in *Proc. NORCHIP Conference*, 1999, pp. 248-255.

A Petri Net based Model for Heterogeneous Embedded Systems

Luis Alejandro Cortés, Petru Eles and Zebo Peng
Department of Computer and Information Science
Linköping University, Linköping, Sweden
Phone: +46 13 281763 Fax: +46 13 282666
E-mail: {luico, petel, zebpe}@ida.liu.se

ABSTRACT

Design of embedded systems must be based on formal representations so that the synthesis process can be carried out systematically. We present PRES, a Petri net based model suited to embedded systems. It can represent several levels of detail using the feature of hierarchical decomposition. This model also includes an explicit notion of time. In PRES tokens hold information and transitions, when fired, perform transformation of data. Concurrency and sequential behavior might be naturally represented in PRES. The representation is formally defined and an example explains different concepts and the semantics of the model.

1. INTRODUCTION

Most modern 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 inherent 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. Design of hardware/software systems is a complex task. We advocate design cycles 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 should be founded upon a clear representation that allows to accomplish the design cycle, based on formal notation. Modeling is an essential issue of any systematic design methodology.

In this work, we propose a model suited to embedded systems. The model, called Petri net based Representation for Embedded Systems (PRES), is an extension to classical Petri nets. It explicitly captures time information, allows representation at different levels of granularity, and supports hierarchical decomposition. Another feature of this model is its expressiveness since the tokens might carry information. Concurrency and sequential behavior are also captured by PRES. As any Petri net based model, PRES is inherently asynchronous.

The rest of this paper is structured as follows. Section 2 addresses related work in the area of modeling for embedded systems, emphasizing on the extensions to Petri nets and approaches similar to PRES. The model is formally defined in Section 3. The semantics of the model is described in Section 4 through a simple example. Finally, some conclusions are outlined in Section 5.

2. RELATED WORK

Many models have been proposed to represent digital systems. These models encompass a broad range of styles, characteristics and application domains. Particularly in the field of hardware/software codesign, a variety of models has been developed and used for system representation.

Many of the computational models used for HW/SW systems are based on extensions to finite-state machines, Petri nets, discrete-event systems, data-flow graphs, communicating processes, among others.

2.1. Extensions to Petri Nets

Petri nets (PN) have been widely used for system modeling in many fields of science over three decades. We do not address here the basic concepts of PNs, but instead we concentrate in this subsection on the main extensions proposed. In our discussion we assume that the reader has a basic knowledge of PNs. [8], [9] are suggested for further reading on PN theory.

Two important intrinsic features of Petri nets are their concurrency and asynchronous nature. These features together with the generality of PNs and their flexibility have stimulated their applications in different areas. However, several drawbacks of the classical PN model have been pointed out along the years.

A major weakness of PNs is the so-called state explosion problem. Petri nets tend to become large even for relatively small systems. The lack of hierarchical decomposition makes it difficult to specify and understand complex systems using the conventional model. To overcome this disadvantage, the classical PN model has been extended introducing the concept of hierarchy [13], [1]. Single elements (transitions and places) may represent a more detailed structure.

The conventional PN model lacks the notion of time. However in many embedded applications time is a critical factor. Several extensions have been proposed in order to capture timing aspects: timed Petri nets [10], time Petri nets [7], and timed place-transition nets [11]. In the first approach, timed PNs, an execution time is associated with each transition, representing the finite duration of a firing. Unlike classical Petri nets, the transition is not instantaneous and the firing rule is modified to make a transition to be fired as soon as it is enabled. In time PNs two values of time are associated with each transition: the minimum and maximum time (starting from the moment the transition has been enabled) in which the transition has to fire, unless it is disabled by the firing of another transition. Finally, in timed place-transition nets, unlike the former cases, the time information is associated to places instead of transitions. The time parameter of each place has the meaning of a delay, so that a token must remain in the place a certain interval of time before it may be removed.

Classical Petri nets lack expressiveness for formulating computations as long as tokens are considered “black dots”. No value is transferred by communications, limiting the modeling power. Allowing tokens to carry information makes it possible to obtain more succinct representations suitable for practical applications. The extensions that include this new dimension to PNs are the so-called high-level Petri nets [4]. High-level PNs include predicate/transition nets and coloured Petri nets. The former introduce the concept of individuals with changing properties and relations [3]. Places (predicates) represent variable properties or relations of individuals, and transitions depict changes of those properties. Graphically, places and transitions are labeled with identifiers which define the net characteristics. Coloured Petri nets have been introduced in [5] and a strong mathematical theory has been built up around them. Transitions describe actions and tokens carry data values. The arcs between transitions/places have attached expressions that describe the behavior of the net. Coloured PNs permit hierarchical constructions. Although time is not explicitly defined in the model, computer tools developed around coloured PNs allow tokens to have time stamps during simulation.

2.2. Modeling Embedded Systems using Petri Nets

As stated before, many applications in distinct areas have successfully used PNs as a representation model. Due to their intrinsic characteristics and particular extensions to the conventional model, PNs might be an interesting representation for embedded systems. We address in

this section some known approaches to the modeling of such systems using Petri nets in the frame of HW/SW codesign. Stoy [12] presents a modeling technique for hardware/software systems. This Petri net representation is based on a parallel model with data-control notation and provides timing information. The model consists of two different but closely related parts: control unit and computational/data part. This representation allows to capture hardware and software in a consistent way. Maciel and Barros [6] use timed Petri nets as intermediate format for the partitioning process: an occam description constitutes the input of the design cycle and is translated into the proposed representation. Timed PNs, in this approach, are associated with dataflow augmented with time information. The definition of sub-nets permits handling hierarchies through special places called ports. A combination of time Petri nets and predicate/transition nets augmented with object-oriented concepts is utilized by Esser *et al.* [2]. Tokens carry data and transitions have associated functions, condition guards, and also time constraints.

3. PETRI NET BASED REPRESENTATION MODEL

In the following we present PRES, a Petri net based model, aimed to represent embedded systems. As mentioned before, it can be used to model a system at different levels of detail using the feature of hierarchical decomposition. The model includes an explicit notion of time. In PRES tokens hold information and transitions, when fired, perform transformation of data. Concurrency and sequential behavior are also naturally represented in PRES.

3.1. Basic Definitions

Definition 1. A *Petri Net based Representation for Embedded Systems* is a five-tuple $PRES = (P, T, I, O, M_0)$ 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, this structure is an *ordinary Petri net*, which means that there exist no multiple arcs, if any, from a place p_i to a transition t_j (or from a transition t_i to a place p_j). Additionally, P and T must be disjoint, i.e. $P \cap T = \varnothing$.

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

Definition 2. A *token* is a pair $k = (v_k, r_k)$ where
v_k 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, a record);
r_k is the *token time*, a finite positive real number representing the time stamp of the token.

Let K be the set of all possible token types for a given system.

Definition 3. A *marking* is a function $M : P \rightarrow \{0, 1\}$ that denotes the absence or presence of tokens in the places of the net.

For our purposes, we will only consider *bounded* Petri nets, i.e. nets where the number of tokens in each place does not exceed a finite number. Specifically, we aim to use structures which are *safe* or *1-bounded*. Since the intended Petri net in this model must be safe, this function M might also express the number of tokens in each place. We will say that a place p is *marked* if $M(p) = 1$. 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 is marked, k^p denotes the token present in p. Thus, the token value of the token in a marked place p will be v_{k^p}, and the token time of the token in p will be r_{k^p}.

Definition 4. The *type function* $\tau : P \rightarrow K$ is a relation that associates a place with a token type. Thus, we will call $\tau(p)$ the token type associated with place p.

It is worth to point 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.

3.2. Description of Systems

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

Definition 6. For every place in the post-set t° of a transition t, there exists an *output function* associated to t. Let us consider the transition t with its pre-set $^\circ t$ and post-set t°. Formally,

$$\forall p_j \in t^\circ \ \exists f_j : \tau(q_1) \times \tau(q_2) \times \ldots \times \tau(q_a) \rightarrow \tau(p_j)$$

with $^\circ t = \{q_1, q_2, \ldots, q_a\}$ and $t^\circ = \{p_1, p_2, \ldots, p_b\}$.

Output functions are very important when describing the behavior of the system to be modeled. They allow systems to be modeled at different levels of granularity with transitions representing simple arithmetic operations or complex algorithms.

Definition 7. For every output function associated to a transition t, there exists a *function delay fd*, a finite positive real number, which represents the execution time (delay) of that function. Formally,

$$\forall f_j \ \exists \ fd_j \in \Re^+$$

with $\Re^+$ the set of positive reals. If no function delay is explicitly defined, it will be assumed 0.

Definition 8. The *guard* G_t of a transition t is the set of boolean *conditions* that must be asserted in order to enable that transition, when all its input places hold tokens. A *condition* of a transition

$$cond_i : \tau(q_1) \times \tau(q_2) \times \ldots \times \tau(q_a) \rightarrow \{0, 1\}$$

is function of the token values in the places of the pre-set of t ($^\circ t = \{q_1, q_2, \ldots, q_a\}$).

The guard G_t of t is the conjunction of all conditions of that transition.There is no restriction in the number of conditions for a certain transition. If *all* conditions are asserted $G_t = 1$, otherwise $G_t = 0$. If no guard is explicitly defined, it will be assumed constantly asserted.

Definition 9. Every transition has a *functionality.* The functionality of a transition t is defined in terms of:

(i) Its *output functions*;
(ii) Its *function delays*;
(iii) Its *guard*.

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

3.3. Dynamic Behavior

Definition 10. 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 $t \in T$ is *enabled iff* (if and only if)

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

$$[\forall q_i \in {}^\circ t \;\; M(q_i) = 1] \wedge [\forall p_j \in t^\circ,\; p_j \notin {}^\circ t \;\; M(p_j) = 0] \wedge [G_t = 1]$$

If the transition t is enabled, we will note it as t^*. Then, the subset of enable transitions, for certain marking M, will be $T^* = \{t \in T | t^*\}$.

Definition 11. Every enabled transition t^* has a *trigger time* tt^* that represents the time instant at which the transition may fire. Each token in the pre-set of an enabled transition has, in general, a different token time. From the point of view of time, the transition could not fire before the tokens are ready. The concept of trigger time is needed to describe how token times are handled when the transition is fired. The trigger time of an enabled transition is the maximum token time of the tokens in its input places,

$$tt^* = max(r_{k^{q_1}}, r_{k^{q_2}}, \ldots, r_{k^{q_a}})$$

where the pre-set of t^* is ${}^\circ t = \{q_1, q_2, \ldots, q_a\}$.

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

Definition 12. The *firing* of an enabled transition changes the marking M into a new marking M^+. As a result of firing the transition t (with pre-set ${}^\circ t = \{q_1, q_2, \ldots, q_a\}$ and post-set $t^\circ = \{p_1, p_2, \ldots, p_b\}$), next events occur simultaneously:

(i) Tokens from its pre-set are removed;

$$\forall q_i \in {}^\circ t \;\; M^+(q_i) = 0$$

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

$$\forall p_j \in t^\circ \;\; M^+(p_j) = 1$$

(iii) Each new token deposited in t° has a token value, which is calculated evaluating the respective output function with the token values of ${}^\circ t$;

$$\forall p_j \in t^\circ \quad v_{k^{p_j}} = f_j(v_{k^{q_1}}, v_{k^{q_2}}, \ldots, v_{k^{q_a}})$$

(iv) Each new token added to t° has a token time, which is the sum of the respective function delay and the trigger time of the transition;

$$\forall p_j \in t^\circ \quad r_{k^{p_j}} = fd_j + tt^*$$

Note that only enabled transitions may fire. The execution time of the functionality of that transition is considered in the time tag of the new tokens.

3.4. Hierarchy

Definition 13. A transition t and its surrounding places (input and output places) can be substituted by a Petri net structure with the same functionality. Similarly, a Petri net structure can be abstracted to a single transition keeping the same net functionality.

This feature allows several levels of granularity as long as transitions might be refined and a more detailed representation may be gotten. Higher levels of abstraction can also be modeled through hierarchical composition.

4. EXAMPLE

This example will not show all the power of the model and its capabilities but rather explain the different definitions aforementioned. The purpose of this very simple net is to illustrate the semantics of our model. The net represents a multiplier which takes two positive integers and produces as output the result of multiplying those numbers. It implements a simple algorithm of iterative sums.

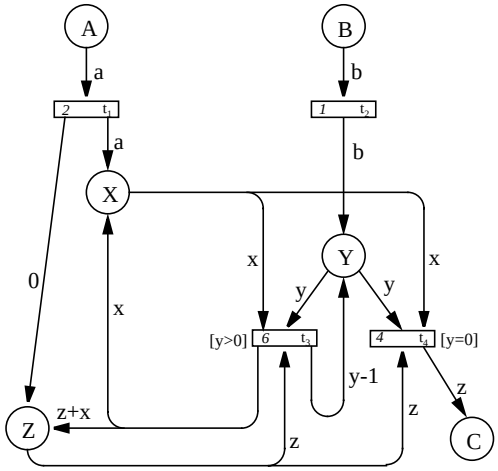
The PRES model of this multiplier is shown in Figure 1. We also show the C description corresponding to this algorithm. Like in classical Petri nets, places are graphically represented by circles, transitions by boxes, and arcs by arrows. In this example $P = \{A, B, X, Y, Z, C\}$ and $T = \{t_1, t_2, t_3, t_4\}$. In this particular case, we consider that the function delays for a given transition are the same, so we can call it transition time *rt* and it is inscribed to the left of transition boxes. We have borrowed notation from Coloured Petri nets [5] to graphically

express output functions and guards. We use inscriptions on the arcs: given a transition, its output functions (inscribed on output arcs) are captured as expressions in terms of the variables written on its input arcs. Guards are enclosed in square brackets and are also functions of the variables on input arcs.

```
int mult(int a,int b)
{
   int x,y,z;
   x=a;
   y=b;
   z=0;
   while (y>0) {
      z=z+x;
      y=y-1;
   }
   return z;
}


c=mult(a,b);
```

(a)



(b)

Figure 1. Multiplier: (a) algorithm; (b) PRES.

Figure 2 shows the behavior of the net for a initial marking M_0 ($M_0(A) = M_0(B) = 1$). Marked places are shaded and enabled transitions are highlighted using thicker lines. Token information is also shown in marked places. In this particular example, all places are associated with a token type integer, which means that, when marked, a place will hold a token whose value is of type integer. To explain the dynamic behavior of this net, we will assume that when several transitions are enabled simultaneously, the one that has minimum trigger time will fire in the next step. If the time trigger of two or more enabled transitions is the same, any of them may fire (one at each step). Let us assume, for the initial marking, $k^A = (5, 0)$ and $k^B = (2, 0)$. Initially, transitions t_1 and t_2 are enabled and both have trigger time $tt_1^* = tt_2^* = 0$. Then either t_1 or t_2 may fire.

Firing t_1 produces the marking shown in Fig. 2(b), where $k^X = (5, 2)$, $k^Z = (0, 2)$ and $k^B = (2, 0)$. Value and time of the new tokens are calculated following Definition 12. It is easy to see that, for this particular system, firing transitions with equal trigger time in any order does not affect the final result.

Fig. 2(c) illustrates some interesting aspects of PRES. Even if each place in the pre-set of t_4 has a token, the transition is not enabled because its guard is not asserted. For the marking in this figure, t_3 is the only enabled transition so it will be fired in the next step. Looking at the token time of tokens in ${}^\circ t_3$, we note that they have different time stamps ($r_{k^X} = r_{k^Z} = 2$, $r_{k^Y} = 1$). Hence, t_3 may not fire before $tt_3^* = max(r_{k^X}, r_{k^Y}, r_{k^Z}) = 2$.

After t_3 fires the marking changes into the one shown in Fig. 2(d). Let us analyze, for instance, the new token in Z, $k^Z = (5, 8)$. The arc (t_3, Z) has the inscription “z+x”, so that the token value in Z is calculated adding the previous token values in places X and Z ($v_{k^Z} = v_{k^Z} + v_{k^X} = 0 + 5 = 5$). The token time in Z is determined as the sum of transition time and trigger time of t_3, $r_{k^Z} = rt_3 + tt_3^* = 6 + 2 = 8$.

Finally, Fig. 2(f) shows the output result of the multiplication (10) and the token time shows the total time needed for the operation (18 time units). The net is not live in this configuration because it is not possible to fire any transition. If this multiplier is part of a larger system, the token in place C will likely be consumed and new tokens will be added to A and B, allowing the net to perform its function again.

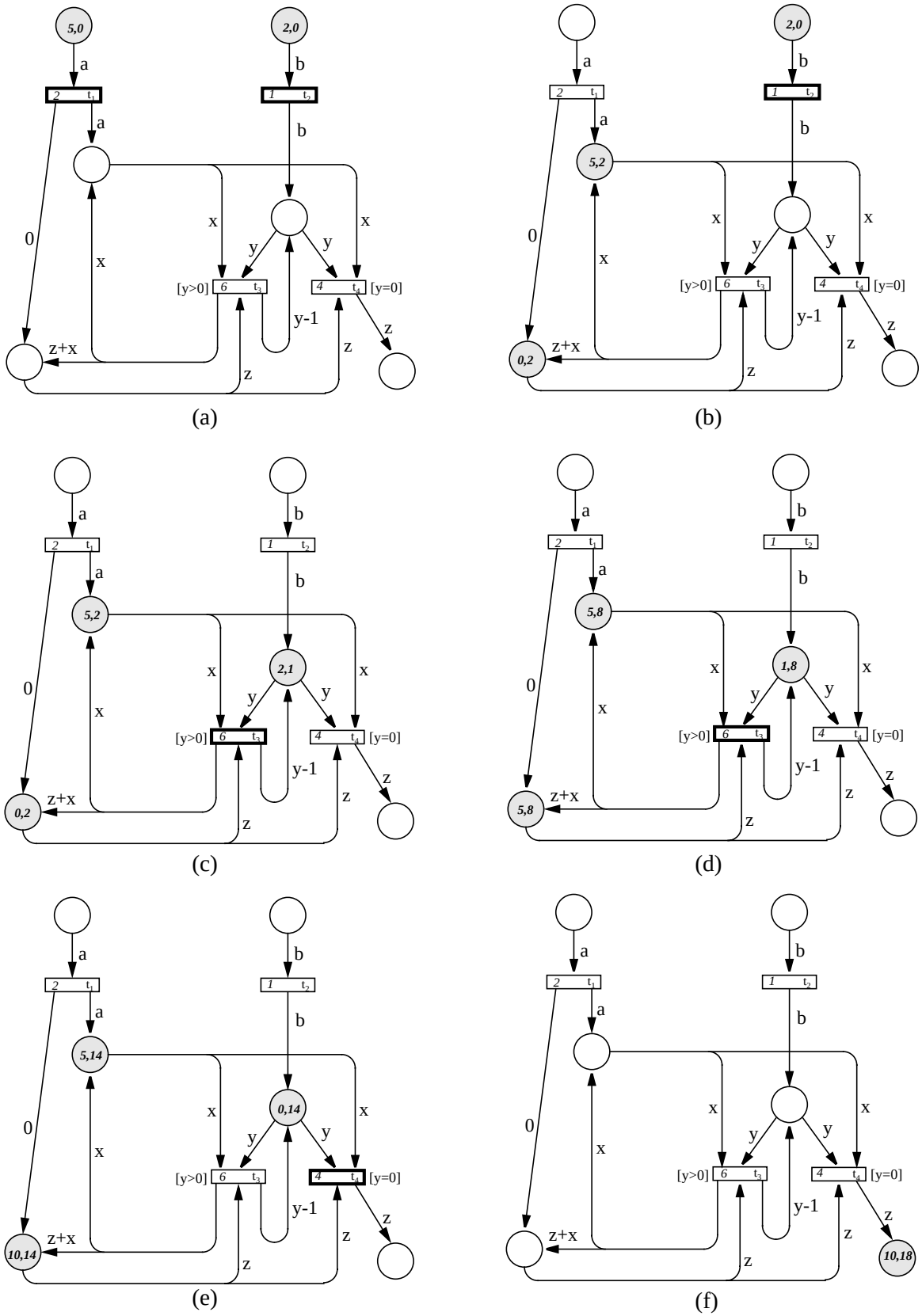


Figure 2. Dynamic Behavior of the Multiplier.

The hierarchy concept is shown in Figure 3. The original net may be abstracted to a higher level net with a single transition. These nets are equivalent. Similarly, transitions may be refined to obtain a more detailed representation of the system.

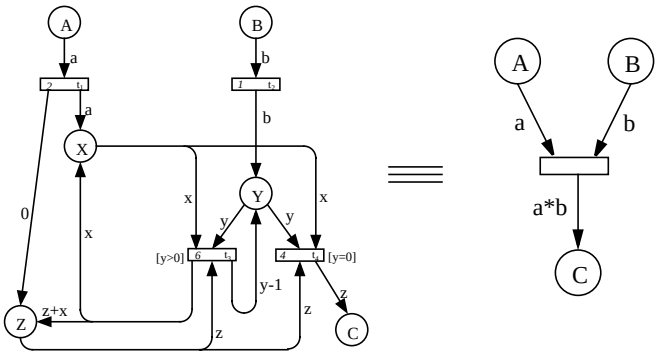


Figure 3. Equivalent PRES Structures.

5. CONCLUSIONS AND FUTURE WORK

We have presented a design representation for embedded systems. It allows to capture relevant information characteristic of such systems. We also presented an example in order to illustrate the modeling capabilities of this representation.

The basic concepts and definitions of our representation have been addressed in this work. Further applications and design methodologies based on this model will be presented later. In the future we will use this representation to develop a formal approach to specification, verification and transformation-based synthesis of heterogeneous electronic systems.

REFERENCES

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

[2] 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.

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

[4] K. Jensen and G. Rozenberg, Eds. *High-level Petri Nets*. Berlin: Springer-Verlag, 1991.

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

[6] P. Maciel and E. Barros, "Capturing Time Constraints by Using Petri Nets in the Context of Hardware/Software Codesign," in *Proc. Intl. Workshop on Rapid System Prototyping*, 1996, pp. 36-41.

[7] 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.

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

[9] J. Peterson, *Petri Net Theory and the Modeling of Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

[10] C. Ramchandani, "Analysis of asynchronous concurrent systems by timed Petri nets," Project MAC, Technical Report 120, Massachusetts Institute of Technology, Cambridge, Feb. 1974.

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

[12] E. Stoy, "A Petri Net Based Unified Representation for Hardware/Software Co-Design," Licentiate Thesis, Dept. of Computer and Information Science, Linköping University, Linköping, 1995.

[13] 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.