Transition Based Synthesis with Modular Encoding of Petri Nets into FPGAs

Arkadiusz BUKOWIEC, Jacek TKACZ, Marian ADAMSKI

Institute of Computer Engineering and Electronics,
Faculty of Electrical Engineering, Computer Science and Telecommunications,
University of Zielona Gora,
Podgorna 50, 65-246 Zielona Gora, Poland

a.bukowiec@iie.uz.zgora.pl, j.tkacz@iie.uz.zgora.pl, m.adamski@iie.uz.zgora.pl

Abstract. *The paper describes a new method for the synthesis of the application specific logic controllers, targeted into the FPGA. The initial steps of the proposed control algorithm rely on the notion of a Petri net, which is an easy way to describe parallel processes. The algorithm is oriented on transition based logic description. It allows easy analysis of dynamics and functioning of the circuit. The logic circuit is also decomposed into logic blocks responsible for particular functions. It leads to the compact implementation with usage of different kind of logic elements like. Additionally such decomposition allows easy analysis of circuit.*

Keywords

Application specific logic controller, FPGA, logic synthesis, Petri net, structural decomposition.

1. Introduction

A Petri net (PN) [10], [9] is one of the most popular models used in formal design and synthesis of the application specific logic controllers (ASLCs) [13], [15], [6]. The digital design of such controllers is very often implemented using field programmable gate arrays (FPGAs) [1], [15], [2], [16]. The most typical implementation of Petri nets in the FPGA devices uses the one-hot local state encoding method, where each place is represented by a flip-flop [11]. Such approaches are oriented towards places based logic description. Additionally, this approach requires hardware implementation of a large number of logic functions and flip-flops included in logic blocks.

In this paper we propose a new method for the synthesis of a Petri net. To allow its effective synthesis, the Petri net is initially converted into Petri macronet [9], [14]. The proposed algorithm is oriented towards transition based logic description. It means, that combinational equations describe transitions [5] in opposite to classical algorithms where they describe places [4]. It easy allows to analyze the dynamics of logic controller by exporting variables that describes transition in Boolean algebra. Additionally, the operations are encoded with a minimal-length binary vector. This encoding allows the realization of logic circuit in compact way. A microoperation decoder can be implemented with the use of embedded memory blocks of an FPGA [12]. It permits the stable work of whole controller.

2. Petri Net

A Petri net [10], [9] 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 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 sets of input and output transitions of a place $p_m \in P$ are defined respectively as follows:

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

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

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

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

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

For a given place p_m the function $M(p_m)$ returns the number of tokens in p_m. A place or a set of places is marked if it contains a token. A transition t_s can be fired if all its input places are marked. Firing a transition removes one token from each input place and puts one token in each output place. When the initial marking M_0 is additionally specified, the Petri net can be represented as a tuple:

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

2.1. Interpreted Petri Net

A Petri net enhanced with an additional feature for information exchange is called an interpreted Petri net [9]. 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 the set of input variables, $X = \{x_1, \ldots, x_L\}$,
- Y is the set of output variables (microoperations - μO), $Y = \{y_1, \ldots, y_N\}$,
- Z is the set of internal communication variables (usually not used, with $Z = \varnothing$).

An interpreted Petri net has a guard condition φ_s associated with every transition t_s. The guard condition φ_s is defined to be a Boolean function of a subset of variables from the sets X and Z. In a special case, the condition φ_s can be defined as 1 (always true). Now, a transition t_s can be fired if all its input places are marked and the current value of the corresponding Boolean function φ_s is equal to 1.

The conjunction ψ_m associated with a place p_m is an elementary conjunction of positive literals formed from output variables from 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. The conjunction ψ_m correspond to microinstruction (μI).

2.2. Macro Petri Net

Macro Petri net is a Petri net where part of the net (subnet) is replaced by one macroplace [9]. It allow to enhance Petri nets with hierarchy [7] 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 [10],
- Two-pole blocks [9],
- Parallel places [10],
- P-blocks [9].

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 [9]. This is macroplaces that have one input and one output and consist of only sequential places. Only Petri macronets with such macroplaces will be used in this article.

3. Idea of Synthesis Method

The idea of proposed synthesis method is based on the modular encoding of places together with functional parallel decomposition of the Petri net-based logic circuit [4]. The novelty of this approach is that places are encoded with use of minimal length code separately inside each macroplace and macroplaces are encoded with use of one-hot encoding. The state of Petri net is determined by concatenation of these codes. Combinational circuit is oriented towards transition generation, and output variables (names of particular microoperations) are placed in configured memories of FPGA. It leads to realization of a logic circuit in double-level architecture (Fig. 1), where the transition coder (TC) of first level is responsible for activation of the transitions:

$$T = \mathrm{TC}(X, Q), \tag{3}$$

The register block (REG) holds a current state of Petri net in the register (RG). It also has additional custom combinational logic (LOGIC) connected to its inputs. This logic is responsible for generation of the next state based on active transitions and current state:

$$Q^* = \mathrm{RG}(\mathrm{LOGIC}(T, Q)), \tag{4}$$

where Q is the set of variables used to store the codes of currently marked places and macroplaces. The internal custom combinational logic of the register also generates the code of microoperation:

$$Z = \mathrm{LOGIC}(T, Q), \tag{5}$$

where Z is the set of variables used to store the codes of currently executed microinstruction. The second level decoder (D) is responsible for generation of microoperations based on microinstruction code and it is implemented using memory blocks. Their functionality can be described by function:

$$Y = \mathrm{D}(Z). \tag{6}$$

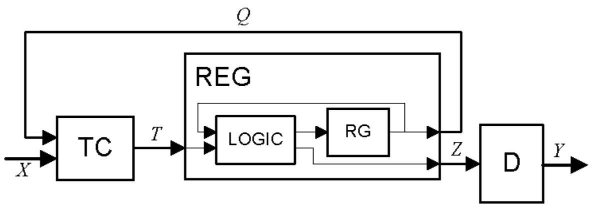


Fig. 1: Logic circuit of Petri net.

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

The entry point to the synthesis method is the interpreted Petri macronet. The outline of synthesis process includes following steps:

- Modular encoding of places. The purpose of this step is to assign the shortest binary local code $K_o(p_m)$ to each place p_m inside each macroplace mp_o, where $o = 1, \ldots, O$ and it is an number of macroplace. Macroplaces are encoded by assigning the one-hot code $K(mp_o)$ to each macroplace mp_o. The global code $C(p_m)$ of global place p_m is determined as concatenation of these codes:

  $$C(p_m) = K(mp_o) * K_o(p_m). \tag{7}$$

  The total required number of variables for encoding is equal to:

  $$R = O + \sum_{o=1}^{O} r_o, \tag{8}$$

  where r_o is a required number of variables for o-th macroplace:

  $$r_o = \lceil log_2(|P_o|) \rceil, \tag{9}$$

  where $P_o \subseteq P$ is a set of places that are placed inside macroplace mp_o.

  To store the macroplace code we use $Q^0 = \{q_1, \ldots, q_O\}$ variables and to store the local place codes we use $Q^o = \{q_{O+r+1}, \ldots, q_{O+r+r_o}\}$ $(r = \sum_{j=1}^{o-1} r_j)$. Now, these variables create set $Q = \bigcup_{o=0}^{O} Q^o$ and $|Q| = R$.

  The process of encoding begins from assigning the one-hot codes to macroplaces. Then, places receive minimal length codes inside each macroplace independently.

- Formation of microinstructions. Let all microoperations create U different microinstructions $Y_u \subseteq Y$, $\Upsilon = \{Y_1, \ldots, Y_U\}$ in the Petri net. Let create O subsets $\Upsilon_o \subseteq \Upsilon$, where Υ_o consists only of microinstructions associated with places from set P_o.

- Encoding of microinstructions. All microinstructions are encoded by binary code $C_o(Y_u)$ separately in each subset Υ_o. The number of variables used is

  $$\rho_o = \lceil log_2(|\Upsilon_o| + 1) \rceil. \tag{10}$$

  To store this code we use $Z^o = \{z_{\rho+1}, \ldots, z_{\rho+\rho_o}\}$ $(\rho = \sum_{j=1}^{o-1} \rho_j)$ variables and $Z = \bigcup_{i=1}^{O} Z^o$. The process of encoding is trivial, and it required to assign binary code $C_o(Y_u)$ to each microinstruction $Y_u \in \Upsilon_o$ starting from value 1. The value 0 is reserved for situation where considered place do not generate any microinstruction. Let assume that particular one microinstruction Y_u can belong to several subsets. In such situation it will receive several codes.

- Formation of conjunctions. Conjunctions describe macroplaces, places and global places. They are needed for easier form of Eq. (3) and Eq. (5) that describe digital circuit. The conjunction describing the macroplace mp_o equals to the affirmation of variable $q_o \in Q^0$. This variable is equal to 1 in the code $K(mp_o)$. The conjunction describing the place p_m consists of affirmation or negation of variables $q_r \in Q^o$ that are used to store the code $K_o(p_m)$ of this place. If variable q_r is equal to 1 in the code $K_o(p_m)$ then affirmation of this variable is used otherwise its negation is used. The conjunction describing the global place p_m consists of the macroplace mp_o and the place p_m conjunction. It corresponds to the code $C(p_m)$.

- Formation of logic equations. Logic equations describe Eq. (3) and Eq. (5) of combinational circuit TC and custom combinational logic LOGIC of register REG. The characteristic function of transition is defined as conjunction of conjunctions of all its input global places and guard condition:

  $$t_s = \bigwedge(\bullet t_s) \wedge \varphi_s. \tag{11}$$

  The function to generate the code of next place calculated by the custom combinational logic LOGIC is defined as:

  $$q_r = q_r \oplus \bigvee(\bullet q_r) \oplus \bigvee(q_r \bullet), \tag{12}$$

  and the function to generate code microoperation is defined as:

  $$z_\rho = \bigvee(P_{z_\rho}), \tag{13}$$

where P_{z_ρ} is a set of global places conjunctions that generate microoperations Y_u represented by the $C_o(Y_u)$ that has variable z_ρ set to 1.

- Formation of memory contents. The memory content can be described as tables or as equations according to the function Eq. (6). In case of tabular description there is required to create O tables. The table consists of two columns. First column is an address and it is described by variables $z_\rho \in Z^o$. The second column is a binary value (vector) of operations. It is based on value of output variables form the set $Y^o = \bigcup_{u=1}^{u=|\Upsilon_o|} Y_u | Y_u \in \Upsilon_o$. In each line of the table, there should be placed a binary value with only these bits y_n set that are in microinstruction Y_u represented by code $C_o(Y_u)$ that equals to the address from the first column of this line.

- Formation of logic circuit and implementation. This step describes the rules of design of the Petri net HDL model and its implementation into FPGA device. Here is applied a bottom-up approach. Conjunctions of places can be described using standard bit-wise operators. Then logic equations can be described with the use of these conjunctions using continuous assignments or procedural assignments as well as bit-wise operators. There should be created a module for circuit TC with inputs X and Q and outputs T. The register REG should be described as R-bits register with an asynchronous set. The typical synthesis template can be used [3]. the decoder D can be described as processes with the `case` statement. As, the embedded memory blocks are synchronous, the sensitivity list of such processes includes only clock signal. The 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 top-level module should describe connections of all components according to the block diagram presented in Fig. 1. Additionally the global reset and clock signals are connected to set and clock inputs of register and reset and clock inputs of decoder. The edge that trigs the decoder has to be opposite to the edge that trigs the register, and then operations are generated during only one clock cycle. The created model of logic circuit can be passed into third-party synthesis tool.

4. Example of Method Application

The method of Petri net synthesis, described in the previous section, is illustrated by its application on Petri net PN_1 (Fig. 2a). This Petri net describes control process of an industrial mixer of aggregate content and water [8]. This Petri net is not complicated and it is a good example to illustrated a synthesis steps. For the synthesis purpose it was compacted into Petri macronet (Fig. 2b).

Firstly, the places have to be encoded (step Modular encoding of places). There is $O = 6$ macroplaces, so it is required to use $r_0 = 6$ variables $Q^0 = \{q_1, \ldots, q_6\}$ to encode macroplaces. Macroplaces contains respectively 2, 1, 3, 1, 2, and 2 places, so it is required to use $r_1 = 1$, $r_2 = 1$, $r_3 = 2$, $r_4 = 1$, $r_5 = 1$, and $r_6 = 1$ variables $Q^1 = \{q_7\}$, $Q^2 = \{q_8\}$, $Q^3 = \{q_9, q_{10}\}$, $Q^4 = \{q_{11}\}$, $Q^5 = \{q_{12}\}$, and $Q^6 = \{q_{13}\}$ to encode places inside each macroplace. In total, it is required to use $R = 13$ variables $Q = \{q_1, \ldots, q_{13}\}$ to encode all places. Macroplaces receive following one-hot codes $K(mp_o)$ using variables from Q^0 subset:

$$K(mp_1) = 1-----;\ K(mp_2) = -1----;$$
$$K(mp_3) = --1---;\ K(mp_4) = ---1--;$$
$$K(mp_5) = ----1-;\ K(mp_6) = -----1;$$

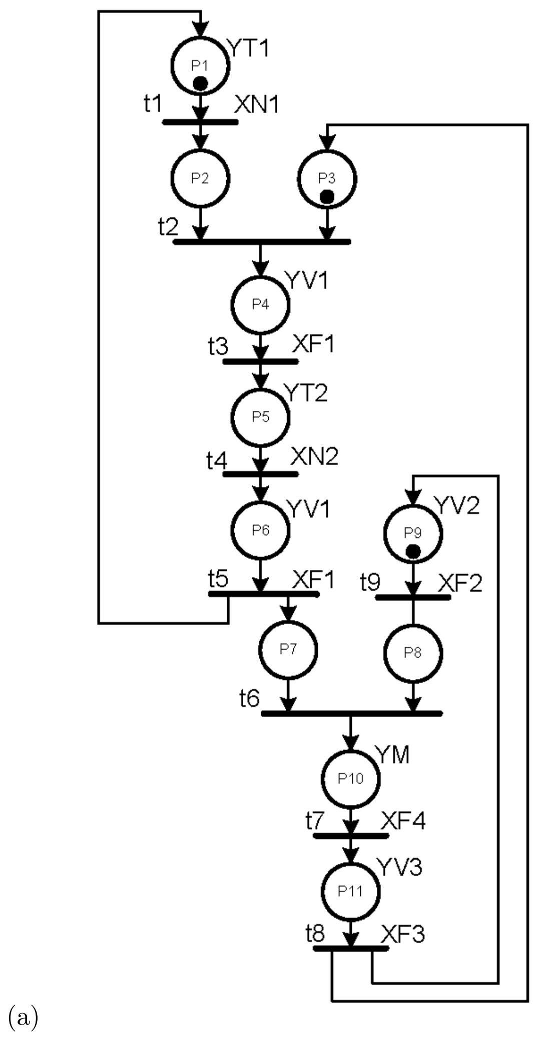
and places receive following binary codes $K_o(p_m)$ inside each macroplace using variables from corresponding Q^o subset:

$$K_1(p_1) = 0;\ K_1(p_2) = 1;\ K_2(p_3) = 1;$$
$$K_3(p_4) = 00;\ K_3(p_5) = 01;\ K_3(p_6) = 10;$$
$$K_4(p_7) = 1;\ K_5(p_8) = 1;\ K_5(p_9) = 0;$$
$$K_6(p_{10}) = 0;\ K_6(p_{11}) = 1;$$

As an alternative, the Gray code can be applied also for places.

When encoding of places is finished the microinstructions can be formed (step Formation of microinstructions) and encoded (step Encoding of microinstructions). The set of microinstructions Υ is formed based on control algorithm and for the example Petri net PN_1 there is $U = 6$ different microinstructions: $Y_1 = \{YT1\}$, $Y_2 = \{YV1\}$, $Y_3 = \{YT2\}$, $Y_4 = \{YV2\}$, $Y_5 = \{YM\}$, $Y_6 = \{YV3\}$, and $\Upsilon = \{Y_1, Y_2, Y_3, Y_4, Y_5, Y_6\}$. This set is divided into $O = 6$ subsets: $\Upsilon_1 = \{Y_1\}$, $\Upsilon_2 = \{\varnothing\}$, $\Upsilon_3 = \{Y_2, Y_3\}$, $\Upsilon_4 = \{\varnothing\}$, $\Upsilon_5 = \{Y_4\}$, and $\Upsilon_6 = \{Y_5, Y_6\}$. Empty sets are omitted in the encoding process and now, microinstructions can be encoded with $\rho_1 = 1$, $\rho_3 = 2$, $\rho_5 = 1$, and $\rho_6 = 2$ variables $Z^1 = \{z_1\}$, $Z^3 = \{z_2, z_3\}$, $Z^5 = \{z_4\}$, and $Z^6 = \{z_5, z_6\}$. The sample encoding can be as follows:

$$C_1(Y_1) = 1;\ C_3(Y_2) = 01;\ C_3(Y_3) = 10;$$
$$C_5(Y_4) = 1;\ C_6(Y_5) = 01;\ C_6(Y_6) = 10;$$



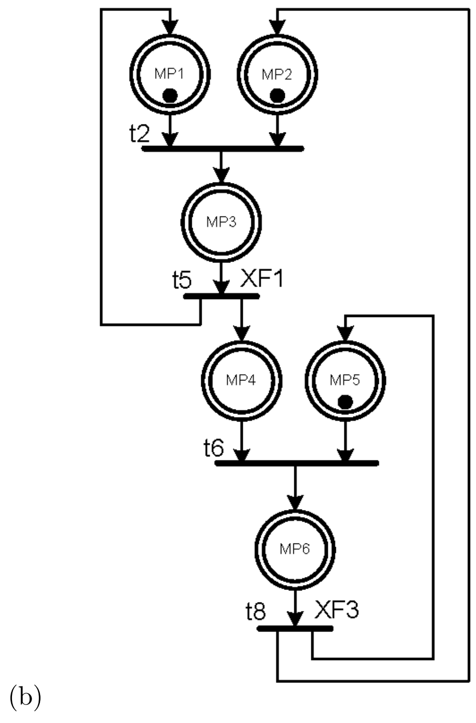


Fig. 2: (a) Petri net PN_1 and (b) its Petri macronet.

When all encodings are finished place conjunctions (step Formation of conjunctions) and logic equations (step Formation of logig equations) can be formed. Place conjunctions are created on the base of the place codes. First place conjunctions of macroplaces have to be denoted and for the Petri net PN_1 that is as follows:

$$mp_1 = q_1;\ mp_2 = q_2;\ mp_3 = q_3;$$
$$mp_4 = q_4;\ mp_5 = q_5;\ mp_6 = q_6;$$

Then, conjunctions of local place inside each macroplace can be formed:

$$lp_1 = \overline{q_7};\ lp_2 = q_7;\ lp_3 = q_8;$$
$$lp_4 = \overline{q_9}\,\overline{q_{10}};\ lp_5 = \overline{q_9}\,q_{10};\ lp_6 = q_9\,\overline{q_{10}};$$
$$lp_7 = q_{11};\ lp_8 = q_{12};\ lp_9 = \overline{q_{12}};$$
$$lp_{10} = \overline{q_{13}};\ lp_{11} = q_{13};$$

Finally, conjunctions of global places can be created based on Eq. (7):

$$p_1 = mp_1\,lp_1;\ p_2 = mp_1\,lp_2;\ p_3 = mp_2\,lp_3;$$
$$p_4 = mp_3\,lp_4;\ p_5 = mp_3\,lp_5;\ p_6 = mp_3\,lp_6;$$
$$p_7 = mp_4\,lp_7;\ p_8 = mp_5\,lp_8;\ p_9 = mp_5\,lp_9;$$
$$p_{10} = mp_6\,p_{10};\ p_{11} = mp_6\,p_{11};$$

After that, logic equations for each transition t_s can be formed:

$$t_1 = p_1\,xn_1;\ t_2 = p_2\,p_3;\ t_3 = p_4\,xf_1;$$
$$t_4 = p_5\,xn_2;\ t_5 = p_6\,xf_1;\ t_6 = p_7\,p_8;$$
$$t_7 = p_{10}\,xf_4;\ t_8 = p_{11}\,xf_3;\ t_9 = p_9\,xf_2$$

and logic equation for each variable q_r can be denoted:

$$q_1 = q_1 \oplus t_5 \oplus t_2;\ q_2 = q_2 \oplus t_8 \oplus t_2;$$
$$q_3 = q_3 \oplus t_2 \oplus t_5;\ q_4 = q_4 \oplus t_5 \oplus t_6;$$
$$q_5 = q_5 \oplus t_8 \oplus t_6;\ q_6 = q_6 \oplus t_6 \oplus t_8;$$
$$q_7 = q_7 \oplus t_1 \oplus t_2;\ q_8 = q_8 \oplus t_8 \oplus t_2;$$
$$q_9 = q_9 \oplus t_4 \oplus t_5;\ q_{10} = q_{10} \oplus t_3 \oplus t_4;$$
$$q_{11} = q_{11} \oplus t_5 \oplus t_6;\ q_{12} = q_{12} \oplus t_9 \oplus t_6;$$
$$q_{13} = q_{13} \oplus t_7 \oplus t_8;$$

and logic equation for each variable z_ρ can be created:

$$z_1 = p_1;\ z_2 = p_5;\ z_3 = p_4 \vee p_6$$
$$z_4 = p_9;\ z_5 = p_{11};\ z_6 = p_{10}$$

Then, the content of operation memory can be formed (step Formation of memory contents). In case of Petri net PN_1 there have to be created four such tables which are shown in Tab. 1. Two macroplaces do not generate any microinstructions so they are omitted.

Finally, the logic circuit can be described (step Formation of logic circuit and implementation) In our approach the VHDL was used. But in similar way it can be also described with the use of Verilog. The module for circuit TC (Fig. 3) uses input variables and

```
entity TC is
  port(XN1, XF1, XN2, XF2, XF3, XF4 : in
       STD_LOGIC;
         Q : in STD_LOGIC_VECTOR(1 to 13);
         T : out STD_LOGIC_VECTOR(1 to 9));
end TC;
architecture TC of TC is
  signal mp : STD_LOGIC_VECTOR(1 to 6);
  signal lp : STD_LOGIC_VECTOR(1 to 11);
  signal p : STD_LOGIC_VECTOR(1 to 11);
begin
  mp(1) <= Q(1);
  mp(2) <= Q(2);
  ...
  mp(6) <= Q(6);
  lp(1) <= not Q(7);
  lp(2) <= Q(7);
  ...
  lp(11) <= Q(13);
  p(1) <= mp(1) and lp(1);
  p(2) <= mp(1) and lp(2);
  ...
  p(11) <= mp(6) and lp(11);
  T(1) <= p(1) and XN1;
  T(2) <= p(2) and p(3);
  ...
  T(9) <= p(9) and XF2;
end TC;
```

Fig. 3: VHDL description of transition coder.

```
entity REG is
  port(clk, res : in STD_LOGIC;
         T : in STD_LOGIC_VECTOR(1 to 9);
         Q : out STD_LOGIC_VECTOR(1 to 13);
         Z : out STD_LOGIC_VECTOR(1 to 6));
end REG;
architecture REG of regREGis
  signal intQ : STD_LOGIC_VECTOR(1 to 13);
  signal mp : STD_LOGIC_VECTOR(1 to 6);
  signal lp : STD_LOGIC_VECTOR(1 to 11);
  signal p : STD_LOGIC_VECTOR(1 to 11);
begin
  RG: process (clk, res) begin
    if res='1' then
      intQ <= "1100100100000";
    elsif (RISING_EDGE(clk)) then
      intQ(1) <= intQ(1) xor T(5) xor T(2);
      ...
      intQ(12) <= intQ(12) xor T(9) xor T(6);
      intQ(13) <= intQ(13) xor T(7) xor T(8);
    end if;
  end process;
  Q <= intQ;
  mp(1) <= ...
  lp(1) <= ...
  p(1) <= ...
  Z(1) <= p(1);
  ...
  Z(6) <= p(10);
end reg;
```

Fig. 4: VHDL description of register.

conjunctions in continuous assignments for transition. Conjunctions are defined as internal signals and they are described as continuous assignments.

```
entity D is
  port(clk, res : in STD_LOGIC;
       Z : in STD_LOGIC_VECTOR(1 to 6);
       YT1, YV1, YT2, YV2, YM, TV3 : out
           STD_LOGIC);
  attribute bram_map: string;
  attribute bram_map of D: entity is "yes";
end D;
architecture D of D is
begin
  O1: process (clk) begin
    if FALLING_EDGE(clk) then
      if res='1' then
        YT1 <= '0';
      else case Z(1) is
        when '0' => YT1 <= '0';
        when '1' => YT1 <= '1';
        when others => YT1 <= '0';
      end case;
      end if;
    end if;
  end process;
  O3: process (clk) begin
    if FALLING_EDGE(clk) then
      if res='1' then
        YV1 <= '0'; YT2 <= '0';
          else case Z(2 to 3) is
        when "00" => YV1 <= '0'; YT2 <= '0';
        when "01" => YV1 <= '1'; YT2 <= '0';
        when "10" => YV1 <= '0'; YT2 <= '1';
        when others => YV1 <= '0'; YT2 <= '0';
      end case;
      end if;
    end if;
  end process;
  O5: process ...
  O6: process ...
end D;
```

Fig. 5: VHDL description of operations decoder.

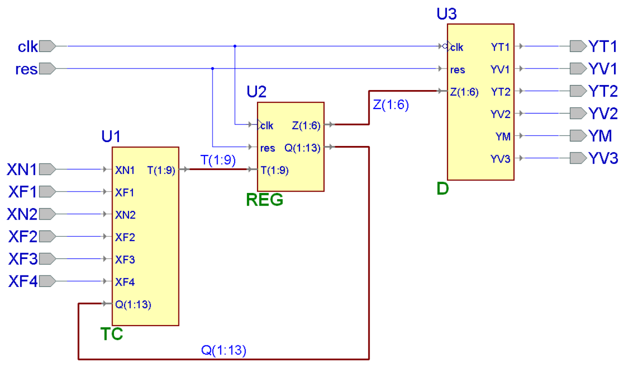


Fig. 6: Block diagram of top-level module.

The module of register REG (Fig. 4) describes the logic of code changes and generate code of microinstruction. It also requires definition of conjunctions as internal signals.

The decoder D (Fig. 5) generates outputs signals.

It can be synthesized as embedded memory block if there is added special synthesis directive. In this file, there is such directive for Xilnix devices that

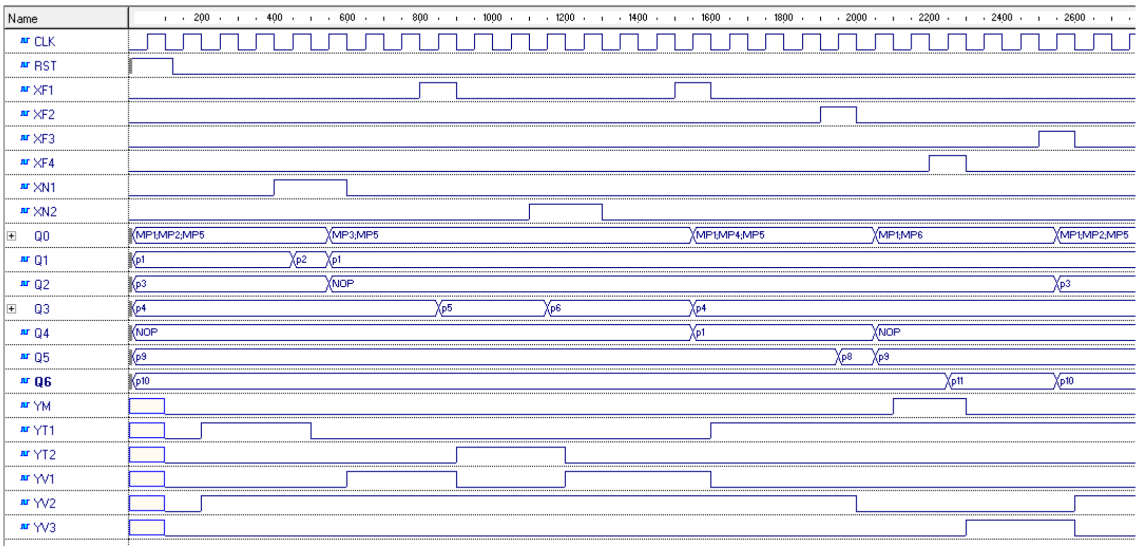


Fig. 7: Simulation results of logic circuit.

sets bram_map attribute to yes. The top-level module (Fig. 6) describes connections of all modules. In our case it is cerated in graphical editor of Active-HDL environment.

The designed circuit is verified in Active-HDL environment with use of a test-bench. The test-bench is described in VHDL and it emulate one cycle of work of an industrial mixer. The simulation results are sown in the Fig. 7.

Tab. 1: Operation memories tables of PN$_1$.

Addr. z_1	μO $YT1$	**Addr.** $z_2\ z_3$	μO $YV1\ YT2$
0	0	00	00
1	1	01	10
		10	01
Addr. z_4	μO $YV2$	**Addr.** $z_5\ z_6$	μO $YM\ YV3$
0	0	00	00
1	1	01	10
		10	01

5. Summary

The paper presents a method of realization of application specific logic controller. A formal description of the method is then accompanied with a simple example. The specification of the control algorithm uses the notion of a Petri net, which allows an easy description of parallel processes. We note that it is possible to apply formal verification methods to test the algorithm. The proposed method of synthesis is based on transition based logic description of the logic circuit and modular encoding of places. It allows to extend formal verification methods by additional analysis the dynamics of the circuit. Additionally the logic circuit is decomposed into three logic blocks responsible for particular functions: dynamic generation of transitions, store the state of the controller and generate output control signals. It allows the compact implementation of logic circuit into FPGA device with usage of different kind of logic elements like: LUTs, flip-flops and embedded memories. Additionally such decomposition allows easy analysis of circuit functioning.

References

[1] BOMAR, B. W. Implementation of microprogrammed control in FPGAs. *IEEE Transactions on Industrial Electronics*. 2002, vol. 49, iss. 2, pp. 415–422. ISSN 0278-0046. DOI: 10.1109/41.993275.

[2] BOROWIK, G., M. RAWSKI, G. LABIAK, A. BUKOWIEC and H. SELVARAJ. Efficient logic controller design. In: *2010 Fifth International Conference on Broadband and Biomedical Communications*. Malaga: IEEE, 2010, pp. 1–6. ISBN 978-1-4244-6952-9. DOI: 10.1109/IB2COM.2010.5723633.

[3] BROWN, S. and Z. VERNESIC. *Fundamentals of digital logic with VHDL design*. New York: McGraw-Hill Higher Education. 2005. ISBN 978-0077221430.

[4] BUKOWIEC, A. and M. ADAMSKI. Synthesis of Petri nets into FPGA with operation flexible memories. In: *2012 IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits*. Tallin: IEEE, 2012, pp. 16–21. ISBN 978-1-4673-1186-1. DOI: 10.1109/DDECS.2012.6219016.

[5] BUKOWIEC, A. and M. ADAMSKI. Transition based synthesis with code markers of Petri nets into GPGAs. In: *12th IFAC Conference on Programmable Devices and Embedded Systems PDeS 2013.* Velke Karlovice: IFAC, 2013. pp. 181–86. ISBN 978-3-902823-53-3. DOI: 10.3182/20130925-3-CZ-3023.00030.

[6] DOLIGALSKI, M. Behavioral specification diversification for logic controllers implemented in FPGA devices. In: *Proceedings of the Annual FPGA Conference on - FPGAworld '12.* New York: ACM Press, 2012, pp. 1–5. ISBN 978-1-4503-1645-3. DOI: 10.1145/2451636.2451642.

[7] ESPARZA, J. and M. SILVA. On the analysis and synthesis of free choice systems. *Advances in Petri Nets 1990.* Berlin: Springer, 1991, pp. 243–286. ISBN 978-3-540-53863-9. DOI: 10.1007/3-540-53863-1_28.

[8] GNIEWEK, L. and J. KLUSKA. Hardware Implementation of Fuzzy Petri Net as a Controller. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics).* 2004, vol. 34, iss. 3, pp. 1315–1324. ISSN 1083-4419. DOI: 10.1109/TSMCB.2003.822956.

[9] KARATKEVICH, A. *Dynamic Analysis of Petri Net-Based Discrete Systems.* Berlin: Springer, 2007. ISBN 978-3-540-71464-4.

[10] MURATA, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE.* 1989, vol. 77, iss. 4, pp. 541–580. ISSN 0018-9219. DOI: 10.1109/5.24143.

[11] PASTOR, E. and J. CORTADELLA. Efficient encoding schemes for symbolic analysis of Petri nets: Properties, analysis and applications. In: *Proceedings Design, Automation and Test in Europe.* Paris: IEEE, 1998, pp. 790–795. ISBN 0-8186-8359-7. DOI: 10.1109/DATE.1998.655948.

[12] RAWSKI, M., G. BOROWIK, T. LUBA, P. TOMASZEWSKI and B. FALKOWSKI. Logic synthesis strategy for FPGAs with embedded memory blocks. In: *Mixed Design of Integrated Circuits & Systems, 2009. MIXDES '09. MIXDES-16th International Conference.* Lodz: IEEE, 2009, pp. 296–301. ISBN 978-1-4244-4798-5.

[13] ROKYTA, P., W. FENGLER and T. HUMMEL. Electronic System Design Automation Using High Level Petri Nets. In: *Hardware Design and Petri Nets.* Boston: Springer, 2000, pp. 193–204. ISBN 978-1-4757-3143-9. DOI: 10.1007/978-1-4757-3143-9_10.

[14] TKACZ, J. and M. ADAMSKI. Macrostate encoding of reconfigurable digital controllers from topological Petri net structure. *Przeglad Elektroniczny.* 2012, vol. 8, no. 8, pp. 137–140. ISSN 0033-2097.

[15] WEGRZYN, M. Implementation of safety critical logic controller by means of FPGA. *Annual Reviews in Control.* 2003, vol. 27, iss. 1, pp. 55–61. ISSN 1367-5788. DOI: 10.1016/S1367-5788(03)00007-5.

[16] WISNIEWSKI, R., A. BARKALOV, L. TITARENKO and W. HALANG. Design of microprogrammed controllers to be implemented in FPGAs. *International Journal of Applied Mathematics and Computer Science.* 2011, vol. 21, iss. 2, pp.401-412. ISSN 1641-876X. DOI: 10.2478/v10006-011-0030-1.

About Authors

Arkadiusz BUKOWIEC received a Bachelor degree in computer engineering from Technical University of Zielona Gora. During these studies he completed industrial practice at Aldec Inc. in Henderson, NV, USA. Then, he received Master degree and a Ph.D. degree in computer science from the University of Zielona Gora. During the master thesis he was working for Aldec Poland. During the Ph.D. studies he spent one semester at Universidade Nova de Lisboa. Since 2003, he has been working at the University of Zielona Gora. His research interests include methods of design, synthesis and verification of digital circuits.

Jacek TKACZ graduated from the University of Zielona Gora and since 2009 works in the Chair of Computer Engineering. Dr. Tkacz's research is devoted to symbolic methods of theorem proving and their application to computer science and electronics. He is also interested in novel design and development technologies for application software, including mobile applications. During the years 1997-2005 he was involved in design and development of the PROLIB software, used by many Polish libraries.

Marian ADAMSKI is a retired head of the Institute of Computer Science and Electronics at the University of Zielona Gora. His research interests include the design of digital systems, understood as digital microsystems, and formal methods in programming of logical controllers. A member of IEEE, IEE, ACM, PTEiTS (Polish Society for Theoretical and Applied Electrical Engineering) and PTI (Polish Computer Science Society).