Designing Moore FSM with Transformation of State Codes

Kamil Mielcarek, Alexander Barkalov, Larisa Titarenko

Designing Moore FSM with Transformation of State Codes

Kamil Mielcarek, AlexanderBarkalov, and Larisa Titarenko

Institute of Metrology, Electronics and Computer Science
Faculty of Computer, Electrical and Control Engineering
University of Zielona Góra,
ul. prof. Z. Szafrana 2, 65-516 Zielona Góra, Poland
{K.Mielcarek,A.Barkalov,L.Titarenko}@imei.uz.zgora.pl

Abstract. A design method is proposed for FPGA-based Moore FSMs. The method is based on transformation of state codes into codes of collections of output functions and classes of pseudoequivalent states. Example of design and results of investigations are given.

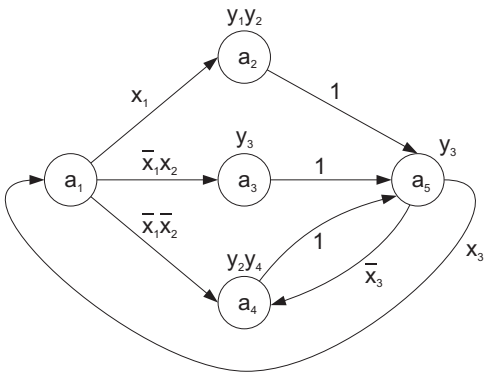
Keywords: Moore FSM, FPGA, look-up table, EMB, design, pseudoequivalent states

1 Introduction

The model of Moore finite state machine (FSM) is often used during the design of control units [2]. One of the important problems of FSM synthesis is need to reduce the hardware consumed by FSM logic circuit [3, 12]. The methods of solution of this problem depend strongly on features of logic elements used for designing the circuits [6, 8]. In this article we discuss the case when field-programmable gate arrays (FPGA) are used to implement the Moore FSM logic circuit.

To implement the FSM logic circuit, it is enough to use two components of FPGA fabric. These components are logic elements (LE) and a matrix of programmable interconnections [1, 21]. An LE consists of a look-up table (LUT) element, a programmable flip-flop and multiplexers. A LUT can be viewed as a random access memory block having S address inputs and a single output. One LUT can keep a truth table of an arbitrary Boolean function having up to S arguments. The flip-flop of LE could be bypassed. It means that the output of LE could be either combinational or registered.

A LUT has rather small number of inputs ($S \leq 6$) [1, 21]. This peculiarity leads to necessity of functional decomposition [14, 16] for Boolean functions having more than S arguments. It results in multilevel logic circuits with complex interconnections. To reduce the hardware amount, it is necessary to diminish the numbers of arguments in Boolean functions representing an FSM logic circuit. We propose one of possible approaches for solution of this problem. Our approach is based on: 1) using the classes of pseudoequivalent states of Moore FSM [6] and 2) encoding of the collections of output functions [4].



$A = \{a_1, ..., a_5\}$, $X = \{x_1, x_2, x_3\}$, $Y = \{y_1, ..., y_4\}$

Fig. 1. State transition graph for Moore FSM S_1

2 Theoretical background

A Moore FSM is characterised by the following sets: $A = \{a_1, ..., a_M\}$ is a set of internal states; $X = \{x_1, ..., x_L\}$ is a set of input variables; $Y = \{y_1, ...y_N\}$ is a set of output functions. Transitions between the states could be represented by a state transition graph (STG) [12]. There is an STG of Moore FSM S_1 shown in Fig. 1.

The vertices of STG correspond to states $a_m \in A$, while the edges to transitions between the states. The transitions are determined by inputs $X_h (h = \overline{1, H})$ equal to conjunctions of some elements $x_e \in X$ (or their complements). There is a collection of output functions (COF) $Y_q \subseteq Y$ marked above a vertex $a_m \in A$. It means that the COF $Y_q ((q = \overline{1, a})$ is generated while an FSM is in the state $a_m \in A$.

The states $a_m \in A$ are encoded using state variables $T_r \in T$, where $T = \{T_1, ..., T_R\}$ is a set of state variables. State codes $K(a_m)$ are assigned during the step of state assignment [12]. The number R is determined as

$$R = \lceil log_2 M \rceil, \tag{1}$$

where $\lceil a \rceil$ is a minimum integer not less than a. To change contents of flip-flops, input memory functions are used forming the set $\Phi = \{D_1, ..., D_R\}$.

To design the Moore FSM logic circuit, it is necessary to find the following systems of Boolean functions:

$$\Phi = \Phi(T, X), \tag{2}$$

$$Y = Y(T). \tag{3}$$

Systems (2)-(3) determine the structural diagram of Moore FSM S_1 (Fig. 2a). This diagram targets the FPGA chips.

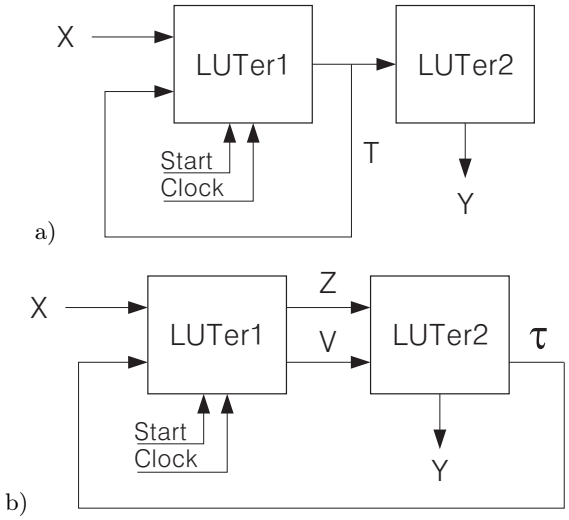


Fig. 2. Structural diagrams of Moore FSM U_1 (a) and U_2 (b)

We use the name "LUTer" to denote a logic circuit consisting from LUTs. If there are flip-flops in a circuit, they are controlled by pulses *Start* and *Clock*. The pulse *Start* loads the zero code into a register formed by these flip-flops. It corresponds to the initial state $a_1 \in A$. The pulse *Clock* allows changing content of flip-flops. It corresponds to a transition between the states.

The LUTer1 implements the system (2), the LUTer2 the system (3). Let the following condition take place:

$$R \leq S. \tag{4}$$

In this case, it is enough a single LUT to implement any function $y_n \in Y$. If the condition (4) is violated, then the number of LUTs exceeds N in the circuit of LUTer2. In this case, the circuit of LUTer2 is multilevel.

In this article, we discuss a design method for reducing the number of LUTs in LUTer2 if the condition (4) is violated. Also, we propose to use pseudoequivalent states [6] to decrease the hardware

3 Main idea of proposed method

The states $a_m, a_s \in A$ are pseudoequivalent states (PES) if identical inputs result in identical next states for both states $a_m, a_s \in A$. For example, there are PES $a_2, a_3, a_4 \in A$ for Moore FSM S_1 (Fig. 1). Let $\Pi_A = \{B_1, ..., B_I\}$ be a partition of the set A by the classes of PES ($I \leq M$).

In the case of S_1, there is $I = 3$. There are the following classes of PES: $B_1 = \{a_1\}$, $B_2 = \{a_2, a_3, a_4\}$, $B_3 = \{a_5\}$.

Let us encode the classes $B_i \in \Pi_A$ by binary codes $K(B_i)$ having R_I bits:

$$R_I = \lceil log_2 I \rceil. \tag{5}$$

Let us use the variables $\tau_r \in \mathcal{T}$ for encoding of the classes $B_i \in \Pi_A$, where $\mathcal{T} = \{\tau_1, ..., \tau_{RI}\}$.

Let it be Q different COF $Y_q \subseteq Y$ for a Moore FSM. Let us encode each COF Y_q by a binary code $K(Y_q)$ having R_Q bits:

$$R_Q = \lceil log_2 Q \rceil. \tag{6}$$

Let us use the variables $z_r \in Z$ for encoding of the COF $Y_q \subseteq Y$, where $|Z| = R_Q$.

Analysis of Fig. 1 shows that there are the following COF in FSM S_1: $Y_1 = \emptyset, Y_2 = \{y_2, y_3\}, Y_3 = \{y_3\}, Y_4 = \{y_2, y_4\}$. So, there is $Q = 4$. It gives $R_Q = 2$ and $Z = \{z_1, z_2\}$.

The COF Y_1 corresponds to the class B_1. The collections Y_2 and Y_3 determine the class B_2. But the COF Y_4 determines two classes, B_2 and B_3. It means that it is necessary to have some identifiers to distinguish the classes B_2 and B_3 for the COF Y_4. In the discussed case, it is enough two identifiers (I_1 and I_2) to distinguish the classes. Let the pair $\langle Y_4, I_2 \rangle$ determine the class B_2, whereas $\langle Y_4, I_2 \rangle$ the class B_3.

In common case, it is enough K identifiers forming the set $ID = \{I_1, ..., I_K\}$. Let us encode them using R_K variables where

$$R_K = \lceil log_2 K \rceil. \tag{7}$$

Let us use variables $v_r = V$ for encoding of identifiers where $|V| = R_K$.

Basing on these preliminaries, we propose the structural diagram of Moore FSM U_2 (Fig. 2b). In U_2, the LUTer1 generates functions

$$Z = Z(\tau, X); \tag{8}$$

$$V = V(\mathcal{T}, X). \tag{9}$$

The LUTer2 implements the systems

$$Y = Y(Z); \tag{10}$$

$$\mathcal{T} = \mathcal{T}(Z, V). \tag{11}$$

Our analysis of the library of standard benchmarks [11] shows that the following conditions take places:

$$R_Q < R; \tag{12}$$

$$R_I < R; \tag{13}$$

$$H(U_2) < H(U_1). \tag{14}$$

We use the symbol $H(U_i)$ to denote the number of terms in the system Φ of $U_i(\overline{1,2})$.

Basing on (12)–(14), we can expect that logic circuits obtained for FSM U_2 consume less

Let us point out that a state of transition $a_s \in A$ is represented by a pair $\langle Y_q, I_k \rangle$, where $A_s \in B_i$ and $Y_q \subseteq Y$ is generated in the state $a_s \in A$. It means that the code $K(a_s)$ is represented as

$$K(a_s) = K(Y_q) * K(I_k). \tag{15}$$

In (15), the symbol $K(Y_q)$ stands for the code of COF Y_q, the symbol $K(I_k)$ for the code of identifier $I_k \in ID$. So, the code $K(a_s)$ should be transformed into corresponding codes from (15).

4 Proposed design method

In this article, we propose a design method for U_2. It includes the following steps:

1. Constructing partition Π_A and collections $Y_q \subseteq Y$ for a given STG.
2. Constructing the pairs $\langle Y_q, I_k \rangle$ for states $a_s \in A$.
3. Encoding of classes $B_i \in \Pi_A$, collections $Y_q \subseteq Y$ and identifiers $I_k \in ID$.
4. Transforming initial STG into class transition graph (CTG).
5. Finding systems (8)–(11).
6. Implementing FSM logic circuit using LUTs of a particular FPGA chip.

Let us apply this method for the Moore FSM S_1. As it is mentioned before, there are the partition $\Pi_A = \{B_1, B_2, B_3\}$ with classes $B_1 = \{a_1\}$, $B_2 = \{a_2, a_3, a_4\}$ and $B_3 = \{a_5\}$ and the collections $Y_1 = \emptyset$, $Y_2 = \{y_1, y_2\}$, $Y_3 = \{y_3\}$ and $Y_4 = \{y_2, y_4\}$. It gives $I = 3$ and $Q = 4$.

There are the following pairs $\langle Y_q, I_k \rangle$ in the discussed case: $\langle Y_1, \emptyset \rangle = a_1, \langle Y_2, \emptyset \rangle = a_2, \langle Y_3, \emptyset \rangle = a_3, \langle Y_4, I_1 \rangle = a_4$ and $\langle Y_4, I_2 \rangle = a_5$. So, there is $K = 2$ and $I = \{I_1, I_2\}$.

Using (1), (5), (6) and (7), we can find that $R = 3$, $R_I = 2$, $R_Q = 2$ and $R_K = 1$. Using (15), we can find that the codes $K(a_s)$ should include $R_Q + R_K = 3$ bits. It gives the set $\Phi = \{D_1, D_2, D_3\}$. Functions D_1, D_2 determine the variables z_1 and z_2. Function D_3 determines the variable v_1. So, there are the following sets: $Z = \{z_1, z_2\}$, $V = \{v_1\}$ and $\mathcal{T} = \{\tau_1, \tau_2\}$.

Let us encode in the trivial way the classes $B_i \in \Pi_A$, the collections $Y_q \subseteq Y$ and identifiers $I_k \in I$. It gives the following codes: $K(B_1) = K(Y_1) = 00$, $K(B_2) = K(Y_2) = 01$, $K(B_3) = K(Y_3) = 10$, $K(Y_4) = 11$, $K(I_1) = 0$ and $K(I_2) = 1$.

Using these codes, we can find the codes $K(a_s)$ corresponding to (15). There are the following codes: $K(a_1) = 00*, K(a_2) = 01*, K(a_3) = 10*, K(a_4) = 110$ and $K(a_5) = 111$. The symbol "*" corresponds to the element $\emptyset$ in the pairs $\langle Y_q, I_K \rangle$.

To construct a CTG, let us replace the states $a_m \in B_i$ by a single vertex $B_i \in \Pi_A$. In the discussed case, the CTG is shown in Fig. 3.

If some class $B_i \in \Pi_A$ has more than one state, then these states are shown inside a particular vertex of CTG. We did it for the class B_2.

There are the following columns in the structure table of U_2: B_i, $K(B_i)$, a_s, $K(a_S)$, X_h, Φ_h, h. This table is constructed on the base of CTG. The column

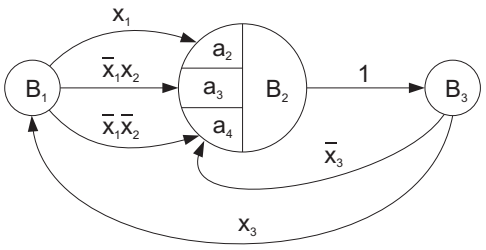


Fig. 3. Class transition graph for Moore FSM S_1

Table 1. Structure table of Moore FSM S_1

B_i	$K(B_i)$	a_s	$K(a_s)$	X_h	Φ_h	h
		a_2	010	x_1	D_2	1
B_1	00	a_3	100	$\bar{x_1}x_2$	D_1	2
		a_4	110	$\bar{x_1}\bar{x_2}$	D_1D_2	3
B_2	$*1$	a_5	111	1	$D_1D_2D_3$	4
B_3	$1*$	a_1	000	x_3	–	5
		a_4	110	$\bar{x_3}$	D_1D_2	6

$K(a_s)$ includes a code of the state of transition (12). The column X_h contains an input signal determining a transition from a state $a_m \in B_i$ into a state $a_s \in A$. The column Φ_h contains input memory functions $D_r \in \Phi$ equal to 1 to load the codes $K(Y_q)$ and $K(I_K)$ into the distributed state register.

The column h contains the number of transition ($h = \overline{1, H(U_2)}$). In the case of S_1, this table has $H(U_2) = 6$ rows (Table 1). We use the combination 11 to optimize the codes of B_2 and B_3. We replace all "*" in state codes by zeros.

The ST is used to derive the functions (8) and (9). In the discussed case, there are the following functions:

$$\begin{aligned} z_1 &= D1 = \bar{\tau_1}\bar{\tau_2}\bar{x_1} \vee \tau_2 \vee \tau_1\bar{x_3}; \\ z_2 &= D_2 = \bar{\tau_1}\bar{\tau_2}x_1 \vee \bar{\tau_1}\bar{\tau_2}\bar{x_2} \vee \tau_2 \vee \tau_1\bar{x_3}; \\ v_1 &= D_3 = \tau_2. \end{aligned} \tag{16}$$

To find the functions (10), we should find disjunctions of COF $Y_q \subseteq Y$ for each function $y_n \in Y$. In the discussed case, there are the following functions:

$$\begin{aligned} y_1 &= Y_2 = \bar{z_1}z_2; \\ y_2 &= Y_2 \vee Y_4 = z_2; \\ y_3 &= Y_3 = z_1\bar{z_2}; \\ y_4 &= Y_4 = z_1z_2. \end{aligned} \tag{17}$$

To find the functions (11), we should find disjunctions of state codes $a_s \in B_i$ for each class B_i: $B_1 = A_1, B_2 = A_2 \vee A_3 \vee A_4$ and $B_3 = A_5$. It gives the following functions: $B_1 = \bar{z_1}\bar{z_2}, B_2 = \tau_2 = \bar{z_1}z_2 \vee z_1\bar{z_2} \vee z_1z_2v_1; B_3 = \tau_1 = z_1z_2v_1$.

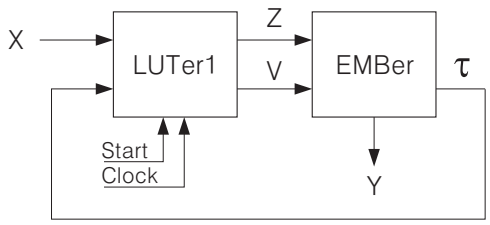


Fig. 4. Structural diagram of Moore FSM U_3

To execute the last step of the proposed design method, such problems should be solved as the mapping, placing and routing [10]. To do it, some industrial packages should be used [1, 21]. In our research, we use program tools of Xilinx to implement the FSM logic circuits.

5 Replacement LUTs by EMBs

It is possible to improve the characteristics of FSM replacing LUTs by embedded memory blocks (EMB) [5, 7, 9, 15, 18–20]. The EMBs of modern FPGA chips [2, 13] have a property of configurability. It means that such parameters as the number of cells and their outputs can be changed. There are the following typical configurations of EMBs: 32K×1, 16K×2, 8K×4, 4K×8, 2K×16, 1K×32 and 512×64 (bits) [2, 13]. So, modern EMBs are very flexible and can be tuned to meet a particular design.

It is possible to implement an FSM circuit using only a single EMB [20]. It can be done if the following condition takes place:

$$(N + R) \cdot 2^{L+R} \leq V_0. \tag{18}$$

In (18), the symbol V_0 stands for the number of cells in the configuration of EMB having $t_F = 1$ outputs. Of course, the condition (18) has place only for rather simple FSMs.

To improve the characteristics of U_1, we propose to replace the LUTer2 by EMBer. The symbol EMBer stands for the circuit implemented with EMBs. Such a replacement leads to the FSM U_3 (Fig. 4).

In this FSM, the EMBer implements the systems (10) and (11). Our analysis of the library [11] shows that the following condition takes place for all standard benchmarks [11]:

$$2^{R_Q+R_K}(N + R_I) \leq V_0. \tag{19}$$

It means that only a single EMB is enough for implementing FSM circuits based on the model U_3. This model can be used till the following condition takes place:

$$S_A \geq R_Q + R_K. \tag{20}$$

Table 2. Table of EMBer of Moore FSM S_1

$K(Y_q)$	$K(I_K)$	Y	$\mathcal{T}$	h	a_m
z_1z_2	v_1	$y_1y_2y_3y_4$	$\tau_1\tau_2$		
00	0	0000	00	1	a_1
00	1	****	**	2	*
01	0	1100	01	3	a_2
01	1	****	**	4	*
10	0	001*	01	5	a_3
10	1	****	**	6	*
11	0	0001	01	7	a_4
11	1	0001	10	8	a_5

In (20), the symbol S_A stands for the number of address bits for some configuration of EMB.

There are the same steps 1–5 in design methods for U_2 and U_3. But there is a difference in the executing the step 6. In the case of U_2 it is necessary to construct the table of EMBer. It includes the following columns: $K(Y_q)$, $K(I_K)$, Y, $\mathcal{T}$, h. The first two columns determine an address of a memory cell. The table includes H_E lines where

$$H_E = 2^{R_Q+R_K}. \tag{21}$$

In the discussed case, there is $R_Q = 2$ and $R_K = 1$. So, there is $H_E = 8$. Table 2 represents the table of EMBer for Moore FSM S_1.

We added the column a_m in Table 2 to show the correspondence among the pairs $\langle Y_q, I_K\rangle$ and states $a_m \in A$. If a pair $\langle K(Y_q), K(I_K)\rangle$ does not correspond to any state, then there are asterisks in the corresponding lines of Table 2. If a state $a_m \in B_i$, then the corresponding line includes the code of this class. For example, there is the relation $a_2, a_3, a_4 \in B_2$, so there is the code 01 in the lines 3,5 and 7 of Table 2.

Let us point out that the structure table of Moore FSM S_1 in the case of U_3 is the same as for U_2 (Table 1). It means that systems (8)–(9) are represented as the system (16) for both models U_2 and U_3 of FSM S_1.

6 Results of investigations

We use the standard benchmarks [11] to investigate efficiency of proposed method. The library LGSynth93 includes 48 finite-state machines represented in the KISS2 format.

To use the benchmarks, we worked out the CAD tool named K2F. It translates the KISS2 file into VHDL model of the Moore FSM. We use Active-HDL environment to synthesize and simulate the FSM. To implement the FSM circuits, we use Xilinx ISE 14.1 package. We choose the FPGA device XC5vex30ff324 by

Table 3. Results of investigation for FSM U_2

BM	U_1 auto	U_1 compact	JEDI	DEMAIN	U_2
cse	74	72	57	63	42
dk16	72	59	49	67	36
ex1	96	111	90	86	56
keyb	84	98	70	74	52
kirkman	77	85	68	70	50
planet	152	208	145	154	105
S1488	210	212	183	185	134
S298	542	495	443	465	315
scf	269	303	232	260	174
styr	178	191	156	158	114
Total	1754	1834	1482	1582	1078
Percentage	163%	170%	137%	147%	100%

Virtex-5 of Xilinx as a target platform. Some results of investigations are shown in Table 3.

There are the numbers of LUTs necessary for implementing logic circuits of different FSMs for given benchmarks in Table 3. There is the name of a benchmark in the column BM. The following methods were taken for comparison with U_2: 1) FSM U_1 designed with method "Auto" of ISE 14.1; 2) FSM U_1 designed with method "Compact" of ISE 14.1; 3) Moore FSM used the state assignment by JEDI from SIS [17]; 4) Moore FSM designed by DEMAIN [13].

We show the most complex benchmarks from [11] into Table 3. As follows from Table 3, the FSM U_2 always produces the best results. The row "Total" gives the overall numbers of LUTs for all benchmarks (for the given design method). The row "Percentage" shows the percentage of the total number of LUTs regarding the FSM U_2. We have taken the total number of LUTs for U_2 as 100%.

Our approach gives better results for 60% of all benchmarks from [11]. If an FSM has less than 20 states, then better results belong to JEDI. Our methods produces better results because: 1) we take into account the existence of PES and 2) we encode the collections of output functions. The JEDI uses PES but there is no encoding of COF for this method. Also, the JEDI uses state variables $T_r \in T$ for representing classes of PES. We use some additional variables $\tau_r \in \mathcal{T}$. Due to these two features, our method produces better circuits than JEDI (and all other methods from Table 3).

We also investigated the efficiency of the model U_3. The same FSMs were taken for comparison with U_3 as it was for the case of U_2. The results of investigations for FSM U_3 are shown in Table 4.

Table 4. Results of investigation for FSM U_3

BM	U_1 auto	U_1 compact	JEDI	DEMAIN	U_3
cse	64	62	47	53	32
dk16	65	52	42	60	29
ex1	84	99	78	74	44
keyb	73	87	59	63	41
kirkman	68	76	59	61	41
planet	128	185	122	131	82
S1488	184	186	157	159	108
S298	529	482	430	452	302
scf	241	275	204	232	146
styr	164	177	142	144	100
Total	1601	1681	1329	1429	925
Percentage	173%	182%	144%	154%	100%

Our analysis of the library [16] shows that it is enough a single EMB to implement output functions in the cases of U_1 auto, U_1 compact, JEDI and DEMAIN. The same is true for the case of U_3. So, there are only numbers of LUTs shown in Table 4.

As follows from Table 4, our approach gives better results for all complex benchmarks from [16]. The reasons for it are stated during the analysis of Table 3. But if EMBs are used, then there is a gain exceeds the gain for the case of LUT-based FSMs. It follows from comparison the rows "Percentage" from Tables 3 and 4.

To explain this conclusion, let us construct a table with comparison of numbers of LUTs necessary in the case of LUT-based and EMB-based FSMs. It is Table 5. There is a ratio of numbers of LUTs for corresponding cells of Table 3 and Table 4.

In all cases, practically the same number of LUTs is replaced by a single EMB. It is equal to N for U_1 auto, U_1 compact, JEDI and DEMAIN. It is equal to $N + R_I$ for U_2 and U_3. So, the less number of LUTs in LUTer1 (or LUTer), the better results can be obtained due to replacement LUTer2 by EMBer. For example, in the case of U_1 auto saving is equal to 12%. But transition from U_2 to U_3 saves in average 22% of LUTs (see the last column of Table 4).

7 Conclusion

The proposed method is based on representing state codes as concatenations of class codes and codes of collections of output functions. It targets LUT-based FSM circuits. Our investigation shows that the method produces FSM circuits having fewer LUTs than the methods used by SIS, DEMAIN and ISE 14.1. The

Table 5. Relations between Tables 3 and 4

BM	U_1 auto	U_1 compact	JEDI	DEMAIN	U_2/U_3
cse	1,16	1,16	1,21	1,19	1,31
dk16	1,11	1,13	1,16	1,12	1,24
ex1	1,14	1,12	1,15	1,16	1,27
keyb	1,15	1,13	1,19	1,17	1,27
kirkman	1,13	1,12	1,15	1,15	1,22
planet	1,18	1,12	1,19	1,17	1,28
S1488	1,14	1,15	1,17	1,16	1,24
S298	1,02	1,03	1,03	1,03	1,04
scf	1,11	1,10	1,14	1,12	1,21
styr	1,09	1,08	1,10	1,10	1,14
Total	11,23	11,14	11,49	11,37	12,22
Average	1,12	1,11	1,15	1,14	1,22

method could be applied for rather complex Moore FSMs having more than 20 states.

The future direction of our research is the development of a method for encoding of collections of output functions. It is necessary if the following condition is violated:

$$R_Q > S. \tag{22}$$

The method should diminish the number of arguments in Boolean function (8). It allows hardware reduction in the circuit of block LUTer2.

References

1. Altera: `http://www.altera.com`, accessed: January, 2016
2. Baranov, S.: Logic Synthesis of Control Automata. Kluwer Academic Publishers (1994)
3. Barkalov, A., Titarenko, L.: Logic Synthesis for FSM-Based Control Units. Springer-Verlag, Berlin (2009)
4. Barkalov, A., Titarenko, L., Barkalov Jr., A.: Structural decomposition as a tool for the optimization of an FPGA-based implementation of a mealy FSM. Cybernetics and Systems Analysis 48(2), 313–322 (2012)
5. Barkalov, A., Titarenko, L., Kołopieńczyk, M.: EMB-based design of Mealy FSM. In: 12th IFAC Conference on Programmable Devices and Embedded Systems. pp. 215–220. Velké Karlovice, Czechy (2013)
6. Barkalov, A., Titarenko, L., Kołopieńczyk, M., Mielcarek, K., Bazydło, G.: Logic synthesis for FPGA-based Finite State Machines, Studies in Systems, Decision and Control, vol. 38. Springer International Publishing, Cham Heidelberg (2015), `http://link.springer.com/book/10.1007/978-3-319-24202-6`

7. Cong, J., Yan, K.: Synthesis for FPGAs with embedded memory blocks. In: Proceedings of the 2000 ACM / SIGDA 8th International Symposium on FPGAs. pp. 75–82 (2000)
8. Czerwinski, R., Kania, D.: Area and speed oriented synthesis of FSMs for PAL-based CPLDs. Microprocessors & Microsystems 36(1), 45–61 (Feb 2012), http://dx.doi.org/10.1016/j.micpro.2011.06.004
9. Garcia-Vargas, I., Senhadji-Navarro, R., Jiménez-Moreno, G., Civit-Balcells, A., Guerra-Gutierrez, P.: ROM-based finite state machine implementation in low cost FPGAs. In: IEEE International Symposium on Industrial Electronics ISIE 2007. pp. 2342–2347. IEEE (2007)
10. Grout, I.: Digital systems design with FPGAs and CPLDs. Elsevier Science, Oxford (2008)
11. LGSynth93: International Workshop on logic synthesis benchmark suite (LGSynth93). TAR, Benchmarks test, Available at http://www.cbl.ncsu.edu:16080/benchmarks/LGSynth93/LGSynth93.tar (1993)
12. Micheli, G.D.: Synthesis and Optimization of Digital Circuits. McGraw–Hill (1994)
13. Nowicka, M., Łuba, T., Rawski, M.: FPGA-based decomposition of boolean functions. algorithms and implementation. In: Proc. of Sixth International Conference on Advanced Computer Systems. pp. 502–509 (1999)
14. Rawski, M., Selvaraj, H., Łuba, T.: An application of functional decomposition in ROM-based FSM implementation in FPGA devices. Journal of System Architecture 51(6–7), 423–434 (2005)
15. Rawski, M., Tomaszewicz, P., Borowski, G., Łuba, T.: Logic Synthesis Method of Digital Circuits Designed for Implementation with Embedded Memory Blocks on FPGAs. In: Adamski, M., Barkalov, A., Węgrzyn, M. (eds.) Design of Digital Systems and Devices. LNEE 79, pp. 121–144. Springer-Verlag, Berlin (2011)
16. Scholl, C.: Functional Decomposition with Application to FPGA Synthesis. Kluwer Academic Publishers, Boston (2001)
17. Sentowich, E., Singh, K., L.Lavango, Moon, C., Murgai, R., Saldanha, A., Savoj, H., P, P.S., Bryton, R., Sangiovanni-Vincentelli, A.: SIS: a system for sequential circuit synthesis. Tech. rep., University of California, Berkely (1992)
18. Sklyarov, V.: Synthesis and implementation of RAM-based finite state machines in FPGAs. In: Proceedings of Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing. pp. 718–728. Springer-Verlag, Villach (2000)
19. Sutter, G., Todorovich, E., López-Buedo, S., Boemo, E.: Low-power FSMs in FPGA: Encoding alternatives. In: Integrated Circuit Design. Power and Timing Modeling, Optimization and Simulation, pp. 363–370. Springer-Verlag (2002)
20. Tiwari, A., Tomko, K.: Saving power by mapping finite-state machines into Embedded Memory Blocks in FPGAs. In: Proceedings of the conference on Design, Automation and Test in Europe - Volume 2. pp. 916–921. IEEE Computer Society (2004)
21. Xilinx: http://www.xilinx.com, accessed: January, 2016