Digital Systems Synthesis based on Direct Translation of Petri Net Model

Viorica SUDACEVSCHI, Victor ABABII, Emilian GUTULEAC, Diana PALII
Technical University of Moldova
str.Stefan cel Mare nr. 168, MD-2004 Chisinau
svm@mail.utm.md, ababii@mail.utm.md, egutuleac@mail.utm.md

***Abstract* — This paper describes a digital system design method based on direct translation of a Petri net model into an FPGA circuit netlist. A proposed CAD tool allows digital system specification, modeling validation and synthesis using ordinary Petri nets. The digital 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.**

***Index Terms* — CAD tool, digital system, direct translation, Hardware Petri Net, Petri Net model, system specification.**

I. INTRODUCTION

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 [1]. 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.

Petri nets implementation methods can be classified into two types: software and hardware. Software implementation represents the emulation of Petri nets using computer software, which usually takes a 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 run-time reconfiguration allows the use of adoptive algorithms that can reduce the time that is necessary for Petri Net simulation. Hardware implementation methods of Petri nets can be divided into direct translation methods [2,3] and logic synthesis methods [4,5]. Logic synthesis methods often suffer 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 specification.

This paper focuses on some of opportunities of Petri nets utilization in digital systems design based on direct translation of the behavioral model into 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 a digital system.

II. PETRI NET SPECIFICATION

a) Petri net definition

A *Petri net (PN)* is a 4-tuple $\left(\boldsymbol{P},\boldsymbol{T},\boldsymbol{F},\boldsymbol{M}^0\right)$ [6, 7], where:

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

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

$\boldsymbol{F}\subseteq\left(\boldsymbol{P}\times\boldsymbol{T}\right)\cup\left(\boldsymbol{T}\times\boldsymbol{P}\right)$ is a flow relation that defines directed arcs from places to transitions $\left(\boldsymbol{P}\times\boldsymbol{T}\right)\notin\varnothing$ and transitions to places $\left(\boldsymbol{T}\times\boldsymbol{P}\right)\notin\varnothing$;

$\boldsymbol{M}^0=\left\{\boldsymbol{m}_1^0,\ \boldsymbol{m}_2^0,\ ...,\ \boldsymbol{m}_N^0\right\}$ 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 transitions. 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 marking. The firing of a transition removes one token from each input place and adds one token into each output place of the transition.

b) Petri net properties

The behavioral properties of PN are very important for modeling, analyzing, verification and validation of digital systems.

A *PN* is said to be *finite* if sets *P* and *T* are finite.

A *PN* is said to be *k-bounded* if in all reachable marking no place has more than k tokens.

A 1-bounded *PN* is called a *safe PN*.

A marking *m* is said to be *reachable* from the initial marking m_0 if exists a sequence of firings σ that transforms

m_0 to 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 m_0 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 $\boldsymbol{m'}$ 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 at any of its allowable states. A *weaker* form of liveness requires a transition to be enabled at least once at some reachable marking. A reachable marking m at which no transition is enabled is called a *deadlock*. A *PN* is said to be *deadlock-free* if its reachability set includes no deadlocks. Presence of deadlocks is regarded as an error in a system which operates in cycles.

c) 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 because 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* [8]. 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* [9]. 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.

III. HARDWARE PETRI NETS MODELS

The computer-based synthesis of the digital system from Petri net level to logic design level requests the adaptation of the Petri net model to its hardware implemented model. The digital system model is considered 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. The arc from processing element $\boldsymbol{P_i}$ to processing element $\boldsymbol{T_j}$ is denoted as $\boldsymbol{P_i \mapsto T_j}$ and the arc from processing element $\boldsymbol{T_j}$ to processing element $\boldsymbol{P_i}$ is denoted as $\boldsymbol{T_j \mapsto P_i}$. In the hardware implemented *PN* model the data flow depends on the topology of the net.

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

$\boldsymbol{RPH = T \cup P \cup A^+ \cup A^- \cup A^S \cup A^T \cup A^I \cup P^{In} \cup P^{Out}}$, where:

$\boldsymbol{T = \{T_1, T_2, ..., T_L\}}$, $\boldsymbol{T \neq \varnothing}$ is a set of processing elements, called transitions;

$\boldsymbol{P = \{P_1, P_2, ..., P_N\}}$, $\boldsymbol{P \neq \varnothing}$ is a set of processing elements, called places;

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

$$\boldsymbol{A_i^+ = \begin{cases} a_{ji}^+ = 1 \mid T_j \mapsto P_i, \\ a_{ji}^+ = 0, \quad otherwise. \end{cases} \quad i = \overline{1,N},\ j = \overline{1,L};}$$

$\boldsymbol{A^- = \{A_i^-,\ i = \overline{1,N}\}}$, $\boldsymbol{A^- \neq \varnothing}$ is a set of arcs $\boldsymbol{T_j \mapsto P_i}$ that represent the condition when the number of tokens in the place is decreased and is defined as follows:

$$\boldsymbol{A_i^- = \begin{cases} a_{ji}^- = 1 \mid T_j \mapsto P_i, \\ a_{ji}^- = 0, \quad otherwise. \end{cases} \quad i = \overline{1,N},\ j = \overline{1,L};}$$

$\boldsymbol{A^S = \{A_j^S,\ j = \overline{1,L}\}}$, $\boldsymbol{A^S \neq \varnothing}$ is a set of *state arcs* $\boldsymbol{P_i \mapsto T_j}$ that determine the enable firing condition of the transition $\boldsymbol{T_j}$ related to the marking of the place $\boldsymbol{P_i}$ the arc is connected with. This set is defined as follows:

$$\boldsymbol{A_j^S = \begin{cases} a_{ij}^S = 1 \mid P_i \mapsto T_j, \\ a_{ij}^S = 0, \quad otherwise. \end{cases} \quad i = \overline{1,N},\ j = \overline{1,L};}$$

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 that connects an input place to a transition means that the transition is only enabled if the input place has a token. The firing changes the marking in the place that is connected to the state arc.

$\boldsymbol{A^T = \{A_j^T,\ j = \overline{1,L}\}}$, $\boldsymbol{A^T \neq \varnothing}$ 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 place that is connected to the test arc.

$$\boldsymbol{A_j^T = \begin{cases} a_{ij}^T = 1 \mid P_i \mapsto T_j, \\ a_{ij}^T = 0, \quad otherwise. \end{cases} \quad i = \overline{1,N},\ j = \overline{1,L};}$$

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

$$\boldsymbol{A_j^I = \begin{cases} a_{ij}^I = 1 \mid P_i \mapsto T_j, \\ a_{ij}^I = 0, \quad otherwise. \end{cases} \quad i = \overline{1,N},\ j = \overline{1,L};}$$

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 between a place and a transition means that the transition is enabled if the input place has no token. The firing does not change the marking in the place that is connected to the inhibitor arc.

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

$\boldsymbol{P^{Out}} = \left\{ \boldsymbol{P_j^{Out}},\ \boldsymbol{j} = \overline{1, \boldsymbol{L^{Out}}} \right\}, \boldsymbol{P^{Out}} \in \boldsymbol{P}$ is a set of processing elements $\boldsymbol{P_j}$ 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 $\boldsymbol{P^{In}}$ and $\boldsymbol{P^{Out}}$ (Figure 1.)

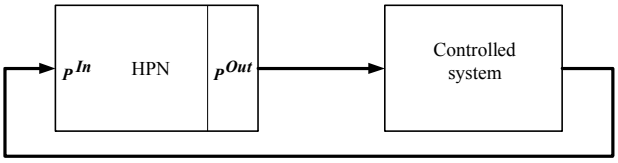


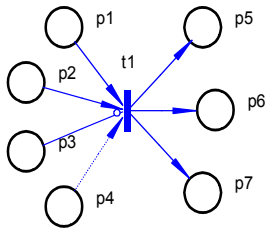
Figure 1. The interaction of the ***HPN*** model with external system.

IV. PROCESSING ELEMENTS SYNTHESIS

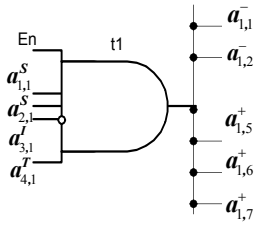
The processing element T prepares the data processing operation. After analyzing of the global state $S^k = \left\{ (m_i, P_i), \forall i = \overline{1, I} \right\}$ at the step k of data processing, the condition for step $k+1$ of data processing operation is formed.

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, P3 is connected by inhibitor arc and P_4 is connected by a test arc. The logic implementation (Figure 2 (b)) represents 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).

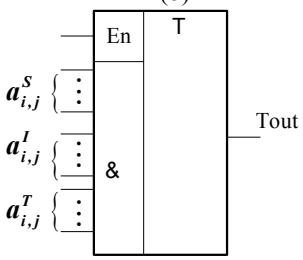
The processing element P stores the state value and performs the increment and decrement operation of the number of tokens. The increment operation occurs when one of the input transitions of the processing element P fires. The decrement operation occurs when one of the output transition of the processing 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:



(a)



(b)



(c)

Figure 2. An example of possible connections to a transition (a), logic implementation of the presented example (b) and logic symbol of the processing element Transition (c).

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

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 to the place P_i, $(\boldsymbol{m_i^{max}}\ \forall\ \boldsymbol{i} = \overline{1, \boldsymbol{N}}) \in \boldsymbol{M^{max}}$ represents 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 3(a) an example of a place with three input and three output transitions is presented. The implementation logic and logic symbol are given in Figure 3(b) and 3(c), respectively, where: *CLK* – clock signal, $SET / RESET\ (S / R)$ - an asynchronous set or reset signal to install the initial marking

M0; *Inc*- inputs used to increment the number of tokens in a place; *Dec*- inputs used to decrement the number of tokens in a place;

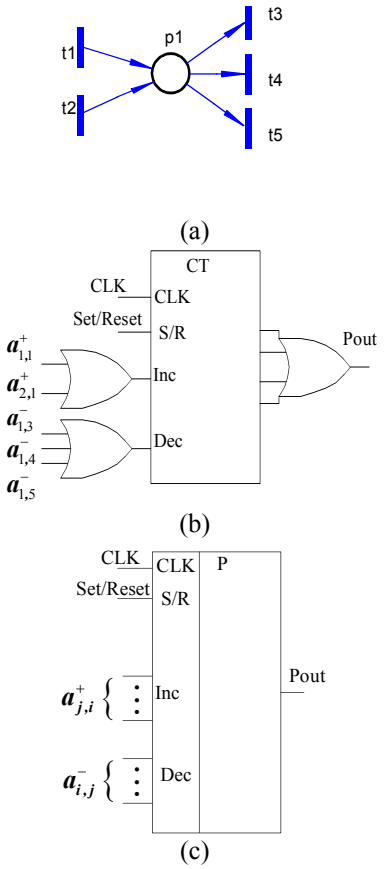


Figure 3. An example of possible connections to a place (a), logic implementation of the presented example (b) and logic symbol of the processing element Place (c).

V. DESIGN FLOW

The digital system design flow is presented in Figure 4.

Design steps description:

PN Models Source – Petri net model that is proposed for analyzing (in graphical form);

VPNP Tool - software tool that allows inserting and modifying the Petri net model interactivelly;

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 of the model such as its reachability, liveness and reversibility;

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

HDL Compiler – HDL code compilation based on matrixes 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.

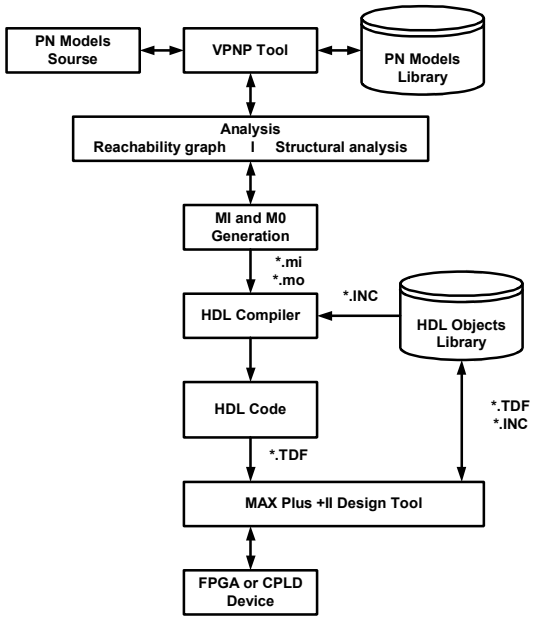


Figure 4. Design flow of the Petri net-based digital system.

VI. DESIGN EXAMPLE

As a design example a parallel to sequential code controller is introduced. A controller consists of a *RAM* for storage of data to be converted, an 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). 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 the address space. The extracted 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 *OEs* and *OEe*. When the conversion operation is finished, signal *EoP* is set.

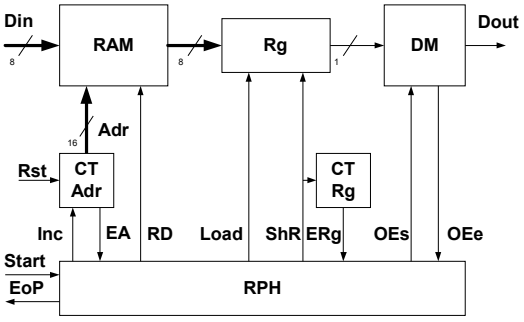


Figure 5. Parallel to sequential code controller.

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.

VII. CONCLUSION

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 behavioral specification of the digital system using 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 constrains are not critical.

REFERENCES

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

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

[3] *J. Carmona, J. Cortadella, and E. Pastor* A structural encoding technique for the synthesis of asynchronous circuits, Fundamenta Informaticae, pp. 135-154, April 2001.

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

[5] *I. Blunno and L. Lavagno.* Automated synthesis of micro-pipelines from behavioural Verilog HDL, Proc. of IEEE Symp. on Adv. Res. in Async. Cir. and Syst. (ASYNC 2000), IEEE CS Press, pp. 84-92.

[6] *Murata T.* Petri nets: Properties, analysis and application, Proceedings of IEEE, 77(4):541-574, April 1989.

[7] *Peterson J.L.* Petri Net Theory and the Modeling of Systems, Prentice Hall, 1981.

[8] *A. Valmari,* Stubborn attack on state explosion, Formal Methods in System Design, 1: 297-322, 1991.

[9] *E. Pastor, O. Roig, J. Cortadella and R. Badia,* Petri net analysis using boolean manipulation, 15th International Conference on Application and Theory of Petri Nets, pages 416-435, Zaragoza, Spain, June 1994.

[10] *McMillan K.L.,* Symbolic Model Checking, Kluwer Academic Publishers, Boston, 1993.

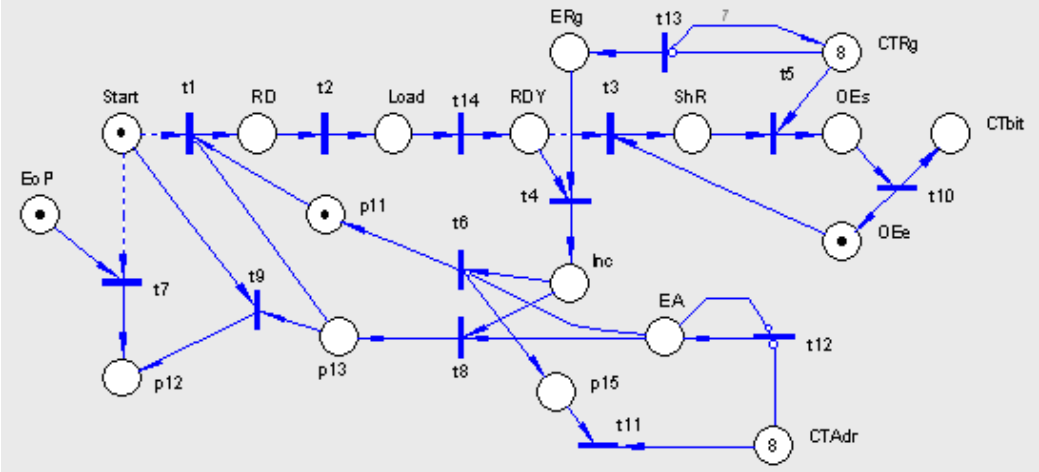


Figure 6. Petri net model for parallel to sequential data conversion.

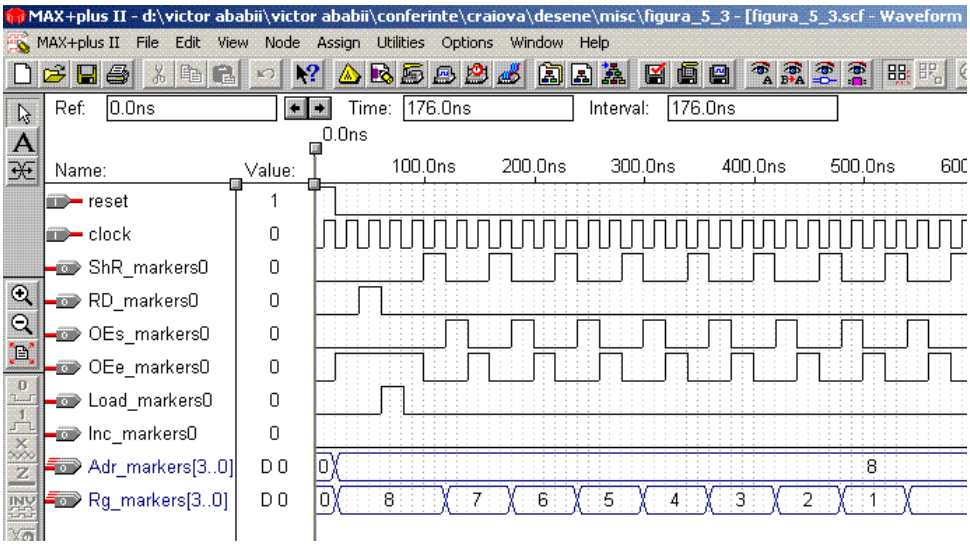


Figure 7. Simulation results.