BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 64, No. 3, 2016
DOI: 10.1515/bpasts-2016-0052

Dual synthesis of Petri net based application specific logic controllers with increased safety

J. TKACZ*, A. BUKOWIEC, and M. ADAMSKI

University of Zielona Góra, 9 Licealna St., 65-417 Zielona Góra, Poland

Abstract. In the paper, design flow of the application specific logic controllers with increased safety by means of Petri nets is proposed. The controller architecture is based on duplicated control unit and comparison results from both units. One specification of control algorithm is used by means of Petri net for both units. The hardware duplication is obtained during dual synthesis process. This process uses two different logic synthesis methods to obtain two different hardware configurations for both control units. Additionally, the dual verification is applied to increase reliability of the control algorithm. Such design flow simplifies the process of realization of control systems with increased safety.

Key words: critical safety, FPGAs, logic controllers, logic synthesis, Petri nets, verification.

1. Introduction

The paper presents some effective techniques used for rigorous computer-based design, synthesis and verification of application specific logic controllers (ASCLs) [1–4], which are dedicated to safety critical application. Typically, reliability is increased if algorithms of control system are designed by different teams and with the use of different methods. It requires the creation of two specifications and their independent verification. But the goal is achieved by means of two different hardware structures and not two different models. The proposed approach simplifies this process by the application of two different logic synthesis methods for one model. One specification is created by one team and then it is verified. Next, it undergoes dual synthesis. This approach also reduces the costs of the design process because only one design team is required.

Algorithm of control unit of digital systems can be described in several forms, e.g. LDs [5, 6], SFCs [7], FSMs [8], statecharts, Petri nets [1, 9], or HDLs [6, 8, 10]. However, for safety critical applications only very well verified methods can be applied [10, 11, 12]. Petri nets, cause-effect tables, decision tables and HDL models with simple construction can be accepted. In presented solution the link between Petri nets and HDL models has been applied, in which the behavior of the controller is specified as hierarchical control interpreted Petri [13, 14] and HDL models are used for simulation, synthesis and implementation. The Petri net was chosen as specification because there are many formal analytic mathematical algorithms available for its analysis, which cannot be achieved for other notations. The Petri net can be used as an intermediate model because there are algorithms for formal model transformation (QVT) available from UML diagrams such as statecharts or SFC. There are also academic tools that allow to specify control algorithms directly as Petri net in case of rapid prototyping. A novel methodology of implementation of concurrent controllers with increased safety is based on dual synthesis of the same Petri net model with the use of different synthesis methods. The first synthesis method is based on modular and symbolic encoding during the logic synthesis [15]. The second one is based on architectural decomposition of a logic circuit [16] and decomposition of Petri net into state machine subnets [16, 17], in which the embedded memory blocks are utilized during synthesis process [4, 8, 16–18]. As a result of both synthesis methods, HDL models are obtained. They will be used for further simulation and implementation. The design process is supplemented by dual verification. It is based on the on simulation of both models and results comparison [10, 19]. Additional methods of testing and analysis [20] can extend verification process at the earlier stage.

The paper consists of nine sections. After Introduction, in Section 2, the theoretical background for control colored interpreted Petri net is presented. Section 3 shows sample application of ASCL which is used as an example in the following sections. The architecture of control system is presented in Section 4 and whole design flow is proposed in Section 5. Then the algorithms of coloring of the Petri net (Sec. 6) and dual synthesis (Sec. 7), including dual verification (Sec. 7.3), are presented. Section 8 shows implementation of the proposed control system with increased safety based on the case study of mixing and transportation of liquid substances. Results of the simulation of the obtained circuit have been also presented. Finally, Section 9 concludes the paper.

2. Control colored interpreted Petri net

2.1. Place-transition net. A simple Petri net [20, 14] is defined as a triple

$$PN = (P, T, F), \tag{1}$$

where:
P is a finite non-empty set of places, $P = \{p_1, \ldots, p_M\}$
T is a finite non-empty set of transitions, $T = \{t_1, \ldots, t_S\}$

*e-mail: j.tkacz@imei.uz.zgora.pl

F is a set of arcs from places to transitions and from transitions to places:

$$\begin{aligned} F &\subseteq (P \times T) \cup (T \times P), \\ P \cap T &= \varnothing. \end{aligned}$$

Sets of input and output transitions of a place $p_m \in P$ are defined respectively as follows:

$$\begin{aligned} \bullet p_m &= \{t_s \in T : (t_s, p_m) \in F\}, \\ p_m \bullet &= \{t_s \in T : (p_m, t_s) \in F\}. \end{aligned}$$

Sets of input and output places of a transition $t_s \in T$ are defined respectively as follows:

$$\begin{aligned} \bullet t_s &= \{p_m \in P : (p_m, t_s) \in F\}, \\ t_s \bullet &= \{p_m \in P : (t_s, p_m) \in F\}. \end{aligned}$$

A marking of a Petri net is defined as a function:

$$M : P \to \mathrm{N}.$$

It describes a number of tokens $M(p_m)$ situated in a place p_m. When a place or a set of places contains a token, it is marked. A transition t_s can be fired if all its input places are marked. Firing of a transition removes tokens from its input places and puts a token in each outputplace. The initial marking M_0 can be specified and the Petri net is defined as follows:

$$PN = (P, T, F, M_0). \tag{2}$$

2.2. Interpreted Petri net. An interpreted Petri net is the one enhanced with feature for information exchange [1, 20]. This exchange is made by use of binary signals. Interpreted Petri nets are used as models of concurrent logic controllers [21]. Two types of interpreted Petri nets can be distinguished: Moore type and Mealy type.

The Boolean variables occurring in the interpreted Petri net can be divided into three sets:
X is a set of input variables, $X = \{x_1, \ldots, x_L\}$,
Y is a set of output variables, $Y = \{y_1, \ldots, y_N\}$,
Z is a set of internal communication variables, typically it is not used and $Z = \varnothing$.
An interpreted Petri net has a condition φ_s associated to every transition t_S. The condition φ_s is defined as Boolean function of the input or internal variables form sets X and Z. In particular case the condition φ_s can be equal to 1 (always true). Now, transition t_S can be fired if all its input places ($\bullet t_S$) are marked and current value of corresponding Boolean function φ_s is equal to 1. Interpreted Petri nets of Mealy and Moore type, like FSMs, differs in the method of output signals generation. In case of Moore type interpreted Petri net, ψ_m is an elementary conjunction of affirmation of some output variables form the set Y. Each such conjunction ψ_m is associated to place p_m. If the place p_m is marked the output variables from corresponding conjunction ψ_m are being set otherwise they are being reset. In case of Mealy type interpreted Petri net ψ_s is an elementary conjunction of affirmation or negation of some output variables form the set Y. When transition t_S is fired variables from corresponding conjunction ψ_s are being set if their affirmation belongs to this conjunction and they are being reset if their negation belongs to this conjunction. The value of non used variables in corresponding conjunction ψ_s remain unchanged.

2.3. Interpreted colored Petri net. A Petri net can also be enhanced by assigning colors to places and transitions [14, 22]. Such Petri net is called colored Petri net or colored interpreted Petri net if both enhancements are applied [1]. These colors help to intuitively and formally validate the consistency of all sequential processes in the Petri net under consideration. Each color recognizes one state machine module. The rules for Petri net coloring are as follows [1, 21]:

- each place and transition must have at least one color,
- if the place has a color each of its input and output transitions must have the same color,
- input places of each transition must hold different colors,
- output places of each transition must hold different colors,
- input and output places of transition must share the same set of colors,
- initially marked places cannot share exactly the same set of colors,
- number of different colors which are shared by the places initially marked is equal to the total number of colors.

Control interpreted Petri net is defined as colored interpreted Petri net that is safe and live. The Petri net is safe if it is one-bounded. k-bounded Petri net does not contain more than k tokens in all reachable markings, so one-boundedness means that each place p_m can contain only one marker. The Petri net is live when all transitions are live. A transition t_S is live if for any marking M', reachable from initial marking M_0, a sequence of transitions exists friable from M' which contains transition t_S [23].

3. Sample application for logic controller

The logic controller has ten inputs ($X = \{x_0, \ldots, x_9\}$) and ten outputs ($Y = \{y_0, \ldots, y_9\}$). Input x_0 is driven by start button. The controlled object (Fig. 1) consists of five tanks and pumps, which are fitted with valves (y_1–y_5) and equipped with level sensors (x_1–x_6) to detect their "fill" and "empty" states. The main tank, which serves as a chemical reactor is integrated with a stirrer (y_7). The product is transported by a tank-carriage (movement left – y_9, right – y_8. Output y_0 says that chemical installation is ready for work. The tank-carriage is filled in the left station (x_7) and emptied in the right station (x_8). Its level sensor (x_9) triggers detection if the carriage is empty. Boolean expressions called guards describe the external conditions for transitions to be enabled. In the example in figure 2 [20] the Petri net places (p_1, p_2, and p_{16}) stands for the local states of concurrent state machine with inputs from the set X and outputs from the set Y. The transitions $T = \{t_1, \ldots, t_9\}$ describe global

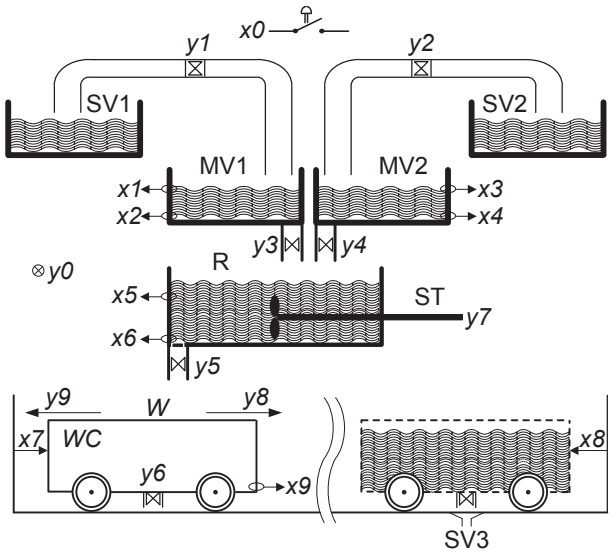


Fig. 1. Object of discrete control system

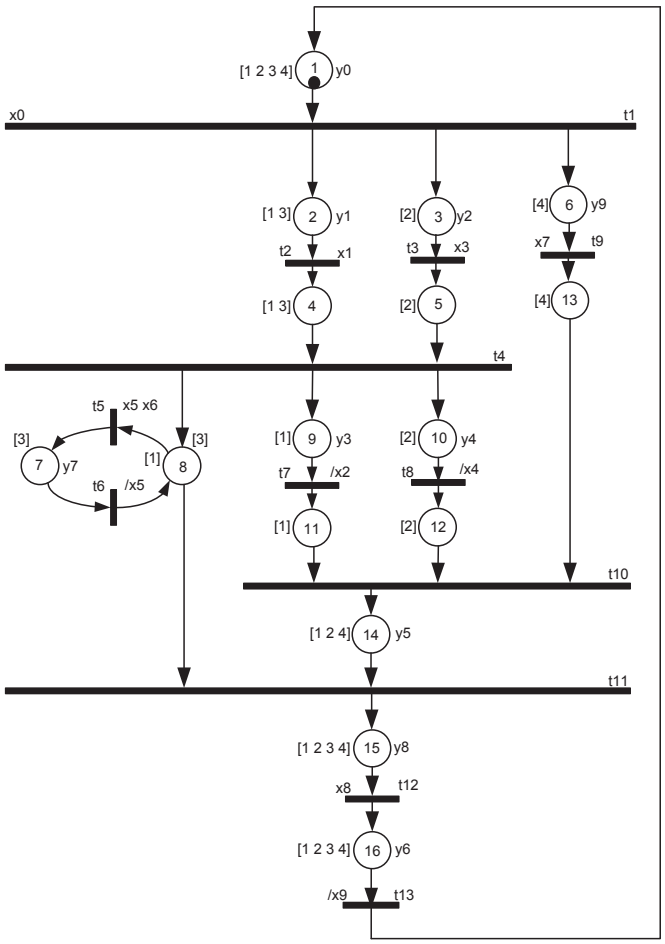


Fig. 2. Control interpreted Petri net

state changes in terms of local changes of the partial states in the Petri net space. The implicit configuration (coordination) places $MP = \{mp_1, \ldots, mp_{11}\}$ detect the Petri net subnets which they dominate. The basic net was reduced to the macronet with macroplaces MP. Transitions with more than one input place or more than one output place, such as t_1, t_4, t_{10}, t_{11}, and t_{13} are called border transitions. Other transitions with one input and one output places are hidden inside first order macroplaces mp_1–mp_7 and mp_9. The macroplaces mp_1 and mp_2 are parallel and form second order macroplace mp_{10}. The second order macroplace mp_{11} is built from two parallel macroplaces mp_4 and mp_5. The configuration place mp_8, which can be redundant during next steps of logic design, points to the third order subnet. The configuration places $\{mp_3, mp_6, mp_7, mp_9, mp_{10}, mp_{11}\}$ are sufficient to represent the activity of all Petri net places, which are dominated by them.

4. Logic controller with increased safety architecture

In the definition [11], the outputs of device with increased safety should be set in the known state in case of accident. Secondly, the device should work properly in normal conditions. This means that ASLC should be designed without any errors and should be able to detect any self-accident. The condition of controller safety state should be secured by properly designed hardware. The known safety state should be achieved automatically after self-accident detection, regardless of algorithm realization by ASCL.

Presented design of ASCL with increased safety is based on duplicated control units architecture (Fig. 3). The duplication of control units is applied in order to detect hardware failure. The similar approach was introduced in [11, 19, 24] for design of PLC. Both control units, A and B, are obtained during dual synthesis and implementation processes from the same Petri net model. For each control a different synthesis method is applied, which is described in next sections. It means that both control

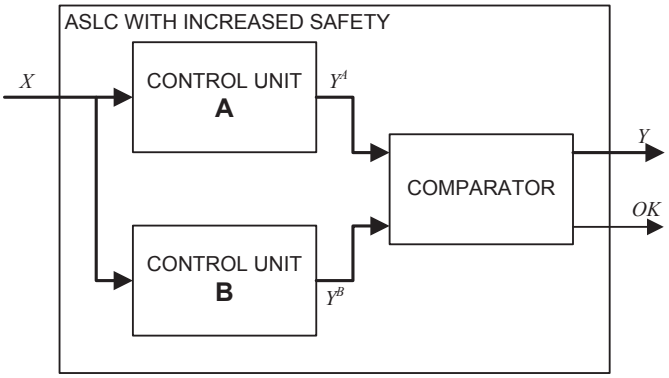


Fig. 3. Architecture of ASCL with increased safety

units realized the same algorithm. They are also connected to the same input signals X. The output signals Y^A and Y^B are compared in the comparator unit. If the comparison is correct then the OK status signal is generated and output signals Y are set to correct values, otherwise output signals Y are reset and the OK status signal is also reset – it indicates an error.

5. Design flow for dependable logic controller with increased safety

Special design flow is proposed (Fig. 4) for such ASCL with increased safety.The entry to the process is well designed control algorithm as a control interpreted Petri net. The Petri net should be initially colored. The coloring algorithm is proposed in next section but different ones can be also considered. First, the dual synthesis is performed. In this process independently two different synthesis methods are used. Each method generates logic description of the control algorithm in hardware description language (HDL). This description is platform-indepen-

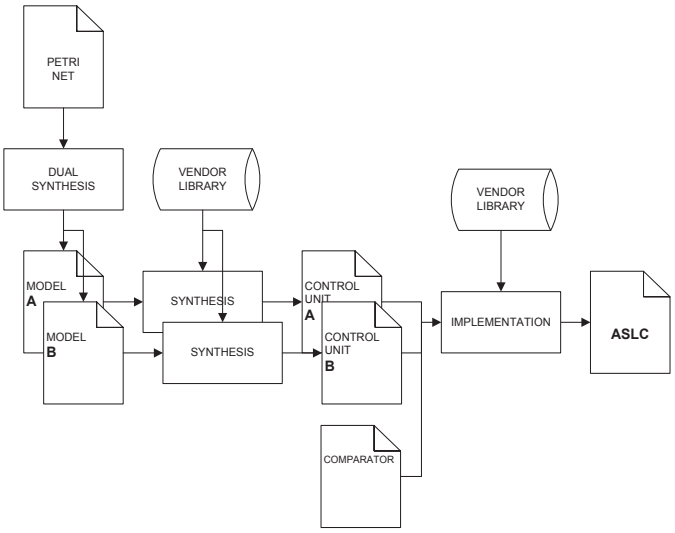


Fig. 4. Design Flow for ASCL with increased safety

dent at this stage. The first one is based on modular microstate encoding and logic description in obtained single module. The second one is based on architectural and parallel decomposition and logic description is divided into several modules. Additionally, to increase reliability, generated models are described in different HDLs – VHDL and Verilog. Then, both models can be passed transferred to the third party synthesis tools. At this stage the device has to be chosen and a suitable vendor library has to be used. Both modules have to be synthesized separately as black boxes. After this, the top level module and comparator should be created. The top level instantiates black boxes of both modules and comparator. Such design is transferred to undergo implementation process. After that the bitstream is obtained.

6. Algorithm of Petri net coloring

There are many different coloring algorithms [20, 22, 25], including:

- manual coloring during specification,
- coloring based on topological structure of the net,
- coloring from concurrency hypergraph made with the use of deduction methods,
- coloring with the use of siphons and traps,
- coloring of discrete space of the net.

Most of them (excluding the manual coloring) can be fully automated and included into design process of ASCL. They discover all possible colorings and for most cases the complexity and time of execution is not acceptable for application in design tools. These methods are typically used for the purpose of verification of heuristic methods. In this case, in respect of the design process of ASCL the best solution is to apply one of heuristic methods of coloring which gives one of the possible satisfied solutions. We decide to apply the heuristic method with backwards in our design process of ASCL.

During the reduction procedure a Petri net is converted into a more general hierarchical description [26]. The following two reduction procedures are the most useful: *Fusion of Series Places* (FSP) and *Fusion of Parallel Places* (FPP) [1, 2, 9, 20–22]. Starting from FSP both techniques are used recursively until the macronet becomes irreducible. The second order macronet is presented in figure 5. It can be colored. It should be noted, that

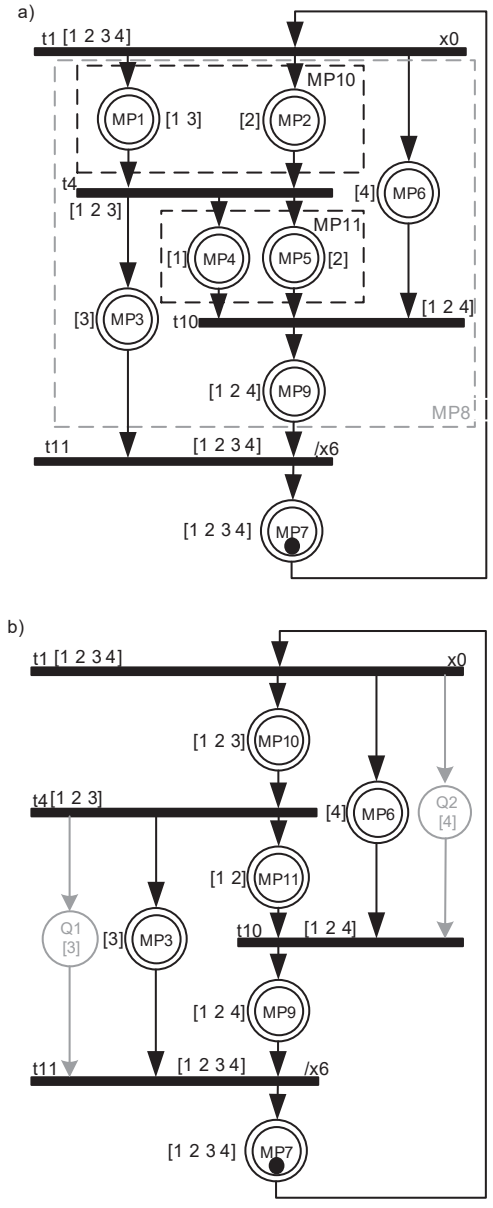


Fig. 5. Colored hierarchical macronet

the macroplaces which are painted with disjoint set of colors are concurrent to each other. The macroplaces sharing the same color are sequentially related to each other.

The recursive SM-coloring with backwards is done according to the rules described in chapter 2.3. The colors $C = \{c_1, \ldots, c_4\}$ (in pictures marked as $[i]$, where i is an index of color), which paints the Petri net places in figure 2 separates four partially overlapping concurrent sequences of local state changes.

The recently developed computer system *iCPN* efficiently reduces the net as well as finds the suitable SM cover of the net from topological structure of interpreted Petri net graph described in the general PNML format. As an another option formal reasoning about Petri net space, which is described in Gentzen propositional logic, makes it possible for the option to paint the net with minimal number of colors.

7. Algorithms of synthesis of Petri net

The automated design flow starts with dual synthesis of designed and initially colored Petri net. Two different synthesis methods are proposed in order to receive two different logic description of the same algorithm. The different logic descriptions are used to implement two control units of ASLC with increased safety. Because outputs of both control units are compared it is required that they produce their outputs at the same time. To satisfy this condition the HDL models have to be prepared in a proper way:

- The state registers of both control units have to be trigged by the same edge of the same clock signal. In proposed solution the rising edge is chosen.
- The outputs have to be registered to avoid mismatch in comparison. The output register usees the same clock that the state register does but it is trigged by the opposite edge instead of registers. It allows to generate outputs in one clock cycle. In proposed solution the falling edge has to be used.
- Both control units have to be reset in the same way. Because the state registers are topically built from standard D-type flip-flops the asynchronous reset is used. For the output register the synchronous reset should be always used in order to generate outputs in the same period of time.

7.1. Monolithic modular logic synthesis. Proposed synthesis method allows to implement control algorithm described by Petri net as a modular logic circuit (Fig. 6). The modular transition coder (MTC) is responsible for the generation of events. The register RG_Q holds encoded global and local states. The Q signal represents encoded global states and the signal Y_P represents encoded by output signals local states. The register RG_Y is used for synchronization purpose and it holds values of output signals. It is trigged by opposite edge instead of the register RG_Q.

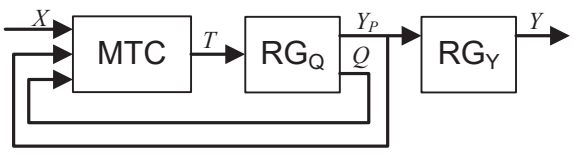


Fig. 6. Modular logic circuit of Petri net

Proposed macrostate encoding is done at second level of hierarchy by means of using the minimal number of coordinates Q_1 and Q_2 (Fig. 5b):

$$\begin{aligned} mp_3 &= Q_1; & mp_{11} &= Q_1 \wedge Q_2; \\ mp_6 &= Q_2; & mp_9 &= Q_1 \wedge \overline{Q_2}; = y_5; \\ mp_{10} &= \overline{Q_1} \wedge Q_2; & mp_7 &= \overline{Q_1} \wedge \overline{Q_2}; \end{aligned}$$

Encoded macrostate changes are described only with use of selected macrostete codes and its boundary transitions:

$$\begin{aligned} @Q_1 &\Leftarrow Q_1 \oplus (t_4 \oplus t_{11}); \\ @Q_2 &\Leftarrow Q_2 \oplus (t_1 \oplus t_{10}); \end{aligned}$$

For simplicity, local places encoding is based on registered controller outputs:

$$\begin{aligned} p_1 &= y_0;\ p_2 = y_1;\ p_3 = y_2; \\ p_4 &= \overline{y_1};\ p_5 = \overline{y_2};\ \ldots;\ p_{16} = y_6; \end{aligned}$$

Excitations of boundary transitions depend on input macroplaces, local places and guards. After substitution of coding terms for macroplaces and local places the expressions takes a form:

$$\begin{aligned} t_1 &\Leftarrow mp_7 \wedge p_1 \wedge x_0 = \overline{Q_1} \wedge \overline{Q_2} \wedge y_0 \wedge x_0; \\ t_4 &\Leftarrow mp_{10} \wedge p_4 \wedge p_5 = \overline{Q_1} \wedge Q_2 \wedge \overline{y_1} \wedge \overline{y_2}; \\ t_{10} &\Leftarrow mp_{11} \wedge mp_6 \wedge p_{11} \wedge p_{12} \wedge p_{13} \\ &\quad = Q_1 \wedge Q_2 \wedge \overline{y_3} \wedge \overline{y_4} \wedge \overline{y_9}; \\ t_{11} &\Leftarrow mp_3 \wedge mp_9 \wedge p_8 \wedge p_{14} \wedge \overline{x_6} \\ &\quad = \wedge \overline{Q_2} \wedge \overline{y_7} \wedge \overline{y_5} \wedge \overline{x_6}; \end{aligned}$$

Given as an example the local transitions for macroplace mp_{10} are:

$$\begin{aligned} t_2 &\Leftarrow mp_{10} \wedge p_2 \wedge x_1 = \overline{Q_1} \wedge Q_2 \wedge y_1 \wedge x_1; \\ t_3 &\Leftarrow mp_{10} \wedge p_3 \wedge x_3 = \overline{Q_1} \wedge Q_2 \wedge y_2 \wedge x_3; \end{aligned}$$

Local state changes for places nested in macroplace mp_{10} are as follows:

$$\begin{aligned} @p_2 &\Leftarrow (p_2 \wedge mp_{10}) \oplus (t_1 \oplus t_2); \\ @p_3 &\Leftarrow (p_3 \wedge mp_{10}) \oplus (t_1 \oplus t_3); \\ @p_4 &\Leftarrow (p_4 \wedge mp_{10}) \oplus (t_2 \oplus t_4); \\ @p_5 &\Leftarrow (p_4 \wedge mp_{10}) \oplus (t_3 \oplus t_4); \end{aligned}$$

For description of local state changes inside macroplace mp_{10} after encoding only two equations are required:

$$\begin{aligned} @y_1 &\Leftarrow (y_1 \wedge \overline{Q_1} \wedge Q_2) \oplus (t_1 \oplus t_2); \\ @y_2 &\Leftarrow (y_2 \wedge \overline{Q_1} \wedge Q_2) \oplus (t_1 \oplus t_3); \end{aligned}$$

The logic circuit is described in single HDL model in VHDL (Fig. 7). According to the architecture presented in the figure 6 the MTC is described with the use of continuous assignments, the register RG_Q is described by the FF process, and the RG_Y register is described by the YY process. Preconditions of global and local transitions are described as simple continuous assignments. The FF process is responsible for the generation of codes of next macro and local states. Because the local states are encoded with the use of output variables there are declared internal copies of these variables, then in the YY process they are synchronously transferred (according to the requirement) to the outputs of control unit. This process is trigged by the opposite edge instead of the FF process. It allows outputs to be generated in one clock cycle without any glitches.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity PN is
port (CLK, RES: in std_logic;
X: in std_logic_vector(9 downto 0);
Y: out std_logic_vector(9 downto 0) );
end PN;

architecture reactor_t of PN is

signal Q1, Q2: std_logic;
signal Y_0, Y_1, Y_2, ..., Y_9: std_logic;
signal T1, T2, T3, ..., T13: std_logic;
begin

--Precondition of global transitions
T1 <= not Q1 and not Q2 and X(0) and Y_0;
T4 <= not Q1 and Q2 and not Y_1 and not Y_2;
T10 <= Q1 and Q2 and not Y_3 and
       not Y_4 and not Y_9;
T11 <= Q1 and not Q2 and Y_5 and not Y_7;

--Precondition of local transitions
T2 <= not Q1 and Q2 and Y_1 and X(1);
T3 <= not Q1 and Q2 and Y_2 and X(3);
T5 <= Q1 and not Y_7 and X(5) and X(6);
...
T13 <= not Q1 and not Q2 and Y_6 and not X(9);

FF:process (CLK,RES) -- Transition firing
begin
   if RES = '1' then
     Q1 <= '0'; Q2 <= '0';
     Y_0 <= '1'; Y_1 <= '0'; ...; Y_9 <= '0';
   elsif rising_edge(CLK) then
     Q1 <= Q1 xor (T4 xor T11);
     Q2 <= Q2 xor (T1 xor T10);

     Y_0 <= (Y_0 and not Q1 and not Q2)
            xor (T13 xor T1);
     Y_1 <= (Y_1 and not Q1 and Q2)
            xor (T1 xor T2);
     Y_2 <= (Y_2 and not Q1 and Q2)
            xor (T1 xor T3);
     ...
     Y_9 <= (Y_9 and Q2) xor (T1 xor T9);
   end if;
end process;

YY: process(CLK)
begin
   if falling_edge(CLK) then
     if RES = '1' then
       Y <= (others => '0');
     else
       -- Outputs
       Y <= Y_9 & Y_8 & ... & Y_0;
     end if;
   end if;
end process;

end reactor_t;
```

Fig. 7. Part of VHDL file

7.2. Architectural distributed synthesis. The entry point to the synthesis method is colored interpreted Petri net of Moore type. This could be achieved by symbolic Petri net coloring from hypergraph of concurrency described in previous section. Places are encoded in concurrent subsets. Since subsets are state machine type, they are recognized by one particular color of colored Petri net. Operations assigned to places are stored in single memory. The memory could be of partitioned into flexible concurrent parts [17]. The logic circuit is realized as distributed two-level architecture (Fig. 8), where the combinational circuits (CC^i) of first level are responsible for the generation of excitation functions:

$$D^i = D^i(X, Q), \tag{3}$$

where $Q = Q^1 \cup Q^2 \cup \ldots \cup Q^I$ is set of variables used to store the codes of marked places. The memory of the circuit is build from I D-type registers RG^i which hold a current state of each subnet. Where, $i = 1, 2, \ldots, I$ and I is a number of colors in a Petri net. The second level common decoder Y is responsible for the generation of operations and it is implemented using memory blocks. Its functionality can by described by function:

$$Y = Y(Q). \tag{4}$$

Such approach allows for balanced usage of different kind of resources available in modern FPGA devices like LUTs and embedded memories.

The entry point to the synthesis method is the colored interpreted Petri net of Moore type with extracted SM subnets. This could be achieved by symbolic Petri net coloring from hypergraph of concurrency described in previous section. The whole synthesis process includes following steps:

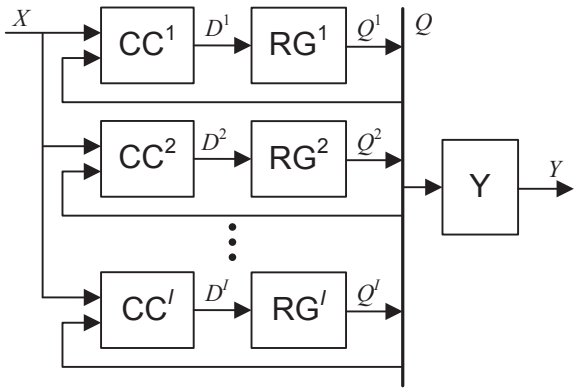


Fig. 8. Distributed logic circuit of Petri net

Encoding of places. The purpose of this step is to assign a binary code to each place. The encoding is being done on minimal number of required bits. One-hot encoding [13] is not acceptable in this method because the place code is also an address of operation memory. Let assume that Petri net is colored with I different colors. Places are encoded separately in each subnet, determined by one color. Let start from the first color ($i = 1$) and assign binary codes to each place colored by it according to the following rules:

- If all places belonging to the C_i color create a marked cycle (marked cycle is a cycle where one place, which belongs to it, is always marked) it is required to use

$$R_i = \lceil \log_2 |P_i| \rceil \tag{5}$$

  bits to encode them, where $P_i \subseteq P$ is a set of places colored by C_i color. Otherwise it is required to use:

$$R_i = \lceil \log_2 (|P_i| + 1) \rceil \tag{6}$$

  bits. Variables form set $Q^i \subset Q$ are used to store this code, where $Q = \{q_0, \ldots, q_{R-1}\}$, where

$$R = \sum_{i=1}^{I} R_i \tag{7}$$

  and $Q^i = \{q_{\rho-R_i}, \ldots, q_{\rho-1}\}$, where $\rho = \sum_{t=1}^{i} R_i$.
- If places do not create a marked cycle the encoding of places starts form code with value 1. The code with value 0 is reserved for situation when none place $p \in P_i$ is marked. Otherwise encoding starts from code with value 0.
- If places create a marked cycle, the place p that is initially marked ($p \in P_i \wedge p \in M_0$) is encoded with code with value 0. If places do not create a marked cycle but any place p is also initially marked it should be encoded with code with having $2^{R_i} - 1$ values. This rule is required to ensure suitable method of logic circuit resetting.

Then, let encode places colored by following colors. Before encoding places colored with next color, first, all places that have already assigned a code are removed from corresponding set P_i:

$$P_i = P_i \setminus \bigcup_{\iota=1}^{i-1} (P_\iota \cap P_i). \tag{8}$$

Then reaming places are encoded according to the same rules as described above.

For example the Petri net the encoding starts from the color C_1. First considered color always creates marked cycle because we do not remove any places from the set P_1. So, it is required to use $R_1 = 3$ bits, according to (5), to encode these places. And possible encoding of these places is shown in table 1. Next, places colored by the color C_2 can be encoded. Because places p_1, p_{14}, p_{15}, and p_{16} from the set P_2 are already encoded because they are in the set P_1, they are removed from the set P_2, and now $P_2 = \{p_3, p_5, p_{10}, p_{12}\}$, according to (8). Now, remaining places do not create marked cycle, so there is required to use $R_2 = 3$ bits, according to (6), to encode these places, and encoding starts from the code 001 (Table 1). In the similar way there places colored by colors C_3 and C_4 are encoded (Table 1).

Table 1
Codes of places of Petri net

Color C_1		Color C_2		Color C_3		Color C_4	
Place	Code $q_2q_1q_0$	Place	Code $q_5q_4q_3$	Place	Code q_7q_6	Place	Code q_9q_8
p_1	000	p_3	001	p_7	01	p_6	01
p_2	001	p_5	010	p_8	10	p_{13}	10
p_4	010	p_{10}	011				
p_9	011	p_{12}	100				
p_{11}	100						
p_{14}	101						
p_{15}	110						
p_{16}	111						

Formation of conjunctions. Conjunctions describe places and transitions. They are needed for easier creation of equations that describe systems (3). The conjunction describing the place p_m consists of affirmation or negation of variables q_r that are used to store the code of this place. If the code has 0 on the r-th bit then negation is used and if it has 1 then affirmation is used. The conjunction describing the transition t_S consists of place conjunctions of input places to this transition and a condition φ_s assigned to this transition.

For the sample Petri net, they are denoted as:

$$\begin{aligned}
p_1 &= \overline{q_2} \wedge \overline{q_1} \wedge \overline{q_0}, \\
p_2 &= \overline{q_2} \wedge \overline{q_1} \wedge q_0, \\
p_3 &= \overline{q_5} \wedge \overline{q_4} \wedge q_3, \\
p_4 &= \overline{q_2} \wedge q_1 \wedge \overline{q_0}, \\
p_5 &= \overline{q_5} \wedge q_4 \wedge \overline{q_3}, \\
p_6 &= \overline{q_9} \wedge q_8, \\
p_7 &= \overline{q_7} \wedge q_6, \\
&\ldots, \\
p_{16} &= q_2 \wedge q_1 \wedge q_0.
\end{aligned}$$

Transition conjunctions are denoted as:

$$\begin{aligned}
t_1 &= p_1 \wedge x_0, \\
t_2 &= p_2 \wedge x_1, \\
t_3 &= p_3 \wedge x_3, \\
t_4 &= p_4 \wedge p_5, \\
t_5 &= p_8 \wedge x_5 \wedge x_6, \\
&\ldots, \\
t_{13} &= p_{16} \wedge \overline{x_9}.
\end{aligned}$$

Formation of logic equations. Logic equations describe functions (3) of combinational circuits CC_i. They are created base on D flip-flop equation and they are build of transition conjunctions and place hold conjunctions. If the variable q_r is set to 1 in the code of the place p then the sum of corresponding variable D_r consists of transition conjunctions of all its input transitions

and the place p hold conjunctions. The place p_m hold conjunction consists of negation of sum of transition conjunctions of all its output transitions and its place conjunction.

For example, the equation for D_0 is denoted as:

$$\begin{aligned} D_0 &= t_1 \vee (\overline{t_2} \wedge p_2) \\ &\vee \quad t_4 \vee (\overline{t_7} \wedge p_9) \\ &\vee \quad t_{10} \vee (\overline{t_{11}} \wedge p_{14}) \\ &\vee \quad t_{12} \vee (\overline{t_{13}} \wedge p_{16}). \end{aligned}$$

Because the variable q_0 is set to 1 in the code of places p_2, p_9, p_{14}, and p_{16} the sum of equation D_0 consists of all input transition conjunctions of these places. In this case they are: t_1, t_7, t_{11}, and t_{13}. Additionally, there have to be added hold condition for all places in the case if place code has to be stored for longer period than one clock cycle. This hold condition is denoted as conjunction of sum of negation of all output transition conjunctions and place conjunction, and, for example, for the place p_2 it is denoted as $\overline{t_2} \wedge p_2$. In the similar way the remaining equations from D_1 to D_9 are formed. Of course, these equations can be minimized after putting conjunctions instead of corresponding variables. But this manipulation will be done automatically during synthesis & implementation process.

Formation of memory content. The memory content can be described as table or as equations according to the system (4). The table consists of two columns. First column is an address and it is described by variables $q_r \in Q$. It should be mentioned that these variables represent superposition of codes of all states from all subnets, not only allowed markings. The second column is an operation. The operation is represented by output variables from the set Y. It should only set these variables that are in elementary conjunctions ψ_m that are associated to the places included in superposition of current address. This table is very often long because it has 2^R lines. The other way to describe this memory is the creation of one logic equation to describe each output variable. It describes output variable as a sum of place conjunctions of places corresponding to the elementary conjunctions ψ_m that consist of corresponding output variable.

Because the table describing memory for the example Petri net should have 1024 lines it was decided to create equations to describe each output. In this case, they are denoted as:

$$\begin{aligned} y_0 &= p_1, \\ y_1 &= p_2, \\ &\ldots, \\ y_9 &= p_6. \end{aligned}$$

The transformation of equations into the memory table will be done automatically during the synthesis & implementation process.

Formation of logic circuit and implementation. This step describes the rules of creation of Petri net HDL model in Verilog and its implementation into FPGA device. Bottom-up approach is applied. Places and transitions conjunctions can be described using standard bit-wise operators. Then logic equations can be described with the use of these conjunctions also using bit-wise operators. There should be created a separate module for each circuit CC^i with inputs X and Q and outputs D^i. As place and transition conjunctions are going to be used in several files one include file *cons.vh* is created that defines their equations (Fig. 9). Then, modules describing combinational circuits using `assign` command are created (Fig. 10).

```
‘define p1 ~Q[2]&~Q[1]&~Q[0]
‘define p2 ~Q[2]&~Q[1]&Q[0]
...
‘define p16 Q[2]&Q[1]&Q[0]

‘define t1 ‘p1&X[0]
‘define t2 ‘p2&X[1]
...
‘define t13 ‘p16&~X[9]
```

Fig. 9. Part of cons.vh file

```
‘include "cons.vh"

module CC1(X, Q, D);
    output [2:0] D;
    input [9:0] X;
    input [9:0] Q;

    assign D[0] = ‘t1 | ~(‘t2)&‘p2
                | ‘t4 | ~(‘t7)&‘p9
                | ‘t10 | ~(‘t11)&‘p14
                | ‘t12 | ~(‘t13)&‘p16;
    ...
endmodule
```

Fig. 10. Part of CC1.v file

The RG^i register should be described as R_i-bits D-type register with asynchronous reset or set. The typical synthesis template can be used. The rules of choosing set or reset are following:

- if places from the set P_i create a marked cycle or any place is not initially marked the register should have an asynchronous reset,
- if places from the set P_i do not create a marked cycle but any place p is initially marked it should have an asynchronous set.

In our case all register should have reset signal (Fig. 11). Of course, we need to create only two registers one 3-bits and one 2-bits and then create two instantiation of each one. The global reset of Petri net logic circuit will be connected to these reset/set inputs. It will assure suitable initialization of circuit. The Y memory can be also described with logic equations. To synthesize it as memory it is required to add special synthesis

```
module RG3bit(CLK, RES, Q, D);
    output [2:0] Q;
    reg    [2:0] Q;
    input CLK;
    input RES;
    input [2:0] D;

    always @(posedge CLK or posedge RES)
        if (RES)
            Q <= 3'b0;
        else
            Q <= D;
endmodule
```

Fig. 11. RG1.v file

directive. The syntax of this directive depends on FPGA vendor. Because, typically, embedded memory blocks are synchronous it is also required to create a clock input and synchronous reset. In our case it is targeted on Xilinix devices and it is described using logic equations. The `always` block with clock signal on the sensitivity list is used and `bram_map` synthesis attribute is set to value `yes` (Fig. 12).

```
`include "cons.vh"

module Y(CLK, RES, Y, Q);
    output [9:0] Y;
    reg    [9:0] Y;
    input CLK;
    input RES ;
    input [9:0] Q ;

    // synthesis attribute bram_map of Y is yes
    always @(negedge CLK)
        if (RES)
            Y = 10'h000;
        else begin
            Y[0] = `p1;
            Y[1] = `p2;
            ...
            Y[9] = `p6;
        end
endmodule
```

Fig. 12. Part of Y.v file

The top-level module (Fig. 13) should describe connections of all modules according to the logic scheme presented in figure 8. Additionally the global reset signal should be connected to reset/set inputs of registers and reset of memory. The global clock signal should be connected to registers and memory. The memory should be trigged by the opposite edge instead of registers. It allows operations to be generated in one clock cycle when the state register value is stabilized. Model of logic circuit created in this way can be transferred to a third-party synthesis tool.

```
module topPN1(CLK, RES, X, Y);
    input CLK;
    input RES;
    input [9:0] X;
    output [9:0] Y;
    wire [9:0] D;
    wire [9:0] Q;

    CC1 cc1(.D(D[2:0]), .Q(Q), .X(X));
    CC2 cc2(.D(D[5:3]), .Q(Q), .X(X));
    CC3 cc3(.D(D[7:6]), .Q(Q), .X(X));
    CC4 cc3(.D(D[9:8]), .Q(Q), .X(X));
    RG3bit rg1(.CLK(CLK), .RES(RES),
               .D(D[2:0]), .Q(Q[2:0]));
    RG3bit rg2(.CLK(CLK), .RES(RES),
               .D(D[5:3]), .Q(Q[5:3]));
    RG2bit rg3(.CLK(CLK), .RES(RES),
               .D(D[7:6]), .Q(Q[7:6]));
    RG2bit rg4(.CLK(CLK), .RES(RES),
               .D(D[9:8]), .Q(Q[9:8]));
    Y y(.CLK(CLK), .RES(RES), .Q(Q), .Y(Y));
endmodule
```

Fig. 13. topPN1.v file

7.3. Dual verification. The control algorithm can be verified by applying dual verification procedure (Fig. 14). This procedure is based on simultaneous simulation of both models [10].

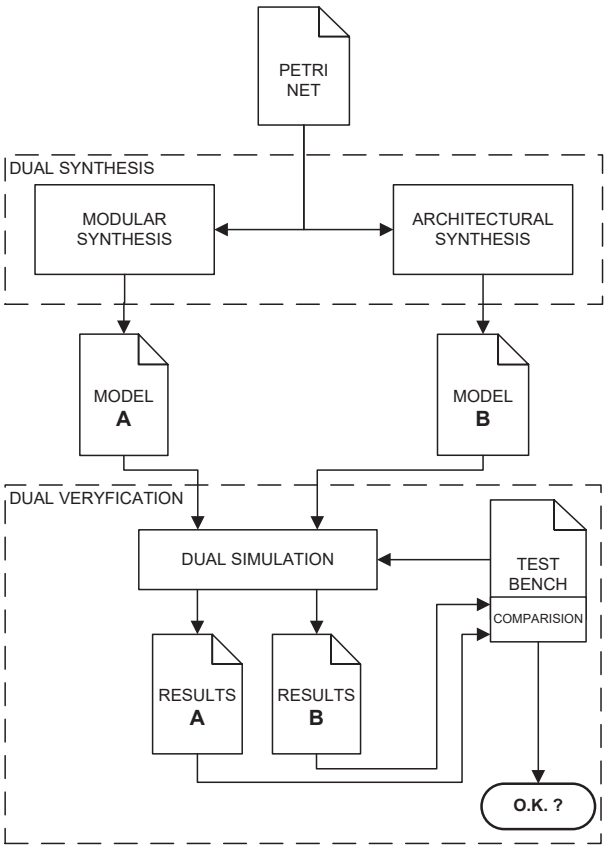


Fig. 14. Dual Verification Flow

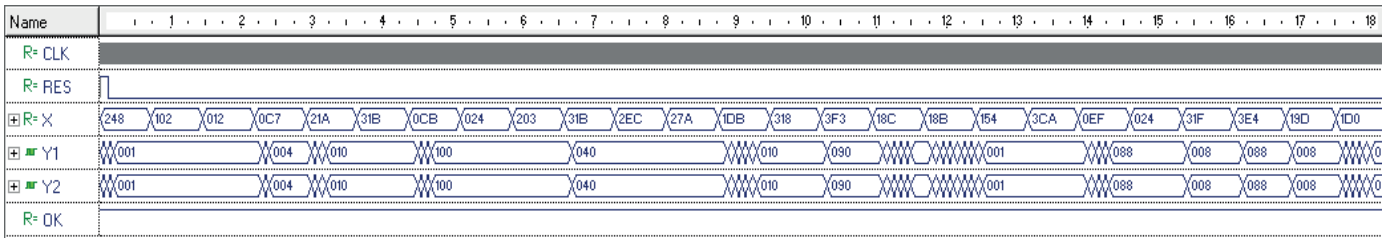


Fig. 15. Dual simulation of Petri net PN_1

For the simulation, a testbench is used. The testbench generates the same input signals for both models at the same time. Additionally, it compares the output signals. If the comparison process does not detect any differences it means that control algorithm is a proper design. Because two different formal methods of synthesis are used, mathematically proofed, there are no possibility to obtain wrong models. If the comparison process detects any differences it means that Petri net model is ambivalence and it has to be corrected by the designer.

The part of dual simulation of a sample Petri net PN_1 is shown in waveform in figure 15. To obtain these results a result comparison testbench was prepared (Fig. 16). There are instantiated both models as unit under tests (UUTs): UUT1 and UUT2. The STIMUL process generates all input signals (CLK, RES, and X) and they are connected to both UUTs. It ensures that both model are stimulated by the same values at the same time. The obtained results are stored in two different variables: Y1, Y2. Then, these variables are compared in the RESCOMP process. If their values are different the OK signal is set to false (represented by 0).

```
module Comp (CLK, RES, Y1, Y2, Y, OK);
   input CLK, RES;
   input [9:0] Y1, Y2;
   output [9:0] Y;
   reg [9:0] Y;
   output OK;
   reg OK;

   always @(posedge CLK or posedge RES)
      if (RES) begin
         Y <= 10'b0;
         OK <= 1'b1;
      end else
      if (Y1 == Y2 && OK) begin
         Y <= Y1;
         OK <= 1'b1;
      end else begin
         Y <= 10'b0;
         OK <= 1'b0;
      end
endmodule
```

Fig. 16. tbrc.v file

In presented the case where one scenario of one full cycle of industrial process was tested. The results obtained from both models are the same.

```
module Comp (CLK, RES, Y1, Y2, Y, OK);
   input CLK, RES;
   input [9:0] Y1, Y2;
   output [9:0] Y;
   reg [9:0] Y;
   output OK;
   reg OK;

   always @(posedge CLK or posedge RES)
      if (RES) begin
         Y <= 10'b0;
         OK <= 1'b1;
      end else
      if (Y1 == Y2 && OK) begin
         Y <= Y1;
         OK <= 1'b1;
      end else begin
         Y <= 10'b0;
         OK <= 1'b0;
      end
endmodule
```

Fig. 17. Comp.v file

8. Implementation of ASLC

The top-level ASLC was constructed based on the architecture presented in figure 3. The HDL description of control unit A and B were obtained during the logic synthesis process. It is required to create additional comparator module. It was described in Verilog HDL (Fig. 18) as a synchronous circuit. The comparator has to be synchronous because both control units could have different delays in setup of output signals. If a comparator detects any differences in respect of compared signals then it sets value of the status signal OK to false and sets all

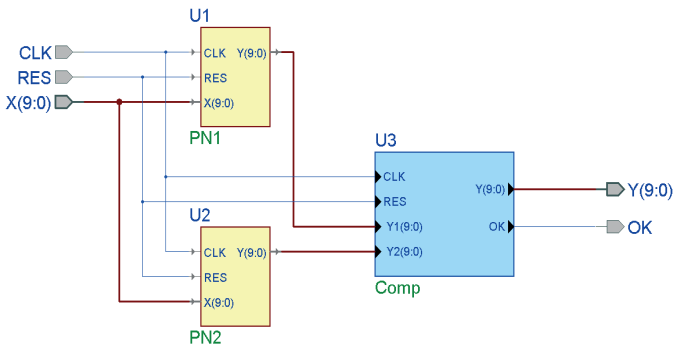
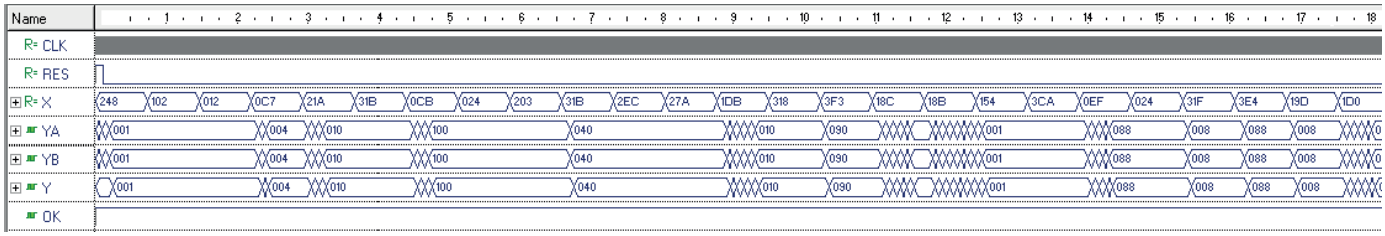


Fig. 18. Top-level of ASLC for Petri net PN_1

a)



b)

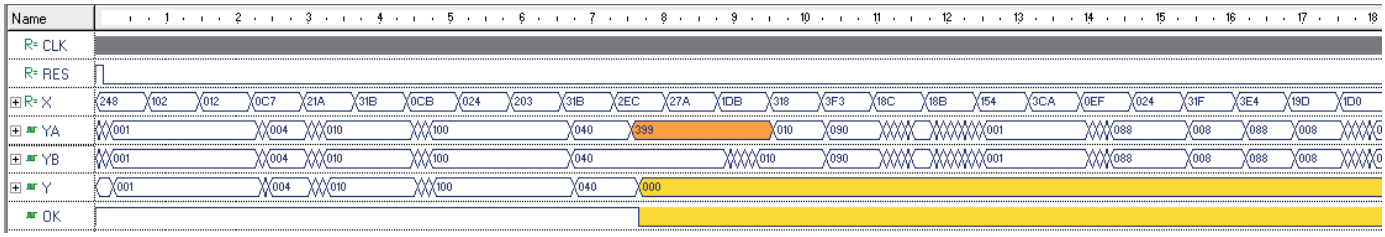


Fig. 19. Simulation of ASCL for Petri net PN_1 (a) without error (b) with error

outputs of the controller to safety state (defined as low values of all outputs). Additionally it holds this output state until the reset of the ASLC takes place.

The top-level module (Fig. 18) was designed in block diagram editor of Active-HDL environment and then converted into Verilog description. Both control units are trigged by the rising edge of the clock signal but they set outputs on the falling edge. The comparator compares these signals also on the rising edge. The usages of both edges allows the final date to be ready in one cycle.

The design of ASLC was simulated in Active-HDL environment with the use of a testbench. The obtained results of the simulation are shown in figure 19. One faultless simulation is presented in the figure 19a and one simulation with injected temporary fault into the YA signal in the figure 19b. Although it could be observed even when the YA signal returns to the correct values the output of the ASLC are still held in the safety state.

The whole design was synthesized & implemented into Xilnix Virtex V50 device with the use of Xilnix ISE with XST. Both control unites were synthesized separately and top-level module was synthesized together with a comparator and black box instantiation of control units. The netlists of control unites were attached to implementation process. Such approach allows for an easy replacement of any control unit. The results of implementation of a sample Petri net are shown in table 2.

Table 2
Synthesis of Petri net PN_1

Resource	Utilization		
	Model A	Model B	ASLC
Slices	17	17	42
Flip-flop	22	10	43
4 input LUT	23	33	71
Block RAM	–	3	3

After the implementation the simulation was repeated with the use of a timing model. During the simulation delays of internal and output signals were observed. The Y1 signal is generated after the falling edge of CLK with 2699 ps average delay, similarly the Y2 signal has 4776 ps average delays. This differences do not affect properly work of controller because they are compared synchronously, after half clock cycle, on rising edge of CLK. The output Y signal of the ASLC is generated after the rising edge of CLK with 8481 ps average delay.

9. Conclusions

Digital controllers for safety critical applications are an important research field. In the presented approach two models are obtained from one specification in different HDLs to ensure that two different synthesis methods are used: monolithic modular synthesis and architectural distributed synthesis. In addition, different description styles are applied: assertion rule-based style [15, 27], and structural RTL description style [4, 19].

The advantages of the proposed solution is that possibility two different models directly from the one specification can be obtained. The dual synthesis process is fully automated. It allows to obtain the architecture of ASLC for systems with increased safety with redundant hardware of control units by one team of designers. In addition to that, the dual simulation is performed to increase reliability of the control algorithm.

References

[1] K. Biliński, M. Adamski, J. Saul, and E. Dagless, "Petri-net-based algorithms for parallel-controller synthesis", *IEEE Proceedings – Computers and Digital Techniques* 141 (6), 405–412 (1994).

[2] C. Girault and R. Valk, *Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications* Springer-Verlag, Berlin/Heidelberg (2003).

[3] A. Khamis, D. Zydek, G. Borowik, and D. S. Naidu, "Control system design based on modern embedded systems", *Lecture Notes in Computer Science* 8112 , 491–498, Springer-Verlag, Berlin/ Heidelberg (2013).

[4] R. Wiśniewski, A. Barkalov, L. Titarenko, and W. Halang, "Design of microprogrammed controllers to be implemented in FPGAs", *International Journal of Applied Mathematics and Computer Science* 21 (2), 401–412 (2011).

[5] A. Milik and E. Hrynkiewicz, "On translation of LD, IL and SFC given according to IEC-61131 for hardware synthesis of reconfigurable controller", *Proceedings of the 19th IFAC World Congress*, 4477–4483, IFAC (2014).

[6] J. Mocha and D. Kania, "Sprzętowa realizacja programu sterowania w strukturach FPGA", *Przegląd Elektrotechniczny* 88 (12a), 95–100 (2012).

[7] A. Milik, "On ladder diagrams compilation and synthesis to FPGA implemented reconfigurable logic controller", *Advances in Electrical and Electronic Engineering* 12 (5), 443–451 (2014).

[8] M. Rawski, P. Tomaszewicz, G. Borowik, and T. Łuba, "Logic synthesis method of digital circuits designed for implementation with embedded memory blocks of FPGAs", *Lecture Notes in Electrical Engineering* 79, 121–144 (2011).

[9] N. Chang, W. H. Kwon, and J. Park, "Hardware implementation of real-time Petri-net-based controllers", *Control Engineering Practice* 6 (7), 889–895 (1998).

[10] M. Węgrzyn, "Implementation of safety critical logic controller by means of FPGA", *Annual Reviews in Control* 27 (1), 55–61 (2003).

[11] W. A. Halang, M. Śnieżek, and S.-K. Jung, "A real-time computing architecture for applications with high safety and predictability requirements", *1st IEEE International Workshop on Real-Time Computing System and Applications RTCSA'94*, 153–157, Seoul, South Korea (1994).

[12] W. A. Halang and M. Adamski, "A programmable electronic system for safety related control applications", *Advances in safety and reliability: Proceedings of the International Conference ESREL'97*, 349–355, Oxford, Pergamon (1997).

[13] N. Marranghello, J. Mirkowski, and K. Bilinski, "Synthesis of synchronous digital systems specified by Petri nets" in A. Yakovlev, L. Gomes, and L. Lavagno (edts.), *Hardware Design and Petri Nets*, 129–150, Kluwer Academic Publishers, Boston (2000).

[14] T. Murata, "Petri nets: Properties, analysis and applications", *Proceedings of the IEEE* 77(4), 541–580 (1989).

[15] M. Adamski and J. Tkacz, "Formal reasoning in logic design of reconfigurable controllers" in *Proceedings of 11th IFAC/ IEEE International Conference on Programmable Devices and Embedded Systems*, 1–6, Brno, Czech Rep. (2012).

[16] A. Bukowiec and M. Adamski, "Logic synthesis for FPGAs of interpreted Petri net with common operation memory", *11th IFAC/IEEE International Conference on Programmable Devices and Embedded Systems PDeS 2012*, 57–62 (2012).

[17] A. Bukowiec and M. Adamski, "Synthesis of Petri nets into FPGA with operation flexible memories" in *Proceedings of the IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits and Systems DDECS'12*, 16–21, Tallinn, Estonia (2012).

[18] G. Borowik, T. Łuba, and B. J. Falkowski, "Logic synthesis method for pattern matching circuits implementation in FPGA with embedded memories", in *12th International Symposium on Design and Diagnostics of Electronic Circuits Systems 2009*, 230–233 (2009).

[19] A. Bukowiec and M. Węgrzyn, "Design of safety critical logic controller using devices integrated microprocessor with FPGA", *Proceedings of SPIE 5775*, 377–384 (2005).

[20] A. Karatkevich, "Dynamic analysis of Petri net-based discrete systems", *Lecture Notes in Control and Information Sciences* 356 (2007).

[21] T. Kozłowski, E. Dagless, J. Saul, M. Adamski, and J. Szajna, "Parallel controller synthesis using Petri nets", *IEEE Proceedings – Computers and Digital Techniques*, 142 (4), 263–271 (1995).

[22] K. Jensen, K. Kristensen, and L. Wells, "Coloured Petri nets and CPN tools for modelling and validation of concurrent systems", *International Journal on Software Tools for Technology Transfer* 9 (3), 213–254 (2007).

[23] V. Savi and X. Xie, "Liveness and boundedness analysis for petri nets with event graph modules", *Lecture Notes in Computer Science* 616, 328–347 (1992).

[24] M. Sałamaj, "Conception of a control unit for critical systems", *International Journal of Electronics and Telecommunications* 59 (4), 363–368 (2013).

[25] J. Tkacz, "State machine type colouring of Petri net by means of using a symbolic deduction method", *Measurement Automation and Monitoring* 53 (5), 120–122 (2007).

[26] M. Doligalski, "Behavioral specification of the logic controllers by means of the hierarchical configurable Petri nets", *Proceedings of 11th IFAC/IEEE International Conference on Programmable Devices and Embedded Systems*, 80–83, Brno (2012).

[27] H. Foster, A. Krolnik, and D. Lacey, *Assertion-Based Design*, Kluwer Academic Publishers, Norwell (2004).