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 an 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 Active HDL (Aldec, USA) and Xilinx 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.

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

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>start the process</td>
</tr>
<tr>
<td>$x_2$</td>
<td>preparation of ingredients in the first container is finished</td>
</tr>
<tr>
<td>$x_3$</td>
<td>preparation of ingredients in the second container is finished</td>
</tr>
<tr>
<td>$x_4$</td>
<td>preparation of tanks is finished</td>
</tr>
<tr>
<td>$x_5$</td>
<td>maximal fluid level in the first container is obtained</td>
</tr>
<tr>
<td>$x_6$</td>
<td>minimal fluid level in the first container is obtained</td>
</tr>
<tr>
<td>$x_7$</td>
<td>maximal fluid level in the second container is obtained</td>
</tr>
<tr>
<td>$x_8$</td>
<td>minimal fluid level in the second container is obtained</td>
</tr>
<tr>
<td>$x_9$</td>
<td>preparation of the drink is finished</td>
</tr>
<tr>
<td>$x_{10}$</td>
<td>filling off the first tank is finished</td>
</tr>
<tr>
<td>$x_{11}$</td>
<td>filling off the second tank is finished</td>
</tr>
<tr>
<td>$x_{12}$</td>
<td>the carriage is in its initial location</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_1$</td>
<td>preparation of the first ingredient</td>
</tr>
<tr>
<td>$y_2$</td>
<td>preparation of the second ingredient</td>
</tr>
<tr>
<td>$y_3$</td>
<td>loading containers</td>
</tr>
<tr>
<td>$y_4$</td>
<td>mixing ingredients</td>
</tr>
<tr>
<td>$y_5$</td>
<td>valve for emptying the first container</td>
</tr>
<tr>
<td>$y_6$</td>
<td>valve for emptying the second container</td>
</tr>
<tr>
<td>$y_7$</td>
<td>valve for filling the first tank</td>
</tr>
<tr>
<td>$y_8$</td>
<td>valve for filling the second tank</td>
</tr>
<tr>
<td>$y_9$</td>
<td>carriage movement to the initial location (right)</td>
</tr>
<tr>
<td>$y_{10}$</td>
<td>valve for filling the first container</td>
</tr>
<tr>
<td>$y_{11}$</td>
<td>valve for filling the second container</td>
</tr>
<tr>
<td>$y_{12}$</td>
<td>carriage movement to the target location (left)</td>
</tr>
</tbody>
</table>

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 $\Rightarrow$ backward implication, symbol $(xor)$ – exclusive or. Capital letters $X_0 - X_{12}$ 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 $T_{i_1}$. If it is necessary next values of registered signals are recognized by a linear temporal logic operator next, denoted as $\Diamond$. In general the current value of a registered signal is presented as $Q$, but its next value is written as $\Diamond Q$ [1], [3], [4].

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.

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 $\Rightarrow$ backward implication, symbol $(xor)$ – exclusive or. Capital letters $X_0 - X_{12}$ 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 $T_{i_1}$. If it is necessary next values of registered signals are recognized by a linear temporal logic operator next, denoted as $\Diamond$. In general the current value of a registered signal is presented as $Q$, but its next value is written as $\Diamond Q$ [1], [3], [4].
The rule-based description of partial state changes in sequential logic for one-hot encoding with direct using of registered outputs $Y_1, \ldots, Y_{11}$ in hierarchical modular array structure can be as follows [12]:

- $\vdash @Y_1 \leftarrow (MP_1$ and $Y_1) \text{xor} (T_2 \text{xor} T_5) : P_5$
- $\vdash @Y_2 \leftarrow (MP_2$ and $Y_2) \text{xor} (T_4 \text{xor} T_6) : P_6$
- $\vdash @Y_3 \leftarrow (MP_3$ and $Y_3) \text{xor} (T_3 \text{xor} T_4) : P_4$
- $\vdash @Y_4 \leftarrow (MP_4$ and $Y_4) \text{xor} (T_8 \text{xor} T_9) : P_{10}$
- $\vdash @Y_5 \leftarrow (MP_5$ and $Y_5) \text{xor} (T_8 \text{xor} T_{10}) : P_{21}$
- $\vdash @Y_6 \leftarrow (MP_6$ and $Y_6) \text{xor} (T_8 \text{xor} T_{11}) : P_{22}$
- $\vdash @Y_7 \leftarrow (MP_6$ and $Y_7) \text{xor} (T_{13} \text{xor} T_{15}) : P_{16}$
- $\vdash @Y_8 \leftarrow (MP_{10}$ and $Y_8) \text{xor} (T_{14} \text{xor} T_{16}) : P_{17}$
- $\vdash @Y_9 \leftarrow (MP_{11}$ and $Y_9) \text{xor} (T_8 \text{xor} T_{10}) : P_{20}$
- $\vdash @Y_{10} \leftarrow (MP_1$ and $Y_{10}) \text{xor} (T_1 \text{xor} T_2) : P_8$
- $\vdash @Y_{11} \leftarrow (MP_2$ and $Y_{11}) \text{xor} (T_1 \text{xor} T_3) : P_3$

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

$$
P_5 = (MP_1 \text{ and } Y_1)
$$
$$
P_6 = (MP_2 \text{ and } Y_2)
$$
$$
\ldots
$$
$$
P_3 = (MP_2 \text{ and } Y_{11})
$$

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

$$
P_1 = (MP_{11} \text{ and not } Y_9)
$$
$$
P_8 = (MP_1 \text{ and not } Y_{10} \text{ and not } Y_1)
$$
$$
\ldots
$$
$$
P_{19} = (MP_{10} \text{ and } Y_8)
$$

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

- $\vdash T_1 \leftarrow MP_{11}$ and not $Y_9$ and $X_3$ and not $X_4$
- $\vdash T_7 \leftarrow MP_3$ and $Y_{12}$ and $X_{13}$
- $\vdash T_8 \leftarrow (MP_4$ and not $Y_{10}$ and not $Y_1)$ and $(MP_2$ and not $Y_{11}$ and not $Y_2)$
- $\vdash T_{12} \leftarrow MP_4$ and not $Y_4$ and not $Y_5$ and not $Y_6$
- $\vdash T_{13} \leftarrow MP_6$ and $MP_7$
- $\vdash T_{14} \leftarrow MP_2$ and $MP_8$
- $\vdash T_{17} \leftarrow MP_9$ and not $Y_7$ and $MP_{10}$ and not $Y_8$

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

- $\vdash T_2 \leftarrow MP_1$ and $Y_{10}$ and $X_5$
- $\vdash T_3 \leftarrow MP_2$ and $Y_{11}$ and $X_7$
- $\vdash T_{18} \leftarrow MP_{11}$ and $Y_9$ and $X_{12}$

The excitations for macroplaces are as follows:

- $\vdash @MP_1 \leftarrow MP_1 \text{xor} (T_3 \text{xor} T_8)$
- $\vdash @MP_2 \leftarrow MP_2 \text{xor} (T_3 \text{xor} T_7)$
- $\vdash @MP_3 \leftarrow MP_3 \text{xor} (T_3 \text{xor} T_7)$
- $\vdash @MP_4 \leftarrow MP_3 \text{xor} (T_3 \text{xor} T_{12})$
- $\vdash @MP_5 \leftarrow MP_6 \text{xor} (T_3 \text{xor} T_{13})$
- $\vdash @MP_6 \leftarrow MP_7 \text{xor} (T_2 \text{xor} T_{13})$
- $\vdash @MP_7 \leftarrow MP_7 \text{xor} (T_2 \text{xor} T_{14})$

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

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

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 $\{c_1, c_2, c_3, c_4\}$. 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 $MP_3[C_4]$ 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 $MP_3[C_2]$ and $MP_b[C_2, C_3]$ 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:

$$M_0 = \{MP_1\}; M_1 = \{MP_1, MP_2, MP_3\}; M_2 = \{MP_3, MP_4, MP_5\}; M_3 = \{MP_3, MP_5, MP_6\}; M_4 = \{MP_3, MP_4, MP_6\}; M_5 = \{MP_3, MP_4, MP_5, MP_6, MP_8\}; M_6 = \{MP_3, MP_5, MP_6, MP_8\}; M_7 = \{MP_5, MP_6, MP_8\}; M_8 = \{MP_5, MP_6, MP_8, MP_10\}; M_9 = \{MP_5, MP_6, MP_8, MP_10\}.$$  

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

$$M \vdash (MP_3), (MP_5, MP_6, MP_8), (MP_5, MP_7, MP_8), (MP_5, MP_6, MP_8), (MP_5, MP_6, MP_8), (MP_5, MP_6, MP_8), (MP_5, MP_6, MP_8), (MP_5, MP_6, MP_8), (MP_5, MP_6, MP_8).$$  

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

$$\vdash (P_1 \text{ or } P_{20}) \text{ and } (P_2 \text{ or } P_3 \text{ or } P_4) \text{ and } (P_5 \text{ or } P_6 \text{ or } P_7) \text{ and } (P_9 \text{ or } P_{10}) \text{ and } \ldots \text{ and } (P_{16} \text{ or } P_{18}) \text{ and } (P_{17} \text{ or } P_{19}) \text{ and } \ldots$$

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 Concurrency 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.

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].
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: \( \{i_3, i_4, i_5, i_6\} \). 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].

\[
\vdash MP_1, MP_2, MP_3, MP_4, MP_5, MP_6, MP_7, MP_8, MP_9, MP_{10}, MP_{11};
\]
\[
\vdash MP_1, MP_2, MP_3, MP_4, MP_5, MP_6, MP_7, MP_8, MP_9, MP_{10}, MP_{11}; \quad (i_1)
\]
\[
\vdash MP_1, MP_2, MP_3, MP_4, MP_5, MP_6, MP_7, MP_8, MP_9, MP_{10}; \quad (i_2)
\]
\[
\vdash MP_2, MP_4, MP_5, MP_6, MP_8, MP_9, MP_{11}; \quad (i_3)
\]
\[
\vdash MP_3, MP_5, MP_6, MP_7, MP_8, MP_9, MP_{10}; \quad (i_4)
\]
\[
\vdash MP_2, MP_3, MP_5, MP_6, MP_7, MP_8, MP_9, MP_{10}; \quad (i_5)
\]
\[
\vdash MP_1, MP_2, MP_3, MP_4, MP_5, MP_6, MP_7, MP_8, MP_9; \quad (i_6)
\]

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).
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 state 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 hyberedges 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 $\{t_1, \ldots, t_6\}$ (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 $\{t_3, t_4, t_1, t_2\}$ 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_{i,0}\{1, 2, 3, 4\} \not\vdash (Q_0 \text{ or } Q_1 \text{ or } \ldots \text{ or } Q_8 \text{ or } Q_9)
\vdash MP_{i,3} \not\vdash (Q_0 \text{ and } Q_5 \text{ and } Q_6
\vdash MP_{2,4} \not\vdash (Q_7 \text{ and } Q_8 \text{ and } Q_9
\vdash MP_{3,1} \not\vdash (Q_0 \text{ and } Q_1) \text{ and } (\text{not } Q_2 \text{ and } Q_3
\vdash MP_{3,4} \not\vdash (Q_0 \text{ and } Q_5 \text{ and } Q_6) \text{ and } (\text{not } Q_7 \text{ and } Q_9 \text{ and } Q_9)
\vdash MP_{2,2} \not\vdash (Q_2 \text{ and } Q_3
\vdash MP_{3,3} \not\vdash (Q_0 \text{ and } Q_0 \text{ and } \text{not } Q_6
\vdash MP_{3,4} \not\vdash (Q_7 \text{ and } Q_8 \text{ and } Q_9
\vdash MP_{3}\{1, 0\} \not\vdash (Q_0 \text{ and } Q_1
\vdash MP_{2,2, 3} \not\vdash (Q_2 \text{ and } not Q_3 \text{ and }
TABLE IV
GROUP SEQUENTS OF TRAPS AND SIPHONS FOR DEFECTED NET

<table>
<thead>
<tr>
<th>Transitions</th>
<th>Traps sequent</th>
<th>Siphons sequent</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁</td>
<td>(MP₁₁ → (MP₁ + MP₂ + MP₅)),</td>
<td>((MP₅ + MP₆ + MP₇) → MP₁₅),</td>
</tr>
<tr>
<td>t₆</td>
<td>(MP₁ + MP₂) → MP₄,</td>
<td>(MP₅ → (MP₁ + MP₂)),</td>
</tr>
<tr>
<td>t₇</td>
<td>(MP₂ → MP₃),</td>
<td>(MP₅ → MP₆),</td>
</tr>
<tr>
<td>t₁₂</td>
<td>(MP₂ → (MP₅ + MP₇)),</td>
<td>((MP₆ + MP₇) → MP₄),</td>
</tr>
<tr>
<td>t₁₃</td>
<td>(MP₆ + MP₈) → MP₉,</td>
<td>(MP₆ → (MP₅ + MP₆)),</td>
</tr>
<tr>
<td>t₁₄</td>
<td>(MP₇ + MP₈) → MP₉₀,</td>
<td>(MP₉₀ → (MP₆ + MP₆)),</td>
</tr>
<tr>
<td>t₁₇</td>
<td>(MP₆ + MP₁₀) → MP₁₃),</td>
<td>(MP₁₁ → (MP₆ + MP₁₀)) ⊢:</td>
</tr>
</tbody>
</table>

Fig. 8. Macronet with defect.

TABLE V
TRAPS AND SIPHONS WITH SELECTED DEFECTS

<table>
<thead>
<tr>
<th>Traps</th>
<th>Siphons</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP₁₅, MP₄, MP₆, MP₉, MP₁₁</td>
<td>MP₁₅, MP₄, MP₆, MP₉, MP₁₁</td>
</tr>
<tr>
<td>MP₁, MP₄, MP₇, MP₁₀, MP₁₁</td>
<td>MP₁, MP₄, MP₇, MP₁₀, MP₁₁</td>
</tr>
<tr>
<td>MP₂, MP₄, MP₆, MP₉, MP₁₁</td>
<td>MP₂, MP₄, MP₆, MP₉, MP₁₁</td>
</tr>
<tr>
<td>MP₂, MP₄, MP₆, MP₉, MP₁₁</td>
<td>MP₂, MP₄, MP₆, MP₉, MP₁₁</td>
</tr>
<tr>
<td>MP₅, MP₆, MP₁₀, MP₁₁</td>
<td>MP₅, MP₆, MP₁₀, MP₁₁</td>
</tr>
<tr>
<td>MP₈, MP₉, MP₁₀, MP₁₁</td>
<td>MP₈, MP₉, MP₁₀, MP₁₁</td>
</tr>
<tr>
<td>MP₈, MP₉, MP₁₀, MP₁₁</td>
<td>MP₈, MP₉, MP₁₀, MP₁₁</td>
</tr>
</tbody>
</table>

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

\[ ⊢ Q₀ \iff Q₀ \oplus T₇ \oplus T₁₇ \]

\[ ⊢ Q₁ \iff Q₁ \oplus T₁ \oplus T₁₄ \]

\[ ⊢ Q₂ \iff Q₂ \oplus T₇ \oplus T₁₇ \]

\[ ⊢ Q₃ \iff Q₃ \oplus T₁ \oplus T₁₃ \]

\[ ⊢ Q₄ \iff Q₄ \oplus T₁₃ \oplus T₁₇ \]

\[ ⊢ Q₅ \iff Q₅ \oplus T₈ \oplus T₁₇ \]

\[ ⊢ Q₆ \iff Q₆ \oplus T₈ \oplus T₁₂ \]

\[ ⊢ Q₇ \iff Q₇ \oplus T₁₄ \oplus T₁₇ \]

\[ ⊢ Q₈ \iff Q₈ \oplus T₈ \oplus T₁₇ \]

\[ ⊢ Q₉ \iff Q₉ \oplus T₈ \oplus T₁₂ \]

The names of macroplaces could be used as aliases in hardware description languages. For rapid prototyping it 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 \( Q₁ \ldots Q₉ \) and distributed output register \( Y₁ \ldots Y₁₂ \). 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 and not q2;
    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_0 <= '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.

Hierarchical encoding using macropalces 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 macropalces 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 macropalces. 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


