CONTROL SYSTEMS DESIGN BASED ON PETRI NETS

This paper describes a structured and flexible method for control system design based on Petri nets models. A proposed CAD tool allows control system specification, modeling validation and synthesis using ordinary Petri nets. The control system synthesis is based on Hardware Petri nets that are composed of two kinds of processing elements (Places and Transitions) and data flow path between them. The use of Hardware Petri nets in CAD tools allows the automation of the FPGA implementation process and substantially reduces the design time and cost. A design example illustrates the proposed method.

Introduction

The electronic technology innovation over the past forty years is best characterized by Moore's Law - chip density doubles in every two years [1]. Such exponential growth rate of chip density allows engineers to pack more complex functionality into a single chip with improving efficiency in terms of area, power, and performance. Leading semiconductor manufacturers are now capable of producing very sophisticated electronic system-on-chips (SoCs). Built at the deep sub-micron level, where the effective impact of interconnects on performance, power and reliability will continue to increase, such systems present a formidable challenge for design and test methods and tools.

The key point raised in the ITRS (International Technology Roadmap for Semiconductors) is that design cost is the greatest threat to the continued phenomenal progress in microelectronics [2]. The only way to overcome this threat is through improving the productivity and efficiency of the design process, particularly by means of design automation and component reuse. The constantly improving CAD tools can help to mitigate the problem by delivering faster simulation, higher capacity formal verification, and better logic synthesis coupled with place-and-route.

The importance of automation of the synthesis process of digital systems steadily grows especially with the constant increase in the complexity of such systems. The use of Petri nets for the specification, analysis and synthesis of digital systems has proved very worthwhile. Petri nets are mathematically well founded and can be used to capture causality relations, concurrency of actions and conflicting conditions from digital systems in a natural and convenient way. It is possible to translate Petri nets to HDL (Hardware Description Language), and vice versa, making it possible to integrate Petri nets tools into existing design environments.

Implementation methods of Petri nets can be classified into two classes: software and hardware. Software implementation represents the emulation of Petri nets using computer software, which usually takes long time. It is widely used in modeling and performance evaluation problems. Hardware implementation of Petri nets is done especially in FPGA (Field Programmable Gate Arrays) circuits. The advantage of FPGA technology is that the interconnection patterns inherent in the Petri net structural description can be very flexibly mapped to the FPGA structure. The possibility of a runtime reconfiguration allows the use of adaptive algorithms that can reduce the time that is necessary
Control systems design based on Petri nets

for Petri Net simulation. Hardware implementation
methods of Petri nets can be divided in direct transla-
tion methods [3,4] and logic synthesis methods [5,6]. Logic
synthesis methods often suf-
der from the state explosion problem because most
modern systems are typically modeled as concurrent
systems. Direct translation methods guarantee
an implementation by construction. The size of the
obtained circuits is linear on the size of the specifica-
tion.

This paper focuses on some of opportunities of
Petri nets utilization in control systems design
based on direct translation of the behavioral model
in FPGA circuits. A proposed CAD tool allows
control system specification, modeling and imple-
mentation using ordinary Petri nets. The synthesiz-
able AHDL code is generated from a Petri net
model. Proposed method makes possible the struc-
tured and flexible FPGA implementation of control
systems.

The paper is organized as follows. Section 2 de-
des Petri nets and their analysis methods. Sec-
section 3 and 4 presents the hardware imple-
mentation using ordinary Petri nets. The synthesiz-
able AHDL code is generated from a Petri net
model. Proposed method makes possible the struc-
tured and flexible FPGA implementation of control
systems.

The behavioral properties of PN are very im-
portant for modeling, analyzing, verification and
validation of computer systems.

A PN is said to be finite if sets P and T are fi-
nite.

A PN is said to be k-bounded if there exists a k
such that at any reachable marking the number of
tokens in any place is not greater than k. A 1-
bounded PN is called a safe PN.

A marking m is said to be reachable from the
initial marking m0 if exists a sequence of firings σ
that transforms m0 in m. The set of markings of a
net that can be reached from its initial marking by
means of all possible firings of transitions is called
the reachability set of the PN. It can be represented
as a graph, called the reachability graph of the net,
with the nodes labeled with the markings and the
arcs labeled with transitions.

A PN is said to be reversible if for any marking
m, the initial marking m0 is reachable from m.

A transition T of a PN is said to be live if for
any reachable marking m there exists a marking
m’ reachable from m at which this transition is
enabled. A PN is said to be live if every transition
is live. This is called a strong form of PN liveness,
in which every operation can be activated at some
state when the system starts in any of its allowable
states. A weaker form of liveness requires only that
a transition can be enabled at least once in some
reachable marking. A reachable marking m at
which no transition is enabled is called a deadlock.
A PN is said to be a deadlock-free if its reachability
set includes no deadlocks. Presence of dead-
locks is regarded as an error in a system which op-
brates in cycles.

Petri net specification of a control system

Petri net definition

A Petri net (PN) is a 4-tuple \( (P, T, F, M^0) \) [7, 8], where:

\[
P = \{ p_1, p_2, ..., p_N \} \text{ is a finite and non-empty set of places;}
\]

\[
T = \{ t_1, t_2, ..., t_L \} \text{ is a finite and non-empty set of transitions } (P \cap T = \emptyset);
\]

\[
F \subseteq (P \times T) \cup (T \times P) \text{ is a flow relation that}
\]

\[
defines directed arcs from places to transitions
\]

\[
(P \times T) \notin \emptyset \text{ and transitions to places}
\]

\[
(T \times P) \notin \emptyset;
\]

\[
M^0 = \{ m_1^0, m_2^0, ..., m_N^0 \} \text{ is the initial marking.}
\]

An ordinary PN is a net where each arc has a
weight that is equal to 1. A PN is represented as a
graph with two types of nodes: circles are used to
denote places and bars or boxes, are used for trans-
itions. Directed arcs between places and transitions
and vice versa, denote the flow relation. A
marking of a PN is depicted with tokens. A transition
is said to be enabled under a given marking, if
all its input places contain at least one token. An
enable transition can fire, producing a new mark-
ing. The firing of a transition removes one token
from each input place and adds one token into each
output place of the transition.

Petri net properties

The behavioral properties of PN are very im-
portant for modeling, analyzing, verification and
validation of computer systems.

A PN is said to be finite if sets P and T are fi-
nite.

A PN is said to be k-bounded if there exists a k
such that at any reachable marking the number of
tokens in any place is not greater than k. A 1-
bounded PN is called a safe PN.

A marking m is said to be reachable from the
initial marking m0 if exists a sequence of firings σ
that transforms m0 in m. The set of markings of a
net that can be reached from its initial marking by
means of all possible firings of transitions is called
the reachability set of the PN. It can be represented
as a graph, called the reachability graph of the net,
with the nodes labeled with the markings and the
arcs labeled with transitions.

A PN is said to be reversible if for any marking
m, the initial marking m0 is reachable from m.

A transition T of a PN is said to be live if for
any reachable marking m there exists a marking
m’ reachable from m at which this transition is
enabled. A PN is said to be live if every transition
is live. This is called a strong form of PN liveness,
in which every operation can be activated at some
state when the system starts in any of its allowable
states. A weaker form of liveness requires only that
a transition can be enabled at least once in some
reachable marking. A reachable marking m at
which no transition is enabled is called a deadlock.
A PN is said to be a deadlock-free if its reachability
set includes no deadlocks. Presence of dead-
locks is regarded as an error in a system which op-
brates in cycles.

Petri net analysis

There are several methods for analyzing the PN
dynamic behavior. The most common one is to
build a reachability set which represent all possible
states of the system. This method is expensive be-
cause the reachable markings may grow exponentially with the number of transitions in the PN.

A few methods have been proposed to overcome the state space explosion. One of them is the stubborn set method [9]. This method partially represents the reachability set. It uses the fact that interleaving (possible orderings of concurrent transitions) lead to the same marking. Although efficient in finding deadlocks, it does not produce a complete representation of the reachable state space. Another method is so called PN symbolic traversal [10]. It uses implicit representation of the reachability set in the form of Binary Decision Diagrams (BDDs) which are canonical representations of Boolean functions in graphical form. This method is efficient in analysis of “state-based” properties such as freedom from deadlock. The third method is called PN unfolding [11]. It is based on representation of full reachability graph using partial orders preserving relations between transition occurrences. A transition occurrence is a unique event associated with a single act of firing of the transition. Since all reachable markings are represented in the PN unfolding, the concurrency relation for two transitions can easily be obtained.

**Hardware implementation of Petri nets**

The computer-based synthesis of the control system from Petri net level to logic design level request the adaptation of the Petri net model to its hardware implemented model. The control system model is considered as a set of processing elements with data flow path between them. The corresponding Petri net model contains two kinds of processing elements $P_i$ and $T_j$. The arcs between them represent the data flow paths. The arc from processing element $P_i$ to processing element $T_j$ is denoted as $P_i \rightarrow T_j$ and the arc from processing element $T_j$ to processing element $P_i$ is denoted as $T_j \rightarrow P_i$. For hardware implemented PN model the data flow depends on the topology of the net.

A Hardware Petri Net (HPN) is defined as a union between sets of processing elements and data flows.

$$RPH = T \cup P \cup A^+ \cup A^- \cup A^i \cup A^f \cup P^in \cup P^out,$$

where:

$$T = \{T_1, T_2, \ldots, T_n\}, \quad T \neq \emptyset$$  is a set of processing elements Transition;

$$P = \{P_1, P_2, \ldots, P_N\}, \quad P \neq \emptyset$$  is a set of processing elements Place;

$$A^+ = \{A^+ i, \ i = 1, N\}, \quad A^+ \neq \emptyset$$  is a set of arcs $T_j \rightarrow P_i$ that represent the condition when the number of tokens in the place is increased and is defined as follows:

$$A^+_j = \begin{cases} \begin{array} {l} a_{ji} = 1, \quad & T_j \rightarrow P_i, \quad i = 1, N, \quad j = 1, L; \\ a_{ji} = 0, \quad & \text{otherwise}. \end{array} \end{cases}$$

$$A^− = \{A^− i, \ i = 1, N\}, \quad A^− \neq \emptyset$$  is a set of arcs $T_j \rightarrow P_i$ that represent the condition when the number of tokens in the place is decreased and is defined as follows:

$$A^−_j = \begin{cases} \begin{array} {l} a_{ji} = 1, \quad & T_j \rightarrow P_i, \quad i = 1, N, \quad j = 1, L; \\ a_{ji} = 0, \quad & \text{otherwise}. \end{array} \end{cases}$$

$$A^+ = \{A^+ s, \ j = 1, L\}, \quad A^+ \neq \emptyset$$  is a set of state arcs $P_i \rightarrow T_j$ that determine the enable firing condition of the transition $T_j$ related to the marking of the place $P_i$. the arc is connected with. This set is defined as follows:

$$A^+_j = \begin{cases} \begin{array} {l} a_{j} = 1, \quad & P_i \rightarrow T_j, \quad i = 1, N, \quad j = 1, L; \\ a_{j} = 0, \quad & \text{otherwise}. \end{array} \end{cases}$$

State arc connects an input place to a transition and has the ability to check whether a place has a token. The presence of a state arc connecting an input place to a transition means that the transition is only enabled if the input place has a token. The firing change the marking in the enabling arc connected place.

$$A^+ = \{A^+ j, \ j = 1, L\}, \quad A^+ \neq \emptyset$$  is a set of test arcs, which has the same function as the set of state arcs, but the firing does not change the marking in the test arc connected place.

$$A^+_j = \begin{cases} \begin{array} {l} a_{j} = 1, \quad & P_i \rightarrow T_j, \quad i = 1, N, \quad j = 1, L; \\ a_{j} = 0, \quad & \text{otherwise}. \end{array} \end{cases}$$
\[ A^j_i = \begin{cases} a^j_i = 1 & | P_i \rightarrow T_j, \ i = 1, N, \ j = 1, L; \\ a^j_i = 0, \ otherwise. \end{cases} \]

Inhibitor arc connects an input place to a transition and has the ability to test whether a place is empty. The presence of an inhibitor arc connecting an input place to a transition means that the transition is enabled if the input place has no token. The firing does not change the marking in the inhibitor arc connected place.

\[ P^{in} = \left\{ P^{in}_j, \ j = 1, L^{in} \right\}, \ P^{in} \in P \] is a set of processing elements \( P_j \) that represent the input signals;

\[ P^{out} = \left\{ P^{out}_j, \ j = 1, L^{out} \right\}, \ P^{out} \in P \] is a set of processing elements \( P_j \) that represent the output signals;

The interaction of the control unit, represented by \( HPN \) and the external system is done through input and output signals that are given by processing elements \( P^{in}, P^{out} \) (figure 1.)

**Processing elements**

The processing element \( T \) prepares the data processing operation. After analyzing of the global state \( S^k = \left\{ \left( m_i, P_i \right), \forall i = 1, L \right\} \) at the step \( k \) of data processing, the condition for step \( k+1 \) of data processing operation is formed.

The behavior of the processing element \( T \) may be described as follows: if in each input place of a transition \( T \) there is a token, then the firing condition of \( T \) occurs. In this case tokens are removed from all input places and are placed into all output places. In figure 2(a) is shown a transition with four input and three output places. \( P_1 \) and \( P_2 \) are connected with \( T_1 \) by state arcs, \( P_3 \) is connected by inhibitor arc and \( P_3 \) is connected by a test arc. The logic implementation (Figure 2 (b)) is an NAND gate with an additional enable input \( En \) that allows the firing of the transition when all its input connections are active. The logic symbol for the processing element Transition is given in Figure 2(c).

![Diagram](image-url)
\[
\begin{align*}
\dot{m}_i^k &= 1 + \sum_{j=1}^{N} (A_{ij}^+ - A_{ij}^-), & i &= 1,2, \ldots, N; \\
\dot{m}_i^k &= 1 - \sum_{j=1}^{N} (A_{ij}^- - A_{ij}^+), & i &= 1,2, \ldots, N; \\
\dot{m}_i^k &= 0, & i &= 1,2, \ldots, N; \\
\dot{m}_i^k &= 1, & i &= 1,2, \ldots, N; \\
\end{align*}
\]

where: \( m_i^k \) is the number of tokens in \( P_i \) at the step \( k \) of data processing, \( L_i^+ \) and \( L_i^- \) are the total number of increment and decrement arcs to the place \( P_i \), \( m_i^{max} = \max_{i=1,N} m_i \) represent the maximal number of tokens that can be stored in \( P_i \). The best way to implement a place is to use a counter with a combinational input logic. In Petri model works as a control system it is enough to check the presence or absence of the tokens in the place.

Design flow

The control system design flow is presented in Figure 4.

![Fig. 4. Design flow of the Petri net-based control system.](image)

Fig. 4. Design flow of the Petri net-based control system.

Design steps description:
- **PN Models Source** – Petri net model that is proposed for analyzing (in graphical form);
- **VPNPN Tool** - software tool that allows inserting and modifying in an interactive mode the Petri net model;
- **PN Models Library** – the library with Petri net models;
- **Analysis (Reachability graph & Structural analysis** – The proposed Petri net model is analyzed in order to determine the set of reachable states and to form the reachability graph. The structural analysis determines the main properties

![Diagram](image)

Fig. 3. An example of possible connections to a place (a), logic implementation of the presented example (b) and logic symbol of the processing element Place (c).

net modeling tasks it is important the exact number of tokens in the place. When a Hardware Petri net
of the model such as its reachability, liveness and reversibility;

**MI and MO generation** – incidence matrix and initial marking generation and their storage in corresponding files;

**HDL Compiler** – HDL code compilation based on matrices MI and MO;

**HDL Objects Library** – the library with standard HDL objects that are used to form HDL code of a Petri net;

**HDL code** – the obtained after compilation HDL code;

**Max Plus + II Design Tool** – MAX+PLUS II software is a fully integrated, architecture-independent package for designing logic with ALTERA programmable logic devices;

**FPGA or CPLD Device** – FPGA or CPLD configuration of the Petri net model.

### Design example
As a design example a parallel to sequential code controller is introduced. A controller consist of a **RAM** for storage of data to be converted, a 8-bit shift right register **Rg**, a data modulation unit **DM**, a memory address counter **CT Adr**, a counter for register **Rg** bits and a Petri net-based control unit **RPH** (Figure 5).

![Fig. 5. Parallel to sequential code controller.](image)

**Fig. 5.** Parallel to sequential code controller.

**DM**, a memory address counter **CT Adr**, a counter for register **Rg** bits and a Petri net-based control unit **RPH** (Figure 5).

![Fig. 6. Petri net model for parallel to sequential data conversion.](image)

**Fig. 6.** Petri net model for parallel to sequential data conversion.

![Fig. 7. Simulation results.](image)

**Fig. 7.** Simulation results.

Науковий вісник Чернівецького університету. 2008. Випуск 423. Фізика.Електроніка. 65
The conversion operation begins when START signal and Rst signal are set. Signal RD initialize the read operation from RAM with address ADR. Inc signal is used to increment the address code. EA is the signal that signalizes the end of address space. The extract data are written in Rg when signal Load is active. The content of Rg is shifted right bit by bit using signal Shr till signal ERg is generated (after eight shifts). Each output bit from Rg is modulated in DM. The modulation time is controlled by signals OEe and OPe. When the conversion operation is finished signal Eop is set.

The corresponding Petri net model that is used as a design entry and simulation results are shown in Figure 6 and Figure 7, respectively. The simulation results are presented for 8-bit data code conversion.

Conclusions

In this paper an approach for the control systems design from Petri nets models has been presented. The use of Petri nets allows interplay of different formal tasks, such as synthesis, verification and performance evaluation, to be carried out within the single modeling framework. The design flow starts with the behavior specification of the control system as a Petri net model. The main properties of the model (reachability, liveness, reversibility) are analyzed using a VPNP software tool. Then the direct translation of the Petri net model into AHDL code is done based on Hardware Petri net analyzing. The use of Hardware Petri nets in CAD tools allows the automation of the FPGA implementation process and substantially reduces the design time and efforts. The method can be used for the synthesis of relatively large circuits when space and speed constraints are not critical.

References


V. Sudacevschi, V. Ababii