INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2013, VOL. 59, NO. 4, PP. 331–339
Manuscript received October 31, 2013; revised November, 2013. DOI: 10.2478/eletel-2013-0040

Structured Mapping of Petri Net States and Events for FPGA Implementations

Jacek Tkacz and Marian Adamski

***Abstract*—The paper presents a new method of structured encoding of global internal states and events in Reconfigurable Logic Controllers, which are directly mapped into Field Programmable Gate Arrays (FPGA). Modular, concurrently decomposed, colored state machine is chosen as a intermediate model, before the mapping of Petri net into an array structure of dedicated but very flexible and reliable digital system. The initial textual specification in formal Gentzen logic serves both as a design description for a rapid prototyping, as well as formal model, suitable for detailed computer-based reasoning about optimized and synthesized logic controller, implemented in configurable hardware. Only the selected linear subset from general, universal propositional Gentzen Logic is necessary to deduce several properties of the net, such as relations of non-concurrency among structurally ordered macroplaces. The goal of this paper is to present the design methodology for modeling and synthesis of discrete controllers using related Petri net theory, rule-based theory (mathematical logic), and VHDL.**

***Keywords*—Configurable logic controllers, interpreted Petri net state space, local and global state encoding, hyperpgraph, logic design, Gentzen sequents, Petri net coloring, FPGA, VHDL**

I. Introduction

THE behavior and internal structure of logic controller can be initially modeled in the form Control Interpreted Petri net, very often related with Sequential Function Charts [1]–[5] or UML state machine diagrams [6], [7]. As the next design step, the topological structure of the Petri net, which can contain structurally ordered macroplaces, places and transitions, can be formally presented in the textual, rule based language, suitable for simulation, validation, verification and a rapid prototyping of reconfigurable logic controllers. The format of logic expressions is taken from professional hardware description language VHDL.

The proposed rigorous digital design process starts from hierarchical concurrent state machine model (HCSM), which has been formally derived from modular, colored control interpreted Petri net [1], [2], [8]–[10]. The colored tokens, arcs, places and transitions separate hierarchically and concurrently related State Machine components. The rule-based textual logic description of Petri net in VHDL syntax is accepted by professional design tools like Acvtive HDL (Aldec, USA) and Xlinx ISE. The flexible, readable template for Petri net description is directly recognized by VHDL compiler and simulator as well as by formal reasoning system. The logic specification is one-to-one mapped into Field Programmable Gate Array macrocells. Combinatorial procedures in formal design of logic controller are supported by Gentzen sequent calculus [8], [9] and formal verification of specification by means of model checking [11].

The rule-based textual description of topological structure of the Petri net with a logical interpretation of binary inputs and outputs of controller are treated together as formal assertions sequents in Gentzen Logic.

The research was financed from budget resources intended for science in 2010–2013 as an own research project No. N N516 513939.

J. Tkacz and M. Adamski are with the Institute of Computer Engineering and Electronics, University of Zielona Góra, Licealna 9, 65-417 Zielona Góra, Poland (e-mails: {j.tkacz; m.adamski}@iie.uz.zgora.pl).

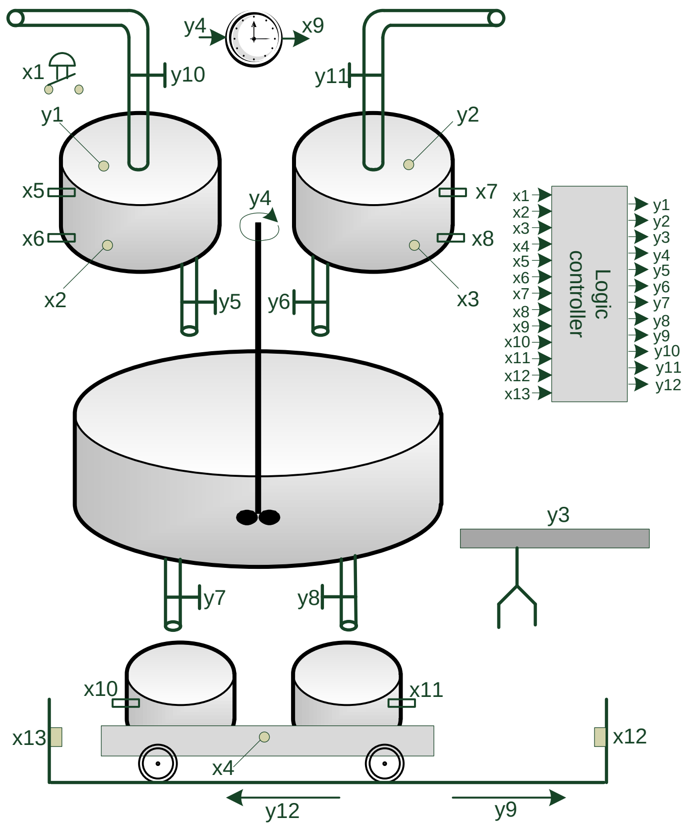

Fig. 1. Control system.

II. Example of Control System

The example of control system for beverage production and distribution is given in Fig. 1. The process can be started by pressing x_1 button. Initially, both tanks placed on the carriage are empty. Valves y_{10} and y_{11} are opened and target tanks are loaded on the carriage y_3. The valves are opened until the containers are filled, and this information is indicated respectively by sensor x_5 and x_7. Meanwhile, loaded containers are transported (signal y_{12}) to a proper location indicated by sensor x_{13}. When the ingredients are ready it is signalized by

TABLE I
LOGIC CONTROLLER INPUTS AND OUTPUTS DESCRIPTION

Signal	Description
	INPUTS
x_1	start the process
x_2	preparation of ingredients in the first container is finished
x_3	preparation of ingredients in the second container is finished
x_4	preparation of tanks is finished
x_5	maximal fluid level in the first container is obtained
x_6	minimal fluid level in the first container is obtained
x_7	maximal fluid level in the second container is obtained
x_8	minimal fluid level in the second container is obtained
x_9	preparation of the drink is finished
x_{10}	filling off the first tank is finished
x_{11}	filling off the second tank is finished
x_{12}	the carriage is in its initial location
x_{13}	the carriage is in its target location
	OUTPUTS
Signal	Description
y_1	preparation of the first ingredient
y_2	preparation of the second ingredient
y_3	loading containers
y_4	mixing ingredients
y_5	valve for emptying the first container
y_6	valve for emptying the second container
y_7	valve for filling the first tank
y_8	valve for filling the second tank
y_9	carriage movement to the initial location (right)
y_{10}	valve for filling the first container
y_{11}	valve for filling the second container
y_{12}	carriage movement to the target location (left)

the sensors x_2 (first container), x_3 (second container) and x_4 (the carriage). After that, the ingredients from both containers are dropped into main tank (signal y_5 and y_6), where they are mixed (active signal y_4). When the drink is well mixed, it is indicated by the sensor x_9. When one of the tanks is ready, it is independenty filled and closed (signals x_{10} and x_{11}). After finishing of both processes, the carriage with tanks moves to its initial position (signal y_9). The sensor x_{12} is active when the carriage reached starting position. A detailed description of the input and output signals are included in Tab. I.

Petri net places $(P_1 - P_{20})$ stand for the local states of concurrent state machine (Fig. 2). The transitions $(t_1 - t_{18})$ describe partial states changes in terms of local state changes of Petri net space. Boolean expressions called guards $(x_1 - x_{13})$ describe the external conditions for transitions to be enable. The Moore type outputs $(y_1 - y_{12})$ are attached to places. In such a way events are related with transitions of the net and inputs of controller.

III. MODULAR STRUCTURED PETRI NET

Usually Petri net places represent local states of Concurrent State Machine, whereas maximal sets of such concurrent Petri net places are assigned to its global states. Very often controller output signals are directly assigned to selected places in the Petri net (Fig. 2), and controller input signals are attached to guards related with particular transitions of the net. The registered outputs can be efficiently used for one-hot encoding of places, which are nested inside recognized macroplaces $MP_1 - MP_{11}$.

In formal description of the Petri net, capital letters stand for the symbols from sequent propositional logic.

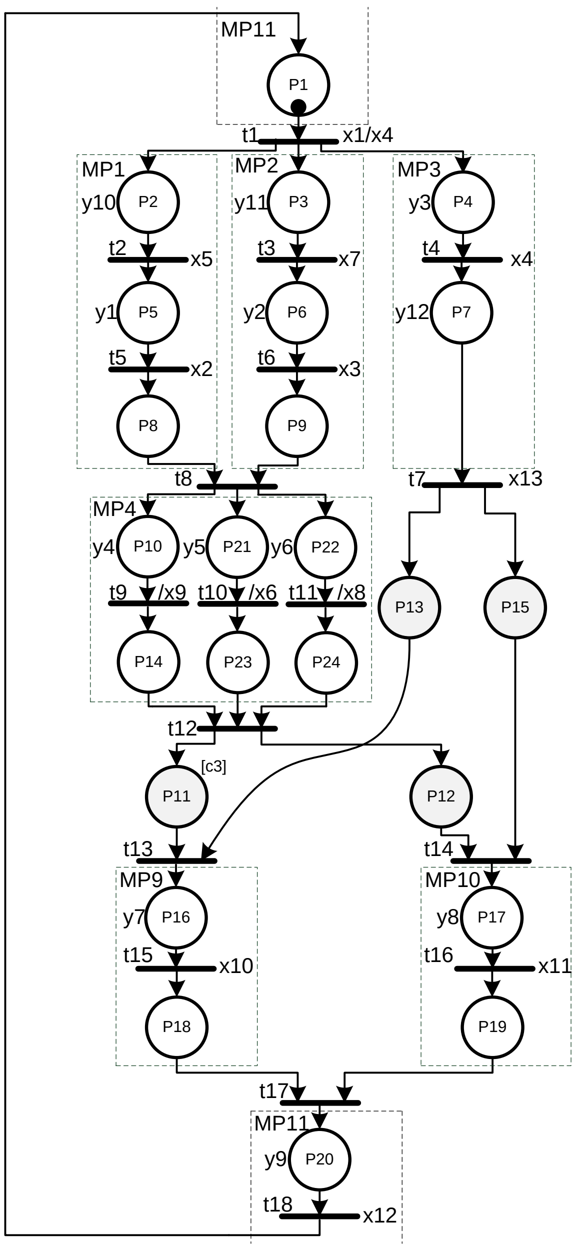

Fig. 2. Modular control interpreted Petri net.

In the new version of the experimental reasoning system Gentzen symbol $\vdash$ describes entailed assertion. Symbol *(*and) denotes conjunction symbol *(*or) denotes disjunction, symbol *(*not) – negation symbol $\Leftarrow$ backward implication, symbol *(*xor) – exclusive or. Capital letters $X_0 - X_{13}$ describe controller inputs and letters $Y_1 - Y_{12}$ – controller outputs. Symbols $MP_1 - MP_{11}$ are the names of macroplaces (Fig. 2, Fig. 2). Petri net places p1-p16 are related to capital letters denoting single bit memory elements: $P_1 - P_{16}$. The name of guarded transition t_i is T_i. If it is necessary next values of registered signals are recognized by a linear temporal logic operator next, denoted as @. In general the current value of a registered signal is presented as Q, but its next value is written as $@Q$ [1], [3], [4].

The rule-based description of partial state changes in sequent logic for one-hot encoding with direct using of registered outputs $Y_1 \dots Y_{11}$ in hierarchial modular array structure can be as follows [12]:

$$\begin{array}{ll}
\vdash @Y_1 \Leftarrow (MP_1 \text{ and } Y_1) \text{ xor } (T_2 \text{ xor } T_5) & : P_5 \\
\vdash @Y_2 \Leftarrow (MP_2 \text{ and } Y_2) \text{ xor } (T_7 \text{ xor } T_6) & : P_6 \\
\vdash @Y_3 \Leftarrow (MP_3 \text{ and } Y_3) \text{ xor } (T_1 \text{ xor } T_4) & : P_4 \\
\vdash @Y_4 \Leftarrow (MP_4 \text{ and } Y_4) \text{ xor } (T_8 \text{ xor } T_9) & : P_{10} \\
\vdash @Y_5 \Leftarrow (MP_4 \text{ and } Y_5) \text{ xor } (T_8 \text{ xor } T_{10}) & : P_{21} \\
\vdash @Y_6 \Leftarrow (MP_4 \text{ and } Y_6) \text{ xor } (T_8 \text{ xor } T_{11}) & : P_{22} \\
\vdash @Y_7 \Leftarrow (MP_9 \text{ and } Y_7) \text{ xor } (T_{13} \text{ xor } T_{15}) & : P_{16} \\
\vdash @Y_8 \Leftarrow (MP_{10} \text{ and } Y_8) \text{ xor } (T_{14} \text{ xor } T_{16}) & : P_{17} \\
\vdash @Y_9 \Leftarrow (MP_{11} \text{ and } Y_9) \text{ xor } (T_8 \text{ xor } T_{10}) & : P_{20} \\
\vdash @Y_{10} \Leftarrow (MP_1 \text{ and } Y_{10}) \text{ xor } (T_1 \text{ xor } T_2) & : P_2 \\
\vdash @Y_{11} \Leftarrow (MP_2 \text{ and } Y_{11}) \text{ xor } (T_1 \text{ xor } T_3) & : P_3
\end{array}$$

$@Y_i$ describe the next value of controller registered output Y_i. Some places such as for example $P_5, P_6, \dots, P_3$ are encoded inside macroplaces as conjunctions of registered signals (MP_j and Y_i):

$$\begin{array}{l}
P_5 = (MP_1 \text{ and } Y_1) \\
P_6 = (MP_2 \text{ and } Y_2) \\
\quad \dots \\
P_3 = (MP_2 \text{ and } Y_{11})
\end{array}$$

The other places, for example $P_1, P_8, \dots P_{18}$ are encoded by means of using negated values of registered outputs signals:

$$\begin{array}{l}
P_1 \;\; = (MP_{11} \text{ and not } Y_9) \\
P_8 \;\; = (MP_1 \text{ and not } Y_10 \text{ and not } Y_1) \\
\quad \dots \\
P_19 = (MP_{10} \text{ and } Y_8)
\end{array}$$

The preconditions of common border transitions for macroplaces $MP_1 - MP_{11}$ includes symbolic names of their encoded input places and attached guards:

$$\begin{array}{l}
\vdash T_1 \Leftarrow MP_{11} \text{ and not } Y_9 \text{ and } X_1 \text{ and not } X_4 \\
\vdash T_7 \Leftarrow MP_3 \text{ and } Y_{12} \text{ and } X_{13} \\
\vdash T_8 \Leftarrow (MP_1 \text{ and not } Y_{10} \text{ and not } Y_1) \text{ and} \\
\qquad (MP_2 \text{ and not } Y_{11} \text{ and not } Y_2) \\
\vdash T_{12} \Leftarrow MP_4 \text{ and not } Y_4 \text{ and not } Y_5 \text{ and not } Y_6 \\
\vdash T_{13} \Leftarrow MP_6 \text{ and } MP_5 \\
\vdash T_{14} \Leftarrow MP_7 \text{ and } MP_8 \\
\vdash T_{17} \Leftarrow MP_9 \text{ and not } Y_7 \text{ and } MP_{10} \text{ and not } Y_8
\end{array}$$

The preconditions of local transitions is built from symbolic names of encoded places and guards:

$$\begin{array}{l}
\vdash T_2 \Leftarrow MP_1 \text{ and } Y_{10} \text{ and } X_5 \\
\vdash T_3 \Leftarrow MP_2 \text{ and } Y_{11} \text{ and } X_7 \\
\quad \dots \\
\vdash T_{18} \Leftarrow MP_{11} \text{ and } Y9 \text{ and } X_{12}
\end{array}$$

The excitations for macroplaces are as follows:

$$\begin{array}{l}
\vdash @MP_1 \Leftarrow MP_1 \text{ xor } (T_1 \text{ xor } T_8) \\
\vdash @MP_2 \Leftarrow MP_2 \text{ xor } (T_1 \text{ xor } T_8) \\
\vdash @MP_3 \Leftarrow MP_3 \text{ xor } (T_1 \text{ xor } T_7) \\
\vdash @MP_4 \Leftarrow MP_4 \text{ xor } (T_8 \text{ xor } T_{12}) \\
\vdash @MP_5 \Leftarrow MP_5 \text{ xor } (T_7 \text{ xor } T_{13}) \\
\vdash @MP_6 \Leftarrow MP_6 \text{ xor } 4(T_{12} \text{ xor } T_{13}) \\
\vdash @MP_7 \Leftarrow MP7 \text{ xor } (T_2 \text{ xor } T_{14}) \\
\quad \dots \\
\vdash @MP_{11} \Leftarrow MP_{11} \text{ xor } (T_{17} \text{ xor } T_1)
\end{array}$$

In previous designs the macroplaces would be one-hot coded by means of using 11 additional state variables $(Q_1 - Q_{11})$.

$$MP_1 = Q_1;\ MP_2 = Q_2;\ \dots;\ MP_{11} = Q_{11}$$

Several optimization techniques could be used for getting smaller number of macrocells [1], [2], [4], [13]. One of them is decomposition of the net into coordinated state machine components, presented in this paper.In that case the number of registered cells (extended T Flip-Flops) could be reduced to nine or even more (Sections VIII, IX).

IV. Colored Macronet

During reduction a Petri net can be converted into a more compact hierarchical description. Two reduction procedures are the most useful: Fusion of Series Places (FSP) and Fusion of Parallel Places (FPP) [5], [9], [14]. Starting from FSP both techniques are used recursively until the macronet becomes irreducible.

Figure 3 shows a macronet related with a control interpreted Petri net from Fig. 2, which describes production management system of milk beverages (Fig. 1). The reachability graph of the macronet is shown in Fig. 4.

A Petri net is enhanced by assigning colors $\{C_1, C_2, C_3, C_4\}$ to places and transitions. Such kind of net is called colored interpreted Petri net [1], [5], [15]. These colors help to intuitively and formally validate the consistency of sequential processes in the considered Petri net. Each color recognizes one state machine module. The rules for Petri net coloring can be found in [2], [3], [5], [15].

The colored second order macronet, which is presented in Fig. 3, can be generated by means of using an experimental computer program (ICPN) developed by team from Institute of Computer Engineering and Electronics, University of Zielona Góra, Poland. The novel effective version is written in C# and it is based on searching and selecting proper state space transversals directly from topological structure of the net. As a result colored structured net from Fig. 2 has been obtained. Another option, which is presented in the next part of the paper (Sections VI, VII), uses automatic reasoning about Petri net space in Gentzen logic systems (Gentzen6 and GentzenDB).

In the considered coloring, macroplaces and transitions are flagged with four colors $\{c_1, c_2, c_3, c_4\}$. Next to each of the macroplaces, numbers of its internal places are provided. It should be noted that the macroplaces, which are painted with disjoint set of colors, for example $MP_1[C_4]$ and $MP_2[C_3]$ and $MP_3[C_1, C_2]$ are concurrent to each other. The macroplaces sharing the same color, for example $MP_5[C_2]$ and $MP_9[C_2, C_3]$ are sequentially related to each other (Fig. 3).

V. Hierarchical Petri Net Space

From the colored Petri net it is possible to deduce equivalent representation of the controller as transition system. It is obtained from reachability graph (Fig. 4) of Petri net or its equivalent macronet (Fig. 3). The vertices of reachability graph describe global states of transition system. If the Petri net is properly colored then vertices contain all colors. There are not vertices which contain two different states with the same color. The number of colors should be minimal and equal to the maximum number of places which are concurrently marked. In such a way the transition system could be treated as an interpreted form of reachability graph (Fig. 4). Its modular and structured form represents global coordination states and macrostates from Fig. 2. The transition system can be in ten global states $M_0 - M_9$. There are superposition of maximal subsets of concurrent local macrostates:

$$M_0 = \{MP_{11}\}; M_1 = \{MP_1, MP_2, MP_3\};$$
$$M_2 = \{MP_3, MP_4\}; M_3 = \{MP_1, MP_2, MP_5, MP_8\};$$
$$M_4 = \{MP_3, MP_6, MP_7\}; M_5 = \{MP_4, MP_5, MP_8\};$$
$$M_6 = \{MP_5, MP_6, MP_7, MP_8\}; M_7 = \{MP_5, MP_6, MP_{10}\};$$
$$M_8 = \{MP_7, MP_8, MP_9\}; M_9 = \{MP_9, MP_{10}\}$$

The monotone characteristic sequent of the Petri net discrete space is as follows:

$$\begin{aligned} M \vdash & (MP_{11}), (MP_1 \textit{ and } MP_2 \textit{ and } MP_3), (MP_3 \textit{ and } MP_4),\\ & (MP_1 \textit{ and } MP_2 \textit{ and } MP_5 \textit{ and } MP_8),\\ & (MP_1 \textit{ and } MP_2 \textit{ and } MP_5 \textit{ and } MP_8),\\ & (MP_3 \textit{ and } MP_6 \textit{ and } MP_7),\\ & (MP_4 \textit{ and } MP_5 \textit{ and } MP_8),\\ & (MP_5 \textit{ and } MP_6 \textit{ and } MP_7 \textit{ and } MP_8),\\ & (MP_5 \textit{ and } MP_6 \textit{ and } MP_{10}),\\ & (MP_7 \textit{ and } MP_8 \textit{ and } MP_9), (MP_9 \textit{ and } MP_{10}); \end{aligned}$$

After further substitution it is possible to get partial or full characteristic functions for any level of hierarchy, even on the local place level:

$$\begin{aligned} \vdash & (P_1 \textit{ or } P_{20}) \textit{ and}\\ & ((P_2 \textit{ or } P_5 \textit{ or } P_8) \textit{ and}\\ & (P_3 \textit{ or } P_6 \textit{ or } P_9) \textit{ and } (P_4 \textit{ or } P_7)) \textit{ and}\\ & \ldots\\ & ((P_{16} \textit{ or } P_{18}) \textit{ and } (P_{17} \textit{ or } P_{19})); \end{aligned}$$

The distribution of the Petri net tokens among places, describes the current global state M. New marking M, after the firing of any enabled transition is the next global state @M. From the current global state M, the modeled controller goes to the next internal global state @M, generating the registered @y output signals.

A. Hypergraphs of Concurency and Sequentiality

In order to determine the SM-subnets covering of the Petri macronet, formal reasoning using propositional Gentzen calculus was used [12], [16]–[19]. As a result of local and global state space analysis, state machine subnets are obtained. The subnets are subsequently marked with different colors. The subnets are accordingly mapped to markers, places, transitions and input and output signals.

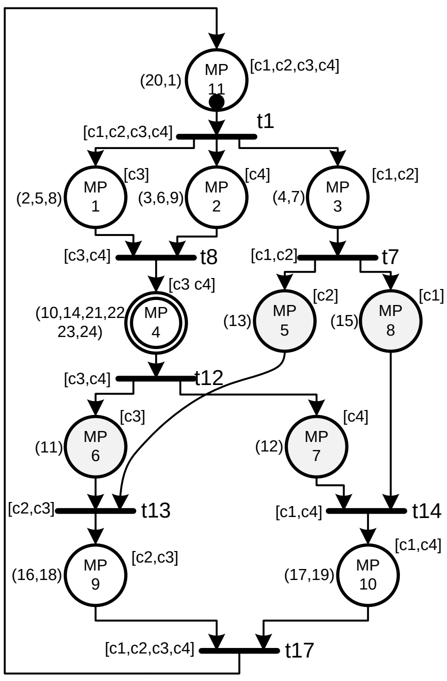

Fig. 3. Petri macronet.

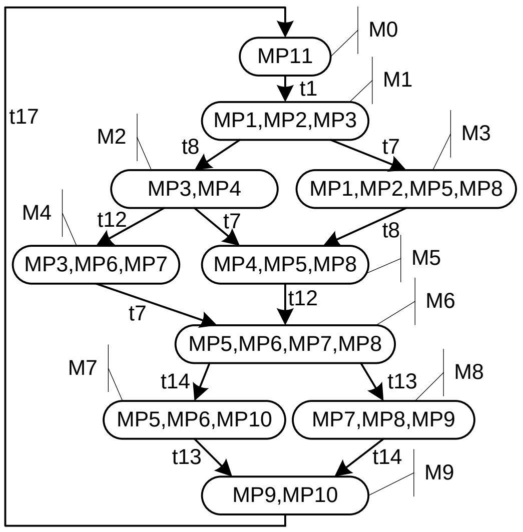

Fig. 4. Reachability graph of macronet.

Hypergraph of concurrency Fig. 5 represents all the global states of the discrete system described by the Petri net. Hyperedges relate concurrent macroplaces belonging to the same global macrostates and correspond to respective vertices of reachability graph (Fig. 4).

Hypergraph of non-concurrency, or sequentiality (Fig. 6), is a complement of hypergraph of concurrency. Its hyperedges correspond to sets of sequential Petri net places, where only one place can be marked, from state machine subnets. It could be calculated as exact transversals of concurrency hypergraph (Fig. 5) [17], [20].

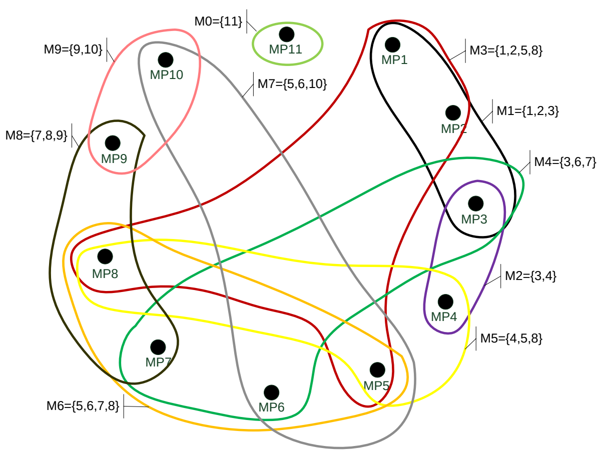

Fig. 5. Concurrency hypergraph.

$$\vdash MP_{11}, (MP_1 \text{ or } MP_2 \text{ or } MP_3), (MP_3 \text{ or } MP_4),$$
$$(MP_1 \text{ or } MP_2 \text{ or } MP_5 \text{ or } MP_8), \ldots, (MP_9 \text{ or } MP_{10})$$

The full hypergraph of sequentiality shown in Fig. 6 contains six hyperedges $\{i_1 - i_6\}$. There are two net covers with exactly four hyperedges: $\{i_3, i_4, i_1, i_2\}, \{i_3, i_4, i_5, i_6\}$. Both covers contain essential hyperedges i_3 and i_4, which are the only ones to cover macroplaces MP_5 and MP_8. The first four sequential subnets $i_1 - i_4$ are sufficient to cover all the macroplaces and places of the net (Fig. 7).

Notice that the non-concurrency graph hyperedges correspond to invariants obtained using ILP linear programming methods. Alternatively they can be read off the marking reachability graph (Fig. 4), as transversals of subsets of places marked in parallel [8], [14].

$$\begin{array}{ll}
\vdash MP_1, MP_4, \mathbf{MP_6}, \mathbf{MP_8}, MP_{10}, MP_{11}; & \\
\vdash MP_1, MP_4, MP_6, MP_9, MP_{11}; & (i_1) \\
\vdash MP_1, \mathbf{MP_4}, \mathbf{MP_5}, MP_7, MP_9, MP_{11}; & \\
\vdash MP_1, MP_4, MP_7, MP_{10}, MP_{11}; & (i_6) \\
\vdash MP_2, MP_4, \mathbf{MP_6}, \mathbf{MP_8}, MP_{10}, MP_{11}; & \\
\vdash MP_2, MP_4, MP_6, MP_9, MP_{11}; & (i_5) \\
\vdash MP_2, \mathbf{MP_4}, \mathbf{MP_5}, MP_7, MP_9, MP_{11}; & \\
\vdash MP_2, MP_4, MP_7, MP_{10}, MP_{11}; & (i_2) \\
\vdash MP_3, \mathbf{MP_5}, \mathbf{MP_7}, MP_{10}, MP_{11}; & \\
\vdash MP_3, MP_5, MP_9, MP_{11}; & (i_4) \\
\vdash \mathbf{MP_3}, \mathbf{MP_6}, MP_8, MP_9, MP_{11}; & \\
\vdash MP_3, MP_8, MP_10, MP_{11}; & (i_3)
\end{array}$$

The depicted hypergraphs could be hierarchical. Macroplace MP_4 corresponds to the sub-hypergraph containing vertices P_{10}, P_{24}, P_{21}, P_{22}, P_{23}, P_{24}. The subnet possible modified by designer corresponding to this macroplace has boundary transitions t_8 and t_{12}, and internal transitions t_9, t_{10}, t_{11}.

Finally selected cover of Petri macronet by state machine components could be represented only by four transversals of macronet state space (Fig. 5). It is described as a reduced

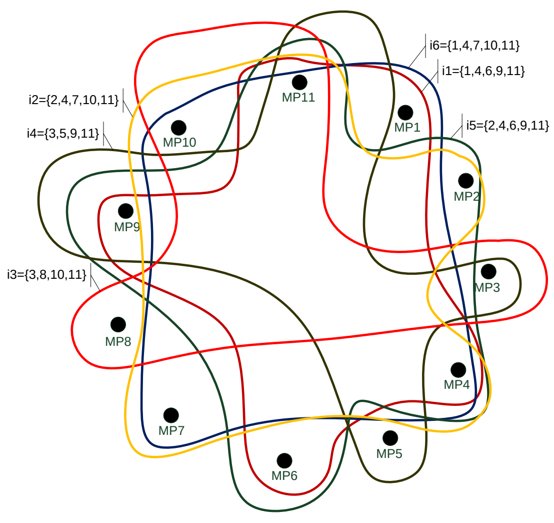

Fig. 6. Non-concurrency hypergraph.

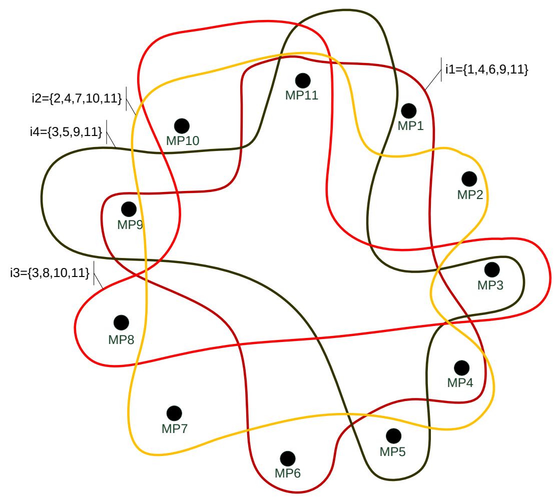

Fig. 7. Non-concurrency hypergraph cover.

hypergraph of non-concurrency (Fig. 7):

$$\begin{array}{l}
\vdash MP_1, MP_4, MP_6, MP_9, MP_{11} \\
\vdash MP_2, MP_4, MP_7, MP_{10}, MP_{11} \\
\vdash MP_3, MP_8, MP_{10}, MP_{11} \\
\vdash MP_3, MP_5, MP_9, MP_{11}
\end{array}$$

VI. Symbolic Method of Hypergraph Generation

What differentiates sequent calculus from other methods, e.g. those used in [1], [18], [19], is that the conversion to clausal form is not required. Moreover, by using cut and consensus a laborious process of results selection is avoided. Siphons and traps that are not minimal are eliminated beforehand.

It is assumed that the Petri net has already been reduced to a hierarchical macronet (Fig. 3), which will be analyzed in the next steps. Gentzen sequents, showing in a symbolic way all the relations of direct transition between input and output places of all transitions, are determined on the basis of the topological structure of an uninterpreted Petri net. Using the method presented in [18], [19] separated sequents of siphons and traps were created (Tab. II, Tab. III).

TABLE II
GROUP SEQUENTS OF TRAPS AND SIPHONS

Transitions	Traps sequent	Siphons sequent
t_1	$(MP_{11} \rightarrow (MP1 + MP2 + MP3)),$	$((MP_1 + MP_2 + MP_3) \rightarrow MP_{11}),$
t_8	$((MP_1 + MP_2) \rightarrow MP_4),$	$(MP_4 \rightarrow (MP_1 + MP_2)),$
t_7	$(MP_3 \rightarrow (MP_5 + MP_8)),$	$((MP_5 + MP_8) \rightarrow MP_3),$
t_{12}	$(MP_4 \rightarrow (MP_6 + MP_7)),$	$((MP_6 + MP_7) \rightarrow MP_4),$
t_{13}	$((MP_5 + MP_6) \rightarrow MP_9),$	$(MP_9 \rightarrow (MP_5 + MP_6)),$
$t14$	$((MP_7 + MP_8) \rightarrow MP_{10}),$	$(MP_{10} \rightarrow (MP_7 + MP_8)),$
$t17$	$((MP_9 + MP_{10}) \rightarrow MP_{11}) \vdash;$	$(MP_{11} \rightarrow (MP_9 + MP_{10})) \vdash;$

TABLE III
TRAPS AND SIPHONS

	Traps	Siphons	Color
i_1	$MP_1, MP_4, MP_6, MP_9, MP_{11}$	$MP_1, MP_4, MP_6, MP_9, MP_{11}$	C_3
i_6	$MP_1, MP_4, MP_7, MP_{10}, MP_{11}$	$MP_1, MP_4, MP_7, MP_{10}, MP_{11}$	C_5
i_5	$MP_2, MP_4, MP_6, MP_9, MP_{11}$	$MP_2, MP_4, MP_6, MP_9, MP_{11}$	C_6
i_2	$MP_2, MP_4, MP_7, MP_{10}, MP_{11}$	$MP_2, MP_4, MP_7, MP_{10}, MP_{11}$	C_4
i_4	$MP_3, MP_5, MP_9, MP_{11}$	$MP_3, MP_5, MP_9, MP_{11}$	C_2
i_3	$MP_3, MP_8, MP_{10}, MP_{11}$	$MP_3, MP_8, MP_{10}, MP_{11}$	C_1

VII. ANALYSIS OF PETRI NETS USING SYMBOLIC METHOD

A. Petri Net Decomposition into SM-Components

In order to check if the Petri net is live, traps equal to siphons (deadlocks) are designed [1], [12], [15]–[17]. Sets of marked traps contained in siphons determine potential state machine subnet, present in the Petri net. Each of the subnets is marked with different color, flagging also its places and transitions. Traps not equal to siphons indicate potential net defects. The net is not live, if all the siphons do not contain traps. We propose the following method of Petri net states space analysis. We use the macronet depicted in Fig. 3 as an example:

1) Using the rule-based, symbolic, description of the Petri net create the group sequent of traps (Tab. II).
2) Reduce the group sequent of traps into single normalized elementary sequents and remove their right sides (Tab. III).
3) Remove the sequents of traps not containing the marked place symbol (no such sequents).
4) Using the rule-based, symbolic, description of the Petri net create the group sequent of siphons (deadlocks) (Tab. II).
5) Reduce the group sequent of siphons into elementary sequents and remove their right sides. Apply the consensus rule to the previously selected sequents of traps in every chosen siphon. If the considered sequent becomes dominated, move to point number 8 (all sequents are dominated).
6) The traps which are equal to siphons determine covering of non-concurrency hypergraph by the hyperegdes corresponding to potential state machine subnets (hyperedges are assigned colors $C_1 - C_6$ (Tab. III).
7) Find the minimal covers of non-concurrency hypergraph (these are $\{C_1, C_2, C_3, C_4\}$ or $\{C_1, C_2, C_5, C_6\}$), move to point number 9.
8) The Petri net is not live (does not apply to the Petri net in Fig. 2).
9) End of algorithm.

After removing the right sides of the reduced sequents listed in the Tab. I and after leaving out only the siphons dominated by marked traps, we obtained six potential automata subnets. These sets correspond to the hyperedges of the sequentiality hypergraph $\{i_1, \ldots, i_6\}$ (Fig. 4). Following point number 5 in the above algorithm, we conclude that the Petri net is live, since each edge of the hypergraph contains a marked trap. Subsets determining traps and siphons are pairwise identical. The cover (i_3, i_4, i_1, i_2) was chosen for Petri net encoding. Subnets were assigned colors according to Tab. III. These colors were placed on the net shown in Fig. 3.

B. Deadlock Detection

The trap and siphon expressions for the net with defect from Fig. 8 are given in the Tab. IV. The net is not covered by initially marked traps contained in siphons. It means that the net is not live. On the other hand it is not fully covered by marked state machine components. Some siphons described in bold (Tab. V) are not equal to traps and show defect in the net.

VIII. PETRI NET MODULAR ENCODING

The macroplaces can be encoded in traditional way inside state machine components. The components are recognized by colors {1,2,3,4}:

$\vdash MP_{11}[1,2,3,4] \Leftarrow \textit{not}\ (Q_0\ \textit{or}\ Q_1\ \textit{or}\ \ldots\ \textit{or}\ Q_8\ \textit{or}\ Q_9)$
$\vdash MP_1[3] \Leftarrow \textit{not}\ Q_4\ \textit{and not}\ Q_5\ \textit{and}\ Q_6$
$\vdash MP_2[4] \Leftarrow \textit{not}\ Q_7\ \textit{and not}\ Q_8\ \textit{and}\ Q_9$
$\vdash MP_3[1,2] \Leftarrow (\textit{not}\ Q_0\ \textit{and}\ Q_1)\ \textit{and}\ (\textit{not}\ Q_2\ \textit{and}\ Q_3)$
$\vdash MP_4[3,4] \Leftarrow (\textit{not}\ Q_4\ \textit{and}\ Q_5\ \textit{and}\ Q_6)\ \textit{and}$
$(\textit{not}\ Q_7\ \textit{and}\ Q_8\ \textit{and}\ Q_9)$
$\vdash MP_5[2] \Leftarrow Q_2\ \textit{and}\ Q_3$
$\vdash MP_6[3] \Leftarrow \textit{not}\ Q_4\ \textit{and}\ Q_5\ \textit{and not}\ Q_6$
$\vdash MP_7[4] \Leftarrow \textit{not}\ Q_7\ \textit{and}\ Q_8\ \textit{not}\ Q_9$
$\vdash MP_8[1] \Leftarrow Q_0\ \textit{and}\ Q_1$
$\vdash MP_9[2,3] \Leftarrow (Q_2\ \textit{and not}\ Q_3)\ \textit{and}$

TABLE IV
GROUP SEQUENTS OF TRAPS AND SIPHONS FOR DEFECTED NET

Transitions	Traps sequent	Siphons sequent
t_1	$(MP_{11} \rightarrow (MP1 + MP2 + MP3)),$	$((MP_1 + MP_2 + MP_3) \rightarrow MP_{11}),$
t_8	$((MP_1 + MP_2) \rightarrow MP_4),$	$(MP_4 \rightarrow (MP_1 + MP_2)),$
t_7	$(MP_3 \rightarrow (MP_8)),$	$((MP_8) \rightarrow MP_3),$
t_{12}	$(MP_4 \rightarrow (MP_6 + MP_7)),$	$((MP_6 + MP_7) \rightarrow MP_4),$
t_{13}	$((MP_6 + MP_8) \rightarrow MP_9),$	$(MP_9 \rightarrow (MP_6 + MP_8)),$
$t14$	$((MP_7 + MP_8) \rightarrow MP_{10}),$	$(MP_{10} \rightarrow (MP_7 + MP_8)),$
$t17$	$((MP_9 + MP_{10}) \rightarrow MP_{11}) \vdash;$	$(MP_{11} \rightarrow (MP_9 + MP_{10})) \vdash;$

TABLE V
TRAPS AND SIPHONS WITH SELECTED DEFECTS

Traps	Siphons
$MP_1, MP_4, MP_6, MP_9, MP_{11}$	$MP_1, MP_4, MP_6, MP_9, MP_{11}$
$MP_1, MP_4, MP_7, MP_{10}, MP_{11}$	$MP_1, MP_4, MP_7, MP_{10}, MP_{11}$
$MP_2, MP_4, MP_6, MP_9, MP_{11}$	$MP_2, MP_4, MP_6, MP_9, MP_{11}$
$MP_2, MP_4, MP_7, MP_{10}, MP_{11}$	$MP_2, MP_4, MP_7, MP_{10}, MP_{11}$
$\mathbf{MP_3, MP_8, MP_9, MP_{10}, MP_{11}}$	$\mathbf{MP_3, MP_8, MP_9, MP_{11}}$
	$\mathbf{MP_3, MP_8, MP_{10}, MP_{11}}$

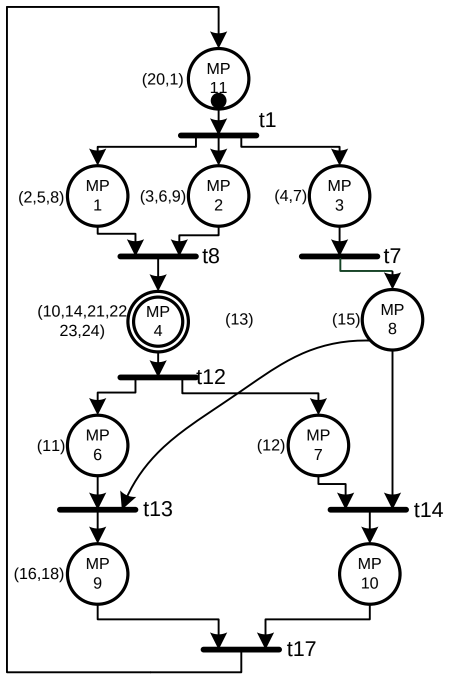

Fig. 8. Macronet with defect.

$(Q_4 \textit{ and } Q_5 \textit{ and not } Q_6)$
$\vdash MP_{10}[1,4] \Leftarrow (Q_0 \textit{ and not } Q_1) \textit{ and}$
$(Q_7 \textit{ and } Q_8 \textit{ and not } Q_9)$

As a result the number of additional encoding variables which are used for state encoding is reduced to nine:

$$\begin{aligned}
&\vdash @Q_0 \Leftarrow Q_0 \textit{ xor } (T_7 \textit{ xor } T_{17})\\
&\vdash @Q_1 \Leftarrow Q_1 \textit{ xor } (T_1 \textit{ xor } T_{14})\\
&\vdash @Q_2 \Leftarrow Q_2 \textit{ xor } (T_7 \textit{ xor} T_{17})\\
&\vdash @Q_3 \Leftarrow Q_3 \textit{ xor } (T_1 \textit{ xor } T_{13})\\
&\vdash @Q_4 \Leftarrow Q_4 \textit{ xor } (T_{13} \textit{ xor } T_{17})\\
&\vdash @Q_5 \Leftarrow Q_5 \textit{ xor } (T_8 \textit{ xor } T_{17})\\
&\vdash @Q_6 \Leftarrow Q_6 \textit{ xor } (T_1 \textit{ xor } T_{12})\\
&\vdash @Q_7 \Leftarrow Q_7 \textit{ xor } (T_{14} \textit{ xor } T_{17})\\
&\vdash @Q_8 \Leftarrow Q_8 \textit{ xor } (T_8 \textit{ xor } T_{17})\\
&\vdash @Q_9 \Leftarrow Q_9 \textit{ xor } (T_8 \textit{ xor } T_{12})
\end{aligned}$$

The names of macroplaces could be used as aliases in hardware description languages. For rapid prototyping is better to use them as flags to observe both local states and macrostates during simulation and modifications [6].

IX. Simulation and Synthesis

Obtained equations are a base for creation a Petri net HDL model in VHDL (Fig. 10). Preconditions of global and local transitions are described as simple continuous assignments. The process FF is responsible for generation of codes of next macro and local states. Because the local states are encoded with use the output variables there are declared internal copy of these variables.

The preferable way of controller rapid prototyping is hierarchical design from a formal assertion-based [21] behavioral description, using professional HDL syntax. One of the possible version of general template [3] is presented in Fig. 10.

For pragmatic reasons the controller is realized as synchronous digital system with distributed encoded state register $Q_1 \ldots Q_9$ and distributed output register $Y_1 \ldots Y_{12}$. In general state register and output register can be merged. All concurrently enable transitions can fire independently, in any order. It is considered that after animation and classical analysis, the implemented interpreted Petri net is checked as safe, live, reversible and without conflicts, which are not solved [10],

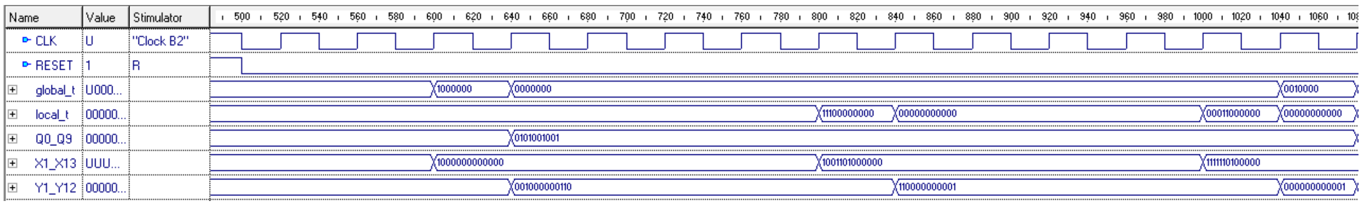

Fig. 9. Simulation results from AHDL tool.

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

entity controller is
    port (CLK, RESET: in std_logic;
        X1, X2, ..., X12, X13: in std_logic;
        Y1, Y2, ...,Y12: out std_logic);
end controller;

architecture controller_ort of controller is
signal q0, q1, ..., q9: std_logic;
signal y_0, y_1, y_2, ..., y_12: std_logic;
signal t1, t2, t3, ..., t18: std_logic;

begin
  --Precondition of  border transitions
  t1 <= not y_9 and x1 and ...  and not q9;
  t7 <= q1 and q3 and y_12 and x13 and not q0
         and not q2;
  ...
  t17 <= q2 and q4 and q5 and q0 and q7 and q8
         and not y_7 and not y_8 and not q6
         and not q6 and not q1 and not q9;

  --Precondition of local transitions
  t2 <= y_10 and x5;
  t3 <= y_11 and x7;
  ...
  t18 <= y_9 and x12;

  FF:process (CLK, RESET) -- Transition firing
  begin
     if RESET = '1' then
         q0 <= '0';  ... ; q9 <= '0';
         y_1 <= '0'; ...; y_12 <= '0';
     elsif rising_edge(CLK) then
         q0 <= q0 xor (t7 xor t17);
         q1 <= q1 xor (t1 xor t14);
         ...
         q9 <= q9 xor (t1 xor t12);

         y_1 <= (y_1 and q6) xor (t2 xor t5);
         y_2 <=  (y_2 and q9) xor (t3 xor t6);
          ...
         y_12 <=(y_12 and q1 and q3) xor (t4 xor t7);
     end if;
  end process;
-- Outputs
Y1 <= y_1;
Y2 <= y_2;
...
Y12 <= y_12;
end controller_ort;
```

Fig. 10. Part of VHDL template.

[19]. Anyway if some transitions of net would be in conflicts or net is not safe (1-bounded), the detected partial state of the net is frozen (state changes stop). Registered outputs can be used both for precondition and local states coding. The simulation results obtained from Active-HDL tool is shown in Fig. 9.

The macroplace-centered and place-centered decomposition and encoding of SM-colored Petri net is preferable from FPGA resources utilization point of view [22], [23]. Additionally it make possible flexible reusing of previously tested, encoded Petri net components. Synthesis after classic hierarchical one-hot state encoding of macroplaces and places needs 35 flip-flops. In case of economical rapid concurrent one-hot local encoding of merged places and registered outputs (Fig. 10) it is necessary to use only 22 shared additional encoding variables. Synthesis result using Xilinx Vertex V50 are presented in Tab. VI.

TABLE VI
DEVICE UTILIZATION SUMMARY

Logic utilization	Used	Available	Utilization
Number of Slices	37	768	4%
Number of FlipFlops	22	1536	1%
Number of LUTs	70	1536	4%
Number of IOBs	27	180	15%
Number of GCLKs	1	4	25%

Hierarchical encoding using macroplaces and registered outputs gives balanced economical synthesis results as well flexibility during redesign of the controller. The coordination places serve also as flags during partial reconfiguration of the net. After modification of the mixing feeder from mechanical part (Fig. 1) it is easy to find local places P_{10}, P_{14}, P_{21}, P_{22}, P_{22} and P_{24}, which are encapsulated in MP_4 and replace them by another subset without destroying the other parts of previous design.

X. Summary

Hierarchical encoding using macroplaces and registered outputs gives economical synthesis results as well flexibility during redesign of the controller. The coordination places serve also as flags during partial reconfiguration of the net. It is important to note that the designed system encoded using the structural (modular) method and described in VDHL hardware description language - can be easily modified by specifying locally only the part of it included into macroplaces. In such a way FPGA implementation, can be easily re-designed [24].

The paper concentrated on behavioral and structural specification of reconfigurable logic controllers (RLC). The initial description is given as a hierarchical modular control interpreted Petri net. On the abstract level of the logic synthesis specification is written in propositional sequent language. It enable us to make all design transformation using formal reasoning methods. Rapid modeling and synthesis in FPGA can be done from expressions written in the hardware description languages, for example VHDL.

REFERENCES

[1] M. Adamski, A. Karatkevich, and M. Węgrzyn, *Design of Embeded Control Systems*. New York: Springer Science+Business Media, Inc., 2005.

[2] J. Tkacz and M. Adamski, "Logic design of structured configurable controllers," in *Proceedings of IEEE 3rd International Conference on Networked Embedded Systems for Every Application NESEA'12*, Liverpool, United Kingdom, 2012, p. 6.

[3] M. Adamski and M. Węgrzyn, "Petri nets mapping into reconfigurable logic controllers," *Electronics and Telecommunications Quarterly*, vol. 55, no. 2, pp. 157–182, 2009.

[4] M. Adamski, "Specification and synthesis of Petri net based reprogrammable logic controller," in *Proceedings of 5th IFAC International Conference on Programmable Devices and Embedded Systems PDeS'01*, Brno, Czech Republic, 2001, pp. 95–100.

[5] K. Biliński, M. Adamski, J. Saul, and E. Dagless, "Petri-net-based algorithms for parallel-controller synthesis," *IEE Proceedings – Computers and Digital Techniques*, vol. 141, no. 6, pp. 405–412, 1994.

[6] M. Doligalski, "Behavioral specification of the logic controllers by means of the hierarchical configurable Petri nets," in *Proceedings of 11th IFAC/IEEE International Conference on Programmable Devices and Embedded Systems PDeS'12*, Brno, Czech Republic, 2012, pp. 80–83.

[7] ——, "Behavioral specification diversification for logic controllers implemented in FPGA devices," in *Proceedings of the 9th Annual FPGA Conference - FPGAworld '12*. Stockholm, Sweden: New York, ACM, 2012, p. 5.

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

[9] A. Yakovlev, L. Gomes, and L. Lavagno, *Hardware Design and Petri Nets*. Boston: Kluwer, 2000.

[10] 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 (STTT)*, vol. 9, no. 3, pp. 213–254, 2007.

[11] I. Grobelna, *Formal Verifikation of Logic Controller Specification by Means of Model Checking*, ser. Lecture Notes in Control and Computer Science. Zielona Góra: University of Zielona Góra Press, Poland, 2013.

[12] 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 PDeS'12*, Brno, Czech Republic, 2012, pp. 1–6.

[13] 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*, Tallinn, Estonia, 2012, pp. 16–21.

[14] J. H. Gallier, *Logic for Computer Science: Foundations of Automatic Theorem Proving*. New York: Harper & Row Publishers, 1985. [Online]. Available: http://www.cis.upenn.edu/ jean/gbooks/logic.html

[15] T. Kozłowski, E. Dagless, J. Saul, M. Adamski, and J. Szajna, "Parallel controller synthesis using Petri nets," *IEE Proceedings – Computers and Digital Techniques*, vol. 142, no. 4, pp. 263–271, 1995.

[16] J. Tkacz and M.Adamski, "Calculation of state machine cover of safe Petri net by means of computer based reasoning," *Measurement Automation and Monitoring*, vol. 57, no. 11, pp. 1397–1400, 2011.

[17] J. Tkacz, "State machine type colouring of Petri net by means of using a symbolic deduction method," *Measurement Automation and Monitoring*, vol. 53, no. 5, pp. 120–122, 2007.

[18] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[19] A. Karatkevich, *Dynamic Analysis of Petri Net-Based Discrete Systems*, ser. Lecture Notes in Control and Information Sciences. Berlin: Springer-Verlag, 2007, vol. 356.

[20] M. Wiśniewska, R. Wiśniewski, and M. Adamski, "Usage of hypergraph theory in decomposition of concurrent automata," *Measurement Automation and Monitoring*, vol. 53, no. 7, pp. 66–68, 2007.

[21] H. Foster, A. Krolnik, and D. Lacey, *Assertion-Based Design*, 2nd ed. Norwell: Kluwer Academic Publishers, 2004.

[22] G. Łabiak, M. Adamski, M. Doligalski, J. Tkacz, and A. Bukowiec, "UML modelling in rigorous design methodology for discrete controllers," *International Journal of Electronics and Telecommunications*, vol. 58, no. 1, pp. 27–34, 2012.

[23] A. Bukowiec, "Synthesis of FSMs based on architectural decomposition with joined multiple encoding," *International Journal of Electronics and Telecommunications*, vol. 58, no. 1, pp. 35–41, 2012.

[24] M. Doligalski, *Behavioral Specification Diversification of Reconfigurable Logic Controllers*, ser. Lecture Notes in Control and Computer Science. Zielona Gra: University of Zielona Gra Press, 2012, vol. 20.