}, \mathbb{C}, \mathbb{F}, I_s>$*, where we have:*

1. $P = \{p_0, \ldots, p_n\}$*, a finite set of places.*
2. $T = \{t_0, \ldots, t_m\}$*, a finite set of transitions.*
3. $pre \in P \to T \nrightarrow (\mathbb{N}^* \times \{\texttt{basic}, \texttt{inhib}, \texttt{test}\})$*, the function associating a weight to place-transition edges.*
4. $post \in T \to P \nrightarrow \mathbb{N}^*$*, the function associating a weight to transition-place edges.*
5. $M_0 \in P \to \mathbb{N}$*, the initial marking of the SITPN.*
6. $\succ \subseteq (T \times T)$*, the priority relation which is a partial order over the set of transitions.*
7. $\mathcal{A} = \{a_0, \ldots, a_i\}$*, a finite set of continuous actions.*
8. $\mathcal{F} = \{f_0, \ldots, f_k\}$*, a finite set of functions (discrete actions).*
9. $\mathcal{C} = \{c_0, \ldots, c_j\}$*, a finite set of conditions.*
10. $\mathbb{A} \in P \to \mathcal{A} \to \mathbb{B}$*, the function associating actions to places.* $\forall p \in P, \forall a \in \mathcal{A}$, $\mathbb{A}(p, a) =$ `true`*, if* a *is associated to* p, $\mathbb{A}(p, a) =$ `false` *otherwise.*
11. $\mathbb{F} \in T \to \mathcal{F} \to \mathbb{B}$*, the function associating functions to transitions.* $\forall t \in T$, $\forall f \in \mathcal{F}$, $\mathbb{F}(t, f) =$ `true`*, if* f *is associated to* t, $\mathbb{F}(t, f) =$ `false` *otherwise.*

12. $\mathbb{C} \in T \to \mathcal{C} \to \{-1, 0, 1\}$, *the function associating conditions to transitions.* $\forall t \in T$, $\forall c \in \mathcal{C}$, $\mathbb{C}(t, c) = 1$, *if* c *is associated to* t, $\mathbb{C}(t, c) = -1$, *if* $\bar{c}$ *is associated to* t, $\mathbb{C}(t, c) = 0$ *otherwise.*

13. $I_s \in T \nrightarrow \mathbb{I}^+$, *the partial function associating static time intervals to transitions, where* $\mathbb{I}^+ \subseteq (\mathbb{N}^* \times (\mathbb{N}^* \sqcup \{\infty\}))$. T_i *denotes the definition domain of* I_s, *i.e. the set of time transitions.*

1.2.2 SITPN State

The SITPN semantics describes the evolution of the state of an SITPN through a number of clock cycles; thus, we must first define the SITPN state structure:

Definition 2 (SITPN State)**.** *For a given* $sitpn \in SITPN$, *let* $S(sitpn)$ *be the set of possible states of* $sitpn$. *An SITPN state* $s \in S(sitpn)$ *is a tuple* $<M, I, reset_t, ex, cond>$, *where:*

1. $M \in P \to \mathbb{N}$ *is the current marking of sitpn.*
2. $I \in T_i \to \mathbb{N}$ *is the function mapping time transitions to their current time counter value.*
3. $reset_t \in T_i \to \mathbb{B}$ *is the function mapping time transitions to time interval reset orders (defined as Booleans).*
4. $ex \in \mathcal{A} \sqcup \mathcal{F} \to \mathbb{B}$ *is the function representing the current activation (resp. execution) state of actions (resp. functions).*
5. $cond \in \mathcal{C} \to \mathbb{B}$ *is the function representing the current value of conditions (defined as Booleans).*

1.2.3 Fired Transitions

Remark 1 (Relations between markings)**.** *For all relation* $\mathcal{R}$ *existing between two marking functions* M *and* M', *the expression* $\mathcal{R}(M, M')$ *is a notation for* $\forall p \in P,\ \mathcal{R}(M(p), M'(p))$. *For instance,* $M' = M - \sum_{t_i \in Pr(t)} pre(t_i)$ *is a notation for* $\forall p \in P,\ M'(p) = M(p) - \sum_{t_i \in Pr(t)} pre(p, t_i)$.

Remark 2 (Sum expressions and arc types)**.** *Many times in this document, we need to express the number of tokens coming in or out of places, after the firing of a certain subset of transitions. To do so, we use two kinds of sum expression:*

1. *The first kind of expression computes a number of output tokens. For instance, for a given place* p, $\sum_{t \in T'} pre(p, t)$ *where* $T' \subseteq T$. *This expression is a notation for* $\sum_{t \in T'} \begin{cases} \omega \text{ if } pre(p,t) = (\omega, \texttt{basic}) \\ 0 \text{ otherwise} \end{cases}$
   Indeed, when computing a sum of output tokens (i.e, resulting of a firing process), we want to add to the sum the weight of the arc between place p *and a transition* $t \in T'$ *only if there exists an arc of type* `basic` *from* p *to* t *(remember that the test and inhibitor never lead to the withdrawal of tokens during the firing process). Otherwise, we add 0 to the sum as it is a neutral element of the addition operator over natural numbers.*

2. *The second kind expression computes a number of input tokens. For instance, for a given place* p, $\sum_{t \in T'} post(p,t)$ *where* $T' \subseteq T$. *This expression is a notation for* $\sum_{t \in T'} \begin{cases} \omega \text{ if } post(t,p) = \omega \\ 0 \text{ otherwise} \end{cases}$. *Here, we add the weight of the arc from* t *to* p *only if there exists such an arc; we add 0 to the sum otherwise.*

Therefore, in the remainder of the document, we will use the conciser notations $\sum_{t \in T'} pre(p,t)$ *to denote output token sums, and* $\sum_{t \in T'} post(t,p)$ *to denote input token sums.*

Definition 3 (Sensitization). *A transition* $t \in T$ *is said to be sensitized by a marking* M, *which is noted* $t \in Sens(M)$, *if and only if* $\forall p \in P, \omega \in \mathbb{N}^*, \big(pre(p,t) = (\omega, \texttt{basic}) \vee pre(p,t) = (\omega, \texttt{test})\big) \Rightarrow M(p) \geq \omega$, *and* $pre(p,t) = (\omega, \texttt{inhib}) \Rightarrow M(p) < \omega$.

Definition 4 (Sensitization by test and basic arcs). *A transition* $t \in T$ *is said to be sensitized by its basic and test arcs at a marking* M, *which is noted* $t \in Sens_{bt}(M)$, *if and only if* $\forall p \in P, \omega \in \mathbb{N}^*, \big(pre(p,t) = (\omega, \texttt{basic}) \vee pre(p,t) = (\omega, \texttt{test})\big) \Rightarrow M(p) \geq \omega$.

Definition 5 (Firability). *A transition* $t \in T$ *is said to be firable at a state* $s = <M, I, reset_t, ex, cond>$, *which is noted* $t \in Firable(s)$, *if and only if* $t \in Sens(M)$, *and* $t \notin T_i$ *or* $I(t) \in I_s(t)$, *and* $\forall c \in \mathcal{C}, \mathbb{C}(t,c) = 1 \Rightarrow cond(c) = 1$ *and* $\mathbb{C}(t,c) = -1 \Rightarrow cond(c) = 0$.

Definition 6 (Fired). *A transition* $t \in T$ *is said to be fired at the SITPN state* $s = <M, I, reset_t, ex, cond>$, *which is noted* $t \in Fired(s)$, *if and only if* $t \in Firable(s)$ *and* $t \in Sens\big(M - \sum_{t_i \in Pr(t)} pre(t_i)\big)$, *where* $Pr(t) = \{t_i \mid t_i \succ t \wedge t_i \in Fired(s)\}$.

1.2.4 SITPN Semantics

Definition 7 (SITPN Semantics). *The semantics of an SITPN is the transition system* $< S, L, E, \rightsquigarrow >$ *where:*

- S *is the set of states of the SITPN.*
- $s_0 =< M_0, O_{\mathbb{N}}, O_{\mathbb{B}}, O_{\mathbb{B}}, O_{\mathbb{B}} >$ *is the initial state of the SITPN, where* M_0 *is the initial marking of the SITPN,* $O_{\mathbb{N}}$ *is a function that always returns 0,* $O_{\mathbb{B}}$ *is a function that always returns false.*
- $L \subseteq Clk \times \mathbb{N}$ *is the set of transition labels, where* $Clk \in \{\uparrow, \downarrow\}$. *A label is a couple* (clk, τ) *composed of a clock event* $clk \in Clk$, *and a time value* $\tau \in \mathbb{N}$ *expressing the current count of clock cycles.*
- $E \in \mathbb{N} \to \mathcal{C} \to \mathbb{B}$ *is the environment function, which gives (Boolean) values to conditions (*$\mathcal{C}$*) depending on the count of clock cycles (*$\mathbb{N}$*).*
- $\rightsquigarrow \subseteq S \times L \times S$ *is the state transition relation, which is noted* $E, \tau \vdash s \overset{clk}{\rightsquigarrow} s'$ *where* $s, s' \in S$ *and* $(clk, \tau) \in L$, *and which is defined as follows:*

– $\forall \tau \in \mathbb{N}, E, \tau \vdash s \overset{\downarrow}{\rightsquigarrow} s'$, *where* $s =< M, I, reset_t, ex, cond >$ *and* $s' =< M, I', reset_t, ex', cond' >$, *if:*

(1) $cond'$ is the function giving the (Boolean) values of conditions that are extracted from the environment at the clock count τ, i.e.:
$\forall c \in \mathcal{C},\ cond'(c) = E(\tau, c)$.

(2) All the actions associated with at least one marked place in the marking M are activated, i.e.:
$\forall a \in \mathcal{A},\ ex'(a) = \sum_{p \in marked(M)} \mathbb{A}(p, a)$ *where* $p \in marked(M) \equiv M(p) > 0$.

(3) All the time transitions that are sensitized by the marking M and received the order to reset their time intervals, have their time counter reset and incremented, i.e.:
$\forall t \in T_i,\ t \in Sens(M) \wedge reset_t(t) = 1 \Rightarrow I'(t) = 1$.

(4) All the time transitions with active time counters that are sensitized by the marking M and did not receive a reset order, have their time counters incremented, i.e.:
$\forall t \in T_i,\ t \in Sens(M) \wedge reset_t(t) = \texttt{false} \wedge (I(t) \leq upper(I_s(t)) \vee upper(I_s(t)) = \infty) \Rightarrow\ I'(t) = I(t) + 1$.

(5) All the time transitions verifying the same conditions as above, but with locked counters, keep having locked counters, i.e.:
$\forall t \in T_i,\ t \in Sens(M) \wedge reset_t(t) = \texttt{false} \wedge I(t) > upper(I_s(t)) \wedge upper(I_s(t)) \neq \infty \Rightarrow\ I'(t) = I(t)$.

(6) All the time transitions that are not sensitized by the marking M have their time counters set to zero, i.e.:
$\forall t \in T_i,\ t \notin Sens(M) \Rightarrow I'(t) = 0$.

– $\forall \tau \in \mathbb{N}, E, \tau \vdash s \overset{\uparrow}{\rightsquigarrow} s'$, *where* $s =< M, I, reset_t, ex, cond >$ *and* $s' =< M', I, reset'_t, ex', cond >$, *if:*

(7) M' is the new marking resulting from the firing of all the transitions contained in $Fired(s)$, i.e.:
$M' = M - \sum_{t \in Fired(s)} pre(t) + \sum_{t \in Fired(s)} post(t)$.

(8) A time transition receives a reset order if it is fired at state s, or, if there exists a place p connected to t by a `basic` *or* `test arc` *and at least one output transition of p is fired and the transient marking of p disables t; no reset order is sent otherwise:*

$$\forall t \in T_i, t \in Fired(s)$$
$$\vee \big(\exists p \in P, \omega \in \mathbb{N}^*,\ pre(p,t) = (\omega, \texttt{basic}) \vee pre(p,t) = (\omega, \texttt{test})$$
$$\wedge \sum_{t_i \in Fired(s)} pre(p, t_i) > 0$$
$$\wedge\, s.M(p) - \sum_{t_i \in Fired(s)} pre(p, t_i) < \omega\big) \Rightarrow reset'_t(t) = \texttt{true},$$
$$\text{and } reset'_t(t) = \texttt{false} \text{ otherwise}.$$

(9) *All functions associated with at least one fired transition ar executed, i.e:*
$\forall f \in \mathcal{F},\ ex'(f) = \sum_{t \in Fired(s)} \mathbb{F}(t,f)$.

1.2.5 SITPN Execution

Definition 8 (SITPN Execution Cycle). *For a given $sitpn \in SITPN$, two states $s, s'' \in S(sitpn)$, a clock cycle count $\tau \in \mathbb{N}$, and an environment $E_c \in \mathbb{N} \to \mathcal{C} \to \mathbb{B}$, $sitpn$ passes from state s to state s'' in one clock cycle, written $E, \tau \vdash sitpn, s \xrightarrow{\uparrow,\downarrow} s''$ iff $\exists s'$ s.t. $E_c, \tau \vdash sitpn, s \xrightarrow{\uparrow} s'$ and $E_c, \tau \vdash sitpn, s' \xrightarrow{\downarrow} s''$.*

Definition 9 (SITPN Execution). *For a given $sitpn \in SITPN$, a starting state $s \in S(sitpn)$, a clock cycle count $\tau \in \mathbb{N}$, and an environment $E_c \in \mathbb{N} \to \mathcal{C} \to \mathbb{B}$, $sitpn$ yields the execution trace θ from starting state s, written $E_c, \tau \vdash sitpn, s \to \theta$, by following the two rules below.*

ExecutionEnd

$$\frac{}{E_c, 0 \vdash sitpn, s \to [\,]}$$

ExecutionLoop

$$\frac{E_c, \tau \vdash sitpn, s \xrightarrow{\uparrow} s' \qquad E_c, \tau \vdash sitpn, s' \xrightarrow{\downarrow} s'' \qquad E_c, \tau - 1 \vdash sitpn, s'' \to \theta}{E_c, \tau \vdash sitpn, s \to (s' :: s'' :: \theta)}\ \tau > 0$$

Definition 10 (SITPN Full Execution). *For a given $sitpn \in SITPN$, a clock cycle count $\tau \in \mathbb{N}$, and an environment $E_c \in \mathbb{N} \to \mathcal{C} \to \mathbb{B}$, $sitpn$ yields the execution trace θ starting from its initial state $s_0 \in S(sitpn)$ (as defined in Def. 7), written $E_c, \tau \vdash sitpn \to \theta$, by following the two rules below.*

FullExec0

$$\frac{}{E_c, 0 \vdash sitpn \xrightarrow{full} [s_0]}$$

FullExecCons

$$\frac{E_c, \tau \vdash s_0 \xrightarrow{\uparrow_0} s_0 \qquad E_c, \tau \vdash s_0 \xrightarrow{\downarrow} s \qquad E_c, \tau - 1 \vdash sitpn, s \to \theta_s}{E_c, \tau \vdash sitpn \xrightarrow{full} (s_0 :: s_0 :: s :: \theta_s)}\ \tau > 0$$

1.2.6 Well-definition of an SITPN

Conflict Definition

In the definition of an SITPN, the priority relation is a mean to solve a situation of conflict in a pair of transitions. We will keep the definition of a conflict as simple as possible. Informally, the transitions of a pair are in conflict if they have an common input place, and if both are linked to this input place by a `basic` arc. Figure 1.12 depicts a situation of conflict between two transitions.

At some point of the execution of the SITPN, the marking possibly enables the two transitions of a conflicting pair in such a manner that the firing of one transition disables the other; then, the conflict is said to be *effective*. The behavior of PNs is fundamentally asynchronous, and a token can only be consumed by one transition. However, in a synchronous setting as the one of the SITPN, all transitions are first elected to be fired, and then all fired at the same time. Therefore, the situation can arise where a same token is

consumed by two transitions, on behalf of them being transitions in effective conflict that are both elected to be fired (e.g, Figure 1.12). To prevent the phenomenon of "double spending", the well-definition property of an *SITPN* enforces the resolution of all conflicts, i.e, to be able to decide which transition in a conflicting pair will be fired when the conflict becomes effective.

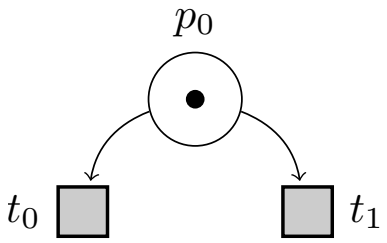


FIGURE 1.12: Example of conflict between two transitions

The formal definition of a conflict is as follows:

Definition 11 (Conflict). *For a given* $sitpn \in SITPN$*, two transitions* $t, t' \in T$ *are in conflict if and only if there exists a place* $p \in P$ *such that* $p \in input(t) \cap input(t')$ *and there exist* $n, m \in \mathbb{N}^*$ *such that* $pre(p,t) = (n, \texttt{basic})$ *and* $pre(p,t') = (m, \texttt{basic})$.

A confict group qualifies a finite set of transitions that are all in conflict with each other through the same place. In Figure 1.12, the set $\{t_0, t_1\}$ is a conflict group. The formal definition of a conflict group is as follows:

Definition 12 (Conflict Group). *For a given* $sitpn \in SITPN$*,* $T_c \subseteq T$ *is a conflict group if and only if there exists a place* p *such that* $\forall t \in output(p), \left(\exists n \in \mathbb{N}^*,\ pre(p,t) = (n, \texttt{basic})\right) \Leftrightarrow t \in T_c$.

Contrary to the statement made in [4, p. 67], we no more consider the notion of conflict as being transitive. To illustrate this, Figure 1.13 shows two conflict groups: $\{t_0, t_1\}$ and $\{t_1, t_2\}$. In a well-defined *SITPN* (see Section 1.2.6), all conflicts in a conflict group must be dealt with, i.e, for all pair of transitions in the group the conflict must be solved. However, we no more consider transitions t_0 and t_2 as in conflict. We argue that even when no conflict resolution technique is applied between transitions in the same situation as t_0 and t_2, the execution of the *SITPN* can neither result in the double-spending of a token, nor in the case where a transition is not elected to be fired even though it ought to be. Therefore, we no more consider the construction of merged conflict group (i.e, conflict groups must be merged into one if their intersection is not empty; e.g, $\{t_0, t_1, t_2\}$ in Figure 1.13) as being necessary.

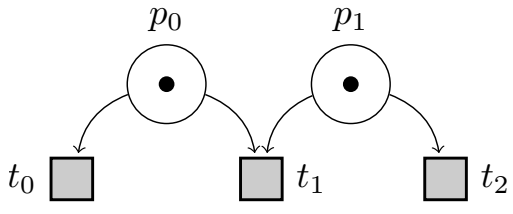


FIGURE 1.13: Example of two separate conflict groups

A given $sitpn \in SITPN$ is well-defined if it enforces some properties needed on the HILECOP source models before the transformation into VHDL. If the properties, layed out in Def. 13, are not ensured, they will lead to compile-time errors during the transformation into VHDL.

Definition 13 (Well-defined SITPN)**.** *A given $sitpn \in SITPN$ is well-defined if:*

- $T \neq \emptyset$*, the set of transitions must not be empty.*
- $P \neq \emptyset$*, the set of places must not be empty.*
- *There is no isolated place, i.e, a place that has neither input nor output transitions:* $\nexists p \in P,\ input(p) = \emptyset \wedge output(p) = \emptyset$*, where* $input(p)$ *(resp.* $output(p)$*) denotes the set of input (resp. output) transitions of* p*.*
- *There is no isolated transition, i.e, a transition that has neither input nor output places:* $\nexists t \in T,\ input(t) = \emptyset \wedge output(t) = \emptyset$*, where* $input(t)$ *(resp.* $output(t)$*) denotes the set of input (resp. output) places of* t*.*
- *All conflicts must be solved by a mean of mutual exclusion: priority relation, mutually exclusive conditions, mutually exclusive time intervals, structural mutual exclusion.*

1.3 Coq implementation of SITPNs

Bibliography

[1] Jose R. Celaya, Alan A. Desrochers, and Robert J. Graves. "Modeling and Analysis of Multi-Agent Systems Using Petri Nets". In: *2007 IEEE International Conference on Systems, Man and Cybernetics*. 2007 IEEE International Conference on Systems, Man and Cybernetics. Oct. 2007, pp. 1439–1444. DOI: 10.1109/ICSMC.2007.4413960.

[2] René David and Hassane Alla. "Petri Nets for Modeling of Dynamic Systems: A Survey". In: *Automatica* 30.2 (Feb. 1, 1994), pp. 175–202. ISSN: 0005-1098. DOI: 10.1016/0005-1098(94)90024-8. URL: https://www.sciencedirect.com/science/article/pii/0005109894900248 (visited on 06/17/2021).

[3] Michel Diaz. *Les Réseaux de Petri: Modèles Fondamentaux*. Hermès science publications, 2001.

[4] Hélène Leroux. "Méthodologie de conception d'architectures numériques complexes : du formalisme à l'implémentation en passant par l'analyse, préservation de la conformité. Application aux neuroprothèses". PhD thesis. Université Montpellier II - Sciences et Techniques du Languedoc, Oct. 28, 2014. URL: https://tel.archives-ouvertes.fr/tel-01766458 (visited on 02/10/2020).

[5] T. Murata. "Petri Nets: Properties, Analysis and Applications". In: *Proceedings of the IEEE* 77.4 (Apr. 1989), pp. 541–580. ISSN: 1558-2256. DOI: 10.1109/5.24143.

[6] Carl Adam Petri. "Kommunikation mit Automaten". In: *http://edoc.sub.uni-hamburg.de/informatik/petri.pdf* (1962). URL: https://edoc.sub.uni-hamburg.de//informatik/volltexte/2011/160/ (visited on 06/10/2021).

[7] Alex Yakovlev and Albert Koelmans. "Petri Nets and Digital Hardware Design". In: *Lecture Notes in Computer Science - LNCS*. Apr. 11, 2006, pp. 154–236. DOI: 10.1007/3-540-65307-4_49.