ON THE USE OF PETRI NETS FOR THE DESCRIPTION OF DIGITAL SYSTEMS

NORIAN MARRANGHELLO¹, WAGNER L. A. DE OLIVEIRA², AND FURIO DAMIANI²

¹ DCCE/IBILCE/UNESP - 15054-000 Rio Preto, Brazil
norian@dcce.ibilce.unesp.br
² DSFI/FEEC/UNICAMP - 13081-970 Campinas, Brazil
{wagner, furio}@dsif.unicamp.br

Abstract — Due to their complexity, nowadays it is virtually inconceivable to design and implement large digital systems without the use of computer-aided design tools. Many petri net extensions have been proposed aiming at describing hardware characteristics as accurately as possible. Among all petri net extensions developed for use with digital systems, only two of them have nearly all characteristics needed to describe such systems in full. They are place chart nets, and petri nets for embedded systems. Using the latter as an example we discuss some issues that may improve the capability of petri nets in dealing with digital systems.

Keywords — Petri Nets; Digital Systems Synthesis; Hardware/Software Codesign.

1 Introduction

Computer-aided design (CAD) tools are not capable of solving problems at the physical level. One has to use an abstract model to represent the problem with a certain degree of confidence transforming the physical problem to the virtual domain. Then the model is analyzed, simulated, and adjusted. Finally, the required system is synthesized from the abstract model, when directives are issued to implement the system (at the physical level).

The effectiveness of CAD tools relies on the languages for the description of the models on which the algorithms operate. Particularly, the problem of designing hardware and software from the same initial specification such that the resulting system present the best possible performance for the intended application has been of great concern to the research community. The optimized performance should be achieved by adequately mapping each function of the system to a hardware or a software component. Therefore, according to Gajski et al. (1997), regarding hardware/software codesign of digital systems a good model must have the following characteristics:
- to represent the functionality of the system in terms of transitions between states;
- to hierarchically represent the behavior of the system;
- to express concurrency existing in the system;
- to represent exceptions in the system;
- to represent asynchronous behaviors;
- to deal with software routines; and
- to represent time.

Researchers have shown (Dittrich, 1995) that some petri net classes are appropriate to modeling complex systems. However, some codesign experts are reluctant to use petri nets in their work. Apparently the reason for such a reluctant behavior relies on the following two facts:
- The first reason would be that these researchers have built their codesign systems based on algorithms and methods that they used in previously made tools, especially those for high-level synthesis. The use of results from former works is a very natural inclination, but cannot be considered as an indicative of the inefficiency of other methods.
- The second reason would be that as a consequence of the use of other methods in their research the knowledge some of those researchers have about petri nets might be limited. Thus, the use of high-level petri nets, more suitable for the representation of hardware/software codesign systems, is not mentioned in any work. All conclusions drawn by those researchers concerning the use of petri nets are based on ordinary (low-level) petri nets, and not properly extended to all classes of petri nets.

The adequate understanding of petri nets by the codesign research community is increasing. Many authors have presented works applying petri nets for codesign, e.g., Erik Stoy (1995), Ricardo Machado et al. (2000), Paulo Maciel et al. (1997), and Jaros³aw Mirkowski and Norian Marranghello (1998).

An example of the usefulness of petri nets to the design and synthesis of digital systems is the tool PETRIFY, developed by a group of European researchers. This tool aids the designer in the process of synthesis and optimization of asynchronous circuits. PETRIFY accepts functional descriptions based on petri nets, signal transition graphs or transition systems. Internally this initial description is transformed either in a petri net or in
a signal transition graph (whichever results in the simpler description) and produces a netlist of an asynchronous controller in the target library, preserving the behavior between input and output of the system. Detailed information on this asynchronous circuit design tool can be found at the URL http://www.ac.upc.es/~vlsi/petrify, of the University of Catalunya, Spain.

As previously mentioned, PETRIFY is cited here only as an example of the usefulness of petri nets to the design and synthesis of digital systems. However, the petri net model used in this tool has the same difficulty in dealing with exceptions as almost all other models used for the representation of concurrent systems. Only recently two petri net extensions capable of dealing with this problem were proposed, they are, petri nets for embedded systems (PNES) (Mirkowski and Yakovlev, 1998) and place chart nets (PCN) (Kishinevsky et al. 1997). While in the former binary tokens are used to deal with the problem at a low level, in the latter no programming constructs are allowed. So far neither model has been properly elaborated.

The purpose of this paper is to discuss some issues related to the description of digital systems by petri nets. To reach this goal we introduce some existing problems using as an example a system described by petri nets for embedded systems. These problems are elaborated and possible solutions are suggested in the final section.

2 Informal description of petri nets for embedded systems

In this section an informal view of petri nets for embedded systems is presented. To begin with we would classify them as ordinary petri nets in the sense that they use boolean tokens, i.e., tokens that can only represent the existence or not of certain datum. However, they incorporate many characteristics usually associated to high-level nets such as hierarchy.

As any other petri net, the notions of state, action, flow relation and token are present in PNES. Nevertheless, contrary to other petri nets, PNES include the notion of modes. A PNES mode is an association of an ordinary PN place to a binary marking, a possibly empty set of programming statements, and a (possibly null) finite time representing the execution time of the statements. They also include a different kind of transition called exception by the authors. Exceptions are associated to non-trivial guards, meaning that there is a mapping of every exception onto a set of boolean conditions defined for the net. Moreover, a set of extended arcs is defined in such a way to allow arcs to connect modes to transitions as well as exceptions.

When hierarchy is considered a supermode comes into play. It is a mode to which a subnet is associated. The entry mode of the subnet has the same input set as the supermode, and the termination mode of the subnet has the same output set of the supermode.

Some of the features mentioned in the introductory section are inherent to PNES as they are an extension of petri nets. For instance, by understanding the functionality of a system as sets of conditions that must be met to enable events which will in turn establish new sets of conditions that will enable new events, and so forth, all petri net extensions (including PNES) represent the functionality of a system in terms of transitions (events) between states (conditions).

Figure-1 shows a PNES model of a simplified view of a processing element (PE) used in a multiprocessor architecture dedicated to circuit simulation (Marranghello, 2000). This multiprocessor is a mesh-connected machine in which the interconnection pattern of several PEs is configured on the fly to match the topology of the circuit to be simulated. Each PE is assigned a circuit element prior to simulation. During simulation each PE will compute the corresponding element's model response to some input stimuli and broadcast the results to other communicating PEs. When calculated results fall within some predetermined error bound the corresponding PE signals the host and continues its computation.

Figure-2 shows the host model for this example. The PEs communicate with each other within the supermode displayed as an oval and labeled as run array of PEs. The computation within each PE is independent from one another. They recompute the assigned model with the data available at the input, be they updated or not. Thus, the operation of the PEs is concurrent and asynchronous. Furthermore, regarding PNES, supermodes are abstractions of lower level sub-nets. When a supermode is entered several instances of the PE model (as the one depicted in Figure-1) are activated. Thus, Figures-1 and -2 combined show the hierarchy available in PNES.

Two kinds of exceptions may occur in a digital system, they are: interrupts and preemptions. Interrupts can be defined as the suspension of some activity by an event, and the resumption of the same activity later on from the state in which it was suspended. Preemptions can be defined as the suspension of some activity by an event, and either its resumption from the starting state or no resumption at all. Preemption is tackled by means of exception transitions, as illustrated in Figure-2, and explained in the sequel. An exception can fire when at least one of its input modes is marked and its guard condition is evaluated to true. In Figure-2 the exception is stability reached and its guard is All_Stable, initially set to false. Whenever a PE
finds its results stable it votes for the stability of the system by sending a signal to the host. If all PEs vote for stability variable All_Stable is set to true. When this occurs the exception stability reached fires removing all tokens from all active instances of the PE model, whichever the state they may be in. After further processing the PEs may be restarted from their initial state. No mechanism for interrupt handling by PNES has been explicitly reported in the literature (Mirkowski and Yakovlev, 1998). However, by using guard in the transitions one can (so to speak) switch them on or off by setting the value of appropriate variables to true or false. Thus, a process can interrupt another by setting to false a variable in the guard of the required transitions, preventing them from firing. When needed the value of such a variable can be reset to true allowing the corresponding transitions to fire when enabled, and in this way restoring the state of the interrupted process.

When done logically, the work permit can for instance be represented by a variable that guards every action in a given portion of the net. The normal working condition of the net would be when the guarding variable is set to true. When there is an interrupt request the variable is set to false blocking the usual net processing, and allowing the interrupting process to take over. Once the interrupting process finishes, the guard variable is reset to true, and processing of the net resumes from where it was prior to interruption.

When done structurally the working permit materializes as a self-loop controller place in the net. Such a place must be input to all transitions representing the process that may be interrupted. The controlling place is filled with a token when the net is started. If any other process in the net needs to interrupt the controlled transitions, it takes the token from the appropriate self-loop controller place. Thus, the corresponding actions are blocked. When the interrupting process concludes its computation, it returns the token to the proper controller place, and the interrupted transitions are allowed to resume normal processing from the state they were prior to interruption.

When done structurally the working permit materializes as a self-loop controller place in the net. Such a place must be input to all transitions representing the process that may be interrupted. The controlling place is filled with a token when the net is started. If any other process in the net needs to interrupt the controlled transitions, it takes the token from the appropriate self-loop controller place. Thus, the corresponding actions are blocked. When the interrupting process concludes its computation, it returns the token to the proper controller place, and the interrupted transitions are allowed to resume normal processing from the state they were prior to interruption.
tokens might either be restored to some sub-set of the input places or not restored at all. This uncertainty, so to say, is what complicates the matter.

Few petri net extensions support programming constructs. Some of those do support such constructs like PNES (Mirkowski and Yakovlev, 1998) associate them to places, others like CPN (Jensen, 1992) aggregate them to transitions. To our knowledge no petri net extension associate programming constructs to both places and transitions. Whether this is a good practice is yet to be discussed. To foster such discussion one argument in favor to this separation is that place related constructs could be used to manipulate data relative to the modeled system while transition related constructs could be used to manipulate data relative to the model itself. This approach renders the control of the net easier and gives more flexibility to data manipulation during analysis.

Most languages for digital systems modeling have no suitable representation of time. Petri nets are no exception. There are different mechanisms that can be used to allow petri nets to deal with time. Some extensions link time mechanisms with transitions specifying a time frame within which the execution of the transition must begin. Other extensions link time mechanisms to places, meaning that it takes some time before the data becomes available for use, i.e., as the occurrence of a transition would be immediate it would represent the time it takes for the corresponding action to produce the required data. In PNES time is associated to places representing the execution time of the behavioral statements assigned to a particular place. To our knowledge no petri net extension associate timing constructs to both places and transitions. The advantage of such an approach would be to augment the expressive power of the extension by allowing one to represent both delays for firing enabled transitions and those for the duration of their execution.

4 A petri net for digital system modeling

Besides existing dynamic and structural analysis techniques, petri net extensions present most of the features needed for a good methodology for digital system modeling as stated in the introduction to this paper. For a language to allow the modeling of a system in an easy way, the tokens should carry complex data types, rather than boolean ones as in place chart nets or petri nets for embedded systems. In this way, the descriptive power of the model would increase. Finally, both time for computation and time for communication have not been simultaneously treated in petri net extensions. In this context, the authors have developed a petri net extension called petri net for digital systems (PNDS). A description of its first version can be found elsewhere (Oliveira and Marranghello, 2000). The main features of the second version can be stated like so:

a) High-level description:
   - Existence of three kinds of places, which can describe storage components, local states of system components, or functional components (behaviors) of the modeled system;
   - Tokens represent, according to the place where they are in, the contents of a cell belonging to a storage unit (described by a storage place), a data value related to the status of a system component (into a local state place), or a data item belonging to an ordered list (data sample) to be processed by a behavior associated to a functional place (such data item can represent an input argument, a data value being changed, or the result of a process);
   - Transitions are able to perform both the changing of modes (removing or adding tokens from/to the places assigned to behaviors/local states) and reading/writing operations involving data values (tokens) belonging to places assigned to storage elements;
   - Special branches are defined according to the places to which they are connected. Branches are labeled with expressions that define what or how tokens must be inserted, removed, read, or written on the assigned places.

b) Hierarchy
   - Models deal with multiple processes in the same structure;
   - One hierarchical instance (a copy of a behavior) for each complete data sample (an ordered list of tokens) present in a functional place.

c) Programming constructs
   - Place related constructs are used to model system inherent data manipulation;
   - Transition related constructs represent control flow manipulation of the net model.

d) Complete exception handling
   - Preemptive mechanisms (to abort local behaviors/states assigned to a place);
   - Interruption mechanisms (to suspend all instances of a behavior assigned to a functional place).

e) Complete time representation
   - Timed and guarded transitions allow the description of events that may occur in a scheduled or random way.

In the following section it is described how PNDS models deal with exception handling and time representation. Further definitions of this
5 Exception handling and time representation by petri nets for digital systems

The occurrence of a particular event can require that a computation mode (behavior or local state) be interrupted or aborted immediately, transferring the net control flow for other computation mode. How to deal with such events as well as how time is represented by petri nets for digital systems is the subject of this section.

Three kinds of places are defined in petri net for digital systems. Type I places are used to indicate that one or more system entities, described by the tokens contained in the place, are in the same local state. Type II places are used to represent system storage units on which read/write operations on data (tokens) can be carried out. Type III places are used to designate that one or more data samples (each of which is represented by a token list) are being processed by one or more functional units of the same unique kind (described by a behavior associated to the place). Further, a special kind of branch called preemptive input branch allows all tokens to be removed from a type I or a type III place, representing the abortion of the mode assigned to a local state descriptor or a system functional unit. Such a branch also allows to reset storage cells of type II places.

Two other kinds of branches are used in petri net for digital systems to model the interruption mechanism. Suspension input branches allow the connected transition to temporarily suspend the mode represented by all active instances of the behavior related to the functional place connected to the source of the branch. Restore output branches have an opposite action, allowing a transition to resume a mode previously suspended. This is possible through the use of a control variable associated to each transition, which is originally true, can be set to false by suspension input branches, and back to true by restore output branches.

Timed transitions make possible to model communication time through the assignment of time intervals to the transitions. This time intervals are defined similarly to Merlin's definition of time. Guarded transitions provide means for the representation of computation time through the use of both time intervals and preemptive mechanisms.

Conclusion

In this paper some issues of the description of digital systems by petri nets are discussed. The main one is the inclusion of mechanisms for the treatment of abstract data types so that the functionality of large-scale systems can be tackled more comfortably. Also, the generalization of the time model used would be desirable to allow for more complete temporal analysis of the systems.

References


Parallel and Distributed Processing Techniques and Applications, pp.1539–1544, June.

