INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2012, VOL. 58, NO. 4, PP. 403–410
Manuscript received November 8, 2012; revised Decemer, 2012. DOI: 10.2478/v10177-012-0055-x

Synthesis of Macro Petri Nets into FPGA with Distributed Memories

Arkadiusz Bukowiec and Marian Adamski

***Abstract*—In this paper a new method of Petri net array-based synthesis is proposed. The method is based on decomposition of colored interpreted macro Petri net into state machine subnets. Each state machine subnet is determined by one color. During the decomposition process macroplaces are expanded or replaced by doublers of macroplace. Such decomposition leads to parallel implementation of a digital system. The structured encoding of places is done by using minimal numbers of bits. Colored microoperations, which are assigned to places, are written into distributed and flexible memories. It leads to realization of a logic circuit in a two-level concurrent structure, where the combinational circuit of the first level is responsible for firing transitions, and the second level memories are used for generation of microoperations. Such an approach allows balanced usage of different kinds of resources available in modern FPGAs.**

***Index Terms*—Decomposition, FGPAs, logic synthesis, Petri nets.**

I. Introduction

APPLICATION specific logic controllers or control units [1], [2] are one of the biggest groups of electronic devices. They can be designed as dedicated software for microcontroller or as dedicated hardware. The second approach gives more possibilities of system integration as system on programmable chip (SoPC) with use of field programmable gate arrays (FPGAs). The most classical way of designed such controllers is application of hardware description languages (HDLs) but it is unconformable for designer and potentially it gives high risk of human mistake. The usage of graphical representation of algorithm is much more conformable [3]–[5]. In this case Petri nets (PNs) [6], [7] are one of the most adequate methods for formal design of application specific logic controllers [1]. It gives easy way for representation of concurrent processes and additionally there could be applied mathematical algorithms for formal analysis and verification of the designed model [8]–[11]. There are also several algorithms of direct synthesis of Petri net model into FPGA devices [12]–[15]. The most typical implementation of Petri nets into FPGA devices use one-hot local state encoding where each single place is represented by a flip-flop [16]. Such an approach requires hardware implementation of a large number of several logic functions and flip-flops included in macrocells.

One of the main features of FPGA is an existence of separated logic elements (look-up tables) with restricted fixed number of inputs. Very frequently logic functions have more arguments than number of inputs of such logic element. It forces a functional decomposition during a synthesis process and consumes a large number of logic elements. One of the methods of decreasing a number of such functions is architectural decomposition of a sequential circuit [17], [18]. Such methods introduce several additional internal variables and very often consume more hardware than typical direct implementation. This issue can be resolved by using logic elements together with embedded memory blocks [19] that are available in modern FPGA devices.

There is proposed the method of synthesis that allows to decrease the number of implemented logic functions depending on inputs and internal variables of Petri net-based logic controller in the paper. The logic functions are classified into two sets. The first set contains functions responsible for describing preconditions and guards of transitions. This set is going to be synthesized with use of logic elements. The second set contains functions responsible for generation of microoperations and it is going to be realized with use of the embedded memory blocks. To permit the minimal local state encoding the Petri net is initially colored [20] and it is compacted into macro Petri net [21]. Macroplaces that are colored by the same color create one state machine module. Consequently, places, represented by these macroplaces, could be encoded by a minimal-length binary vector. This encoding also allows a reasonable decomposition of a microoperation decoder into several concurrently working distributed memories. Each memory block controls only microoperations that belong to the subnet with the same color. A new procedure of extracting subnets supplements the known methods of Petri net SM coloring [1], [13], [16], [22]–[24]. In such a way it leads to balanced usage of all kinds of logic resources of the FPGA device. Very frequently such a method gives also an effective utilization of all FPGA resources by a whole digital system.

II. Petri Net and Its Extensions

A simple Petri net [6], [7] 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, $T = \{t_1, \ldots, t_S\}$
- F is a set of flow relations called arcs from places to transitions and from transitions to places:

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

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

A. Bukowiec 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: {a.bukowiec, m.adamski}@iie.uz.zgora.pl).

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

$$\bullet p_m \quad = \{t_s \in T : (t_s, p_m) \in F\},$$
$$p_m \bullet \quad = \{t_s \in T : (p_m, t_s) \in F\}.$$

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

$$\bullet t_s \quad = \{p_m \in P : (p_m, t_s) \in F\},$$
$$t_s \bullet \quad = \{p_m \in P : (t_s, p_m) \in F\}.$$

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 contain a token it is marked. A transition t_s can be fired if all its input places are marked. Firing of a a transition removes tokens from its input places and puts one token in each output place. There can be specified the initial marking M_0, then the Petri net is defined as a tupe:

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

A. Colored Petri Net

A Petri net can be enhanced by assigning colors to places and transitions [1], [6], [20]. In state machine (SM) colored Petri net colors help to validate intuitively and formally the consistency of all sequential processes covering the considered Petri net. Each color recognizes one SM-subnet. The rules for Petri net coloring are as follows [13], [22]:

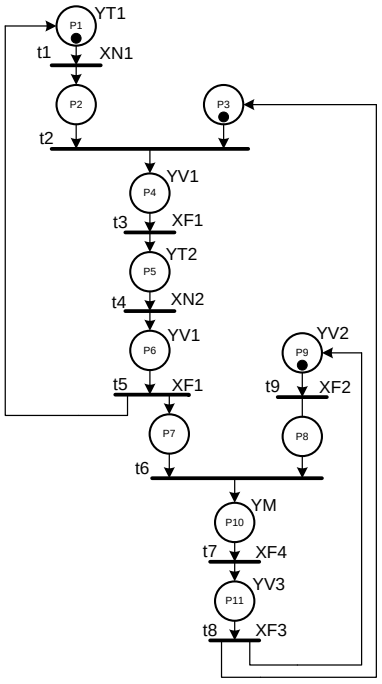
- each place and transition must have at least one color,
- if the place has a color each of its input and output transition must have the same color,
- input places of each transition must hold different colors,
- output places of each transition must hold different colors,
- input and output places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

B. Interpreted Petri Net

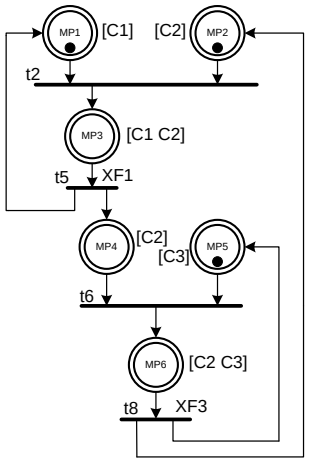
An interpreted Petri net is a Petri net enhanced with an additional feature for information exchange [7]. Such a Petri net is called interpreted Petri net or a colored interpreted Petri net if both enhancements are applied. This exchange is made by use of binary signals. Interpreted Petri nets are used as models of concurrent logic controllers.

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 $Z = \varnothing$.



(a) Interpreted Petri net



(b) Colored interpreted macro Petri net

Fig. 1. Example of Petri net PN_1.

The interpreted Petri net has a guard condition φ_s associated with every transition t_s. The condition φ_s is defined as the Boolean function of some variables form sets X and Z. In the particular case the condition φ_s can be defined as 1 (always true). Now, transition t_s can be fired if all its input places are marked and current value of corresponding Boolean function φ_s is equal to 1. Conjunction ψ_m is associated with place p_m. ψ_m is an elementary conjunction of affirmation of some output variables form the set Y. If the place p_m is marked the output variables from corresponding conjunction ψ_m are set and other variables are reset.

C. Macro Petri Net

Macro Petri net is a Petri net where part of the net (subnet) is replaced by one macroplace [7]. It allows to enhance Petri

TABLE I
COLORS OF MACRO PETRI NET MPN_1

Color C_i	Macroplaces P_i	Transitions T_i	Outputs Y_i
C_1	$\{mp_1, mp_3\}$	$\{t_2, t_5\}$	$\{yt_1, yt_2, yv_1\}$
C_2	$\{mp_2, mp_3, mp_4, mp_6\}$	$\{t_2, t_5, t_6\}$	$\{yt_2, yv_1, yv_3\}$
C_3	$\{mp_5, mp_6\}$	$\{t_6, t_8\}$	$\{yv_2, yv_3, ym\}$

nets with hierarchy [8] and it simplifies algorithms of coloring and verification of Petri net. There are many classes of subnets that could be replaced by macroplace, for e.g.:

- State machine subnets [6],
- Two-pole blocks [25],
- Parallel places [6],
- P-blocks [7].

These classes create to many possibilities of merging Petri net into macro Petri net. For the synthesis purpose, the best solution is application of mono-active macroplaces [25]. This is macroplaces that have one input and one output and consist of only sequential places. Only macro Petri nets with such macroplaces will be used in this article.

The example of Petri net PN_1 is given in Fig. 1a. This net contains $M = 11$ places and $S = 9$ transitions. The initial marking is defined as $M_0 = \{p_1, p_3, p_9\}$. The set of input variables is $X = \{xn_1, xn_2, xf_1, xf_2, xf_3, xf_4\}$ and output variables is $Y = \{yt_1, yt_2, yv_1, yv_2, yv_3, ym\}$. It can be merged into macro Petri net MPN_1 (Fig. 1b) and then it is colored by $I = 3$ colors and covered by three SM-macrosubnets C_1, C_2, and C_3 (Tab. I). It can be noticed that some places, eg. mp_3 or mp_6, and some transitions, eg. t_2 or t_8, are colored by more than one color and they belong to several SM-macrosubnets.

III. IDEA OF SYNTHESIS METHOD

The idea of proposed synthesis method is based on the minimal local states encoding of places together with functional parallel decomposition of the Petri net-based logic circuit. Places are encoded separately in every colored subset. Output variables (names of particular microoperations) assigned to places are placed in configured memories of FPGA. It leads to realization of a logic circuit in two-level structure (Fig. 2), where the combinational circuits (CC^i) of first level are responsible for generation of the excitation functions: where $Q = Q^1 \cup Q^2 \cup \cdots \cup Q^I$ is the set of variables used to store the codes of currently marked places. The memory of the circuit is built from I concurrent colored D-type registers RG^i which hold a current state of each subnet. Here, $i = 1, 2, \ldots, I$ is a number of color of SM-subnet in Petri net colored by I colors. The second level decoders Y^i are responsible for generation of microoperations and they are implemented using memory blocks. Their functionality can be described by function:

$$Y^i = Y^i(Q^i). \tag{3}$$

Such approach allows to use logic elements and embedded memory blocks available in modern FPGA devices in a balanced way.

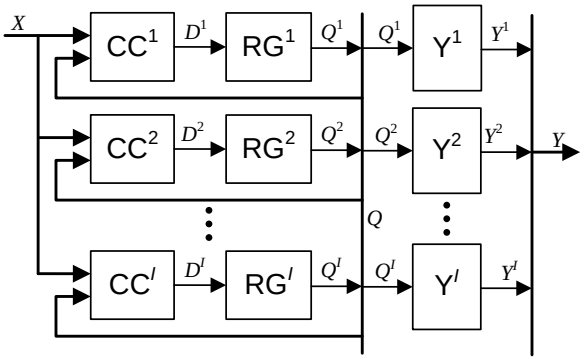


Fig. 2. Logic circuit of Petri net.

$$D^i = D^i(X, Q), \tag{4}$$

The entry point to the synthesis method is the colored interpreted macro Petri net. There are many algorithms of SM-coloring Petri nets, for example, the one described in: [23], [24]. The outline of synthesis process includes following steps:

1) *Formation of subnets.* The purpose of this step is to extract subnets from the colored macro Petri net. Let assume that the macro Petri net is colored with I different colors. Let us start from the first color ($i = 1$). All macro places colored by this color are expanded into corresponding subnets and these subnets create the first SM-subnet. Next subnets are created in a similar way. All macroplaces, which have been previously selected (and expanded) by already created colored SM-subnets are replaced by double of macroplaces. The doubles (clones) of these macroplaces appear in a new colored SM-subnet and they do not have any output signals assigned. Other macroplaces (not previously selected) are expanded into corresponding subnets and these subnets appear n a new M-subnet. There can be several doubles of macroplaces in one subnet but if some of them occur in a sequence then they can be replaced by one double of all macroplaces. The doubles are treated as normal place in the next steps. The procedure of formations of subnets expands each macro place only once. If any macro place is colored by more than one color it is replaced by its double in followings SM-subnets. In comparison with method where the colored Petri net is an entry point to the synthesis method [26], now, there is required to apply procedure od macroplaces expansion less times than procedure of places merging when Petri net is considered. Now, also outputs are partitioned by colors and each output belongs only to one SM-subnet.
2) *Encoding of places.* The purpose of this step is to assign the shortest binary code to each place. The encoding is done on minimal number of required bits. Places are encoded separately in each SM-subnet. It is required to use

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

bits to encode them, where $P_i \subseteq P \cup MP_i$ is a set of places in a subnet that was created based on the color

C_i. MP_i is the set of doubles added to this subnet. There are used variables from set $Q^i \subset Q$ to store this code, where $Q = \{q_0, \ldots, q_{R-1}\}$, where

$$R = \sum_{i=1}^{I} R_i \tag{6}$$

and $Q^i = \{q_{\rho-R_i}, \ldots, q_{\rho-1}\}$, where $\rho = \sum_{i=1}^{i} R_i$. Places that belong to the initial marking set M_0 get code equal to 0. If the subnet does not include any place from the initial marking set M_0 the code equal to 0 should be assigned to the double of macroplace that has replaced such place.

3) *Formation of conjunctions.* Conjunctions describe places, preconditions of transitions and place hold-conditions. They are needed for easier creation of equations that describe digital systems (4). The conjunction describing the place p consists only of affirmation or negation of variables q_r that are used to store the code of this place. If the r-th bit of the code equals 0 then negation is used and if it equals 1 then affirmation is used. The characteristic functions describing the pre-condition of transition t consists of place conjunctions of input places to this transition (from all subnets) and a guard condition φ assigned to this transition. The hold-conjunction of place p consists of negation of the sum of transition conjunctions of all its output transitions (from all subnets) and its place conjunction.
4) *Formation of logic equations.* Logic equations describe functions (4) of combinational circuits CC^i. They create the base for D flip-flop equations, which are built from conjunctions describing preconditions of transitions 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.
5) *Formation of memory contents.* The memory content can be described as a table or as equations according to the system (3). There is required to create I such tables. The table consists of two columns. First column is an address and it is described by variables $q_r \in Q^i$. The second column is an operation. The operation is represented by output variables form the set $Y^i \subseteq Y$. Here, the set Y^i consists only of variables y_n that are under control of the subnet colored by the color C_i. It means, they are in elementary conjunctions ψ that are associated with the places belonging to the set P^i. In each line of the table, there should be placed only these variables y_n that are in elementary conjunction ψ that is associated with the place represented by address from the first column of this line. The other way to describe this memory is to create one logic equation to describe each output variable. It describes output variable as a sum of place conjunctions of the places corresponding to the elementary conjunctions ψ that contain corresponding output variable.
6) *Formation of logic circuit and implementation.* This step describes the rules of creation of the Petri net HDL model and its implementation into FPGA device. Here is applied a bottom-up approach. Conjunctions of places and transitions can be described using standard bit-wise operators. Then logic equations can be described with the use of these conjunctions using continuous assignments as well as bit-wise operators. There should be created a separate module for each circuit CC^i with inputs X and Q and outputs D^i (Fig. 3). The register RG^i should be described as R_i-bits D-type register with an asynchronous reset (Fig. 4). The typical synthesis template can be used [27]. Each memory Y^i can be described as a process with the case" statement (Fig. 5). As, the embedded memory blocks are synchronous, the sensitivity list of this process includes clock signal. The

```
entity cci is
 port(X : in STD_LOGIC_VECTOR(L-1 downto 0);
      Q : in STD_LOGIC_VECTOR(R-1 downto 0);
      D : out STD_LOGIC_VECTOR(Ri-1 downto 0)
     );
end cci;
architecture cci_arch of cci is
 signal p : STD_LOGIC_VECTOR(1 to M);
 signal mp : STD_LOGIC_VECTOR(1 to Mm);
 signal t : STD_LOGIC_VECTOR(1 to S);
 signal hp : STD_LOGIC_VECTOR(1 to M);
 signal hmp : STD_LOGIC_VECTOR(1 to Mm);
begin
    p(1) <= ...;
    ...
    mp(1) <= ...;
    ...
    t(1) <= ...;
    ...
    hp(1) <= ...;
    ...
    hmp(1) <= ...;
    ...
    D(0) <= ...;
    ...
end cci_arch;
```

Fig. 3. Template of combinational circuit in VHDL.

```
entity RGi is
 port(CLK : in STD_LOGIC;
      RES : in STD_LOGIC;
      D : in STD_LOGIC_VECTOR(Ri-1 downto 0);
      Q : out STD_LOGIC_VECTOR(Ri-1 downto 0)
     );
end RGi;
architecture RGi_arch of RGi is
begin
   process (CLK, RES) begin
      if RES='1' then
        Q <= (others => '0');
      elsif (CLK'event and CLK='1') then
        Q <= D;
      end if;
    end process;
end RGi_arch;
```

Fig. 4. Template of register in VHDL.

```
entity yi is
 port(CLK : in STD_LOGIC;
      RES : in STD_LOGIC;
      Q : in STD_LOGIC_VECTOR(Ri-1 downto 0);
      Y : out STD_LOGIC_VECTOR(Ni-1 downto 0)
     );
 attribute bram_map: string;
 attribute bram_map of yi: entity is ''yes'';
end yi;
architecture yi_mem of yi is
begin
   process (CLK) begin
      if (CLK'event and CLK='0') then
        if RES='1' then
          Y <= ...;
        else
          case Q is
              when ... => Y <= ...;
              ...
              when OTHERS => Y <= ...;
          end case;
        end if;
      end if;
   end process;
end yi_mem;
```

Fig. 5. Template of microoperation decoder in VHDL.

reset has to be realized as a synchronous one because typical memory blocks do not support any asynchronous control signal. To ensure that such a described module could be synthesized as a memory block it is required to set the value of the special synthesis directive. The syntax of this directive depends on FPGA vendor. The presented template (Fig. 5) include such directive for Xilnix devices. The top-level module should describe connections of all modules according to the block diagram presented in Fig. 2. Additionally the global reset signal is connected to reset inputs of registers and memories. The global clock signal is connected to the clock inputs of registers and memories. The edge that trigs the memories has to be opposite to the edge that trigs the registers. Operations are generated during only one clock cycle [19]. The created model of logic circuit can be passed into third-party synthesis tool.

IV. Example of Method Application

The method of Petri net synthesis, described in the previous section, is illustrated by its application on macro Petri net MPN_1 (Fig. 1b). The Petri net PN_1 (Fig. 1a) describes control process of industrial mixer of aggregate content and water (Fig. 6) [28], [29]. Outputs of the controller are connected into valves of tanks and engine of mixer. Inputs gives information about state of tanks, scale, timer and flow meter.

The macro Petri net MPN_1 is colored using three colors, and such colored macro Petri net can be an entry point to the synthesis method.

Firstly, subnets have to be formatted (step 1.) The first SM-subnet (Fig. 7) consists of expanded all macroplaces colored by color C_1. The second one (Fig. 7) has one double of macroplace dmp_1. This double of macroplace replaces the

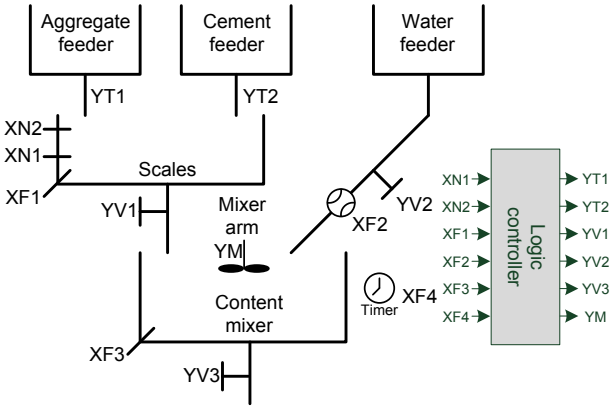
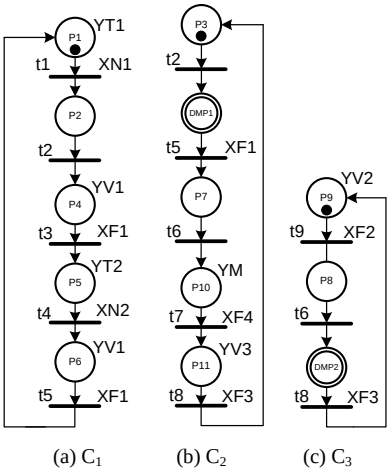


Fig. 6. Object of application specific logic controller.

TABLE II
Received Partitions of Petri Net PN_1

Color C_i	Places P_i	Transitions T_i	Outputs Y^i
C_1	$\{p_1, p_2, p_4, p_5, p_6\}$	$\{t_1, t_2, t_3, t_4, t_5\}$	$\{yt_1, yt_2, yv_1\}$
C_2	$\{p_3, dmp_1, p_7, p_{10}, p_{11}\}$	$\{t_2, t_5, t_6, t_7, t_8\}$	$\{yv_3, ym\}$
C_3	$\{p_9, p_8, dmp_2\}$	$\{t_6, t_8, t_9\}$	$\{yv_2\}$



(a) C_1 (b) C_2 (c) C_3

Fig. 7. Extracted subnets of macro Petri net MPN_1.

macroplace mp_3 because this macroplace have been already expanded in the first subnet. Other macroplaces (mp_2 and mp_4) are expanded in the second SM-subnet. There is also created one double of macroplace dmp_2 in the third subnet (Fig. 7) and it replaces the macroplaces mp_6. Obtained subnets combine macro Petri net MPN_1 with Petri net PN_1 (Tab. II) and they are shown in Fig. 7. It has to be mentioned that output signals are now only under control of one SM-subnet.

Then, the places could be encoded (step 2.) It is required to use $R_1 = 3$, $R_2 = 3$, and $R_3 = 2$ bits, according to (5), to encode all places. And possible encoding of these places is shown in Tab. III. As, $M_0 = \{p_1, p_3, p_9\}$, places p_1, p_3, and p_9 receive codes equal to 0.

When encoding is finished conjunctions can be created (step 3.) The place conjunctions are created on the base of the place

TABLE III
ENCODING OF PLACES OF SM-SUBNETS

Color C_1		Color C_2		Color C_3	
Place	Code $q_2q_1q_0$	Place	Code $q_5q_4q_3$	Place	Code q_7q_6
p_1	000	p_3	000	p_9	00
p_2	001	dmp_1	001	p_8	01
p_4	010	p_7	010	dmp_2	10
p_5	011	p_{10}	011		
p_6	100	p_{11}	100		

codes. For the Petri net PN_1, they are denoted as:

$$\begin{aligned} p_1 &= \overline{q_2} \wedge \overline{q_1} \wedge \overline{q_0}, \\ p_2 &= \overline{q_2} \wedge \overline{q_1} \wedge q_0, \\ &\ldots, \\ dmp_2 &= q_7 \wedge \overline{q_6}. \end{aligned}$$

Each Precondition of the transition is created as conjunction. This conjunction itself consists of conjunctions of transition input places and of a transition condition. They are denoted as:

$$\begin{aligned} t_1 &= p_1 \wedge xn_1, \\ t_2 &= p_2 \wedge p_3, \\ &\ldots, \\ t_8 &= p_{11} \wedge dmp_2 \wedge xf_3, \\ t_9 &= p_9 \wedge xf_2. \end{aligned}$$

Each place hold-conjunction is created as conjunction. This conjunction itself consists of negation of sum of all conjunctions of transitions of output places and of a place conjunction. They are denoted as:

$$\begin{aligned} hp_1 &= \overline{t_1} \wedge p_1, \\ hp_2 &= \overline{t_2} \wedge p_2, \\ &\ldots, \\ hmp_2 &= \overline{t_8} \wedge dmp_2. \end{aligned}$$

Next, logic equation describing combinational circuits can be formed (step 4.) There have to be created equations for each D_r variable. For example, the equation for D_0 is denoted as:

$$\begin{aligned} D_0 &= t_1 \vee hp_2 \\ &\vee \quad t_3 \vee hp_5. \end{aligned}$$

As the variable q_0 is equal to 1 in the code of places p_2, and p_5 the sum of equation D_0 consists of all input transition conjunctions of these places. In this case they are: t_1, and t_3. Additionally, there has to be added hold-condition for all places if the place code has to be stored in register for longer period than one clock cycle. In the similar way there are formed remaining equations for D_1 to D_7. Of course, these equations can be minimized after putting conjunctions instead of corresponding variables. But this manipulation will be done automatically during synthesis and implementation process by third party CAD tools.

TABLE IV
OPERATION MEMORIES TABLES OF PETRI NET PN_1

Memory Y^1		Memory Y^2		Memory Y^3	
Address $q_2q_1q_0$	Operation $yt_1yt_2yv_1$	Address $q_5q_4q_3$	Operation yv_3ym	Address q_7q_6	Operation yv_2
000	100	000	00	00	1
001	000	001	00	01	0
010	001	010	00	10	0
011	010	011	01		
100	001	011	10		

TABLE V
PARAMETERS OF LOGIC CIRCUIT OF PETRI NET PN_1

	PN	PN-MOs
Number of logic equations	17	8
Number of Flip-Flops	11	8
Memory bits	–	52

Then, contents of operation memories can be formed (step 5.) There is required to create separate table for each SM-subnet. In case of Petri net PN_1 there are presented three such tables which are shown in Tab. IV.

Finally, the logic circuit can be built (step 6.) In our case it was described in VHDL using presented templates (Figs. 3–5). But in similar way it can be also described with the use of Verilog. The top-level module (Fig. 8) was drew using block diagram environment of Active-HDL. The key parameters of the received logic circuit are shown in the column PN-MOs in Tab. V. For the comparison purpose, the parameters of logic circuit obtained as a result of standard method of synthesis oriented on the places are shown in the column PN. The standard method requires more logic equations because it is necessary to create one equation per each place and one equation per each output. Proposed method reduces the number of those expressions. Now, there is required to create logic equation only for excitation functions because minimal encoding of the places has been applied and there is no need of creations output equations because they are generated by distributed memories.

The simulation of whole design was also preformed in Active-HDL. The sample results (Fig. 9) shown one cycle of whole mixing process.

V. SUMMARY

The method of synthesis of application specific logic controllers into FPGAs with embedded memory blocks was presented in this article. The Petri net as graphical representation of algorithm [3], [29]–[31] is used as entry point to the synthesis method. The logic circuit is obtained by application of the architectural decomposition methods [18], [32]. There are implemented separate blocks to generate logic equations for each subnet and to generate outputs of each subnet. Then the special method of logic synthesis is proposed. The digital design is based on minimal encoding of places in each subnet separately. As the output functions are extracted to autonomic blocks they can be implemented as concurrent embedded memory blocks. It leads to balanced usage of different kinds of logic resources of FPGA device. The presented method is

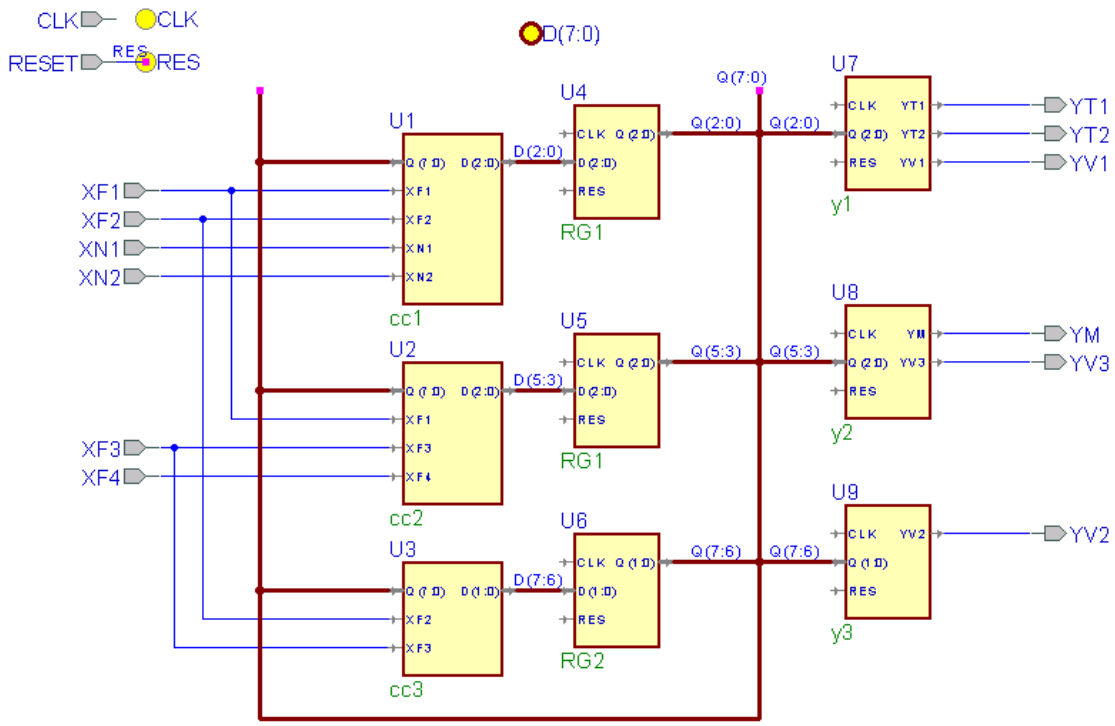


Fig. 8. Top-level module of Petri net PN_1.

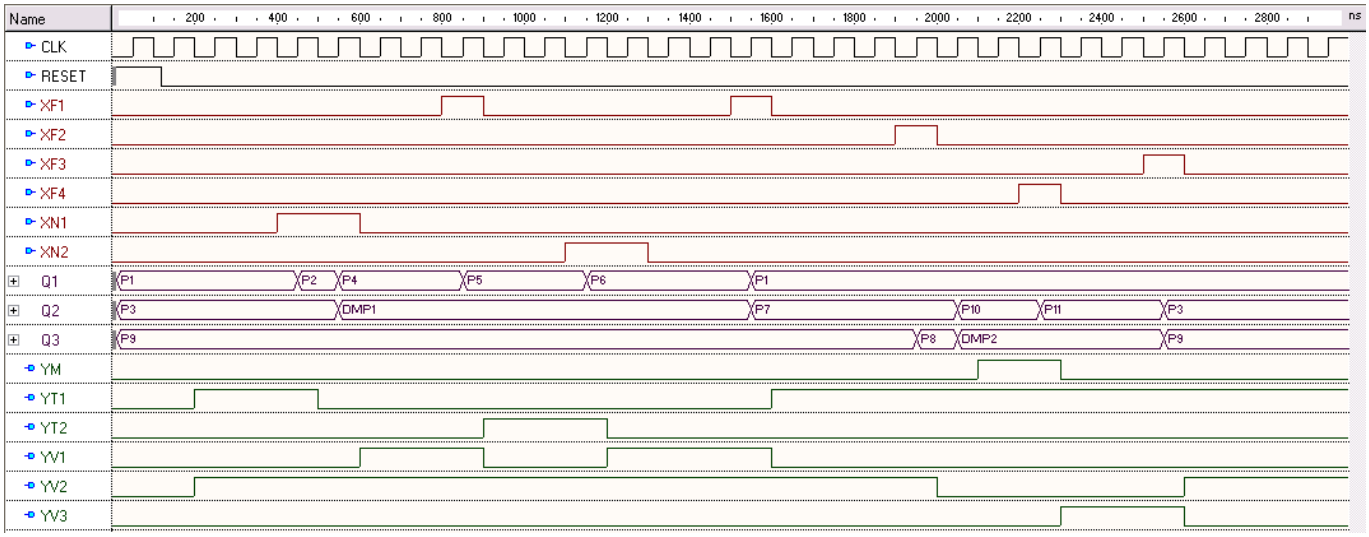


Fig. 9. Simulation of Petri net PN_1.

fully automated and it could be easily integrated with design tools in CAD system.

The method was illustrated by a simple example, which shows the outline of the proposed method. It is dedicated to highly concurrent Petri nets with a large number of inputs and outputs. It gives special benefits when output equations are complicated and excitation functions has high number of arguments comparing with application of standard method of synthesis.

The application of macro Petri net as an entry point to the synthesis method allows to simplify the process of extracting SM-subnets. The state encoding based on separated SM-components is presented, among the others in [33] and in several subsequent papers. Such kind of procedures were also proposed in several early papers related to parallel controller design [7], [16]. The main disadvantage of the method presented in [33] is that the codes of Petri net places can only recognize the current local states of this Petri net and consequently its global state. They cannot be used to determine the next states of the petri net. Unfortunately, the characteristic functions of some places are represented as disjunctions of conjunctions of encoding variables. Different methods and

strategies of the parallel decompositions of safe Petri net SM-components can be found in [6], [8], [23], [24].

REFERENCES

[1] M. Węgrzyn, P. Wolański, M. Adamski, and J. Monteiro, "Coloured Petri net model of application specific logic controller programs," in *Proceedings of IEEE International Symposium on Industrial Electronics ISIE'97*, vol. 1. Guimarães, Portugal: Piscataway, 1997, pp. 158–163.

[2] M. Adamski, "Application specific logic controllers for safety critical systems," in *14th World Congress of IFAC International Federation of Automatic Control*, vol. Q: Transportation Systems: Computer Control. Beijing, China: Oxford, International Federation of Automatic Control, 1999, pp. 519–524.

[3] D. Drusinsky and D. Harel, "Using statecharts for hardware description and synthesis," *IEEE Transactions on Computer-Aided Design*, vol. 8, no. 7, pp. 798–807, 1989.

[4] G. Łabiak, "From UML statecharts to FPGA - the HiCoS approach," in *Proceedings of the Forum on Specification & Design Languages FDL'03*. Frankfurt, Germany: ECSI, 2003, pp. 354–363.

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

[6] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=24143

[7] A. Karatkevich, *Dynamic Analysis of Petri Net-Based Discrete Systems*, ser. Lecture Notes in Control and Information Sciences. Berlin: Springer-Verlag, 2007, vol. 356, DOI: 10.1007/978-3-540-71560-3.

[8] J. Esparza and M. Silva, "On the analysis and synthesis of free choice systems," in *Advances in Petri Nets 1990*, ser. Lecture Notes in Computer Science, G. Rozenberg, Ed. Berlin/Heidelberg: Springer-Verlag, 1991, vol. 483, pp. 243–286.

[9] K. Barkaoui and M. Minoux, "A polynomial-time graph algorithm to decide liveness of some basic classes of bounded Petri nets," in *Application and Theory of Petri Nets*, ser. Lecture Notes in Computer Science, K. Jensen, Ed. Berlin/Heidelberg: Springer, 1992, vol. 616, pp. 62–75, DOI: 10.1007/3-540-55676-1_4.

[10] A. Karatkevich and T. Gratkowski, "Analysis of the operational Petri nets by a distributed system," in *Proceedings of the International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science TCSET'04*, Lviv Polytechnic National University. Lviv, Ukraine: Lviv, Publishing House of Lviv Polytechnic, 2004, pp. 319–322.

[11] A. Węgrzyn, "Parallel algorithm for computation of deadlocks and traps in Petri nets," in *10th IEEE International Conference Emering Technologies and Factory Automation ETFA'05*, vol. 1, Universita di Catania. Catania, Italy: Piscataway, IEEE Operation Center, 2005, pp. 143–148.

[12] M. Adamski, "Petri nets in ASIC design," *Applied Mathematics and Computer Science*, vol. 3, no. 1, pp. 169–179, 1993.

[13] 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.

[14] L. Gomes, A. Costa, J. Barros, and P. Lima, "From Petri net models to VHDL implementation of digital controllers," in *33rd Annual Conference of the IEEE Industrial Electronics Society IECON'07*. Taipei, Taiwan: IEEE, 2007, pp. 94–99.

[15] M. Węgrzyn and A. Węgrzyn, "Penlogic – system for concurrent logic controllers design," in *Design of Digital Systems and Devices*, ser. Lecture Notes in Electrical Engineering, M. Adamski, A. Barkalov, and M. Węgrzyn, Eds. Berlin: Springer-Verlag, 2011, vol. 79, pp. 215–228.

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

[17] T. Łuba, M. Rawski, and Z. Jachna, "Functional decomposition as a universal method of logic synthesis for digital circuits," in *Proceedings of the 9th International Conference Mixed Design of Integrated Circuits and Systems MixDes'02*, Wrocław, Poland, 2002, pp. 285–290.

[18] A. Bukowiec and A. Barkalov, "Structural decomposition of finite state machines," *Electronics and Telecommunications Quarterly*, vol. 55, no. 2, pp. 243–267, 2009.

[19] A. Bukowiec, *Synthesis of Finite State Machines for FPGA devices based on Architectural Decomposition*, ser. Lecture Notes in Control and Computer Science. Zielona Góra: University of Zielona Góra Press, 2009, vol. 13.

[20] 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.

[21] 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.

[22] 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.

[23] A. Węgrzyn, "On decomposition of Petri net by means of coloring," in *Proceedings of IEEE East-West Design & Test Workshop EWDTW'06*, Sochi, Russia, 2006, pp. 407–413.

[24] 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.

[25] A. Karatkevich, "On macroplaces in Petri nets," in *Proceedings of IEEE East-West Design & Test Symposium EWDTS'08*. Lviv, Ukraine: IEEE, 2008, pp. 418–422.

[26] 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, DOI: 10.1109/DDECS.2012.6219016.

[27] K. Skahill, *VHDL for Programmable Logic*. Redwood City: Addison-Wesley Publishing, 1996.

[28] L. Gniewek and J. Kluska, "Hardware implementation of fuzzy Petri net as a controller," *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, vol. 34, no. 3, pp. 1315–1324, 2004.

[29] 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, DOI: 10.2478/v10177-012-0004-8.

[30] G. Borowik, M. Rawski, G. Łabiak, A. Bukowiec, and H. Selvaraj, "Efficient logic controller design," in *Fifth International Conference on Broadband and Biomedical Communications IB2Com'10*, Malaga, Spain, 2010, pp. [CD–ROM].

[31] 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.

[32] A. Barkalov and L. Titarenko, *Logic Synthesis for FSM-based Control Units*, ser. Lecture Notes in Electrical Engineering. Berlin: Springer-Verlag, 2009, vol. 53.

[33] E. Pastor and J. Cortadella, "Efficient encoding schemes for symbolic analysis of Petri nets," in *Proceedings of the Conference on Design, Automation and Test in Europe DATE'98*. Paris, France: IEEE Computer Society Press, 1998, pp. 790–795.