HDL Implementation from Petri Nets Description

Viorica SUDACEVSCHI, Victor ABABII, Emilian GUTULEAC, Valentin NEGURA
Technical University of Moldova
str.Stefan cel Mare, 168, MD-2012, Chisinau
svm@mail.utm.md; avv@mail.utm.md; egutuleac@mail.utm.md; vnegura@yahoo.fr

***Abstract* — This paper describes the digital systems synthesis based on direct mapping of Petri nets model into FPGA circuit. A design flow that includes the specification of the system using Synchronous Petri Nets, verification of the behavioral properties of the model, generation of the mathematical model of Hard Petri Nets (HPN), used for automatic generation of the AHDL code is described. The direct mapping approach avoids algorithmic complexity inherent in logic synthesis based on state encoding and substantially reduces the design time and cost. The method used for modeling and implementation of the digital systems was validated using MAX+PLUS II design environment.**

***Index Terms* — AHDL, digital system, direct mapping, FPGA, MAX+PLUS II, Hard Petri Nets.**

I. INTRODUCTION

Advances in semiconductor technology over the last four decades have resulted in a nearly constant compound growth in transistor density of approximately 46% per year [1]. This remarkable achievement has not been matched by an equivalent increase in integrated circuit designer productivity, leading to a design gap as illustrated in Figure 1.

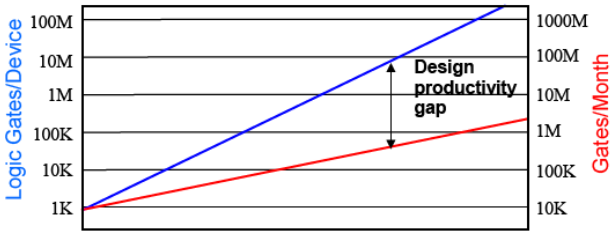

Figure 1. The design productivity gap.

The design gap represents the disparity between the transistors available to a designer and the ability to use them effectively in a design. Increases in productivity are limited by the spiraling system complexity engendered by increased transistor counts. Traditional design methods do not scale to match the increased complexity.

On the other side, the growth of design productivity leading to a so called "verification crisis". According to a Collett international study, the rate of first silicon success is steadily declining, dropping to 35% in 2003; 70% of re-spun designs contain functional bugs [2].

One of the ways to overcome this threat is through improving the productivity and efficiency of the design process, particularly by means of new synthesis approaches that can transform a behavioral specification into an adequate implementation.

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.

Two main approaches to digital systems design based on Petri nets are direct mapping [3] and logic synthesis [4]. Logic synthesis methods often suffer from the state explosion problem because most modern systems are typically modeled as concurrent systems. Direct mapping methods guarantee an implementation by construction. The size of the obtained circuits is linear on the size of the specification.

This paper focuses on some of opportunities of Petri nets utilization in digital systems synthesis based on direct mapping of the behavioral model in FPGA circuits. A proposed CAD tool allows digital system specification, modeling and implementation using ordinary Petri nets. The synthesizable AHDL code is generated from a Petri net model. Proposed method makes possible the structured and flexible FPGA implementation of digital systems.

II. DIRECT MAPING OF A PETRI NET MODEL

In logic synthesis approach *boolean* equations for the output signals of the circuit are derived using minimization methods. This approach suffers from excessive computation complexity and memory requirements. The circuit optimization often involves analysis and recalculation of the whole state space. Thus it cannot be applied to large specifications. There is no transparent correspondence between the elements of the original specification, the intermediate representation of the state space and the components of the resultant circuit, which complicates the checking of circuit functionality.

The main idea of the direct mapping approach is that a Petri net model of a system is converted into a circuit netlist in such a way that the graph nodes correspond to the circuit elements and graph arcs correspond to the interconnects (Figure 2).

The direct mapping method has a linear algorithmic complexity, is not affected by state explosion, so large digital systems can be constructed at low cost. Direct mapping facilities checking of the functional correctness of the implementation because of the transparent correspondence between the elements of the initial

specification and the components of the resultant circuit. Notwithstanding all advantages, this approach is insufficiently studied and existing techniques for direct mapping often produce large circuits with inefficient interface to the environment.

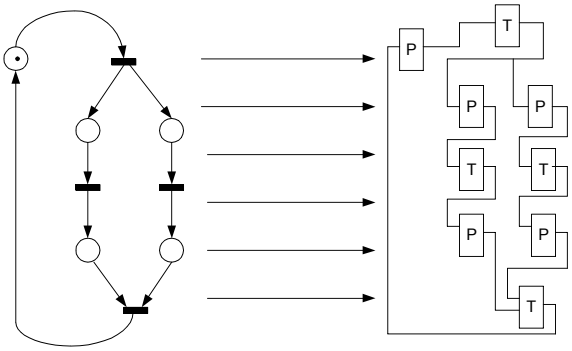

Figure 2. Direct mapping of a Petri net model to the circuit.

III. SYNCHRONOUS PETRI NETS

In order to describe digital systems behavior, to perform their verification and synthesis, a new extension of ordinary Petri nets, namely Synchronous Petri nets, is proposed.

A *Synchronous Petri net (SPN)* is a 6-tuple *(P, T, A,M_0, M_{max}, C),* where:

$\boldsymbol{P} = \{ \boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_N \}$ is a finite and non-empty set of *places*;

$\boldsymbol{T} = \{ \boldsymbol{t}_1, \boldsymbol{t}_2, ..., \boldsymbol{t}_L \}$ is a finite and non-empty set of *transitions* ($\boldsymbol{P} \cap \boldsymbol{T} = \varnothing$);

$\boldsymbol{A} \subseteq (\boldsymbol{P} \times \boldsymbol{T}) \cup (\boldsymbol{T} \times \boldsymbol{P})$ is a set of arcs that consists of three subsets: $\boldsymbol{A} = \boldsymbol{A}^N \cup \boldsymbol{A}^I \cup \boldsymbol{A}^T$, $\boldsymbol{A}^N \cap \boldsymbol{A}^I \cap \boldsymbol{A}^T = \{\varnothing\}$, $\boldsymbol{A}^N$ - normal arcs, $\boldsymbol{A}^I$ - inhibition arcs, $\boldsymbol{A}^T$ - test arcs;

$\boldsymbol{M}_0 = \{ \boldsymbol{M}_0^{P_1}, \boldsymbol{M}_0^{P_2}, \ldots, \boldsymbol{M}_0^{P_N} \}$ is the *initial marking*, defined as an initial number of tokens in each place;

$\boldsymbol{M}_{max} = \{ \boldsymbol{M}_{max}^{P_1}, \boldsymbol{M}_{max}^{P_2}, \ldots, \boldsymbol{M}_{max}^{P_N} \}$ is the *maximal marking*, defined as a maximal number of tokens in each place;

C is the *synchronization variable* that enable the transitions firing.

Subset $\boldsymbol{A}^N$ defines the normal arcs, through which tokens are removed from each input place and are added to each output place. Subsets $\boldsymbol{A}^I$ and $\boldsymbol{A}^T$ are necessary in behavioral analyze but do not remove or add tokens.

IV. HARDWARE IMPLEMENTATION OF PETRI NETS

The computer-based synthesis of the digital system from Petri net level to logic design level request the adaptation of the Petri net model to its hardware implemented model. The digital 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 $\boldsymbol{P}_i$ and $\boldsymbol{T}_j$. The arcs between them represent the data flow paths. For hardware implemented *SPN* model the data flow depends on the topology of the net.

A ***Hardware Petri Net (HPN)*** [5] is defined as reunion between sets of processing elements and data flows:

$\boldsymbol{RPH} = < \boldsymbol{T}, \boldsymbol{P}, \boldsymbol{A}^+, \boldsymbol{A}^-, \boldsymbol{A}^S, \boldsymbol{A}^T, \boldsymbol{A}^I, \boldsymbol{P}^{In}, \boldsymbol{P}^{Out}, \boldsymbol{M}_0, \boldsymbol{M}_{max}, \boldsymbol{C} >$, where:

$\boldsymbol{T} = \{\boldsymbol{T}_1, \boldsymbol{T}_2, ..., \boldsymbol{T}_L\}$, $\boldsymbol{T} \neq \varnothing$ is a set of processing elements that correspond to transition nodes;

$\boldsymbol{P} = \{\boldsymbol{P}_1, \boldsymbol{P}_2, ..., \boldsymbol{P}_N\}$, $\boldsymbol{P} \neq \varnothing$ is a set of processing elements that correspond to place nodes;

$\boldsymbol{A}^+ = \{\boldsymbol{A}_i^+, \boldsymbol{i} = \overline{1, \boldsymbol{N}}\}$, $\boldsymbol{A}^+ \neq \varnothing$ is a set of *increment connections* of the number of tokens in position processing elements and is defined as follows:

$$\boldsymbol{A}^+ = \left\{ \boldsymbol{a}_{ji}^+ \;\middle|\; \begin{array}{l} \boldsymbol{a}_{ji}^+ = 1, \;\; \boldsymbol{t}_j \xrightarrow{a^N} \boldsymbol{p}_i \\ \boldsymbol{a}_{ji}^+ = 0, \; \boldsymbol{otherwise} \end{array} \quad \boldsymbol{j} = \overline{1, \boldsymbol{L}}, \boldsymbol{i} = \overline{1, \boldsymbol{N}} \right\};$$

$\boldsymbol{A}^- = \{\boldsymbol{A}_i^-, \boldsymbol{i} = \overline{1, \boldsymbol{N}}\}$, $\boldsymbol{A}^- \neq \varnothing$ is a set of *decrement connections* of the number of tokens in position processing elements and is defined as follows:

$$\boldsymbol{A}^- = \left\{ \boldsymbol{a}_{ji}^- \;\middle|\; \begin{array}{l} \boldsymbol{a}_{ji}^- = 1, \quad \boldsymbol{p}_i \xrightarrow{a^N} \boldsymbol{t}_j \\ \boldsymbol{a}_{ji}^- = 0, \; \boldsymbol{otherwise} \end{array} \quad \boldsymbol{j} = \overline{1, \boldsymbol{L}}, \boldsymbol{i} = \overline{1, \boldsymbol{N}} \right\}$$

;

$\boldsymbol{A}^S = \{\boldsymbol{A}_j^S, \boldsymbol{j} = \overline{1, \boldsymbol{L}}\}$, $\boldsymbol{A}^S \neq \varnothing$ is a set of *state connections* that determine the enable firing condition of the transition $\boldsymbol{T}_j$ related to the marking of the input place $\boldsymbol{P}_i$. This set is defined as follows:

$$\boldsymbol{A}^s = \left\{ \boldsymbol{a}_{ij}^s \;\middle|\; \begin{array}{l} \boldsymbol{a}_{ij}^s = 1, \quad \boldsymbol{p}_i \xrightarrow{a^N} \boldsymbol{t}_j \\ \boldsymbol{a}_{ij}^s = 0, \; \boldsymbol{otherwise} \end{array} \quad \boldsymbol{i} = \overline{1, \boldsymbol{N}} \;\; , \boldsymbol{j} = \overline{1, \boldsymbol{L}} \right\}$$

State connection has the ability to check whether a place has a token. The assertion of a state connection means that the transition is only enabled if the input place has a token. The transition firing changes the marking in the input place.

$\boldsymbol{A}^T = \{\boldsymbol{A}_j^T, \boldsymbol{j} = \overline{1, \boldsymbol{L}}\}$, $\boldsymbol{A}^T \neq \varnothing$ is a set of *test connections*, which has the same function as the set of state connections, but the transition firing does not change the marking in the input place.

$$\boldsymbol{A}^T = \left\{ \boldsymbol{a}_{ij}^T \;\middle|\; \begin{array}{l} \boldsymbol{a}_{ij}^T = 1, \quad \boldsymbol{p}_i \xrightarrow{a^T} \boldsymbol{t}_j \\ \boldsymbol{a}_{ij}^T = 0, \quad \boldsymbol{otherwise} \end{array} \quad \boldsymbol{i} = \overline{1, \boldsymbol{N}}, \; \boldsymbol{j} = \overline{1, \boldsymbol{L}} \right\}$$

$\boldsymbol{A}^I = \{\boldsymbol{A}_j^I, \boldsymbol{j} = \overline{1, \boldsymbol{L}}\}$, $\boldsymbol{A}^I \neq \varnothing$ is a set of inhibitor connections, which provides an enabling function, when the place stores no tokens. It is defined as follows:

$$\boldsymbol{A}^I = \left\{ \boldsymbol{a}_{ij}^I \;\middle|\; \begin{array}{l} \boldsymbol{a}_{ij}^I = 1, \quad \boldsymbol{p}_i \xrightarrow{a^I} \boldsymbol{t}_j \\ \boldsymbol{a}_{ij}^I = 0, \quad \boldsymbol{otherwise} \end{array} \quad \boldsymbol{i} = \overline{1, \boldsymbol{N}}, \; \boldsymbol{j} = \overline{1, \boldsymbol{L}} \right\}$$

; Inhibitor connection has the ability to test whether a place is empty. The assertion of an inhibitor connection means that the transition is enabled if the input place has no token. The firing does not change the marking in the input place.

$\boldsymbol{P^{In}} = \left\{ \boldsymbol{P_j^{In}},\ \boldsymbol{j} = \overline{1, \boldsymbol{L^I}} \right\}, \boldsymbol{P^{In}} \in \boldsymbol{P}$ is a set of processing elements $\boldsymbol{P_j}$ that represent the input signals in the digital system;

$\boldsymbol{P^{Out}} = \left\{ \boldsymbol{P_j^{Out}},\ \boldsymbol{j} = \overline{1, \boldsymbol{L^O}} \right\}, \boldsymbol{P^{Out}} \in \boldsymbol{P}$ is a set of processing elements $\boldsymbol{P_j}$ that represent the output signals in the digital system;

$\boldsymbol{M}_0 = \{ \boldsymbol{M}_0^{P_1}, \boldsymbol{M}_0^{P_2}, \ldots, \boldsymbol{M}_0^{P_N} \}$ - is the initial marking;

$\boldsymbol{M}_{max} = \{ \boldsymbol{M}_{max}^{P_1}, \boldsymbol{M}_{max}^{P_2}, \ldots, \boldsymbol{M}_{max}^{P_N} \}$ - is the maximal marking;

$\boldsymbol{C}$ - is the synchronization variable.

A conclusion section is not compulsory. Make sure that the whole text of your paper observes the textual arrangement on this page.

V. PROCESSING ELEMENTS

The processing element T prepares the data processing operation. After analyzing of the global state $S^k = \{(m_i, P_i), \forall i = \overline{1, I}\}$ 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 3(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, P3 is connected by inhibitor arc and P_4 is connected by a test arc. The logic implementation of a processing element is shown in Figure 3 (b)).

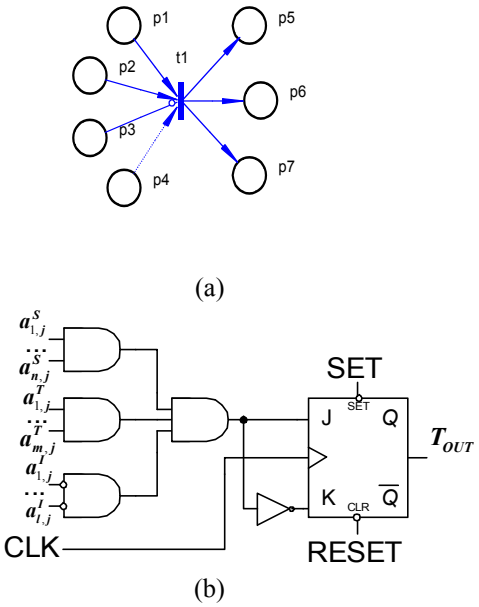

Figure 3. An example of possible connections to a transition (a), logic implementation of the processing element Transition (b).

The processing element P stores the state value and performs the increment and decrement operation of the number of tokens. The increment operation occurs when one of the input transitions of the processing element P fires. The decrement operation occurs when one of the output transition of the processing element P fires. The number of tokens in P at the step $k+1$ of data processing, denoted by m_i^{k+1}, is changed according to the following rules:

$$\boldsymbol{m_i^{k+1}} = \begin{cases} \boldsymbol{m_i^k} + 1 \ \Big| \ \sum_{j=1}^{L_i^+} (\boldsymbol{A^+_{ij}}) = 1, \ \forall \ \boldsymbol{m_i^k} < \boldsymbol{m_i^{\max}}; \\ \boldsymbol{m_i^k} - 1 \ \Big| \ \sum_{j=1}^{L_i^-} (\boldsymbol{A^-_{ij}}) = 1 \ \forall \ \boldsymbol{m_i^k} > 0; \\ \boldsymbol{m_i^k} \ \Big| \ \sum_{j=1}^{L_i^+} (\boldsymbol{A^+_{ij}}) = 0 \ \ \& \ \ \sum_{j=1}^{L_i^-} (\boldsymbol{A^-_{ij}}) = 0; \\ \boldsymbol{m_i^k} \ \Big| \ \sum_{j=1}^{L_i^+} (\boldsymbol{A^+_{ij}}) = 1 \ \ \& \ \ \sum_{j=1}^{L_i^-} (\boldsymbol{A^-_{ij}}) = 1; \end{cases} , \boldsymbol{i} = \overline{1, \boldsymbol{N}}$$

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 of the place P_i, $(\boldsymbol{m_i^{max}} \ \forall \ \boldsymbol{i} = \overline{\boldsymbol{1, N}}) \in \boldsymbol{M^{max}}$ 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 net modeling tasks it is important the exact number of tokens in the place. When a Hardware Petri net model works as a digital system it is enough to check the presence or absence of the tokens in the place. In Figure 4(a) an example of a place with three input and three output transitions is presented. The logic implementation is given in Figure 4(b).

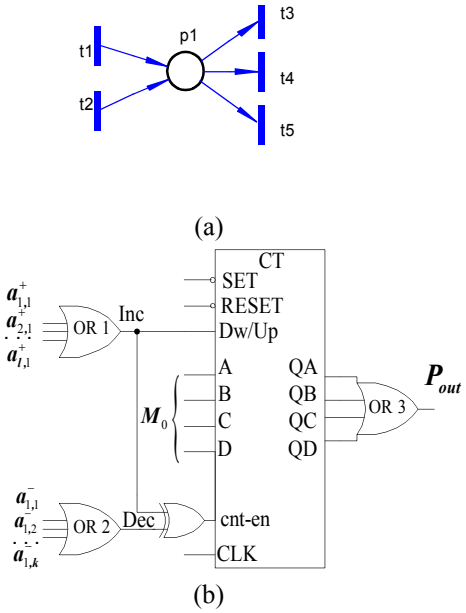

Figure 4. An example of possible connections to a place (a), logic implementation of the processing element Place (b).

AHDL codes for both processing elements were elaborated. These codes were executed and simulated using MAX+PLUS II design tool [6].

VI. DESIGN FLOW

The digital system design flow is presented in Figure 5.

System specification is done using SPN model.

The proposed Petri net model is analyzed in order to determine the set of reachable states and to form the reachability graph in VPNP (*Visual Petri Net +*) environment. The behavioral analysis determines the main properties of the model such as its reachability, liveness and reversibility. In the result, an XML code of the Petri net model is obtained.

According to this code HPN model is generated. This model is translated to an AHDL code of the analyzed digital system.

This code is executed and simulated using MAX+PLUS II design tool. In the result, the gatelevel netlist that can be implemented into FPGA circuit is obtained.

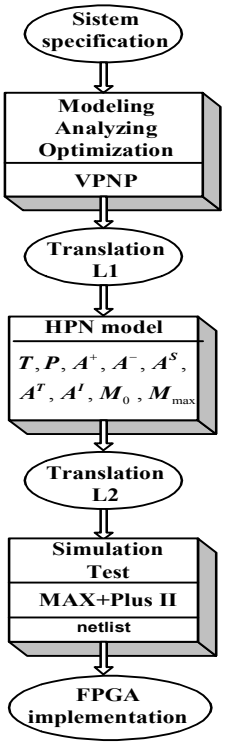

Figure 5. Design flow of the Petri net-based digital systems synthesis.

VII. DESIGN EXAMPLE

As a design example a parallel to sequential code controller is presented.

A controller consist of a *RAM* memory for storage of data to be converted, a 8-bit shift right register *Rg*, a memory address counter *CT Adr* and a Petri net-based control unit *RPH* (Figure 6).

The conversion operation begins when *START* signal (initiation of data transfer operation) and *Reset* signal (*CT Adr* and *RPH* reset) are asserted. Signal *RD* initialize the read operation from *RAM* memory on address *ADR*. *Inc* signal is used to increment the address code. *EA* is the signal that determines the end of the address space. The extracted data are written in *Rg* when signal *Load* is asserted. Signal *ShR* is used to shift right the content of *Rg*. The number of shifts is controlled by internal RPH counter. When the conversion operation is finished signal *EoP* is asserted.

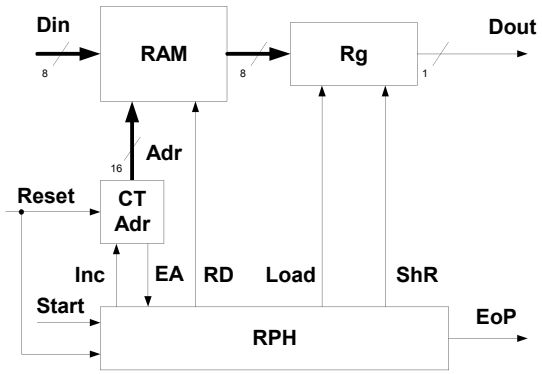

Figure 6. Parallel to sequential code controller.

The corresponding Petri net model that is used as a design input is shown in Figure 7. The simulation results are shown in Figure 8. For simulation was done the conversion of three 4-bits binary words.

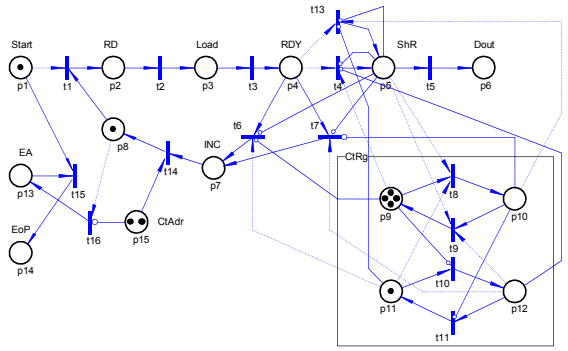

Figure 7. Petri net model of the parallel to sequential code controller.

VIII. CONCLUSIONS

In this paper an approach for the digital 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 digital 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 mapping of the Petri net model into AHDL code is done. 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 constrains are not critical.

REFERENCES

[1] Semiconductor Industry Association, "International Technology Roadmap for Semiconductors, 2005 Edition."

[2] Jun Yuan, Carl Pixley, Adnan Aziz, Constraint-based verification, Springer, 2006, 253 pages.

[3] A. Bystrov and A. Yakovlev, Asynchronous Circuit Synthesis by Direct Mapping: Interfacing to Environment, Proc. ASYNC'02, Manchester, April 2002.

[4] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A.Yakovlev: Logic Synthesis of Asynchronous Controllers and Interfaces. Springer Verlag (2002).

[5] V. Sudacevschi, V. Ababii, V. Negură, A Hardware Implementation of Safe Petri Net Models. Proceedings of the 8th International Conference on Development and Application Systems, Suceava, Romania, 25-27 May, 2006, p. 9-13.

[6] V.Ababii, V. Sudacevschi, Safe Petri Nets Models Mapping into FPGA Using HDL Code. *The International Symposium on Systems Theory, SINTES 12,* October 20-22, 2005, Craiova, Romania, Vol.4, p. 697-699.

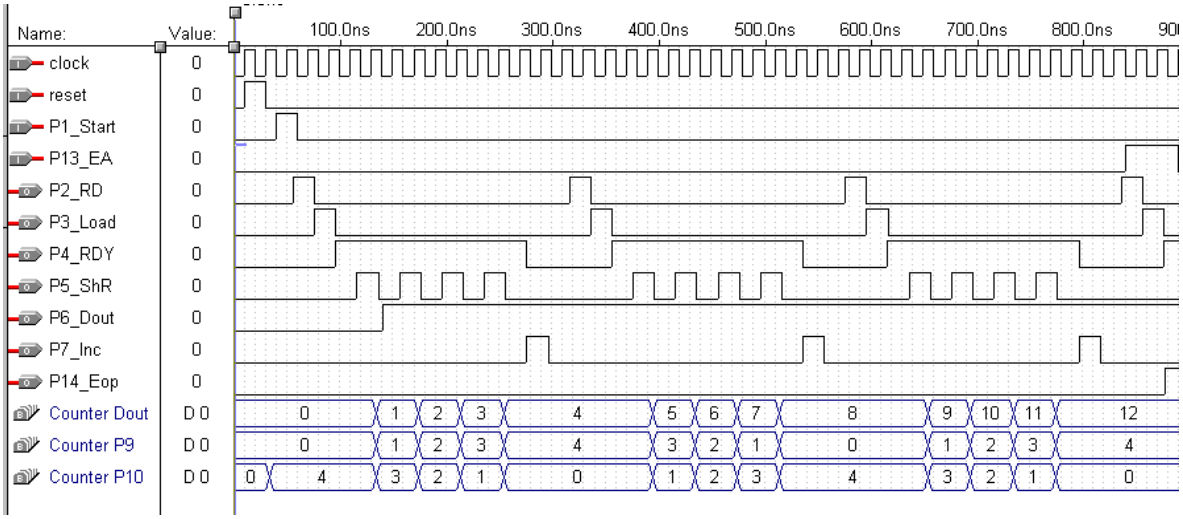

Figure 8. Simulation results.