

Formal Coverification of Embedded Systems using Model Checking

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

The complexity of embedded systems is increasing rapidly. In consequence, new verification techniques that overcome the limitations of traditional methods and are suitable for hardware/software systems are needed. In this paper we introduce a computational model for embedded systems based on Petri nets, called PRES. We present an approach to coverification of both the hardware and software parts of an embedded system represented by PRES. We use symbolic model checking to prove the correctness of such systems, specifying properties in CTL and verifying whether they are satisfied. This coverification method permits to reason formally about design properties as well as timing requirements. A medical monitoring system illustrates the feasibility of our approach on practical applications.

1. Introduction

Current electronic systems are typically constituted of application-specific hardware components and software running on programmable platforms. The inherent heterogeneity of this kind of systems makes them very complex and difficult to verify. Moreover, the increasing demand on high-performance products has boosted the levels of sophistication of such systems.

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. First, these techniques may cover just a small fraction of the system behavior. Second, long simulation times and 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 designs. They might overcome some of the limitations of traditional validation methods. At the same time, formal verification can give a better understanding of the system behavior, help to uncover ambiguities, and reveal new insights of the system.

Formal methods have been extensively used in software development [10] and hardware verification as well [14].

However, they are not commonplace in embedded systems design. There is a lack of techniques for formal verification of hardware/software systems. In this paper we present an approach to coverification of embedded systems using symbolic model checking, based on a Petri net representation. With this approach it is possible to validate properties of the system as well as timing requirements. Design properties are specified as CTL (Computation Tree Logic) formulas and the model checker determines whether they are satisfied. We model the system using PRES (Petri net based Representation for Embedded Systems), a notation capable of capturing relevant information characteristic to embedded systems.

The rest of this paper is organized as follows. Section 2 addresses related approaches to modeling of embedded systems using Petri nets as well as formal methods suitable for HW/SW codesign. Section 3 describes shortly the main features of symbolic model checking. The underlying computational model that we use to represent embedded systems is introduced in Section 4. In Section 5 we present our approach to formal coverification. Section 6 shows the verification of a medical monitoring system. Finally, some conclusions are drawn in Section 7.

2. Related Work

The increasing complexity of embedded systems poses a challenge in verifying their correctness. Some verification approaches, suitable to hardware/software systems, have been proposed recently. Alur *et. al* [1] present a model checking procedure based on the Hybrid Automata model: given a system represented as communicating machines with real-valued variables, the method shows whether an ICTL-formula (Integrator Computation Tree Logic), specifying system requirements, is satisfied. Using the same model, Hybrid Automata, another coverification method is proposed in [12], where complex systems can be analyzed using a simplification strategy to verify individually the hardware, the software and the interface. Balarin *et. al* [2] introduce a verification methodology based on Codesign Finite State Machines (CFSMs), in which CFSMs are translated into traditional state automata. This technique checks if all possible sequences of inputs and outputs of the system satisfy the desired properties. To do so, those sequences that

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

meet the requirements constitute the language of another automaton, reducing the problem to the verification of language containment between automata. In [11], a partitioned system, described using a Pascal-like language, is the input to the proposed coverification framework in which CTL and TCTL formulas are evaluated in order to check behavioral and timing properties. An approach to symbolic model checking of process networks and related models is proposed in [19], where IDD (Interval Decision Diagrams) are used to represent multi-valued functions.

On the other hand, related work in the area of Petri nets (PNs) includes [20], which presents a BDD-based model checker for safe nets. Although the approach is intended to verify Petri nets in general, with no particular interest in embedded systems and without dealing with time information, it studies different forms of describing PNs using the SMV system [17], developed at Carnegie Mellon University. An interesting approach used for analysis and verification of bounded Petri nets is presented in [16]. Using the efficiency of BDDs to represent sets of markings and reduction rules to transform PNs, this technique can be used for reachability analyses and verification of some properties of PNs with large state spaces.

Many models have been proposed to represent HW/SW systems. Particularly Petri nets have been extended to capture significant information of such systems. Maciel *et. al* [15] introduce an intermediate model for hardware/software codesign, extending Petri nets to analyze certain properties used in the partitioning process. Stoy [18] presents a modeling technique based on PNs notation, where timed Petri nets with restricted transition rules are used to represent control flow in both hardware and software.

3. Symbolic Model Checking

Model checking is an approach to formal verification used to determine whether the model of a system satisfies certain required properties. Clarke *et. al* [6] introduced a model checking algorithm for formulas specified in the temporal logic CTL (Computation Tree Logic). CTL is based on propositional logic of branching time, that is, a logic where time may split into more than one possible future using a discrete model of time. Formulas in CTL are composed of atomic propositions, boolean connectors, and temporal operators. Temporal operators consist of forward-time operators (**G** globally, **F** in the future, **X** next time, and **U** until) preceded by a path quantifier (**A** all computation paths, and **E** some computation path). Thus formulas may describe properties of computation paths over labeled state-transition structures. This algorithm, however, requires the entire state transition graph to be constructed, causing a serious state explosion problem.

One way to overcome the state explosion is to represent symbolically the transition relation instead of explicit enumeration. A compact and efficient form of representing

boolean formulas and transition relations is using ordered binary decision diagrams (BDDs). BDDs are canonical representations that make boolean manipulations much simpler computationally [3]. Symbolic model checking [4] makes use of BDDs to represent sets of states and the transition relation, and the algorithm employs fixed-point techniques that manipulate sets using their characteristic functions encoded as BDDs. Therefore, it is possible to reason about designs with large state spaces without constructing the state graph of the system. SMV [17] is one of the available tools that uses the BDD-based symbolic model checking algorithm. This model checker has an input language that allows to describe systems using boolean, scalar or fixed-array data types, and boolean and basic scalar operations. The CTL formulas to be checked are also specified in the SMV language and may express liveness or deadlock-freedom, safety, and fairness, among other properties. In our experiments we have used the SMV tool to verify the correctness of designs represented in PRES.

4. Petri Net based Model

One of the issues in applying formal methods to embedded systems design is the underlying computational model. This model has to be formally defined in order to allow reasoning about the properties of the system that it represents. The notation we use to model such systems is PRES (Petri net based Representation for Embedded Systems). PRES is an extension to Petri nets which allows to capture important features of HW/SW systems. Some of the characteristics of this model will be illustrated using the example shown in Figure 1. The net represents a patient monitoring system as introduced in [9] and studied in [5].

The patient monitor measures physiological phenomena and analyzes this information. If the system detects abnormal conditions on the patient, it activates aural and visual alarms. The patient condition information is displayed and recorded as well. The functionality of the system can be captured as a set of processes. The *acquire* process reads information from the sensors. Usually this information contains spurious data that must be debugged. *filter* processes such data and eliminates false information received from the sensors. Once the information has been filtered, the processes that detect anomalous conditions on blood pressure, heart rate, or temperature may start depending on the data available. For instance, a possible anomaly in the blood pressure will activate the process *blood* in order to study the data. If, after analyzing the information, an irregular condition of the patient is encountered, the process *alarm* will be executed and an audio signal (process *audio*) will be triggered. The information resulted from the *filter* process is displayed on a screen and recorded by the processes *display* and *recorder* respectively. The specification of the patient monitoring system includes a timing constraint which states that the data from sensors must be sampled every 15 ms and

acquisition of new information requires the system to finish its functionality before the next execution.

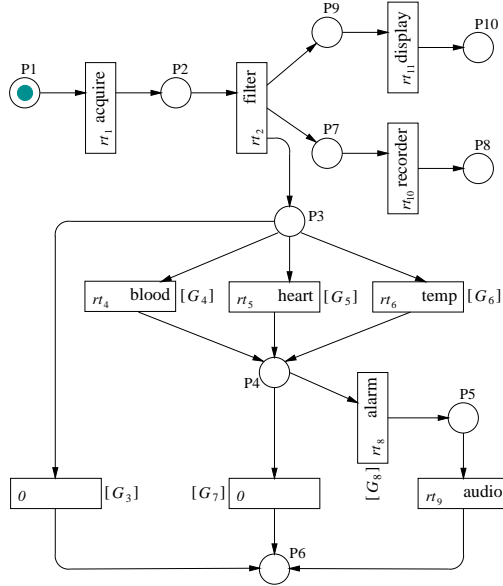


Figure 1. Medical monitoring system

In the following we briefly describe the computational model. A complete and formal definition of this representation can be found in [7]. A *Petri net based Representation for Embedded Systems* is constituted by a finite non-empty set P of *places*, a finite non-empty set T of *transitions*, a finite non-empty set I of *input arcs*, a finite non-empty set O of *output arcs*, and the initial *marking* M_0 of the net. Like in classical Petri nets, places are graphically represented by circles, transitions by boxes, and arcs by arrows. The medical monitoring system is modeled in PRES as shown in Figure 1, where the operations performed in the processes are captured by transitions and the data dependence between them is given by the structure of the net. The transitions have been named after the processes. A *marking* M is a function that denotes the absence or presence of tokens in places of the net. The model requires the net to be *safe* or *1-bounded*, i.e. no more than one token is allowed in a place. The marking M_0 , for the model of the monitoring system in Figure 1, shows $P1$ as the only place initially marked.

In PRES, a *token* is a pair $k = \langle v, r \rangle$ where v is the *token value* (this value may be of any type), and r is the *token time* (a non-negative real-valued time stamp). In this manner tokens themselves carry data and time information. There exists a *type function* τ that associates a *token type* to every place. This is the type of value that a token may bear in that place. The token type related to a certain place is an intrinsic property of that place and will not change during the dynamic behavior of the net.

For every transition t , there exists a *transition function* associated to t . Transition functions have as arguments to-

ken values of tokens in places of the pre-set¹ of the transition. Transition 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 being associated with simple arithmetic operations or complex algorithms. For instance, in Figure 1, there is one transition function associated to transition *filter*, which defines token values of new tokens in $P3$, $P7$ and $P9$, when *filter* is fired (executed). This function represents what has been earlier called the process *filter*.

For every transition t , there exist *minimum* and *maximum transition delays*, non-negative real numbers, which represent the lower and upper limits for the execution time (delay) of the function associated to that transition. In this paper we restrict ourselves to the case in which minimum and maximum transition delays are equal. Under the above assumption, in the example of Figure 1 such a time is captured as “transition delay” and is inscribed in the respective transition box. Thus rt_6 represents the execution time of the function associated to transition *temp* (in Section 6, where we study alternative implementations of the system, particular values will be assigned to transition delays).

Each transition t in the net may also have a *guard* G which represents a *condition* that must be satisfied in order to enable that transition, when all its input places hold tokens. Guards are functions of token values of tokens in the pre-set of a given transition. In Figure 1, for example, G_4 represents the condition that must be fulfilled to execute the process *blood*. In Figure 1 there are two transitions that have no name attached: we have introduced them in order to model the situation in which no abnormal condition on the patient is detected. The associated execution time is zero because there are no activities to be performed in this case.

In PRES, every transition has a *behavior*. The behavior of a transition t is defined in terms of its *transition function* and its *transition delay*. Intuitively, this behavior describes “what happens” when the transition 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 ${}^{\circ}t$ and depositing one token into each element of the post-set t° . These tokens, placed into t° , have values and time stamps which depend on the previous tokens in ${}^{\circ}t$ and the behavior of t . When a transition fires, all the tokens in its output places get the same token value and token time.

A transition t is said to be *enabled* if all places of its pre-set are marked, its output places different from the input ones² are empty, and its guard is asserted. It means that the enabling rule of classical PNs has been modified to fit the characteristics of this specific model. Every enabled transition has a *trigger time* tt^* that represents the time instant at

¹The pre-set ${}^{\circ}t$ of a transition t is the set of *input places* of t . Similarly, the post-set t° of a transition t is the set of *output places* of t .

²A place may be, at the same time, input and output of a transition.

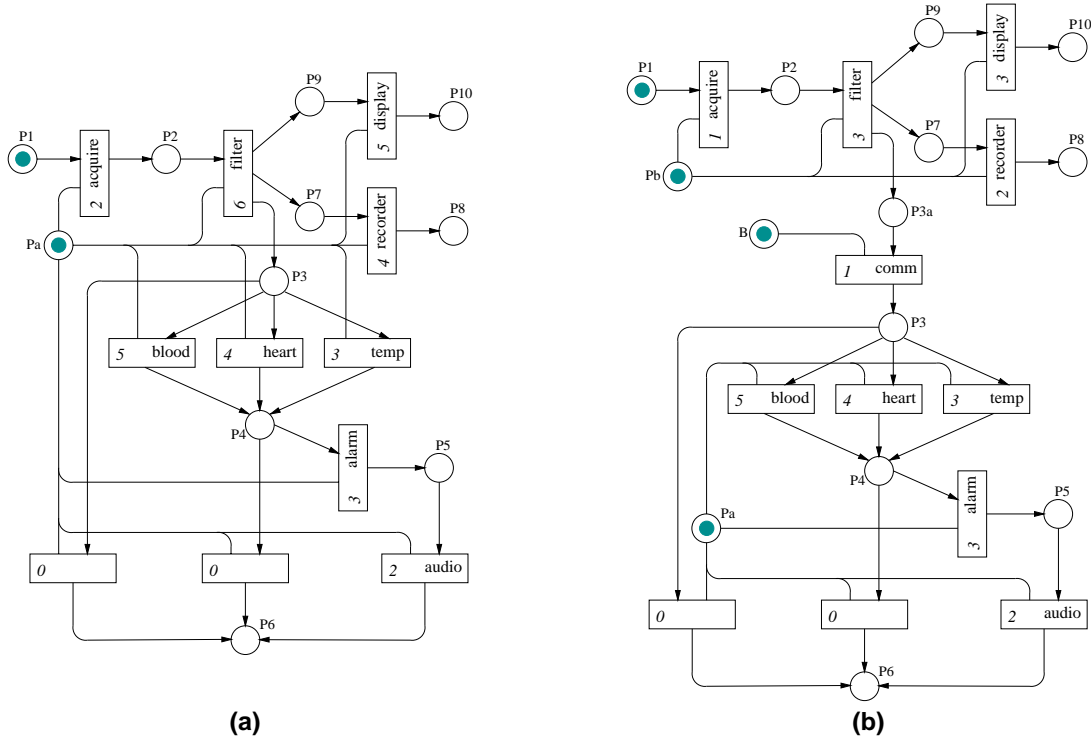


Figure 2. Different implementations of the patient monitoring system

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 all 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.

The *firing* of an enabled transition changes a marking M into a new marking M^t . As a result of firing a given transition t , the following events occur: tokens from its pre-set are removed; one token is added to each place of its post-set; each new token deposited in t° gets a token value, which is calculated by evaluating the transition function with the token values of tokens in ${}^\circ t$ as arguments; and each new token added to t° gets a token time, which is the sum of the transition delay and the trigger time of the transition. For instance, referring to Figure 2(a), suppose that *acquire* is fired and the system reaches the state in which Pa and $P2$ are the only marked places. Then, *filter* will be the only enabled transition in the net. When *filter* fires, tokens will be deposited in $P3$, $P7$, $P9$ and Pa (at the same time that tokens are removed from $P2$ and Pa) and their token values are calculated by evaluating the respective transition function with previous token values of $P2$ and Pa as arguments. These new tokens will have identical token time, that is, the sum of the trigger time (maximum token time of previous tokens in $P2$ and Pa) and the transition delay (6 time units).

There is an aspect worth to pinpoint regarding the timing semantics of the model: the time stamps of tokens capture the time elapsed since the starting instant of the system (assuming that all token times are zero in the initial marking); there is not such a concept like clock-on-the-wall that enforces a strict order in the firing of transitions³.

In summary, PRES is a Petri net based model with extensions to capture features of embedded systems: the model includes an explicit notion of time; tokens, in our notation, hold information and transitions—when fired—perform transformation of data; the representation also supports hierarchical decomposition. The reader is referred to [7] for a formal definition of PRES.

5. Coverification of Embedded Systems

The coverification method presented in this work is based on the model introduced in the previous section. The purpose of the approach presented in this paper is to reason about embedded systems using PRES as underlying representation. There are several types of analysis that can be performed on systems represented in PRES. A given marking, i.e. absence or presence of tokens in places of the net, may represent the state of the system at a certain moment in

³ A slight modification of PRES is proposed in [8] in order to have a strict temporal order in the firing of transitions.

the dynamic behavior of the net. Based on this, different properties can be studied. For instance, in a landing gear controller of an airplane, the wheel door must not close while the plane is landing—under any circumstance. This sort of safety requirement might be formally proven by checking that the places which represent such a dangerous state are never marked simultaneously. Sometimes, the designer could also be interested in proving that the system eventually reaches a certain state whose marking represents the completion of a task.

The kind of analysis described above, called *reachability analysis*, is very useful but says nothing about timing aspects nor does it deal with token values. In many embedded applications, however, time is an essential factor. Moreover, in hard real-time systems, where deadlines should not be missed, it is crucial to reason quantitatively about temporal properties to assure the correctness of the design. Therefore, it is needed not only to check that a certain state will eventually be reached but also to ensure that this will occur within some bound on time. In PRES, time information is attached to tokens, so that we can analyze quantitative timing properties: we may, for instance, prove that a given place will eventually be marked in the future and that its time stamp, for any possible condition, will be less than a certain time value that represents a temporal constraint. Such a study will be called *time analysis*.

A third type of analysis for systems modeled in PRES involves reasoning about values of tokens in marked places. This type of *behavior analysis* is not part of the coverification method proposed here. In this work we address just reachability and time analyses. In other words, we concentrate on the absence/presence of tokens in the places of the net and their time stamps, but we do not deal with the values of those tokens. We assume that transition functions (see Section 4) are correctly defined.

As it has been mentioned above, in a PRES model a place may hold at most one token for a certain marking. Thus it is possible to encode a marking—or a set of markings—as a boolean function where the variables correspond to places of the net. Boolean functions can be straightforwardly represented by BDDs. Firing a transition in a Petri net changes the marking into a new one, which is a variation in the state of the system. It is possible to build the BDD that represents the transition relation of the system and then compute efficiently the reachable states using BDDs [3], [13]. With such a BDD-based representation we can formally verify properties, specified in CTL, using symbolic model checking [4] and accomplish reachability analyses. In our experiments, we use the SMV tool (a BDD-based symbolic model checker) [17] and its input language to describe and verify systems modeled in PRES.

A program in SMV describes both the system and the specification (properties to verify). The system is described as a collection of “modules”. Each module may contain variables, its initial state, and assignments of variables for

the next state. A “process” is an instance of a module, in such a way that the model checker executes a step by choosing non-deterministically a process and then executing all assignment statements of that process in parallel.

To translate a PRES model into the SMV input language, we declare in the `main` module a boolean as well as an integer variable for each place of the net. The boolean variable represents absence/presence of tokens in that place, while the integer one represents the time stamp of the token when the particular place is marked (we restrict ourselves to integer token stamps). We instantiate each transition as a process that has as parameters its input and output places as well as time stamps of tokens in those places. In the `main` module we also define the initial marking of the net, assigning initial values to the variables that represent places and to time stamps of tokens in initially marked places.

We describe each transition of the Petri net as a module that adds/removes tokens (changes the marking) when it is executed (fires). Figure 5 illustrates the description of the *blood* process corresponding to the implementation shown in Figure 2(b). When a transition fires, it changes the marking of the system removing tokens from its input places and adding new tokens to output places. This is captured using `next` assignments for input/output places of the transition. Thus if the transition is enabled (`enabled := P3 & Pa & !P4`), execution of *blood* will assign boolean values to *P3*, *Pa* and *P4* according to the transition firing rules, that is, 1 to output places and 0 to input places.

As stated in the definition of PRES, time stamps of new tokens are calculated as the sum of the trigger time and the transition delay, e.g. `(trigger_time + tran_delay) mod 28`. In this case, we are using integer addition “modulo 28” because integer variables in SMV must be bounded when they are defined. The bound of variables for time stamps, in the example of Figure 2(b), is 27. This is an upper bound on the value of the time stamp that a token may have in the net (this is, of course, assuming that time stamps of all tokens in the initial marking are zero). We need estimation based procedures to calculate such an upper bound. These procedures must be accurate enough to assure that the performance of the verification method is not adversely affected by a too pessimistic estimation of the upper bound for token times. The larger the value of this bound, the longer is the computation time needed to verify timing properties. For instance, Table 1 shows the verification time for different upper bounds of time stamps for the very same patient monitoring system of Figure 2(b). These times have been obtained by running the SMV system on an UltraSPARC-III@440 MHz processor. The properties verified in these cases are explained in detail in Section 6. The complexity of this problem grows exponentially in the size of the bound for time stamps. This becomes a limitation of our approach when large time stamps are needed to characterize the timing aspects of a system.

In the situation in which the net representing the system does not have loops, as it is the case of the models of the medical system in Figure 2, an upper bound could be trivially determined summing all transition times. In the case of representations containing loops, we need procedures that consider the number of times a loop may execute in order to estimate such an upper bound.

Table 1. Coverification of the monitoring system

Upper Bound	Time [s]
27	1.7
55	12.7
111	98.0
223	1300.1

In PRES each token has two components: a value and a time stamp. Time stamps, according to the definition of the model, may be non-negative reals. However, the reader might have noted, that in order to use the SMV system we need to restrict the time to discrete (integer) values. Though this aspect indeed limits the modeling power of PRES, there exists a wide spectrum of systems that can be still represented adequately in PRES when time stamps, as well as function delays, are treated as integer values.

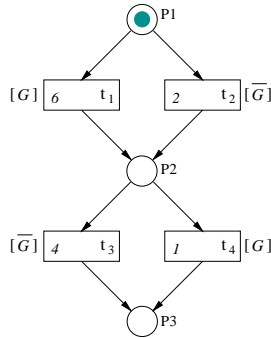


Figure 3. PRES model with dependent guards

Another important issue to point out is that, since we are not dealing with token values, only certain kind of systems that include guards in their models may be analyzed using this approach. On the other hand, models in which transitions bear no guard may be straightforwardly studied. Some systems modeled in PRES, which include guards, have the particular characteristic that there exist sets of “complementary” guards that are “independent” from one set to another. This aspect is best explained in reference to the monitoring system of Figure 1. For this example, G_3, G_4, G_5 and G_6 are complementary because whenever $P3$ is marked, only one of these guards will be asserted. Similarly, G_7 and G_8 are complementary. In this particular example, there is no

relation between the sets $\{G_3, G_4, G_5, G_6\}$ and $\{G_7, G_8\}$, which means that they are independent. For PRES models with independent sets of complementary guards, these guards can be ignored without affecting the reachability and time analyses. In this case, the model will exhibit non-determinism when firing transitions whose guards have been dropped.

To study further the issue of guards in our approach, consider the simple net of Figure 3. Suppose that we can express all guards in terms of the token value of the initial token in $P1$, and we can write G and \bar{G} to denote complementary guards, however, these sets are not independent. It is clear that ignoring the guards will alter the analysis results of the system. When we drop the guards and analyze the model, we find out that in the worst case $P3$ will get a token with time stamp $r_3 = 10$, whereas in reality the worst case will be a token in $P3$ with time stamp $r_3 = 7$.

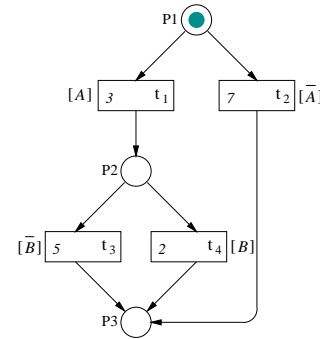


Figure 4. PRES model with independent guards

On the other hand, consider the model shown in Figure 4 with two independent sets $\{A, \bar{A}\}$ and $\{B, \bar{B}\}$ of complementary guards. If we ignore the guards in this case, the model will exhibit non-determinism but this will not affect the reachability and time analyses. For instance, for both cases (considering guards and dropping them), in the worst case $P3$ will get a token with time stamp $r_3 = 8$.

6. Coverification of the Medical Monitoring System

In this section we show the coverification of the medical monitoring application described above for two possible implementations of the system. This illustrates practically a transformational design space exploration methodology based on formal methods. We consider first, in Figure 2(a), an implementation using a single programmable processor. Note that values have been assigned to transition times. These times are the estimated worst case execution times of the respective functions on the selected processor. For example, the transition time for *heart* is 4 ms. The reader can also notice that, by reason of the considerations explained in

the previous section, the guards have been ignored. The place Pa (initially marked) models the processor: this place is both input and output of all transitions which capture processes mapped onto that processor. We use lines with no arrowheads to indicate this bidirectional flow relation between transitions and Pa . Place $P1$ models the data read from the sensors, $P8$ and $P10$ the information to be recorded and displayed respectively, and $P6$ is the indicator which shows that the analysis has been performed. $P1$, $P6$, $P8$, and $P10$ are the places through which the system interacts with its environment. Initially $P1$ and Pa are marked and time stamps for tokens in $P1$ and Pa are $r_1 = 0$ and $r_a = 0$. If the system operates properly, a new token will be added to $P1$ after the patient monitor finishes its functionality.

```

MODULE blood(P3,time_P3,Pa,time_Pa,P4,time_P4)
ASSIGN
  next(P3) := case
    enabled : 0;
    l : P3;
  esac;
  next(Pa) := case
    enabled : 1;
    l : Pa;
  esac;
  next(P4) := case
    enabled : 1;
    l : P4;
  esac;
  next(time_Pa) := case
    enabled : (trigger_time + tran_delay) mod 28;
    l : time_Pa;
  esac;
  next(time_P4) := case
    enabled : (trigger_time + tran_delay) mod 28;
    l : time_P4;
  esac;
DEFINE
  tran_delay := 5;
  trigger_time := case
    (time_P3 >= time_Pa) : time_P3;
    (time_Pa >= time_P3) : time_Pa;
  esac;
  enabled := P3 & Pa & !P4;
FAIRNESS running

```

Figure 5. Description of *blood* using SMV

We must first verify that the system, under any circumstance, will complete its functionality, that is, $P6$, $P8$, and $P10$ will eventually be marked. Using the SMV tool, this property can be expressed as a CTL formula preceded by the keyword SPEC:

```
SPEC AF (P6 & P8 & P10)
```

which reads “eventually ($P6$ & $P8$ & $P10$) for all computation paths”, i.e. the state in which $P6$, $P8$, and $P10$ are simultaneously marked is inevitable. This formula holds for both representations in Figure 2. The second property, which concerns our design, is the constraint of a maximum delay of 15 ms. We have to formally verify that when $P6$, $P8$ and $P10$ are marked (which has been shown to be true) the time stamps of tokens in these places are less than or equal 15. We may express this constraint in our description as three CTL formulas:

```
SPEC AF (P6 & (time_P6 <= 15))
SPEC AF (P8 & (time_P8 <= 15))
```

```
SPEC AF (P10 & (time_P10 <= 15))
```

All three formulas above turn out to be false, for the model in Figure 2(a), and SMV gives counter-examples. As the implementation in Figure 2(a) does not meet the time constraint, we consider an alternative architecture. Figure 2(b) models the patient monitoring system implemented using one programmable processor (represented by Pa) and one hardware component (Pb). Processes *acquire*, *filter*, *recorder* and *display* are mapped onto Pb while the other processes onto Pa . A new transition (*comm*) has been introduced in the model to consider the cost of inter-processor communication. This process *comm* is the only one that utilizes bus resources (place B). Note that execution times of processes mapped onto Pb have changed with respect to the previous design alternative. For the model in Figure 2(b) we have formally verified, using symbolic model checking through the SMV system, that the properties mentioned above do hold for all possible situations. This implementation has superior performance because of parallelism and lower execution times for the hardware.

7. Conclusions

We have presented PRES, a Petri net based model with extensions to capture important features of embedded systems. The model is simple, intuitive and can be easily handled by the designer. We introduced an approach to formal verification of embedded systems using symbolic model checking with PRES as underlying computational model. Thus, coverification is possible dealing with timing properties.

It has been shown how PRES models can be translated into the input formalism of a model checker in a relatively simple manner. A patient monitoring system has been studied to illustrate the applicability of the coverification approach to practical systems. Transformations during design space exploration can be smoothly captured in PRES and properties to be checked can be derived directly from the model in an easy manner.

Our main contribution lies in modeling embedded systems in such a way that the representation is adequate to be analyzed using formal methods. The model that we use is a Petri net based notation in which tokens bear both value and time stamp. We address in this paper a coverification method that allows to reason formally about the presence/absence of tokens in places of the net and their time stamps, but we do not deal with their token values. This is a problem worth for further research.

References

- [1] R. Alur, T. A. Henzinger, and P.-H. Ho, “Automatic Symbolic Verification of Embedded Systems,” in *IEEE Trans. Software Engineering*, vol. 22, pp. 181-201, March 1996.

- [2] F. Balarin, H. Hsieh, A. Jurecska, L. Lavagno, and A. Sangiovanni-Vincentelli, "Formal Verification of Embedded Systems based on CFSM Networks," in *Proc. DAC*, 1996, pp. 568-571.
- [3] R. E. Bryant, "Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams," in *ACM Computing Surveys*, vol. 24, pp. 293-318, Sept. 1992.
- [4] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill, "Symbolic Model Checking for Sequential Circuit Verification," in *IEEE Trans. CAD of Integrated Circuits and Systems*, vol. 13, pp. 401-424, April 1994.
- [5] S. Campos, E. M. Clarke, W. Marrero, and M. Minea, "Timing Analysis of Industrial Real-Time Systems," in *Proc. Workshop on Industrial-Strength Formal Specification Techniques*, 1995, pp. 97-107.
- [6] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications," in *ACM Trans. on Programming Languages and Systems*, vol. 8, pp. 244-263, April 1986.
- [7] 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.
- [8] L. A. Cortés, P. Eles, and Z. Peng, "Verification of Embedded Systems using a Petri Net based Representation," to appear in *Proc. Intl. Symposium on System Synthesis*, 2000.
- [9] P. J. Drongowski, "Software architecture in realtime systems," in *Proc. Workshop on Real-Time Applications*, 1993, pp. 198-203.
- [10] J. D. Gannon, J. M. Purtilo, and M. V. Zelkowitz, *Software Specification: A Comparison of Formal Methods*. Norwood, NJ: Ablex Publishing, 1994.
- [11] E. H. A. Garcez and W. Rosenstiel, "CVF - Coverification Framework," in *Proc. Brazilian Symposium on Integrated Circuit Design*, 1998, pp. 103-106.
- [12] P.-A. Hsiung, "Hardware-Software Coverification of Concurrent Embedded Real-Time Systems," in *Proc. Euromicro RTS*, 1999, pp. 216-223.
- [13] A. J. Hu, "Formal Hardware Verification with BDDs: An Introduction," in *Proc. Pacific Rim Conference on Communications, Computers and Signal Processing*, 1997, pp. 677-682.
- [14] C. Kern and M. R. Greenstreet, "Formal Verification in Hardware Design: A Survey," in *ACM Trans. on Design Automation of Electronic Systems*, vol. 4, pp. 123-193, April 1999.
- [15] 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.
- [16] E. Pastor, O. Roig, J. Cortadella, and R. M. Badia, "Petri Net Analysis Using Boolean Manipulation," in *Application and Theory of Petri Nets 1994*, R. Valette, Ed. LNCS 815, Berlin: Springer-Verlag, 1994, pp. 416-435.
- [17] The SMV System, <http://www.cs.cmu.edu/~mod-elcheck/smv.html>
- [18] 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.
- [19] K. Strehl and L. Thiele, "Symbolic Model Checking of Process Networks Using Interval Diagrams Techniques," in *Proc. IC-CAD*, 1998, pp. 686-692.
- [20] G. Wimmel, "A BDD-based Model Checker for the PEP Tool," Major Individual Project Report, Dept. of Computing Science, University of Newcastle, Newcastle, May 1997.