in *Proc. ICECCS*, 2000, pp. 134-142.

Definitions of Equivalence for Transformational Synthesis of Embedded Systems

Luis Alejandro Cortés, Petru Eles, and Zebo Peng
Dept. of Computer and Information Science
Linköping University, Linköping, Sweden
{luico,petel,zebpe}@ida.liu.se

Abstract

Design of embedded systems is a complex task that requires design cycles founded upon formal notation, so that the synthesis from specification to implementation can be carried out systematically. In this paper we present a computational model for embedded systems based on Petri nets called PRES+. It includes an explicit notion of time and allows a concise formulation of models. Tokens, in our notation, hold information and transitions—when fired—perform transformation of data. Based on this model we define several notions of equivalence (reachable, behavioral, time, and total), which provide the framework for transformational synthesis of embedded systems. Different representations of an Ethernet network coprocessor are studied in order to illustrate the applicability of PRES+ and the definitions of equivalence on practical systems.

1. Introduction

Many electronic systems consist of dedicated hardware elements and software running on specific platforms. Such systems are obviously heterogeneous, i.e. are composed of elements with distinct properties. At the same time, such systems are typically embedded, that is, they are part of larger systems and interact continuously with their environment. The inherent heterogeneity of this kind of systems makes them very complex and consequently difficult to design. Moreover, the strong demand on high-performance products has boosted the levels of sophistication of such systems. The ever increasing complexity of embedded systems poses a challenge in the different phases of their design process.

Design of hardware/software systems is a complex task. Design cycles should be based on formal models so that the synthesis from specification to implementation can be carried out systematically. In order to devise systems that meet the performance, cost, and reliability goals, the design process must be founded upon a formal representation that allows to accomplish the whole design cycle. Modeling is an essential issue of any systematic design methodology. In this work we introduce the formal definition of PRES+ (Petri net based Representation for Embedded Systems), a notation capable of capturing relevant information characteristic to embedded systems.

Synthesis of systems involves translating a specification at a certain level of abstraction into an implementation at a more detailed level of granularity. Hence techniques to transform an abstract description into a more refined model are essential in the synthesis process. Conversely, in order to facilitate the analysis of complex systems, higher levels of abstraction might be achieved through hierarchical composition preserving the system properties. Therefore, transformations can either refine a description or abstract it.

Transformational synthesis approaches perform a number of (small) transformation steps to the initial specification until a certain implementation is achieved. Such a stepwise approach to the synthesis of digital systems has some advantages with respect to the traditional synthesis approach, for instance it is more efficient when using optimization heuristics in the synthesis process [7]. Another appealing feature of this kind of synthesis process is the fact that different design alternatives might be explored if various valid transformations or sequences of transformations are applied to the initial specification. Thus a variety of analysis, such as cost, performance, etc., may be accomplished.

Transformation-based synthesis is also becoming very attractive since design processes might be linked to formal verification methods. For the levels of complexity typical to modern electronic systems, traditional validation techniques like simulation and testing are neither sufficient nor viable to verify their correctness: such techniques may cover just a small fraction of the system behavior; additionally, bugs found late in prototyping phases have a negative impact on time-to-market. Formal methods are becoming a practical alternative to ensure the correctness of digital designs, overcoming some of the limitations of traditional validation techniques [3]. Through a rigorous analysis of the system, it is possible to come to a better understanding of its behavior, uncover inconsistencies or ambiguities, and obtain new insights.

A transformation is considered valid or not, depending on the properties that it preserves. A sound transformation is one that changes a representation into an equivalent one.

This research is sponsored by the Swedish National Board for Industrial and Technical Development (NUTEK) in the frame of the SAVE project.

Here the concept of equivalence plays a key role. When we claim that two systems are equivalent, it is very important to understand the significance of equivalence and point out the frame in which such an equivalence is encompassed. Then we can say that a particular type of transformation will preserve the properties of a system according to the concept of equivalence in which that transformation is contrived. Thereupon, clear definitions of equivalence should be stated in order to study transformation-based design processes.

In this work we define several notions of equivalence (reachable, behavioral, time, and total) for embedded systems represented using PRES+. The principal idea behind the definitions of equivalence presented here is to provide a formal framework for transformational synthesis of embedded systems. Thus we can show whether one description is equivalent to another and then define valid transformations in the synthesis process. Establishing distinct levels of equivalence allows to study several properties of a system—especially which of them are preserved—when it is represented by different descriptions.

This paper consists of two main parts. In the first one, Section 3, we formally define the computational model that we use to represent embedded systems, and we use a simple example to illustrate the semantics of PRES+. In the second part, we define the concepts of equivalence (Section 4) and study two representations of a network coprocessor to show how such notions are applicable to practical systems (Section 5). Finally, Section 6 outlines some conclusions and gives insights for future work.

2. Related Work

The frame of reference to study the validity of a given transformation is the model used to represent the system. Many computational models have been proposed in the literature to represent digital systems. These models encompass a broad range of styles, characteristics, and application domains. Particularly in the field of embedded system design, a variety of models has been developed and used for system representation [13], [6]. Their features largely differ even though they all are computational models intended for heterogeneous embedded systems.

Particularly, Petri nets (PNs) might be an interesting representation for this kind of systems: PNs, for instance, may fairly represent parallel as well as sequential activities, and may easily capture non-deterministic constructions. In embedded systems design, Petri nets have been extended in various ways to fit their most relevant traits, e.g. notion of time, and we can find several PN-based models with different flavors: Maciel *et. al* [14] introduce an intermediate model for hardware/software codesign, extending Petri nets to analyze certain properties used in the partitioning process; Stoy [23] presents a modeling technique where timed Petri nets with restricted transition rules are used to represent control flow in both hardware and software; Esser *et al.* [8] utilize a combination of time Petri nets and predicate/transition nets augmented with object-oriented concepts as model of computation during the design of embedded systems; Coloured Petri nets have been also used to model embedded systems [1].

On the other hand, several notions of equivalence have been addressed in various contexts for different sorts of systems. In the field of digital design there have been many proposed approaches using transformations [6]. Specifically, many of the concepts of equivalence and transformations defined for PNs attempt to overcome the state explosion problem, namely Petri nets tend to become complex even for relatively small systems. Murata [16] proposes a set of basic transformations—aimed to reduce the complexity of traditional Petri nets—that preserve liveness, safeness, and boundedness. This includes fusion of places, fusion of transitions, and elimination of self-loops. Similarly, Berthelot [2] studies several transformations on PNs, the conditions under which those transformations are applicable, and the preservation of a rich set of properties. Felder *et. al* [9] study Time PNs and introduce a set of transformation rules that preserve the correctness of the system in respect to bounded-invariance and bounded-response properties. An application of transformations of nets to deal with the problem of state explosion is the study of optimal schedules for complex systems [17]. The system, modeled by a Petri net, is reduced until at least one optimal solution to the scheduling problem remains.

Pomello [20] gives an overview of notions of equivalence for Petri nets of concurrent systems. Several concepts of equivalence are re-defined to capture and study the concurrency degree of systems. The ideas considered there are based on equivalence from an external point of view, that is to say, system properties are perceived by an observer that is usually the environment with which the system interacts.

Nakagawa *et al.* [18] present a method to abstract nets based on equivalence of firing sequences of a specific subset of transitions. Thus net reduction rules may be obtained in order to generate an equivalent Petri net from another one, preserving such firing sequences.

3. Petri Net based Representation for Embedded Systems

Many of the computational models used for hardware/software systems are based on extensions to finite-state machines, Petri nets, discrete-event systems, data-flow graphs, the so-called synchronous/reactive models, and communicating processes, among others. Several different models coexist in the scenario of HW/SW codesign. However, Petri nets seem to be particularly appealing: PNs have been widely used for system modeling in many fields of science; PNs are a well-understood graphical and mathematical tool; powerful formal theories, defining its structure and firing rules, have been developed around this model.

As stated before, many applications in different areas have successfully used PNs as a representation model. Due to their intrinsic characteristics and particular extensions of the conventional model, PNs might be an interesting notation for representation of embedded systems. The main extensions—proposed separately in distinct contexts—that are relevant for embedded systems include the notion of time [21], [15], [22], the concept of hierarchy [24], [5], and the modeling power of high-level PNs [10], [12].

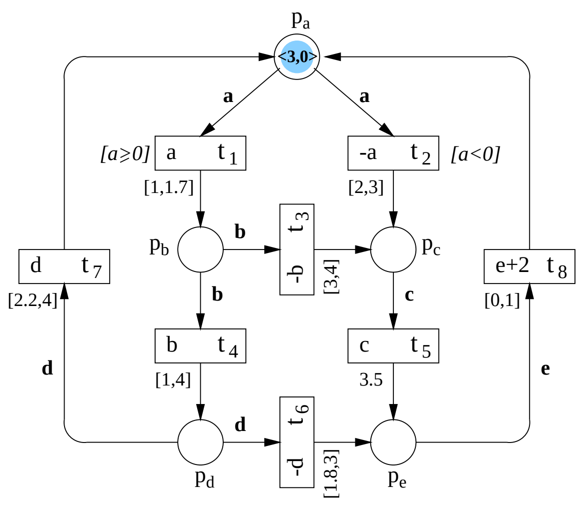


Figure 1. A PRES+ model

The notation we use to model embedded systems is PRES+ (Petri net based Representation for Embedded Systems). PRES+ is a slightly modified version of the model introduced in [4]. Its is a computational model based on Petri nets that allows to capture important characteristics of embedded systems. In the following we introduce the formal definition of PRES+. Figure 1 shows a simple example used to illustrate the main features of this representation.

3.1. Basic Definitions

Definition 1. A *PRES+* model is a five-tuple $N = (P, T, I, O, M_0)$ where
$P = \{p_1, p_2, \ldots, p_m\}$ is a finite non-empty set of *places*;
$T = \{t_1, t_2, \ldots, t_n\}$ is a finite non-empty set of *transitions*;
$I \subseteq P \times T$ is a finite non-empty set of *input arcs* which define the flow relation between places and transitions;
$O \subseteq T \times P$ is a finite non-empty set of *output arcs* which define the flow relation between transitions and places;
M_0 is the initial *marking* of the net (see Definition 3). ▪

Defined in this way, *N* is an *ordinary Petri net*, which means that there exist no multiple arcs. Like in classical Petri nets, places are graphically represented by circles, transitions by boxes, and arcs by arrows. For the example in Figure 1, $P = \{p_a, p_b, p_c, p_d, p_e\}$ and $T = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8\}$.

Properties, characteristics, and behavior of PRES+ will be introduced and defined in detail in what follows.

Definition 2. A *token* is a pair $k = \langle v, r \rangle$ where
v is the *token value*. This value may be of any type, e.g. boolean, integer, etc., or user-defined type of any complexity (for instance a structure, a set, or a record). The type of this value is referred to as *token type*;
r is the *token time*, a non-negative real number representing the time stamp of the token.

The set *K* denotes the set of all possible token types for a given system. ▪

Definition 3. A *marking* $M : P \rightarrow \{0, 1\}$ is a function that denotes the absence or presence of tokens in the places of the net. A PRES+ net *N* is *safe* or *1-bounded*, that is, a place may hold at most one token for a certain marking. $M(p) = 1$ whenever the place p is *marked*, otherwise $M(p) = 0$. ▪

Note that a marking *M* implicitly assigns one token *k* to each marked place. We introduce the following notation which will be useful in defining the dynamic behavior of PRES+: when a place p_i is marked, k_{p_i} or simply k_i denote the token present in p_i. The token value of the token k_i is denoted v_{p_i} or v_i, and the token time of the token k_i is denoted r_{p_i} or r_i. For instance, given the initial marking M_0 in Figure 1, the token $k_a = \langle 3, 0 \rangle$ has a value $v_a = 3$ and a time stamp $r_a = 0$. For the sake of simplicity, in the examples we use the short notation *w* to denote the token value v_w.

Definition 4. The *type function* $\tau : P \rightarrow K$ associates a place with a token type. $\tau(p)$ denotes the token type associated with the place *p*. The token type is the type of value that a token may bear in that place. ▪

It is worth pointing out that the token type related to a certain place is fixed, that is, it is an intrinsic property of that place and will not change during the dynamic behavior of the net. For the example in Figure 1, all places have token type *integer*.

Definition 5. The *pre-set* $^\circ t = \{p \in P | (p, t) \in I\}$ of a transition *t* is the set of *input places* of *t*. Similarly, the *post-set* $t^\circ = \{p \in P | (t, p) \in O\}$ of a transition *t* is the set of *output places* of *t*. ▪

In our example we have, for instance, $^\circ t_3 = \{p_b\}$ and $t_3{}^\circ = \{p_c\}$.

3.2. Description of Functionality

Definition 6. All output places of a given transition have the same token type,

$$if\ p, q \in t^\circ \Rightarrow \tau(p) = \tau(q)$$

▪

Definition 7. For every transition *t*, there exists a *transition function* f associated to *t*. Formally,

$$\forall t \in T \ \exists\ f : \tau(p_1) \times \tau(p_2) \times \ldots \times \tau(p_a) \rightarrow \tau(q)$$

where $^\circ t = \{p_1, p_2, \ldots, p_a\}$ and $q \in t^\circ$. ▪

Transition functions are very important when describing the behavior of the system to be represented. They allow systems to be modeled at different levels of granularity with transitions representing simple arithmetic operations or complex algorithms. In Figure 1 we inscribe transition functions inside transition boxes: the transition functions associated to t_6 and t_8, for example, are given by $f_6(d) = -d$ and $f_8(e) = e + 2$ respectively. We use inscriptions on the input arcs of a transition in order to denote the arguments of its transition function and/or its guard.

Definition 8. For every transition *t*, there exist a *minimum transition delay* d^- and a *maximum transition delay* d^+, which are non-negative real numbers and represent, respectively, the lower and upper limits for the execution time (delay) of the function associated to the transition. Formally,

$$\forall t \in T \ \ \exists\, d^-, d^+ \in \Re_0^+ \text{ such that } d^- \le d^+$$

with $\Re_0^+$ being the set of non-negative real numbers. ■

We inscribe transition delays as $[d^-, d^+]$ close to the respective transition. Thus the minimum transition delay d_1^- of t_1 is 1, and its maximum transition delay d_1^+ is 1.7 time units. Note that when $d^-=d^+=d$ we just inscribe the value *d*, like in the case of the transition delay $d_5 = 3.5$.

Definition 9. The *guard G* of a transition *t* is the (necessary) condition that must be satisfied in order to enable that transition, when all its input places hold tokens. The guard

$$G : \tau(p_1) \times \tau(p_2) \times \ldots \times \tau(p_a) \to \{0, 1\}$$

of a transition *t* is a function of the token values in the places of its pre-set $°t = \{p_1, p_2, \ldots, p_a\}$. If the condition holds *G* = 1, otherwise *G* = 0. ■

For instance, in the example of Figure 1, $a < 0$ represents the guard G_2.

Definition 10. Every transition has a *functionality*. The functionality of a transition *t* is defined in terms of:

(i) Its *transition function f*;

(ii) Its *minimum* and *maximum transition delays* d^- and d^+. ■

Intuitively, this functionality describes the "behavior" of the transition when it fires. Unlike the classical Petri net model, each token holds a value and a time stamp. When a transition *t* is fired, the marking *M* will generally change by removing all the tokens from the pre-set $°t$ and depositing one token into each element of the post-set $t°$. These tokens, added to $t°$, have values and time stamps that depend on the previous tokens in $°t$ and the functionality of *t*.

3.3. Dynamic Behavior

Definition 11. A transition *t* is said to be *enabled* if all places of its pre-set are marked, its output places different from the input ones[1] are empty, and its guard is asserted. Formally, for a given marking *M*, a transition t is enabled *iff* (if and only if)

(i) $\forall p \in °t \ \ M(p) = 1$

(ii) $\forall q \in (t° - °t) \ \ M(q) = 0$

(iii) $G = 1$

■

[1] A place may be, at the same time, input and output of a transition.

Note that, for the initial marking, t_2 is not enabled even though its only input place is marked and its output place is not. This is because $G_2 = 0$ $(3 \nless 0)$.

Definition 12. Every enabled transition *t* has an *enabling time et* that represents the time instant at which the transition becomes enabled. The enabling time *et* of a (enabled) transition is the maximum token time of the tokens in its input places,

$$et = max(r_1, r_2, \ldots, r_a)$$

where the pre-set of *t* is $°t = \{p_1, p_2, \ldots, p_a\}$. ■

Note that this enabling time varies during the execution of the net and, if the transition is not enabled, it does not make sense.

Definition 13. The *earliest trigger time* tt^- and the *latest trigger time* tt^+ of an enabled transition are the lower and upper time bounds for the firing of the transition,

$$tt^- = et + d^-$$
$$tt^+ = et + d^+$$

An enabled transition *t* may not fire before its earliest trigger time tt^- and must fire before or at its latest trigger time tt^+, unless *t* becomes disabled by the firing of another transition. ■

Assuming for instance that t_1 fires at 1 time units, the token in p_a is removed and accordingly a new token $k_b = \langle 3, 1 \rangle$ is deposited in p_b. Thus the enabling time for both t_3 and t_4 becomes 1 time units. In consequence t_3 may not fire before 4 time units and must fire before or at 5 time units, unless it becomes disabled by the firing of t_4.

Definition 14. The *firing* of an enabled transition changes a marking *M* into a new marking M^+. As a result of firing the transition *t*, with pre-set $°t = \{p_1, p_2, \ldots, p_a\}$, the following events occur:

(i) Tokens from its pre-set (which are not in its post-set) are removed;

$$\forall p \in (°t - t°) \ \ M^+(p) = 0$$

(ii) One token is added to each place of its post-set;

$$\forall q \in t° \ \ M^+(q) = 1$$

(iii) Each new token deposited in $t°$ has a token value, which is calculated by evaluating the transition function with the token values of tokens in $°t$ as arguments;

$$\forall q_i \in t° \quad v_i = f(v_1, v_2, \ldots, v_a)$$

(iv) Each new token added to $t°$ has a token time that is the time instant at which the transition *t* fires;

$$\forall q_i \in t° \quad r_i = tt^* \text{ where } tt^* \in [tt^-, tt^+]$$

■

The execution time of the function of a transition is considered in the time stamp of the new tokens. Note that, when a transition fires, all the tokens in its output places get the same token value and token time. The token time of a token represents the time at which it was "created". The system time (the global time), in a marking *M*, is given by the maximum token time of all tokens in the net.

When used to model embedded systems, the representation introduced above has several interesting features to be highlighted, some of them inherited from the classical Petri net model:

- Non-determinism may be naturally represented by PRES+. Non-determinism can be used as a powerful mechanism to express succinctly the behavior of certain systems and then reduce the complexity of the model.
- Parallel or concurrent activities may be easily expressed in terms of Petri nets. We recall that concurrency is present in most embedded systems.
- Since tokens carry information in our model, PRES+ overcomes the lack of expressiveness of classical Petri nets, where tokens are considered as "black dots".
- Time is a critical factor in many embedded applications. Our model captures timing aspects by associating lower and upper limits to the duration of activities related to transitions and keeping time information in token stamps.
- PRES+ has been also extended by introducing the concept of hierarchy. However, we will not further discuss this particular feature in this paper.

Summarizing, PRES+ is a model to be used in the design cycle of embedded systems. Our representation is an extension to the classical PN model that overcomes some of the drawbacks of Petri nets when modeling embedded systems: it captures explicitly timing information; PRES+ allows representations at different levels of granularity; our model is more expressive since tokens might carry information. Furthermore, the model is simple, intuitive, and can be easily handled by the designer.

4. Equivalence in Embedded Systems

In what follows we define the four notions of equivalence mentioned in Section 1. These ideas are built upon the model presented in Section 3. The underlying motivation of the following definitions is to set a formal framework to compare PRES+ representations, and thus establish the validity of transformations in the synthesis process of embedded systems. The study of an Ethernet coprocessor (Section 5) will further illustrate these concepts of equivalence.

Definition 15. A place p is said to be an *in-port* if

$$\forall t \in T \quad (t, p) \notin O$$

that is, there is no transition t for which p is output place. The set of in-ports is noted as *inP*. Similarly, a place p is said to be an *out-port* if

$$\forall t \in T \quad (p, t) \notin I$$

that is, there is no transition t for which p is input place. *outP* denotes the set of out-ports. ■

Note: In some cases, the designer could be only interested in some part of the net. In that case it is possible to re-define the concepts of *in-port* and *out-port* in such a manner that the following notions of equivalence apply only to that part of the system which concerns the designer. Thus, according to his needs, the designer can re-define *inP* as a subset composed of *some* (instead of *all*) places that have no transition as input. Similarly *outP* could be re-defined as the subset of *some* places with no transition as output.

Definition 16. The *reachability set* $R(N)$ of a net N is the set of all markings reachable from M_0. ■

Definition 17. A net N_1 is *reachable-equivalent* or *R-equivalent* to another net N_2 *iff*

(i) There exist bijections $f_{in} : inP_1 \to inP_2$ and $f_{out} : outP_1 \to outP_2$ that define one-to-one correspondences between in(out)-ports of N_1 and N_2;

(ii) For the initial marking, in both N_1 and N_2, all in-ports are marked and all out-ports are empty

$$\forall p \in inP_1 \quad M_{1,0}(p) = 1$$
$$\forall q \in outP_1 \quad M_{1,0}(q) = 0$$
$$\forall pp \in inP_2 \quad M_{2,0}(pp) = 1$$
$$\forall qq \in outP_2 \quad M_{2,0}(qq) = 0$$

where $M_{1,0}$ and $M_{2,0}$ are the initial markings of N_1 and N_2 respectively;

(iii) For all $M'_1 \in R(N_1)$ such that

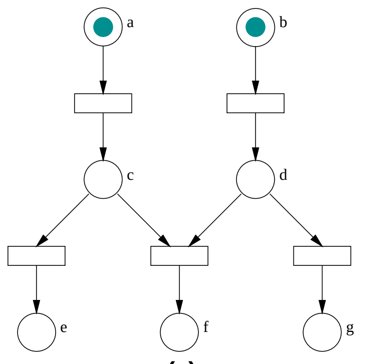
$$\forall p \in inP_1 \quad M'_1(p) = 0$$
$$\forall a \in (P_1 - inP_1 - outP_1) \quad M'_1(a) = M_{1,0}(a)$$

there exists $M'_2 \in R(N_2)$ such that

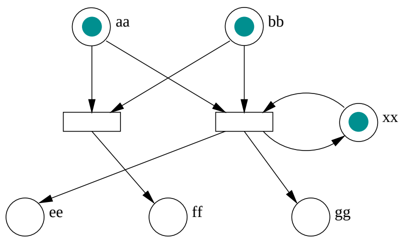
$$\forall pp \in inP_2 \quad M'_2(pp) = 0$$
$$\forall b \in (P_2 - inP_2 - outP_2) \quad M'_2(b) = M_{2,0}(b)$$
$$\forall qq \in outP_2 \quad M'_2(qq) = M'_1(q) \text{ where } qq = f_{out}(q)$$

and vice versa.

■



(a)



(b)

Figure 2. R-equivalent nets

When two nets are R-equivalent, we use the notation $N_1 \overset{R}{=} N_2$. The concept of R-equivalence will be illustrated with reference to the example shown in Figure 2. Let us consider the nets N_1 and N_2 in Figures 2(a) and 2(b) respectively. According to Definition 15, $P_1 = \{a, b, c, d, e, f,$

$g\}$, $inP_1 = \{a, b\}$, $outP_1 = \{e, f, g\}$, $P_2 = \{aa, bb, xx, ee, ff, gg\}$, $inP_2 = \{aa, bb\}$, and $outP_2 = \{ee, ff, gg\}$. One-to-one correspondences between in(out)-ports are defined by $aa = f_{in}(a)$, $bb = f_{in}(b)$, $ee = f_{out}(e)$, $ff = f_{out}(f)$, and $gg = f_{out}(g)$. Using the notation $M = (M(p_1)\ M(p_2)\ \dots\ M(p_m))$ to express the marking of a net, we can write the initial markings as $M_{1,0} = (1\ 1\ 0\ 0\ 0\ 0\ 0)$ and $M_{2,0} = (1\ 1\ 1\ 0\ 0\ 0)$, which fulfill the condition (ii) in Definition 17. A simple analysis of reachability shows that there exist two markings in the reachability set $R(N_1)$ satisfying the first part of condition (iii) in Definition 17 (in-ports are not marked and other places, different from in-ports and out-ports, have the same marking as $M_{1,0}$), namely $M'_1 = (0\ 0\ 0\ 0\ 1\ 0\ 1)$ and $M''_1 = (0\ 0\ 0\ 0\ 0\ 1\ 0)$. For each one of these markings, there exists a marking in $R(N_2)$ that fulfills the second part of condition (iii) given in Definition 17, $M'_2 = (0\ 0\ 1\ 1\ 0\ 1)$ and $M''_2 = (0\ 0\ 1\ 0\ 1\ 0)$ respectively. In a similar way, for each one of the markings M'_2 and M''_2 above, there exists a marking in $R(N_1)$ satisfying the second part of Definition 17(iii). Hence N_1 and N_2 are R-equivalent.

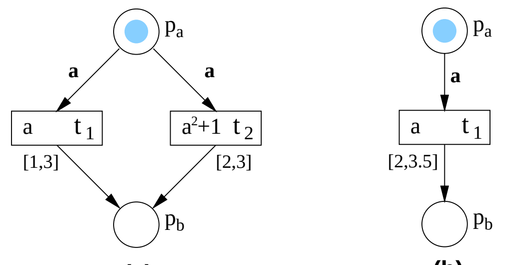


Figure 3. R-equivalent nets with different "behavior"

Before defining the concepts of *behavioral-equivalence* and *time-equivalence*, we will study the simple nets N_1 and N_2 shown in Figures 3(a) and 3(b) respectively. Having $P_1 = \{p_a, p_b\}$ and $P_2 = \{p_a, p_b\}$, the initial markings are $M_{1,0} = (1\ 0)$ and $M_{2,0} = (1\ 0)$. It is straightforward to note that $M'_1 = (0\ 1)$ and $M'_2 = (0\ 1)$ fulfill the conditions established in Definition 17 and therefore $N_1 =^R N_2$. However, note that N_1 has different "behaviors". Assuming, for both nets in the initial marking, that the token in p_a is $k_a = \langle a, 0\rangle$, it is clear that the very same marking $M'_1 = (0\ 1)$ may associate different tokens to the place p_b: when t_1 fires the token in p_b will be $k_b = \langle a, r_b^{i}\rangle$ with $r_b^{i} \in [1,3]$, but when t_2 fires the token in p_b will be $k_b = \langle a^2+1, r_b^{ii}\rangle$ with $r_b^{ii} \in [2,3]$. The reason of this behavior is the non-determinism of N_1. On the other hand, for N_2 the marking $M'_2 = (0\ 1)$ associates to p_b the token $k_b = \langle a, r_b\rangle$ with $r_b \in [2,3.5]$.

As shown in the example of Figure 3(a), a certain marking M' (see condition (iii) in Definition 17) might associate different tokens to the same place. In other words, a marking denotes absence or presence of tokens in a certain place but says nothing about token value/time when the place is marked. Note, for instance, that for the net N_1 in Figure 3(a) the set of all possible tokens in the out-port p_b, for the marking $M'_1 = (0\ 1)$, is given by $\{\langle a, r_b\rangle | r_b \in [1,3]\} \cup \{\langle a^2+1, r_b\rangle | r_b \in [2,3]\}$.

We denote with $outP'_1$ and $outP'_2$ the subsets, of $outP_1$ and $outP_2$ respectively, in which out-ports are marked in M'_1 and M'_2 (condition (iii) in Definition 17),

$$outP'_1 = \{q \in outP_1 | M'_1(q) = 1\}$$
$$outP'_2 = \{qq \in outP_2 | M'_2(qq) = 1\}$$

The concepts of $outP'_1$ and $outP'_2$ are needed in the following definitions.

Definition 18. Two nets, N_1 and N_2, are *behavioral-equivalent* or *B-equivalent iff*

(i) The nets are R-equivalent;

$$N_1 =^R N_2$$

(ii) For the initial marking stated in Definition 17(ii),

$$\forall pp \in inP_2 \quad v_{pp} = v_p \text{ where } pp = f_{in}(p)$$

(iii) For those markings that fulfill the condition (iii) in Definition 17, it holds that for all k_q such that

$$q \in outP'_1$$

there exists k_{qq} such that

$$qq \in outP'_2$$
$$v_{qq} = v_q \text{ where } qq = f_{out}(q)$$

and vice versa.

■

The expression $N_1 =^B N_2$ denotes that the two nets are B-equivalent.

Definition 19. Two nets, N_1 and N_2, are *time-equivalent* or *T-equivalent iff*

(i) The nets are R-equivalent;

$$N_1 =^R N_2$$

(ii) For the initial marking stated in Definition 17(ii),

$$\forall pp \in inP_2 \quad r_{pp} = r_p \text{ where } pp = f_{in}(p)$$

(iii) For those markings that fulfill the condition (iii) in Definition 17, it holds that for all k_q such that

$$q \in outP'_1$$

there exists k_{qq} such that

$$qq \in outP'_2$$
$$r_{qq} = r_q \text{ where } qq = f_{out}(q)$$

and vice versa.

■

Two T-equivalent nets are noted as $N_1 =^T N_2$.

Definition 20. Two nets, N_1 and N_2, are *total-equivalent* or *§-equivalent iff*

(i) The nets are B-equivalent;

$$N_1 =^B N_2$$

(ii) The nets are T-equivalent;

$$N_1 =^T N_2$$

■

We denote this strong equivalence as $N_1 =^{\S} N_2$.

Figure 4 shows the relation between the different concepts of equivalence introduced above. The graph captures the dependence between the notions of equivalence. Thus, for instance, R-equivalence is necessary for T-equivalence and also for B-equivalence. Similarly, §-equivalence implies all other equivalences. §-equivalence is the strongest notion of equivalence defined in this work. However, observe that the stronger the equivalence, the more difficult to validate a transformation based on such an equivalence.

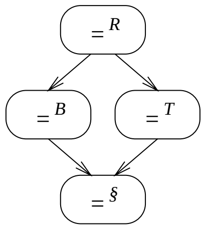


Figure 4. Relation between notions of equivalence

5. The Example: An Ethernet Network Coprocessor

In order to illustrate the concepts of equivalence defined above, let us analyze a high-level representation of an Ethernet network coprocessor. We have chosen this application because it has been discussed several times in the context of hardware/software codesign [11], [19]. The system is a network coprocessor that manages communication functions in an Ethernet link, off-loading the host computer from these activities. Briefly, the CPU sends instructions to the coprocessor. Based on these, if a frame of data must be transmitted, the Ethernet coprocessor loads the data from a location of memory, specified by the CPU, and sends the frame over the network. The coprocessor might continually receive data from the network and store it in a local memory, unless the CPU deactivates this function. In Figure 5 we give a representation of this system formulated in PRES+, which consists of three functional blocks (*execution unit*, *transmit unit* and *receive unit*). Each functional block is represented by a transition. The model includes places through which the system interacts with its environment, in this case the host CPU, the Ethernet link, and a local memory.

We consider as a first alternative a possible implementation of such a system completely in hardware. The execution times of each one of the functional units can be estimated, and the duration of its operations can be assigned as lower and upper limits to the respective transitions in the PRES+ model. For the sake of clarity, Figure 5 does not show inscriptions on the arcs, nonetheless all transitions do have transition functions associated to them. Thus, for example, there exists a transition function corresponding to *execution unit*, defined in terms of the token values of tokens in places *INSTR* and *COLLISION*. We assume that each transition has its functionality completely defined. Figure 5 shows neither token values nor token times for the initial marking. We assume that all time stamps of tokens in the initial marking are 0. For instance, the token in the place *DATA_SEND* for the initial marking is given by $k_{DATA\_SEND} = \langle data\_send, 0 \rangle$, where *data_send* has a valid type corresponding to the token type of that place.

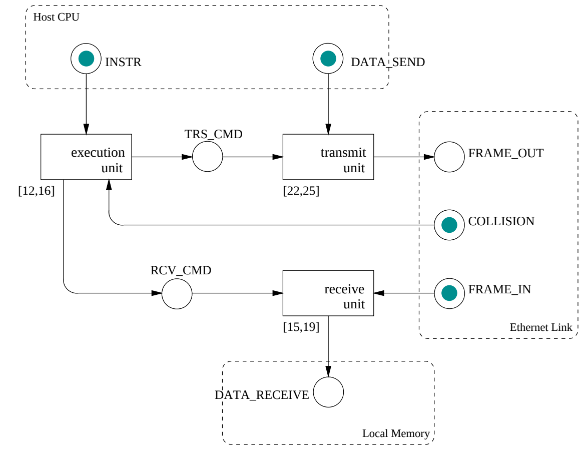


Figure 5. PRES+ model of an ethernet network coprocessor

Now suppose that, in order to explore the design space, we want to study another possible implementation of the Ethernet coprocessor as a heterogeneous hardware/software system. Having different functional blocks in our system, we can analyze several alternatives. Figure 6 shows a possible partition of the system on two processing engines. Consider that *P1* represents a programmable processor and *P2* is a hardware component. The new places *P1* and *P2* are introduced in the Petri net of Figure 6 to indicate the allocation of resources in the system and the way that functional units are mapped onto different processors (programmable and hardware). In order to consider the cost of inter-processor communication, a new transition *comm* has been added to the model. In this way, we consider the communication time, estimated between 2 and 3 time units, assigning such limits as minimum and maximum transition delays. The transition *comm* just “transmits” to *receive unit* the data coming from *execution unit*. The place *B* represents the only system bus. Since the blocks to be mapped onto software (*execution unit* and *transmit unit*) differ in implementation with respect to the previous design alternative (Figure 5), their execution times have been changed in the model to take into account the characteristics of this design. However, transition functions are the same for corresponding blocks in Figures 5 and 6.

Although the nets in Figures 5 and 6 are relatively simple, they serve our purposes of illustrating the concepts of equivalence as defined above. It is clear that Figure 6 is a refinement of the representation given in Figure 5: it is a typical step in transformational synthesis.

In order to prove the R-equivalence of the nets in Figures 5 and 6, we have to check the requirements stated in Definition 17: both nets have four in-ports and two out-ports and, then, it is possible to define bijections f_{in} and f_{out} representing one-to-one relations between these places; in both nets, for the initial marking, in-ports are marked and out-ports are not; a reachability analysis—simple for this level of abstraction but, in general, complex for more detailed systems—shows that both nets have reachable markings in which all out-ports are marked, in-ports are empty, and other places have the very same marking they had in the initial state (Definition 17). Consequently the nets are R-equivalent.

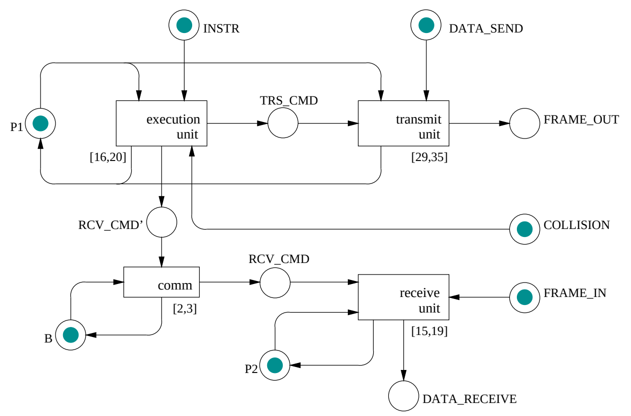


Figure 6. Refinement of the Ethernet coprocessor

The B-equivalence of these models can be noticed intuitively: since the refinement to the first representation affected only the mapping of functional blocks, it did not influence any behavioral aspect. Note that transition functions have been preserved. However, a formal demonstration of B-equivalence for such a complex system including its explicit functionality is beyond the scope of this paper.

The non-T-equivalence can be illustrated observing that for the marking characterized in Definition 17(iii), there exists at least one token in the first net that has no match in the second one. For instance, in the PRES+ model of Figure 5, when *DATA_RECEIVE* is marked its token will have a token time in the interval [27,35]. However, there does not exist a token in *DATA_RECEIVE* in Figure 6, for such a marking, whose token time is $r = 27$ time units, because the interval for such a token time will be [33,42]. Finally, since the nets are not T-equivalent, they are not §-equivalent either.

6. Concluding Remarks

Many electronic systems consist of dedicated hardware and software running on specific platforms. At the same time, such systems are typically embedded, that is, they are part of larger systems and interact continuously with their environment. The complexity of these systems is such that, in order to reason about their properties, formal notations and models are needed. Design cycles should be based on formal representations so that the synthesis of a design from specification to implementation can be carried out systematically.

We have presented PRES+, a Petri net based representation for embedded systems. It captures important features of this kind of systems: time related information, concurrency, and sequential behavior as well. The model also allows representations at different levels of granularity. In PRES+ tokens might carry information which makes the model more expressive in comparison to classical Petri nets. Furthermore, the model is simple, intuitive, and can be easily handled by the designer.

We introduced several notions of equivalence for embedded systems represented in PRES+. Thus we provided a formal framework to study embedded systems using PRES+ models. In transformational synthesis approaches unambiguous concepts of equivalence are necessary because they permit to define the validity of transformations (abstractions/refinements). Establishing different levels of equivalence allows to study several properties of a system.

The study of different representations of an Ethernet network coprocessor has illustrated the applicability of the proposed computational model and the notions of equivalence defined above on the transformation-based synthesis of a practical embedded system.

References

[1] L. P. M. Benders and M. P. J. Stevens, “Petri Net Modelling in Embedded System Design,” in *Proc. European Computer Conference*, 1992, pp. 612-617.

[2] G. Berthelot, “Checking Properties of Nets using Transformations,” in *Advances in Petri Nets 1985*, G. Rozenberg, Ed. *LNCS 222*, Berlin: Springer-Verlag, 1986, pp. 19-40.

[3] E. M. Clarke and J. M. Wing, “Formal Methods: State of the Art and Future Directions,” Technical Report CMU-CS-96-178, School of Computer Science, Carnegie Mellon University, Pittsburgh, Sept. 1996.

[4] L. A. Cortés, P. Eles, and Z. Peng, “A Petri Net based Model for Heterogeneous Embedded Systems,” in *Proc. NORCHIP Conference*, 1999, pp. 248-255.

[5] G. Dittrich, “Modeling of Complex Systems Using Hierarchically Represented Petri Nets,” in *Proc. Intl. Conference on Systems, Man, and Cybernetics*, 1995, pp. 2694-2699.

[6] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli, “Design of Embedded Systems: Formal Models, Validation, and Synthesis,” in *Proc. IEEE*, vol. 85, pp. 366-390, March 1997.

[7] P. Eles, K. Kuchcinski, and Z. Peng, *System Synthesis with VHDL*. Dordrecht: Kluwer, 1998.

[8] R. Esser, J. Teich, and L. Thiele, “CodeSign: An embedded system design environment,” in *IEE Proc. Computers and Digital Techniques*, vol. 145, pp. 171-180, May 1998.

[9] M. Felder, C. Ghezzi, and M. Pezzè, “Analyzing refinements of state based specifications: the case of TB nets,” in *Proc. Intl. Symposium on Software Testing and Analysis*, 1993, pp. 28-39.

[10] H. J. Genrich and K. Lautenbach, “System modelling with high-level Petri nets,” in *Theoretical Computer Science*, vol. 13, pp. 109-136, Jan. 1981.

[11] R. K. Gupta and G. De Micheli, “System Synthesis via Hardware-Software Co-design,” Technical Report CSL-TR-92-548,

Dept. EECS, Stanford University, Stanford, Oct. 1992.

[12] K. Jensen, *Coloured Petri Nets*. Berlin: Springer-Verlag, 1992.

[13] L. Lavagno, A. Sangiovanni-Vincentelli, and E. Sentovich, “Models of Computation for Embedded System Design,” in *NATO ASI Proc. on System Synthesis*, 1998, pp. 1-57.

[14] P. Maciel, E. Barros, and W. Rosenstiel, “A Petri Net Model for Hardware/Software Codesign,” in *Design Automation for Embedded Systems*, vol. 4, pp. 243-310, Oct. 1999.

[15] P. M. Merlin and D. J. Farber, “Recoverability of Communication Protocols—Implications of a Theoretical Study,” in *IEEE Trans. Communications*, vol. COM-24, pp. 1036-1042, Sept. 1976.

[16] T. Murata, “Petri Nets: Analysis and Applications,” in *Proc. IEEE*, vol. 77, pp. 541-580, April 1989.

[17] J. C. Murgaza, H. Camus, J.-C. Gentina, E. Teruel, and M. Silva, “Reducing the Computational Complexity of Scheduling Problems in Petri Nets by means of Transformation Rules,” in *Proc. Intl. Conference on Systems, Man, and Cybernetics*, 1998, pp. 19-25.

[18] M. Nakagawa, D.-I. S. Lee, S. Kumagai, and S. Kodama, “Equivalent Net Abstraction and Firing Sequence Preservation,” in *Proc. ISCAS*, 1995, pp. 513-516.

[19] S. Narayan, F. Vahid, and D. D. Gajski, “Modeling with SpecCharts,” Technical Report #90-20, Dept. of Information and Computer Science, University of California, Irvine, July 1990 (revised Oct. 1992).

[20] L. Pomello, “Some Equivalence Notions for Concurrent Systems: An Overview,” in *Advances in Petri Nets 1985*, G. Rozenberg, Ed. *LNCS 222*, Berlin: Springer-Verlag, 1986, pp. 381-400.

[21] C. Ramchandani, “Analysis of Asynchronous Concurrent Systems by Timed Petri Nets,” Project MAC, Technical Report 120, Massachusetts Institute of Technology, Cambridge, February 1974.

[22] J. Sifakis, “Performance Evaluation of Systems using Nets,” in *Net Theory and Applications*, W. Brauer, Ed. *LNCS 84*, Berlin: Springer-Verlag, 1980, pp. 307-319.

[23] E. Stoy and Z. Peng, “An Integrated Modelling Technique for Hardware/Software Systems,” in *Proc. ISCAS*, 1994, pp. 399-402.

[24] W. M. Zuberek and I. Bluemke, “Hierarchies of Place/Transitions Refinements in Petri Nets,” in *Proc. Conference on Emerging on Technologies and Factory Automation*, 1996, pp. 355-360.