www.czasopisma.pan.pl PAN www.journals.pan.pl
INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2012, VOL. 58, NO. 1, PP. 35–41
Manuscript received December 31, 2011; revised March 2012. DOI: 10.2478/v10177-012-0005-7

Synthesis of FSMs Based on Architectural Decomposition with Joined Multiple Encoding

Arkadiusz Bukowiec

***Abstract*—The method of synthesis of the logic circuit of finite state machine (FSM) with Mealy's outputs is proposed in this paper. Proposed method is based on the innovate encoding of microinstructions split into subsets. Code of microinstruction is represented as a part of current state code and code of microinstruction inside of current subset. It leads to realization of FSM as s double-level structure. It leads to diminishing of number of variables required for encoding of microinstructions. Such approach permits to decrease the number of required outputs of combinational part of FSM.**

***Keywords*—Boolean algebra, circuit synthesis, Field Programmable Gate Arrays, sequential circuits.**

I. Introduction

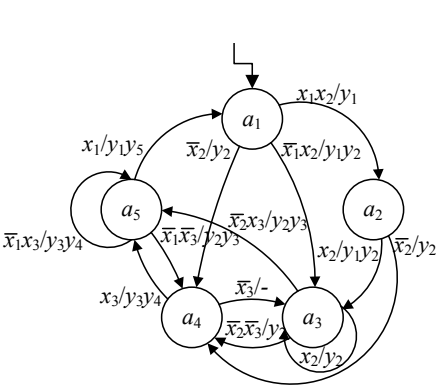
FINITE state machines (FSMs) with Mealy's outputs [1], [2] are one of the most popular method of control units (CUs) design. Nowadays, field programmable gate arrays (FPGAs) are used very often for implementation of logic circuits of FSMs [3], [4]. One of the main features of FPGA is existence of logic elements with restricted number of inputs [5]. On the other hand, logic functions of FSMs have much more arguments than number of inputs of typical logic element. This imbalance leads to necessity of decomposition of logic functions describing the behavior of FSM [6], [7], [8]. The negative results of functional decomposition are both increasing a number of levels of the FSM circuit and decreasing of a digital system performance in comparison to single-level implementation of control unit.

One of methods of decreasing a number of logic functions depending on big number of arguments is multi-level implementation of FSM [9], [10]. Such methods required additional internal variables and very often consume more hardware then single-level implementation of FSM. But, this issue can be resolved by usage of both, logic elements and embedded memory blocks, that are available in modern FPGA devices.

The method of decreasing of a number of functions depending of logic conditions and internal variables of FSM is proposed in given article. There is proposed method of joined multiple encoding of microinstructions. A set of microinstruction is divided into subsets based on a current state [11]. Then, subset are joined into pairs [12]. Each pair is identified based on a part of state code [13]. Next, microinstruction are encoded separately in each pair of subsets. The microinstruction encoding leads to decrease the number of implemented logic functions by combinational part of the logic circuit. And the joining of subsets leads to decrease size of memory decoder. In overall it leads to balanced usage of different kind of logic resources.

This work was supported by the Ministry of Science and Higher Education of Poland. Research grant no. N516 513939 for years 2010-2013.

A. Bukowiec is with the Institute of Computer Engineering and Electronics, University of Zielona Góra, Licealna 9, 65-417 Zielona Góra, Poland (e-mail: a.bukowiec@iie.uz.zgora.pl).



(a) State diagram

```
.i 3
.o 5
.s 5
.p 13
.r a1
11- a1 a2 10000
01- a1 a3 11000
-0- a1 a4 01000
-1- a2 a3 11000
-0- a2 a4 01000
-1- a3 a3 01000
-00 a3 a4 01000
-01 a3 a5 01100
--1 a4 a5 00110
--0 a4 a3 00000
1-- a5 a1 10001
0-1 a5 a5 00110
0-0 a5 a4 01100
```

(b) KISS2

Fig. 1. Example of FSM S_1.

II. Rudiments

A finite state machine is a mathematical model of behavior composed of a finite set of input symbols, a finite set of states, a finite set of output symbols, transitions and actions. It is represented as 6-tuple:

$$S = \langle X, Y, A, a_1, \delta, \omega \rangle, \tag{1}$$

where:

- X is a finite set of input symbols, $X = \{x_1, \ldots, x_L\}$;
- Y is a finite set of output symbols, $Y = \{y_1, \ldots, y_N\}$;
- A is a finite non empty set of states, $A = \{a_1, \ldots, a_M\}$;
- a_1 is the initial state, $a_1 \in A$;
- δ is a transition function, defined as a function of a state and input symbols:

$$\delta : A \times X \to A; \tag{2}$$

- ω is an output function, in case of Moore model [14] defined as a function of a state:

$$\omega : A \to Y, \tag{3}$$

and in case of Mealy model [15] defined as a function of a state and input symbols:

$$\omega : A \times X \to Y. \tag{4}$$

One of the most popular methods of representation of FSMs is KISS2 text format [16]. It is a text file (Fig. 1b)

representation of one-dimensional state transition table [1], [2]. A file in this format consists of two parts: header and table. The header includes information about the number of inputs `.i`, the number of outputs `.o`, the number of table products `.p`, the number of states `.s`, and the initial state (optional) `.r`. The table describes the behavior (transitions) of a FSM. It consists of four columns: a logic condition, a current state, a next state, and output variables. The '–' sign in logic condition means that this input variable does not affect this transition. The '0' value means that negation of this variable should be placed in a logic condition and the '1' value that its affirmation should be placed in a logic condition.

III. Synthesis Methods

Logic circuit of Mealy FSM can be described as system of logic functions:

$$\begin{aligned} \Phi &= \Phi(X, Q), \\ Y &= Y(X, Q), \end{aligned} \tag{5}$$

where $Q = \{Q_1, \ldots, Q_R\}$ is a set of internal variables, that are used to encode states of FSM $a_m \in A$, and $R = \log_2 M$; $\Phi = \{D_1, \ldots, D_R\}$ is a set of excitation functions. This system is formed based on transition table (e.g. described in KISS2 file) during logic synthesis process. It is also the basis to form single-level logic circuit of FSM, called P (Fig. 2a) [1], [2]. Here the combinational circuit P implements system of excitation functions and microoperations (5) and it is implemented with use of logic elements in FPGAs. The register RG implements the memory of FSM and it has D type inputs as a rule.

In this structure, the total number of logic functions implemented by combinational circuit P is equal to:

$$n_{\mathrm{P}}(\mathrm{P}) = R + N. \tag{6}$$

One of the known methods of reduction of this value is application of the maximal encoding of microinstructions [2]. Let transition table has T different microinstructions $Y_t \subseteq Y$. Let encode each microinstruction Y_t by binary code $K(Y_t)$ with $N_1 = \lceil \log_2 T \rceil$ bits, where $N_1 < N$. Let use variables $z_n \in Z = \{z_1, \ldots, z_{N_1}\}$ for representation of these codes. In this case, the logic circuit of FSM can be implemented with a double-level structure PY (Fig. 2b) [1], [2]. Here, the combinational circuit P implements system of excitation functions and microinstructions encoding:

$$\begin{aligned} \Phi &= \Phi(X, Q), \\ Z &= Z(X, Q), \end{aligned} \tag{7}$$

but the number of implemented logic functions is reduced to:

$$n_{\mathrm{PY}}(\mathrm{P}) = R + N_1. \tag{8}$$

The register RG has exactly the same function like in a previous structure. The additional circuit Y implements system of microinstruction decoder:

$$Y = Y(Z), \tag{9}$$

and, because of its regular structure, it can be implemented with use of embedded blocks in FPGAs. However, the value of (8) is still relatively big in comparison to (6) and it does not assure reduction of the number of required logic elements to implementation of combinational circuit P [17]. It makes that application of this structure in FPGAs not effective.

Further reduction of the number of implemented logic functions can be satisfied by application of multiple encoding of microinstructions [10], [13]. Let divide set of microinstructions $\Upsilon = \{Y_1, \ldots, Y_T\}$ into subsets based on current state a_m. It leads to existence of M subsets $\Upsilon(a_m) \subseteq \Upsilon$, $\boldsymbol{\Upsilon} = \{\Upsilon(a_1), \ldots \Upsilon(a_M)\}$ and microinstruction $Y_t \in \Upsilon(a_m)$ iff it is executed during any transition from state a_m. Let encode each microinstruction $Y_t \in \Upsilon(a_m)$ by binary code $K_m(Y_t)$ with $N_0 = \lceil \log_2 T_0 \rceil$ bits where:

$$T_0 = \max(|\Upsilon(a_1)|, \ldots, |\Upsilon(a_M)|). \tag{10}$$

Let use variables $\psi_n \in \Psi = \{\psi_1, \ldots, \psi_{N_0}\}$ for representation of these codes. In this case code of microinstruction $K(Y_t)$ is represented by concatenation of multiple code of microinstruction $K_m(Y_t)$ and code of current state $K(a_m)$:

$$K(Y_t) = K_m(Y_t) * K(a_m). \tag{11}$$

Digital circuit of FSM with such encoding can be implemented with a double-level structure PY_0 (Fig. 2c) [17], [11]. Now, the combinational circuit P implements system of excitation functions and microinstructions multiple encoding:

$$\begin{aligned} \Phi &= \Phi(X, Q), \\ \Psi &= \Psi(X, Q), \end{aligned} \tag{12}$$

and the number of implemented logic functions is equal to:

$$n_{\mathrm{PY}_0}(\mathrm{P}) = R + N_0. \tag{13}$$

The register RG has exactly the same function like in previous structures. There is also additional circuit Y. It implements system of microinstruction multiple decoder:

$$Y = Y(\Psi, Q), \tag{14}$$

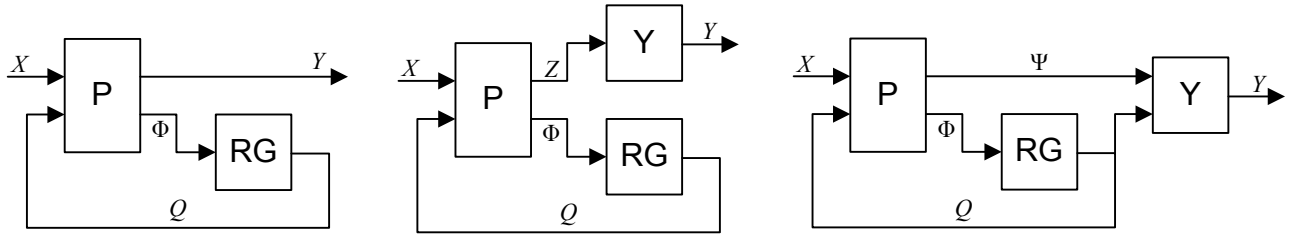
and it also can be implemented with use of embedded blocks in FPGAs. In this case, the value of (13) gives possibility to reduce the number of required logic elements for implementation of combinational circuit P [17]. Unfortunately, implementation of the microinstruction multiple decoder Y, represented by system (14), can lead to not effective usage of embedded memory block of FPGAs.

IV. Synthesis Methods with Joined Multiple Encoding

The idea of presented in this article method of synthesis is based on joining of microinstructions subsets into pair. It leads to possibility of identification of microinstruction only with use of a part of a current state code. This solution causes that the microinstruction decoder memory size is decreased twice and there is no need to implement any additional logic functions by combinational circuit.

Let join subsets $\Upsilon(a_m)$ into pairs $\Upsilon_{m'}^{m''} = \Upsilon(a_{m'}) \cup \Upsilon(a_{m''})$[1] and all such pair create a set $\boldsymbol{\Upsilon}^{\mathbf{P}}$. The number of elements of all pairs $\Upsilon_{m'}^{m''}$ should be equalized. Iff $M < 2^{\lceil \log_2 M \rceil}$

[1]Each pair is represented as a sum of two subsets.



(a) P (b) PY (c) PY_0

Fig. 2. Structures of logic circuit of FSM.

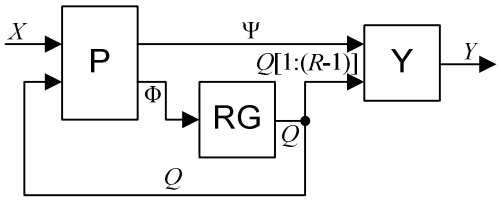


Fig. 3. Structure of logic circuit of PY_J FSM.

then $2^{\lceil \log_2 M \rceil} - M$ the biggest subsets $\Upsilon(a_m)$ should be joined into pair with empty set $\varnothing$ ($\Upsilon^{\varnothing}_{m'} = \Upsilon(a_{m'}) \cup \varnothing$)). The remain subsets are joined into pair by the rule: *The biggest with the smallest*. Next, let encode states $a_{m'}$ and $a_{m''}$ of each pair by binary codes $K(a_{m'})$ and $L(a_{m''})$ with R bits. These codes should differ only on the least significant bit Q_R. Now, let encode microinstruction for each pair of subsets by binary code $K^{m''}_{m'}(Y_t)$ on $N_C = \lceil \log_2 T_C \rceil$ bits, where

$$T_C = \max_{m'=1,m''=1}^{M,M}(|\Upsilon(^{m''}_{m'})|). \tag{15}$$

Application of this is effective only if condition:

$$N_C = N_0 \tag{16}$$

is satisfied. Otherwise, the number of logic function implemented by combinational circuit is increased and length of multiple code of microinstruction is also increased. It causes also that memory size is not reduced. But, for typical control algorithms condition (16) should be satisfied.

Let us use variables $\psi_n \in \Psi = \{\psi_1, \ldots, \psi_{N_C}\}$ for representation of the code $K^{m''}_{m'}(Y_t)$. When such encoding is applied, there is required to use concatenation of multiple code of microinstruction $K^{m''}_{m'}(Y_t)$ and part of the code of current state $K(a_m)[1:(R-1)]$:

$$K(Y_t) = K^{m''}_{m'}(Y_t) * K(a_m)[1:(R-1)]. \tag{17}$$

for one to one representation of the code of microinstruction $K(Y_t)$.

For this encoding, the logic circuit of FSM can be implemented with a double-level structure PY_J (Fig. 3). Here, the combinational circuit P implements system of excitation functions and microinstructions joined multiple encoding:

$$\begin{aligned} \Phi &= \Phi(X, Q), \\ \Psi &= \Psi(X, Q), \end{aligned} \tag{18}$$

```
1: T_0 = 0
2: Υ = ∅
3: for m = 1 to M do
4:     Υ(States[m]) = ∅
5:     for h = 1 to H do
6:         if AM(h)==States[m] then
7:             Υ(a_m) →ADD(YH(h))
8:         end if
9:     end for
10:    Υ →ADD(Υ(States[m]))
11:    if (Υ(States[m]) →COUNT> T_0) then
12:        T_0 = Υ(States[m]) →COUNT
13:    end if
14: end for
```

Fig. 4. Algorithm of creation and division of microinstructions.

and the number of implemented logic functions is equal to:

$$n_{PY_J}(P) = R + N_C. \tag{19}$$

Because of (16) this value is unchanged in comparison to previous method. And it still can be implemented using logic blocks of FPGA without big impact on the logic blocks number. The register RG has exactly the same function like in previous structures. There is also circuit Y. It implements system of microinstruction joined multiple decoder:

$$Y = Y(\Psi, Q[1:R-1]), \tag{20}$$

and it also can be implemented with use of embedded blocks in FPGAs. Because the address word is shorter by one bit in comparison to PY_0 structure the size of memory is decreased twice.

The whole synthesis process includes following steps:

1. *Creation and division of microinstructions set.* Let us create set of microinstructions $\Upsilon = \{Y_1, \ldots, Y_T\}$ by readout all unique microinstructions Y_t from transition table. Then, this set is divided into M subsets based on current state A_m. Each subset $\Upsilon(a_m) \subseteq \Upsilon$ consists only of microinstructions that are executed during any transition from state a_m. Implemented algorithm (Fig. 4) is optimized and creates divided subsets directly from transition table.
2. *Joining of microinstruction subsets into pairs.* Let us join all subsets $\Upsilon(a_m)$ into pair $\Upsilon^{m''}_{m'}$ by applying rule described above and i9mplemented in algorithm shown in Figure 5.

1: $\boldsymbol{\Upsilon} \rightarrow$SORT
2: $\boldsymbol{\Upsilon}^{\mathbf{P}} = \varnothing$
3: $M_P = 2^{\lceil \log_2 M \rceil} - M$
4: **if** $M_P > 0$ **then**
5: **for** $m = 1$ to M_P **do**
6: $\Upsilon^P(m) = \varnothing$
7: $\Upsilon^P(m) \rightarrow a_{m'} = \boldsymbol{\Upsilon}[m] \rightarrow a_m$
8: $\Upsilon^P(m) \rightarrow$ADDELEM($\boldsymbol{\Upsilon}[m]$)
9: $\Upsilon^P(m) \rightarrow a_{m''} = \boldsymbol{\Upsilon}[m] \rightarrow \varnothing$
10: $\boldsymbol{\Upsilon}^{\mathbf{P}} \rightarrow$ADD($\Upsilon^P(m)$)
11: **end for**
12: **end if**
13: **for** $m = M_P + 1$ to $\frac{M - M_P}{2}$ **do**
14: $\Upsilon^P(m) = \varnothing$
15: $\Upsilon^P(m) \rightarrow a_{m'} = \boldsymbol{\Upsilon}[m] \rightarrow a_m$
16: $\Upsilon^P(m) \rightarrow$ADDELEM($\boldsymbol{\Upsilon}[m]$)
17: $\Upsilon^P(m) \rightarrow a_{m''} = \boldsymbol{\Upsilon}[m] \rightarrow a_m$
18: $\Upsilon^P(m) \rightarrow$ADDELEM($\boldsymbol{\Upsilon}[M - (m - (M_P + 1))]$)
19: $\boldsymbol{\Upsilon}^{\mathbf{P}} \rightarrow$ADD($\Upsilon^P(m)$)
20: **end for**

Fig. 5. Algorithm of joining of microinstructions.

1: $\mathbf{K}(\boldsymbol{\Upsilon}^{\mathbf{P}}) = \varnothing$
2: **for** $m = 1$ to $\left|\boldsymbol{\Upsilon}^{\mathbf{P}}\right|$ **do**
3: $K(\Upsilon^P(m)) = \varnothing$
4: **for** $t = 1$ to $\left|\boldsymbol{\Upsilon}^{\mathbf{P}}[m]\right|$ **do**
5: $K(\Upsilon^P(m)) \rightarrow$ADD(INTTOBIN($t$,$N_C$))
6: **end for**
7: $\mathbf{K}(\boldsymbol{\Upsilon}^{\mathbf{P}}) \rightarrow$ADD($K(\Upsilon^P(m))$)
8: **end for**

Fig. 6. Algorithm of encoding of microinstructions.

1: **for** $m = 1$ to $\left|\boldsymbol{\Upsilon}^{\mathbf{P}}\right|$ **do**
2: $K(A)[\boldsymbol{\Upsilon}^{\mathbf{P}}[m] \rightarrow a_{m'} \rightarrow m] =$INTTOBIN($(m-1) * 2$,$R$)
3: **if** $\boldsymbol{\Upsilon}^{\mathbf{P}}[m] \rightarrow a_{m''} \neq \varnothing$ **then**
4: $K(A)[\boldsymbol{\Upsilon}^{\mathbf{P}}[m] \rightarrow a_{m''} \rightarrow m] =$INTTOBIN($(m * 2 - 1$,$R$)
5: **end if**
6: **end for**

Fig. 7. Algorithm of encoding of states.

3. *Encoding of microinstructions.* Let us encode each microinstruction Y_t in each pair $\Upsilon_{m'}^{m''}$ by binary code $K_{m'}^{m''}(Y_t)$ (Fig. 6).
4. *Encoding of states.* There is required a special encoding of states to satisfy the possibility of encoding of microinstruction with usage of partial code of current state. So, let us encode states $a_{m'}$ and $a_{m''}$ defining each pair $\Upsilon_{m'}^{m''}$ by following binary codes $K(a_{m'})$ and $K(a_{m''})$. It assures that these codes differ only on least significant bit Q_R. The algorithm presented in Figure 7 overwrites existing trivial binary encoding that is used by other synthesis methods by new codes.
5. *Formation of direct structural table of FSM* $\mathrm{PY_J}$. This table is formatted based on original transition table by adding columns with: code of current state $K(a_m)$, code of next state $K(a_s)$, excitation functions Φ_h that are equal to 1 to switch the FSM memory form code $K(a_m)$ to $K(a_s)$, and by replacing microinstruction column Y_h by column Ψ_h. The column Ψ_h consists variables that are equal to 1 in adequate code $K_{m'}^{m''}(Y_t)$. This table is a basis for formation of the system (18). The application for synthesis omits this step because all data for this table is created in previous steps and the table is required only for presentation purpose and manual synthesis. The application store the FSM model in table that are red from KISS2 file and additional collections.
6. *Formation of table of microinstruction decoder.* This table has columns: $K_{m'}^{m''}(Y_t)$, $K(a_m)[1:(R-1)]$, $K(Y_t)$. This table is basis for formation of the system (20). From similar reason like in previous step this table is also not created by the application for synthesis. The application creates the description of Y circuit in Verilog HDL in this step.
7. *Formation of logic equations.* This step is required to form Boolean equations describing the system (18). They are created based on direct structural table of FSM $\mathrm{PY_J}$ as sum of products in typical way [1], [2]. The application for synthesis builds equations in Verilog HDL. These equations create description of P circuit.
8. *Implementation of logic circuit into FPGA.* The combinational circuit P and the register RG are implemented with use of standard logic blocks. The circuit P with use of look-up tables and the register RG with D type flip-flops. The decoder Y is implemented with use of embedded memory blocks. The address is represented by (17) and microinstructions from sets $\Upsilon_{m'}^{m''}$ creates content of this memory.

   To satisfy such implementation, the whole circuit could be described with use of HDLs in approbate way, and then it should be passed into third party synthesis & implementation tools. The logic equations of the system (18) of combinational circuit P should be described with use of continuous assignment. The register RG should be described as R-bits D type flip-flop with use of standard synthesis template [18], [19]. The circuit Y has to be described as process with clock signal on the sensitivity list. It should be trigged by opposite edge of clock signal than the register RG. It satisfies that outputs are stable after one clock cycle [17]. The reset signal should be described as synchronous one and the content of the memory could be described with use of `case` statement with address as a selector. Additionally, there have to be added special synthesis directive to permit implementation with use of embedded memory blocks. The syntax of this directive depends on selected FPGA vendor. The top-level module could be described as connection of instantiated components.

V. METHOD APPLICATION ON EXAMPLE

The synthesis process described in previous section will be illustrated on example FSM $\mathrm{S_1}$ (Fig. 1) [17]. There is $M = 5$ states in this FSM and they create the set $A = \{a_1, \ldots, a_5\}$.

There is also $N = 5$ microoperations $Y = \{y_1, \ldots, y_5\}$. These microoperations create $T = 7$ microinstructions

$$\Upsilon = \left\{ \begin{array}{l} Y_1 = \{y_1\}, Y_2 = \{y_1, y_2\}, Y_3 = \{y_2\}, \\ Y_4 = \{y_2, y_3\}, Y_5 = \{y_3, y_4\}, \\ Y_6 = \varnothing, Y_7 = \{y_1, y_5\} \end{array} \right\}.$$

There is required to use $N_1 = \lceil \log_2 T \rceil = 3$ bits to encode all microinstructions in case of application of PY structure (Fig. 2b). It means, that the combinational circuit P have to implement $n_{\mathrm{PY}}(\mathrm{P}) = R + N_1 = 6$ logic functions and the decoder Y can be implemented as a memory block with 3-bit address and 5-bit word.

The application of PY_0 structure (Fig. 2c) is also possible. In this case, there is required to use $N_0 = \lceil \log_2 T_0 \rceil = 2$ bits to encode microinstructions. And now, that the combinational circuit P has to implement $n_{\mathrm{PY}_0}(\mathrm{P}) = R + N_0 = 5$ Boolean functions and the decoder Y can be implemented with use of a memory block with 5-bit address and and 5-bit word. It can be noticed that there is required to store more words in the memory but size of memory block does not change because the number of available block is not exceeded. Of course in other cases it can be exceeded and then the application of proposed method with joined encoding can be helpful.

The synthesis with joined multiple encoding of microinstructions into $\mathrm{PY_J}$ structure (Fig. 3) starts with division of microinstructions set into subsets based on current state (*step 1*). In our example, there is $M = 5$ such subsets: $\Upsilon(a_1) = \{Y_1, Y_2, Y_3\}$, $\Upsilon(a_2) = \{Y_2, Y_3\}$, $\Upsilon(a_3) = \{Y_3, Y_4\}$, $\Upsilon(a_4) = \{Y_5, Y_6\}$, and $\Upsilon(a_5) = \{Y_4, Y_5, Y_7\}$. Next, there is required to join these subsets into pairs (*step 2*). This is the most important step of this synthesis method. Because, for the example FSM S_1, the condition $M < 2^{\lceil \log_2 M \rceil}$ is satisfied the three biggest subsets are joined with empty set $\varnothing$ into pair. So, there are created four pairs: $\Upsilon_1^{\varnothing} = \{Y_1, Y_2, Y_3\}$, $\Upsilon_5^{\varnothing} = \{Y_4, Y_5, Y_7\}$, $\Upsilon_2^{\varnothing} = \{Y_1, Y_2\}$, and $\Upsilon_3^4 = \{Y_3, Y_4, Y_5, Y_6\}$. Now, the microinstruction can be encoded with binary code $K_{m'}^{m''}(Y_t)$ (*step 3*). In our case there is required to use $N_C = \lceil \log_2 T_C \rceil = 2$ bits to represent this code and encoding looks as follow: $K_1^{\varnothing}(Y_1) = 00$, $K_1^{\varnothing}(Y_2) = 01$, $K_1^{\varnothing}(Y_3) = 10$, $K_5^{\varnothing}(Y_4) = 00$, $K_5^{\varnothing}(Y_5) = 01$, $K_5^{\varnothing}(Y_7) = 10$, $K_2^{\varnothing}(Y_1) = 00$, $K_2^{\varnothing}(Y_2) = 01$, $K_3^4(Y_3) = 00$, $K_3^4(Y_4) = 01$, $K_3^4(Y_5) = 10$, and $K_3^4(Y_6) = 11$. To encode microinstructions there is required to use adequate state encoding (*step 4*). Because codes of states $K(a_{m'})$ and $K(a_{m''})$ have to differ on one last least significant bit Q_R the states could be encoded as follow: $K(a_1) = 000$, $K(a_2) = 010$, $K(a_3) = 100$, $K(a_1) = 101$,and $K(a_5) = 110$ for our example. It can be noticed that codes 001, 111, and 011 can not be assigned to any state because subsets $\Upsilon(a_{m'})$ depending on states a_1, a_5, and a_2 are joined with empty set $\varnothing$. Now, the transformed direct structural table of FSM $\mathrm{PY_J}$ (*step 5.*) can be created. It is presented in Table I for example FSM S_1. There is also required to create table of the decoder Y (*step 6*). It describes content of the memory. It is presented in Table II for example FSM S_1. Based on the transformed DST (Tab. I) there can be formed logic equations of system (18) (*step 7*). For our example FSM S_1, there can

TABLE I
TRANSFORMED DST OF THE MEALY FSM S_1

a_m	$K(a_m)$	a_s	$K(a_s)$	X_h	Ψ_h	Φ_h	h
a_1	000	a_2	010	$x_1\ x_2$	$--$	D_2	1
		a_3	100	$\overline{x_1}\ x_2$	ψ_2	D_1	2
		a_4	101	$\overline{x_2}$	ψ_1	$D_1\ D_3$	3
a_2	010	a_3	100	x_2	$--$	D_1	4
		a_4	101	$\overline{x_2}$	ψ_2	$D_1\ D_3$	5
a_3	100	a_3	100	x_2	$--$	D_1	6
		a_4	101	$\overline{x_2}\ \overline{x_3}$	$--$	$D_1\ D_3$	7
		a_5	110	$\overline{x_2}\ x_3$	ψ_2	$D_1\ D_2$	8
a_4	101	a_5	110	x_3	ψ_1	$D_1\ D_2$	9
		a_3	100	$\overline{x_3}$	$\psi_1\ \psi_2$	D_1	10
a_5	110	a_1	000	x_1	ψ_1	$--$	11
		a_5	110	$\overline{x_1}\ x_3$	ψ_2	$D_1\ D_2$	12
		a_4	101	$\overline{x_1}\ \overline{x_3}$	$--$	$D_1\ D_3$	13

TABLE II
TABLE OF DECODER Y

$K(a_m)[1:2]$ Q_1Q_2	$K_{m'}^{m''}(Y_t)$ $\psi_1\psi_2$	Y_t $y_1y_2y_3y_4y_5$	t_0
00	00	10000	1
00	01	11000	2
00	10	01000	3
01	00	11000	4
01	01	01000	5
10	00	01000	6
10	01	01100	7
10	10	00110	8
10	11	00000	9
11	00	01100	10
11	01	00110	11
11	10	10001	12

TABLE III
PARAMETERS OF MEALY FSM S_1

	P	PY	PY_0	$\mathrm{PY_J}$
$n(\mathrm{P})$	8	6	5	5
$n(\mathrm{RG})$	3	3	3	3
$n(\mathrm{Y})$	0	40	160	80

be formed, for example:

$$\begin{aligned} D_3 &= \overline{Q_1}\,\overline{Q_2}\,\overline{Q_3}\,\overline{x_2} + \overline{Q_1}\,Q_2\,\overline{Q_3}\,\overline{x_2} + Q_1\,\overline{Q_2}\,\overline{Q_3}\,\overline{x_2}\,\overline{x_3} \\ &\quad + Q_1\,Q_2\,\overline{Q_3}\,\overline{x_1}\,\overline{x_3}, \\ \psi_2 &= \overline{Q_1}\,\overline{Q_2}\,\overline{Q_3}\,\overline{x_1}\,x_2 + \overline{Q_1}\,Q_2\,\overline{Q_3}\,\overline{x_2} + Q_1\,\overline{Q_2}\,\overline{Q_3}\,\overline{x_2}\,x_3 \\ &\quad + Q_1\,\overline{Q_2}\,Q_3\,\overline{x_2}\,\overline{x_3} + Q_1\,Q_2\,\overline{Q_3}\,\overline{x_1}\,x_3. \end{aligned}$$

It can be noticed that these equations are not minimized ones. This proceed is not required at logic synthesis level because it would be preformed by third party synthesis & implementation tools in next step. This step finishes logic synthesis process. Now, the key parameters, like number of logic equations or number of memory bits, of logic circuit can be calculated. The Table III presents these parameters for FSM S_1, where

- $n(\mathrm{P})$ is a number of logic equations implemented by the combinational circuit P,
- $n(\mathrm{RG})$ is a number of D type flip-flops in the register RG,
- $n(\mathrm{Y})$ is a number of used memory bits in the decoder Y.

```
module S1_P (x, Q, psi, D);
   input [1:3] x;
   input [1:3] Q;
   output [1:2] psi;
   output [1:3] D;

   assign psi[1]=~x[2] & ~Q[1] & ~Q[2] & ~Q[3]
      | x[3] & Q[1] & ~Q[2] & Q[3]
      | ~x[3] & Q[1] & ~Q[2] & Q[3]
      | x[1] & Q[1] & Q[2] & ~Q[3];
   ...
   assign D[3]=~x[2] & ~Q[1] & ~Q[2] & ~Q[3]
      | ~x[2] & ~Q[1] & Q[2] & ~Q[3]
      | ~x[2] & ~x[3] & Q[1] & ~Q[2] & ~Q[3]
      | ~x[1] & ~x[3] & Q[1] & Q[2] & ~Q[3] ;
endmodule
```

Fig. 8. Verilog description of combinational circuit P.

```
module S1_RG (clk, res, D, Q);
   input clk, res;
   input [1:3] D;
   output [1:3] Q;
   reg [1:3] Q;

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

Fig. 9. Verilog description of register RG.

Now, the logic circuit can be prepared for implementation (*step 8*). First, the whole circuit is described in Verilog HDL. The combinational circuit P is described using continues assignment based on equations received in *step 7* (Fig. 8). The register RG is described as 3-bits D type flip-flop with asynchronous reset signal and trigged by rising edge of clock signal (Fig. 9). There was used standard synthesis template [18], [19] for this purpose. The circuit Y is described with use of one process trigged by falling edge of the clock signal (Fig. 10). The content of the memory is described with use of `case` statement and it was taken from the table of decoder Y (Tab. II) that was received in *step 6*. There is also set special synthesis attribute `bram_map` to `yes` to satisfy implementation into embedded memory blocks. This is attribute for Xilnix devices, and in case od different vendor devices it should be replaced by different one. Finally, the top-level module is described as instantiation of already described components (Fig. 11). Their connections should be adequate to the logic structure presented in Figure 3. Now, such description can be passed into third party synthesis & implementation tools. In this case, there was used Xilnix ISE with XST, and the obtained results are shown in Table IV.

The application of this method with joined multiple encoding reduce the memory size by two and do not affect the number of implemented logic functions in comparison with well known method multiple encoding. The presented in Table III parameters of example FSM S_1 shows this

```
module S1_Y (clk, psi, Q, y);
   input clk;
   input [1:2] psi;
   input [1:2] Q;
   output [1:5] y;
   reg [1:5] y;

   // synthesis attribute bram_map of S1_Y is
      yes
   always @(negedge clk)
      case ({Q,psi})
         4'b0000: y = 5'b10000;
         4'b0001: y = 5'b11000;
         4'b0010: y = 5'b01000;
         ...
      endcase
endmodule
```

Fig. 10. Verilog description of decoder Y.

```
module S1 (clk, res, x, y);
   input clk, res;
   input [1:3] x;
   output [1:5] y;
   wire [1:3] d;
   wire [1:3] q;
   wire [1:2] psi;

   S1_RG UD (.clk(clk), .res(res), .D(d),
             .Q(q));
   S1_P  UP (.x(x), .Q(q),
             .D(d), .psi(psi));
   S1_Y  UY (.clk(clk), .psi(psi), .Q(q[1:2]),
             .y(y));
endmodule
```

Fig. 11. Verilog description of top-level module.

TABLE IV
IMPLEMENTATION RESULTS OF MEALY FSM S_1

	P	PY	PY_0	PY_J
Slices	10	9	8	7
LUTs	18	16	14	13
FFs	3	3	3	3
BRAMs	–	1	1	1

dependence. Although, the implementation results depends also on functional decomposition of logic equations process and state encoding, and it cause that the number of utilized logic elements could change. It have to be mentioned that encoding of states for both synthesis method is different because the proposed method with joined multiple encoding required special state encoding algorithm. Additionally, the number of embedded memory blocks is the same for all synthesized structures because of small size of example FSM S_1. In case of bigger examples, the value of $n(Y)$ parameter has also influence on number of utilized embedded memory blocks.

VI. Summary

There was proposed method of synthesis and double-level structure of a digital device implementing an FSM. This structure is dedicated to presented synthesis method. The synthesis method is based on the multiple encoding of microinstructions of a state machine and structural decomposition of its logic circuit. This method was adapted for synthesis process into FPGA devices. It takes advantage of features of new FPGA devices like embedded memory blocks. The utilization of such resources leads to reduce the number of required standard logic blocks, like LUTs, for implementation of a control unit. The proposed method is also oriented on reduction of memory size.

References

[1] S. I. Baranov, *Logic Synthesis for Control Automat*. Boston: Kluwer Academic Publishers, 1994.

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

[3] Z. Salcic, *VHDL and FPLDs in Digital Systems Design, Prototyping and Customization*. Boston: Kluwer Academic Publishers, 1998.

[4] H. Kubátová, "Finite state machine implementation in FPGAs," in *Design of Embedded Control Systems*, M. Adamski, A. Karatkevich, and M. Węgrzyn, Eds. New York: Springer, 2005, pp. 177–187.

[5] J. Jenkins, *Designing with FPGAs and CPLDs*. Upper Saddle River, NJ: Prentice Hall, 1994.

[6] C. Scholl, *Functional Decomposition with Application to FPGA Synthesis*. Boston: Kluwer Academic Publishers, 2001.

[7] M. Rawski, H. Selvaraj, T. Łuba, and P. Szotkowski, "Application of symbolic functional decomposition concept in FSM implementation targeting FPGA devices," in *Proceedings of the 6th International Conference on Computational Intelligence and Multimedia Applications ICCIMA'05*, Las Vegas, NV, 2005, pp. 153–158.

[8] G. Borowik, M. Rawski, G. abiak, 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].

[9] M. Adamski and A. Barkalov, *Architectural and Sequential Synthesis of Digital Devices*. Zielona Góra: University of Zielona Góra Press, 2006.

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

[11] A. Bukowiec, "Synthesis of Mealy FSM with multiple shared encoding of microinstructions and internal states," in *Proceedings of IFAC Workshop on Programmable Devices and Embedded Systems PDeS'06*, Brno, Czech Republic, 2006, pp. 95–100.

[12] ——, "Architectural synthesis of FSMs with joined multiple encoding," *Electrical Review*, vol. Vol. 2011, no. No. 11, pp. 150–153, 2011.

[13] A. Bukowiec, A. Barkalov, and L. Titarenko, "FSMs implementation into FPGAs with multiple encoding of states," in *Proceedings of IEEE East-West Design & Test Symposium EWDTS'08*. Lviv, Ukraine: IEEE, 2008, pp. 72–75.

[14] E. F. Moore, "Gedanken-experiments on sequential machines," in *Automata Studies*, ser. Annals of Mathematical Studies, C. E. Shannon and J. McCarthy, Eds. Princeton, NJ: Princeton University Press, 1956, vol. 34, pp. 129–153.

[15] G. H. Mealy, "A method for synthesizing sequential circuits," *Bell System Technical Journal*, vol. Vol. 34, no. No. 5, pp. 1045–1079, 1955.

[16] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide. version 3.0." Microelectronics Center of North Carolina, Research Triangle Park, NC, Tech. Rep. 1991-IWLS-UG-Saeyang, 1991. [Online]. Available: http://jupiter3.csc.ncsu.edu/~brglez/Cite-BibFiles-Reprints-home/Cite-BibFiles-Reprints-Central/BibValidateCentralDB/Cite-ForWebPosting/1991-IWLSUG-Saeyang/1991-IWLSUG-Saeyang_guide.pdf

[17] 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.

[18] P. Eles, K. Kuchcinski, and Z. Peng, *System Synthesis with VHDL*. Norwell: Springer, 1998.

[19] D. Thomas and P. Moorby, *The Verilog Hardware Description Language*, 5th ed. Norwell, MA: Kluwer Academic Publishers, 2002.