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 component architecture 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).

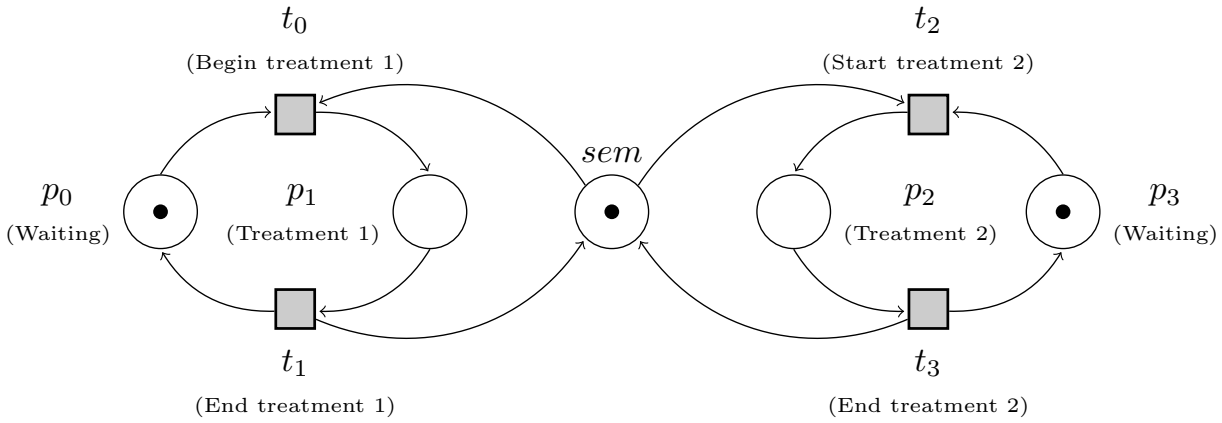


FIGURE 1.1: An example of Petri net - A semaphor to prevent the parallel execution of *Treatment 1* and *Treatment 2*.

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 will 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. Other parameters affect the firability of transitions, some of them will be presented in 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 from one state of the system to another.

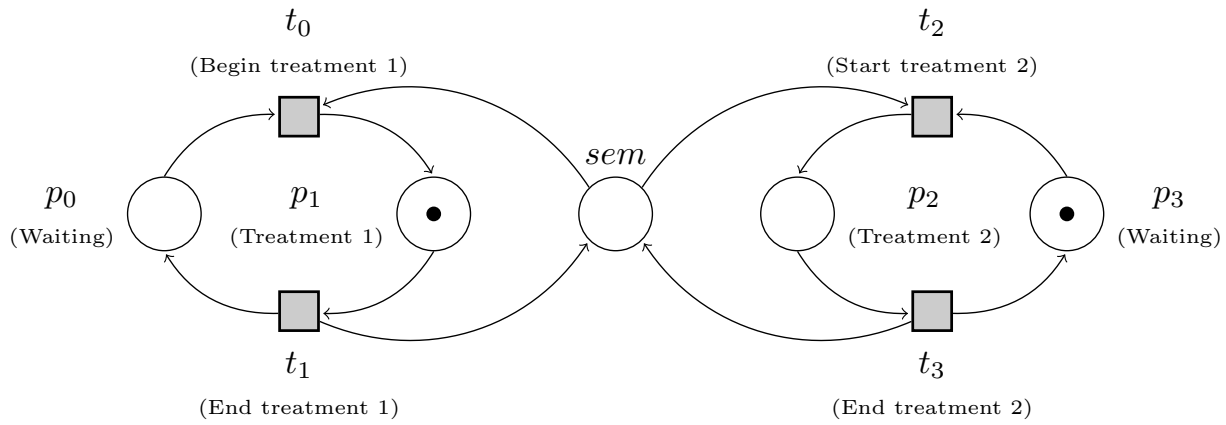


FIGURE 1.2: The PN resulting of the firing of transition t_0.

Inhibitor and test edges

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.

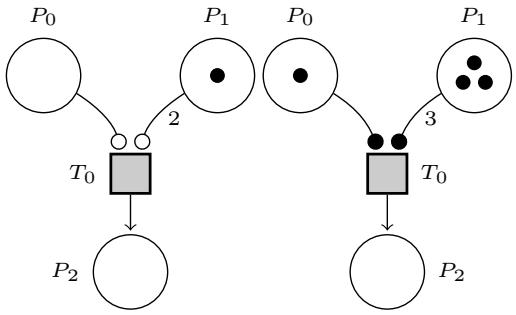


FIGURE 1.3: An example of inhibitor and test arcs.

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, in Figure 1.3.a, 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. In the same way, in Figure 1.3.b, transition t_0 is sensitized because there is at least one token in place p_0 and three tokens in place p_1.

The purpose of Hilecop Petri nets is to model the behavior of electronic components involved in an electronic system architecture. Therefore, to meet the requirements of electronic system modelling, Hilecop Petri nets combine the properties of multiple classes of Petri nets, which are presented here.

Interpreted Petri nets

Interpreted Petri nets (IPN) are useful to design the interaction between a system and its outside environment. As they describe electronic systems which are not completely cut from the outside world, Hilecop Petri nets are in need of interpretation. Thus, 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. Actions 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, calculated using both internal variable and external signal values. If the value of condition c associated to transition t is *false* then transition t is not firable.

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

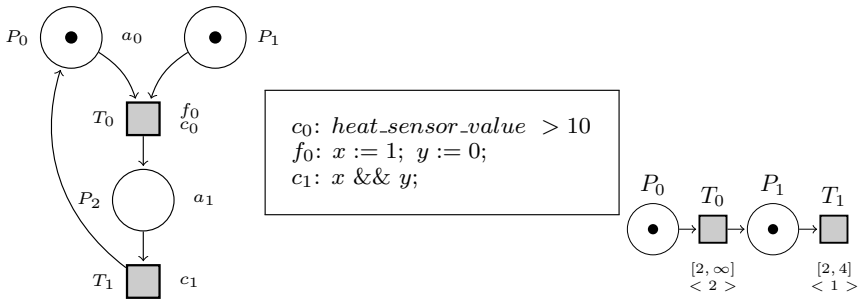


FIGURE 1.4: Different classes of Petri nets.

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.

Time Petri nets

In a time Petri net (TPN), time intervals, and time counters are associated to transitions. The goal of associating a time interval to a transition is to constraint the firing of this transition to a certain time window. As shown in figure 1.4.b, time intervals are of the form $[a, b]$, where $a \in \mathbb{N}^*$ and $b \in \{\mathbb{N}^* \cup \infty\}$. Time counters are represented between diamond brackets. For each sensitized transition associated with a time interval, time counters are incremented at a certain time step, previously defined by the modeller. Then, time counters are reset when transitions are fired or somehow disabled. In time Petri nets, a transition is firable only if its time counter value is within its time interval. For instance, in figure, only transition t_0 is firable.

1.1.2 Particularities of SITPNs

This section goal is to show how the choices regarding the properties of Hilecop Petri nets are strongly related to their final FPGA-based implementation.

Synchronous execution.

A clock signal regulates the evolution of Hilecop Petri nets, meaning that the evolution of an Hilecop Petri net is *synchronized* with two clock events: the rising edge and the falling edge of the signal. Figure 1.5 depicts the process of transition firing and global state evolution, following the clock signal.

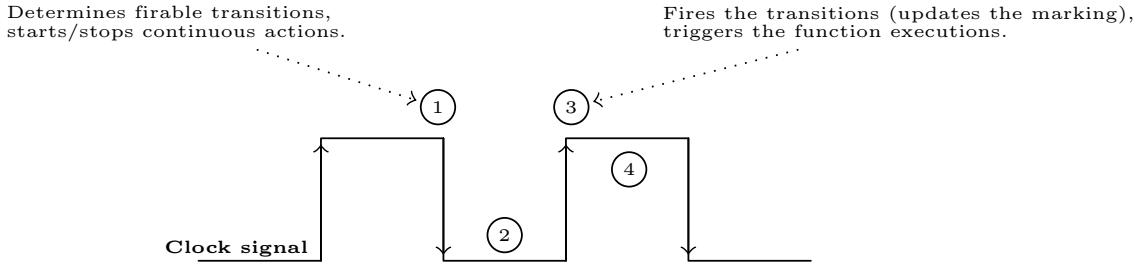
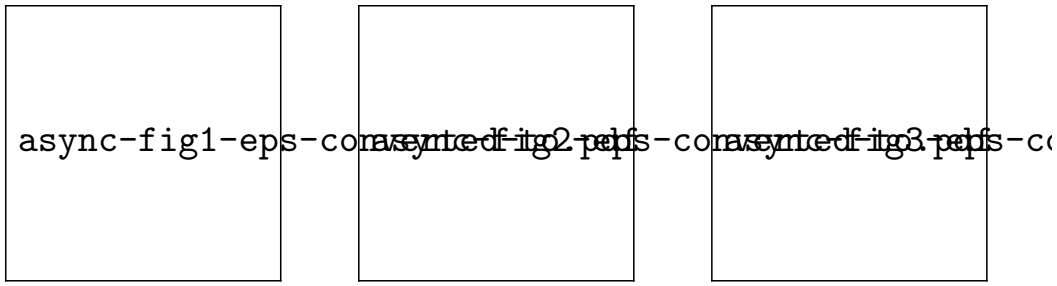


FIGURE 1.5: Evolution of the Hilecop Petri nets synchronized with a clock signal.

As shown in figure 1.5, the marking evolution process is divided in two steps. First, on ①, firable transitions are collected and actions are either started or stopped depending on the marking of the associated places. Then, on ③, all previously collected firable transitions are fired, and the associated functions are executed. The implementation of places and transitions on FPGA is responsible for the separation of transition firing in two steps [4].

Why do we need a synchronous execution for Hilecop Petri nets? While functions execute themselves instantaneously at an abstract level, it is not the case when implemented on FPGA. Indeed, on FPGA, the execution of functions takes a certain amount of time related to the propagation of electric signals inside the electronic circuit. Figure illustrates the effect of the time taken by the execution of function f_0.



(a) An example of Petri net.

(b) Case: T_1 is fired before the end of f_0.

(c) Case: f_0 terminates before taking the firing decision for T_1.

FIGURE 1.6: Effect of the duration of functions in firing decisions [4].

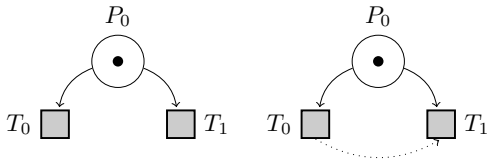
As it is standard in Petri net theory, the marking evolves in an *asynchronous* way, that is, transitions are fired one at a time. In figure, t_0 is fired before t_1, which triggers the execution of f_0. Therefore, two cases arise:

- f_0 has not completed yet when the decision to fire t_1 is taken, although the execution of f_0 affects the firing t_1.
- f_0 has completed when the decision to fire t_1 is taken.

One can see how the amount of time taken by the execution of functions influences the evolution of a Petri net implemented on FPGA. If the evolution of Hilecop Petri nets is asynchronous, there are no automatic means to ensure that f_0 completes every time before the firing of t_1 **[leroux]**. At the end of the Hilecop production line, when Petri nets are implemented on FPGA, it is possible leveraging an electronic circuit analyzer to determine the longest signal propagation path of the system. Then, it is easy to deduce an upper time bound for all treatments –which are nothing more than signal propagations– taking place in the physical system. Eventually, the clock signal frequency will be set accordingly, to permit that all functions have enough time to be completely executed within half a clock period (④ in figure 1.5). Thus the use of a clock signal and the orientation of Hilecop Petri nets toward a synchronous execution are mandatory.

Conflict resolution.

The choice of a synchronous evolution is not without consequences. Indeed, some issues arise when trying to express an *or* branching with Hilecop Petri nets, as shown by figure 1.7.a.



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

FIGURE 1.7: Transitions in structural conflict.

The semantics of synchronous execution is that all transitions are fired at the same time. Then, in figure 1.7.a, 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. In the situation depicted by figure 1.7.a, t_0 and t_1 are said to be in *structural conflict* with each other, meaning they have one of their pre-places in common. To resolve structural conflicts, priorities are drawn between transitions which will determine a firing order in case of conflict. In figure 1.7.b, the dotted arrow represents the priority relation between transition t_0 and t_1. Here, t_0 has a higher firing priority than t_1.

1.2 Formal definition of the SITPN structure and semantics

1.2.1 SITPN structure and well-definition

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 strict order over the set of transitions.*
7. $\mathcal{A} = \{a_0, \ldots, a_i\}$, *a set of continuous actions.*
8. $\mathcal{C} = \{c_0, \ldots, c_j\}$, *a set of conditions.*
9. $\mathcal{F} = \{f_0, \ldots, f_k\}$, *a set of functions (discrete actions).*
10. $\mathbb{A} \in P \to \mathcal{A} \to \mathbb{B}$, *the function associating actions to places.* $\forall p \in P, \forall a \in \mathcal{A}, \mathbb{A}(p, a) = \texttt{true}$, *if a is associated to p,* $\mathbb{A}(p, a) = \texttt{false}$ *otherwise.*
11. $\mathbb{F} \in T \to \mathcal{F} \to \mathbb{B}$, *the function associating functions to transitions.* $\forall t \in T, \forall f \in \mathcal{F}, \mathbb{F}(t, f) = \texttt{true}$, *if f is associated to t,* $\mathbb{F}(t, f) = \texttt{false}$ *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.*

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 1.8 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 1.8). 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.

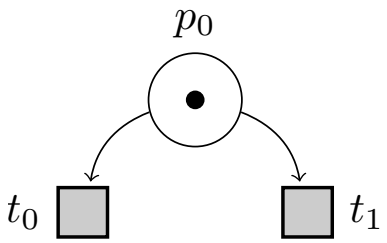


FIGURE 1.8: Example of conflict between two transitions

The formal definition of a conflict is as follows:

Definition 2 (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 1.8, the set $\{t_0, t_1\}$ is a conflict group. The formal definition of a conflict group is as follows:

Definition 3 (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 1.9 shows two conflict groups: $\{t_0, t_1\}$ and $\{t_1, t_2\}$. In a well-defined $SITPN$ (see Section 1.2.1), 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 1.9) as being necessary.

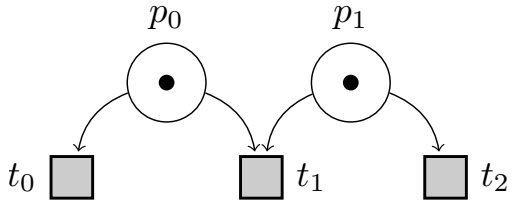


FIGURE 1.9: Example of two separate conflict groups

Well-definition of an SITPN

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. 4, are not ensured, they will lead to compile-time errors during the transformation into VHDL.

Definition 4 (Well-defined SITPN). *A given $sitpn \in SITPN$ is well-defined if and only 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.*

1.2.2 SITPN State

Definition 5 (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. $I \in T_i \to \mathbb{N} \sqcup \{\psi\}$ *is the function mapping time transitions to their current time counter value or to the value* locked.
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* $\sum_{t \in T'} \begin{cases} \omega \ if\ pre(p,t) = (\omega, \texttt{basic}) \\ 0\ otherwise \end{cases}$. *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 6 (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 7 (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 8 (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 9 (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 10 (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:*

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

(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 11 (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* $E, \tau \vdash sitpn, s \overset{\uparrow,\downarrow}{\longrightarrow} s''$ *iff* $\exists s'$ *s.t.* $E_c, \tau \vdash sitpn, s \overset{\uparrow}{\rightarrow} s'$ *and* $E_c, \tau \vdash sitpn, s' \overset{\downarrow}{\rightarrow} s''$.

Definition 12 (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.*

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

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

Definition 13 (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. 10), written $E_c, \tau \vdash sitpn \to \theta$, by following the two rules below.*

$$\frac{}{E_c, 0 \vdash sitpn \xrightarrow{full} [s_0]}\ \textsc{FullExec0}$$

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

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/volltexte/2011/160/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.