Int. J. Appl. Math. Comput. Sci., 2020, Vol. 30, No. 4, 745–759
DOI: 10.34768/amcs-2020-0055

IMPROVING CHARACTERISTICS OF LUT–BASED MEALY FSMS

ALEXANDER BARKALOV [a], LARYSA TITARENKO [a], KAMIL MIELCAREK [a,*]

[a] Institute of Metrology, Electronics and Computer Science
University of Zielona Góra
ul. Szafrana 2, 65-516 Zielona Góra, Poland
e-mail: `{A.Barkalov,L.Titarenko,K.Mielcarek}@imei.uz.zgora.pl`

Practically, any digital system includes sequential blocks represented using a model of finite state machine (FSM). It is very important to improve such FSM characteristics as the number of logic elements used, operating frequency and consumed energy. The paper proposes a novel technology-dependent design method targeting a decrease in the number of look-up table (LUT) elements and their levels in logic circuits of FPGA-based Mealy FSMs. It produces FSM circuits having three levels of logic blocks. Also, it produces circuits with regular systems of interconnections between the levels of logic. The method is based on dividing the set of internal states into two subsets. Each subset corresponds to a unique part of an FSM circuit. Only a single LUT is required for implementing each function generated by the first part of the circuit. The second part is represented by a multi-level circuit. The proposed method belongs to the group of two-fold state assignment methods. Each internal state is encoded as an element of the set of states and as an element of some of its subsets. A binary state assignment is used for states corresponding to the first part of the FSM circuit. The one-hot assignment is used for states corresponding to the second part. An example of FSM synthesis with the proposed method is shown. The experiments with standard benchmarks are conducted to analyze the efficiency of the proposed method. The results of experiments show that the proposed approach leads to diminishing the number of LUTs in the circuits of rather complex Mealy FSMs having more than 15 internal states. The positive property of this method is a reduction in energy consumption (without any overhead cost) and an increase in operating frequency compared with other investigated methods.

Keywords: FPGA, LUT, Mealy FSM, structural decomposition, two-fold state assignment, energy consumption.

1. Introduction

Various sequential blocks are widely used in modern digital systems (Gajski *et al.*, 2009; Micheli, 1994). Very often, sequential blocks are represented using models of finite state machines (FSMs) (Baranov, 2008; Micheli, 1994). For example, FSMs are used for implementing: (i) hardware-software interfaces of embedded systems (Gajski *et al.*, 2009), (ii) complex functions such as hyper-tangent and exponentiation functions (Brown and Card, 2001; Li *et al.*, 2014), (iii) activation functions in deep neural networks (Li *et al.*, 2017; Xie *et al.*, 2017), (iv) some blocks for integral stochastic computing (Ardakani *et al.*, 2017), (v) different stages of cascaded digital processing systems (Rafla and Gauba, 2010; Glaser *et al.*, 2011; Das and Priya, 2018). Also, they are used for the synthesis of control units of digital systems (Czerwiński and Kania, 2013; Sklyarov *et al.*, 2014; Baranov, 1994; Kubica *et al.*, 2019; Opara *et al.*, 2019; Kubica and Kania, 2017; Opara and Kania, 2010; Nowicka *et al.*, 1999; Barkalov and Barkalov Jr., 2005). An analysis of the literature (Sutter *et al.*, 2002; Cong and Yan, 2000; Sklyarov, 2000; Garcia-Vargas *et al.*, 2007; Tiwari and Tomko, 2004) shows that Mealy FSMs are used very often in logic design. Based on this analysis, we choose this model in our current research.

Nowadays, the field programmable gate arrays (FPGAs) are widely used for implementing FSM logic circuits (Maxfield, 2004; Grout, 2008). The majority of FPGAs are based on look-up table (LUT) elements connected with programmable flip-flops (Altera, 2020; Xilinx, 2015).

To compare outcomes of different FSM-based design methods, basically, three metrics are used. These are (i) the chip area occupied by an FSM circuit, (ii) the performance and (iii) the consumed energy (Barkalov *et al.*, 2020b; Czerwiński and Kania, 2013). In the case

---

[*] Corresponding author

of LUT-based FSMs, the chip area is proportional to the number of LUTs in the circuit. We use the number of LUTs to compare different solutions.

As a rule, the number of inputs S_L of a LUT is rather small ($S_L \leq 6$) (Altera, 2020; Xilinx, 2015). It leads to the necessity of functional decomposition for Boolean functions representing the FSM logic circuit (Scholl, 2001; Kam *et al.*, 2010; Nowicka *et al.*, 1999). It results in an increase of the number in layers of LUTs in the circuit and in complication for interconnections. In turn, it results in increasing the propagation time and power consumption (Sutter *et al.*, 2002; Wu *et al.*, 2000).

The functional decomposition (Rawski *et al.*, 2005; Michalski and Kokosiński, 2016) leads to multi-level FSM circuits with irregular connections, when the same variables appear at different levels of a circuit. This complicates the process of routing and may lead to an increase in the overall length of interconnections. In fact, this leads to an increase in power consumption (Sklyarov *et al.*, 2014). To make the interconnections more regular, it is possible to use methods of structural decomposition (Barkalov *et al.*, 2020b).

In this article, we propose a design method for hardware reduction in FPGA-based Mealy FSMs. The method is based on the joint use of two-fold state assignment (Barkalov *et al.*, 2020b) and one-hot state assignment (Kubatova and Becvar, 2002; Kołopieńczyk *et al.*, 2017). The *main contribution of the article* is a novel approach extending the scope of Mealy FSM synthesis methods based on two-fold state assignment (Barkalov *et al.*, 2018; 2020b). Our experiments with standard benchmarks (LGSynth93, 1993) show that this approach allows improving all three main characteristics of LUT-based Mealy FSMs compared with FSM circuits obtained using other known design methods.

2. Background of Mealy FSMs

A Mealy FSM is defined as the sextuple $S = (X, Y, A, \delta, \lambda, a_1)$ (Baranov, 2008; Micheli, 1994), where $X = \{x_1, \ldots, x_L\}$ is a finite set of inputs, $Y = \{y_1, \ldots, y_N\}$ is a finite set of outputs, $A = \{a_1, \ldots, a_M\}$ is a finite set of states, $\delta : A \times X \to A$ is the transition function, $\lambda : A \times X \to Y$ is the output function, $a_1 \in A$ is the “reset” state.

A Mealy FSM can be represented using different approaches. They include state transition graphs, state transition tables (Baranov, 1994; Micheli, 1994), binary decision diagrams (Opara *et al.*, 2019; Kubica and Kania, 2017), and-inversion graphs (Testa *et al.*, 2019; Mishchenko and Brayton, 2006; 2007) used in ABC systems (Brayton and Mishchenko, 2010). To explain our approach, we choose state transition tables (STT). This form of representation is the closest to the KISS2 format (Barkalov *et al.*, 2015) used in our investigations.

An STT includes the following columns (Baranov, 1994): a_m is the current state; a_s is the state of transition (a next state); X_h is a conjunction of inputs (or their compliments) determining a transition from a_m to a_s; h is a transition number ($h \in \{1, \ldots, H\}$). For example, the STT (Table 1) represents some Mealy FSM S_1.

Using Table 1, the following parameters of S_1 can be found: the number of inputs $L = 10$, the number of outputs $N = 11$, the number of states $M = 10$, the number of transitions $H = 23$.

When the set of states is constructed, the step of state assignment should be executed (Micheli, 1994). During this step, each state $a_m \in A$ is represented by its code $K(a_m)$ having R bits. The variables $T_r \in T$ are used for state assignment, where T is a set of state variables. The method of one-hot state assignment is very popular in the FPGA-based design of FSMs (Kubatova and Becvar, 2002). But very often, a binary state assignment is more preferable, in which

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

A special register (RG) is used to keep the state codes. It includes R flip-flops with mutual synchronization pulse *Clock* and mutual clearing pulse *Start*. As a rule, D flip-flops are used for implementing RGs (Baranov, 2008; Czerwiński and Kania, 2013). To change the content of the RG, input memory functions $D_r \in \Phi$ are used, where $\Phi = \{D_1, \ldots, D_R\}$.

Table 1. Structure table of the Mealy FSM corresponding to GSA Γ_1.

a_m	a_s	X_h	Y_h	h
a_1	a_2	1	y_1y_2	1
a_2	a_3	x_1x_2	y_1	2
	a_3	$x_1\bar{x}_2$	y_2y_3	3
	a_3	$\bar{x}_1x_3$	y_3y_4	4
	a_3	$\bar{x}_1\bar{x}_3$	y_5	5
a_3	a_4	x_3	y_6y_7	6
	a_4	$\bar{x}_3x_4$	y_3y_5	7
	a_3	$\bar{x}_3\bar{x}_4$	y_5	8
a_4	a_5	1	y_2y_3	9
a_5	a_6	x_7	y_3y_6	10
	a_3	$\bar{x}_7$	y_5	11
a_6	a_7	x_5	y_1y_7	12
	a_7	$\bar{x}_5$	y_8	13
a_7	a_8	x_6	y_7y_8	14
	a_7	$\bar{x}_6$	y_8	15
a_8	a_9	1	y_7	16
a_9	a_{10}	1	y_1	17
a_{10}	a_3	$x_3x_7x_8x_9x_{10}$	y_3y_4	18
	a_1	$x_2x_7x_8x_9\bar{x}_{10}$	y_7y_9	19
	a_5	$x_3x_7x_8\bar{x}_9$	$y_{10}y_{11}$	20
	a_8	$x_3x_7\bar{x}_8$	y_2y_3	21
	a_9	$x_3\bar{x}_7$	y_{11}	22
	a_1	$\bar{x}_3$	–	23

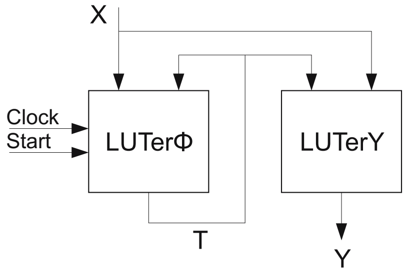

Fig. 1. Structural diagram of Mealy FSM U_1.

To design a logic circuit of a Mealy FSM, a structure table (ST) should be constructed (Baranov, 1994). It is an expansion of an STT by the columns with codes of current and next states ($K(a_m)$ and $K(a_s)$, respectively). Also, an ST includes a column Φ_h with symbols $D_r \in \Phi$ corresponding to 1's in the code $K(a_s)$ from the h-th row of ST ($h \in \{1,\dots,H\}$).

This table forms a basis for deriving functions

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

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

The functions (2) and (3) are used for implementing the FSM logic circuit.

3. State-of-the-art

The trivial structural diagram of Mealy FSM U_1 is shown in Fig. 1. Here the symbol LUTer determines a circuit implemented with LUTs.

In FSM U_1, the LUTerΦ implements the system (2), and the LUTerY the system (3). If a function D_r is generated as the output of some LUT, then this output is connected with a flip-flop. These flip-flops form a state register RG distributed among the logic elements. This explains the presence of pulses $Clock$ and $Start$ as inputs of LUTerΦ.

The process of FSM design has always been associated with the necessity of solving optimization problems (Micheli, 1994). As a rule, when designing FPGA-based FSMs, four basic optimization problems arise (Barkalov *et al.*, 2015; Khatri and Gulati, 2011). They are (i) a decrease in the chip area occupied by an FSM circuit (hardware reduction), (ii) the reduction in the signal propagation time (an increase in the clock frequency); (iii) reduction in power consumption, and (iv) an improvement of testability. In this article, we consider the first of these problems.

The main disadvantage of U_1 is the following: each function $D_r \in \Phi$ and $y_n \in Y$ could depend on up to $L + R$ arguments. The analysis of the library of LGSynth93 (LGSynth93, 1993) shows that for some benchmark FSMs we have $L + R \geq 20$. At the same time, for modern LUTs $S_L \leq 6$ (Altera, 2020; Xilinx, 2015). Thus, the following condition very often takes place:

$$L + R \gg S_L. \tag{4}$$

If (4) is satisfied for some FSM, then the problem of hardware reduction arises.

There are four main approaches to solving this problem, namely:

(a) optimal state assignment (Baranov, 2008; Micheli, 1994; Kam *et al.*, 2010),

(b) functional decomposition of Boolean functions representing an FSM circuit (Scholl, 2001; Nowicka *et al.*, 1999; Rawski *et al.*, 2005; Machado and Cortadella, 2020),

(c) the replacement of LUTs by embedded memory blocks (EMB) (Sklyarov *et al.*, 2014; Sutter *et al*, 2002; Cong and Yan, 2000; Sklyarov, 2000; Garcia-Vargas and Senhadji-Navarro, 2015; Tiwari and Tomko, 2004; Barkalov *et al.*, 2015; 2020a; Rawski *et al.*, 2011; Kołopieńczyki *et al.*, 2017),

(d) structural decomposition of an FSM circuit (Sklyarov *et al.*, 2014; Barkalov and Titarenko, 2009; Kołopieńczyk *et al.*, 2017; Barkalov *et al.*, 2020b).

We shall understand the optimal state assignment as a process of obtaining state codes allowing to reduce the number of arguments in functions (2) and (3). These functions are represented as sum-of-products (SOP) (Micheli, 1994). Each product term F_h is represented as

$$F_h = A_m X_h, \quad (h \in \{1,\dots,H\}). \tag{5}$$

In (5), the symbol A_m stands for the conjunction of state variables corresponding to the state code $K(a_m)$ from the h-th row of ST.

The number of bits in $K(a_m)$ can range from $\lceil \log_2 M \rceil$ to M. If $R = M$, it is a one-hot state assignment (Sutter *et al.*, 2002). When the one-hot method is used, only a single state variable forms a conjunction $A_m (m \in \{1,\dots,M\})$. It allows decreasing the number of arguments in terms (5). This leads to circuits with fewer LUTs and layers of logic than in the case of binary encoding. This approach is used in the ABC system by Berkeley (Brayton and Mishchenko, 2010; ABC System, 2020). The results of Sutter *et al.* (2002) show that one-hot is “attractive for large FSMs, but a better implementation of small machines can be obtained using binary encoding.” The results of investigations reported by Sklyarov (2000) show that binary encoding gives better results if $L > 10$.

One of the most popular state assignment algorithms is JEDI, which is distributed with the system SIS

(Sentowich *et al.*, 1992). It targets a multi-level logic implementation. It maximizes either the size of common cubes in logic functions (the input dominant algorithm) or the number of common cubes in a logic function (the output dominant algorithm).

Modern industrial packages use a lot of different state assignment strategies. For example, the following methods are used in the design tool XST of Xilinx (Xilinx, 2020a): the automatic state assignment, one-hot, compact, Gray codes, Johnson codes, speed encoding. Also, all these methods are implemented in the CAD tool Vivado of Xilinx (Vivado, 2020).

Therefore, there are a lot of state assignment methods. It is really difficult to say which is the best for a particular FSM.

Functional decomposition is very popular in the FSM design (Scholl, 2001; Rawski *et al.*, 2005; Rawski *et al.*, 2011; Machado and Cortadella, 2020). If the number of arguments for some function exceeds S_L, then the original function is broken down into smaller and smaller components. There are three basic approaches in this area: serial, parallel and balanced decompositions. In each step of the serial decomposition, the numbers of circuit levels and input-output delays are increasing. In the parallel decomposition, these characteristics are minimized. The balanced decomposition allows finding a solution which maximizes advantages and minimizes disadvantages of two previous strategies (Michalski and Kokosiński, 2016). These approaches are used, for example, in the systems DEMAIN (Rawski *et al.*, 1997) or PKmin (PKmin, 2020). Obviously, there are program tools for functional decomposition in any CAD targeting FPGA-based design.

Modern FPGAs have a lot of embedded memory blocks (Altera, 2020; Xilinx, 2015). Using EMBs allows for an improvement of main characteristics of FSM circuits (Sklyarov, 2000). Because of this, there are many design methods targeting EMB-based FSMs (Sklyarov *et al.*, 2014; Baranov, 1994; Cong and Yan, 2000; Sklyarov, 2000; Garcia-Vargas *et al.*, 2007; Tiwari and Tomko, 2004; Rawski *et al.*, 2011; Kołopieńczyk *et al.*, 2017; Barkalov *et al.*, 2020a; Borowik, 2018).

The EMBs have a property of configurability. This means that parameters such as the number of cells and their outputs could be changed by a designer (Grout, 2008). Typical configurations of EMBs are the following: 32K×1, 16K×2, 8K×4, 4K×8, 2K×16, 1K×32, 512×64, 256×128 (bits) (Altera, 2020; Xilinx, 2015). Thus, modern EMBs are very flexible and can be tuned to meet a particular FSM.

A survey of different approaches to EMB-based design can be found in the work of Garcia-Vargas and Senhadji-Navarro (2015). Let us point out that these methods could be used only if there are “free” EMBs, which are not used for implementation of other parts of a digital system.

In the case of structural decomposition, an FSM circuit is represented by several blocks (Barkalov *et al.*, 2020b). Each block implements additional functions different from (2) and (3). The methods of structural decomposition are characterized by the following: systems of additional functions are implemented as separate blocks of FSM circuits. Each block has its own inputs and outputs different from the inputs and outputs of other blocks. This allows obtaining FSM circuits with more regular interconnections than for their counterparts based on functional decomposition.

Let us point out that the methods of structural decomposition are not widely used in FPGA-based design. But we think that this approach has a good potential. They can be used together with methods of functional decomposition and resynthesis (Testa *et al.*, 2019; Mishchenko and Brayton, 2006; 2011). These three groups of methods complement each other. Using them together can improve the characteristics of FSM circuits.

In this article we propose a design method targeting LUT-based Mealy FSMs. The method is based on a structural decomposition of the FSM circuit. It is a technology-dependent method because it takes into account the number of LUT's inputs S_L.

The proposed method is an evolution of ideas from (Barkalov *et al.*, 2020b; 2018). We divide an initial FSM in two parts. The two-fold state assignment (Barkalov *et al.*, 2020b) is used in the first part. The one-hot state assignment and functional decomposition are used in the second part. This leads to a Mealy FSM U_2 discussed in the next section.

4. Main idea of the proposed method

Let a Mealy FSM S be presented by its STT. Encode states $a_m \in A$ by binary codes $K(a_m)$ having $R = \lceil \log_2 M \rceil$ bits. Transform the STT into an ST of Mealy FSM S.

Let $X(a_m) \subseteq X$ be a set of inputs determining transitions from a state $a_m \in A$. Represent the set A as $A_0 \cup A_R$ where $A_0 \cap A_R = \emptyset$. If $|X(a_m)|$ is less than S_L, then $a_m \in A_R$; otherwise, $a_m \in A_0$

Encode states $a_m \in A_0$ by one-hot codes $C_0(a_m)$ having $R_0 = |A_0|$ bits. Use variables $\beta_r \in B = \{\beta_1, \ldots, \beta_{R_0}\}$ to encode these states.

Construct a partition $\Pi_A = \{A^1, \ldots, A^I\}$ of the set A_R such that the following condition takes place:

$$R_i + L_i \le S_L \quad (i \in \{1, \ldots, I\}). \tag{6}$$

In (6), the symbol R_i stands for the number of state variables necessary to encode the states $a_m \in A^i$, the symbol L_i is the number of inputs $x_e \in X^i$ determining transitions from the states $a_m \in A^i$.

Encode the states $a_m \in A^i$ by binary codes $C_R(a_m)$ having R_i bits:

$$R_i = \left\lceil \log_2 \left(|A^i| + 1 \right) \right\rceil \quad (i \in \{1, \dots, I\}). \tag{7}$$

Use the state variables $\tau_r \in \tau = \{\tau_1, \dots, \tau_{R_A}\}$ to encode the states $a_m \in A^i$. The following relation takes place: $R_A = R_1 + R_2 + \dots + R_I$.

The set A_0 determines a subtable ST_0 of the initial ST. Each class $A^i \in \Pi_A$ determines a subtable ST_i of the initial ST. Using tables ST_0–ST_I, we can find sets $X^i \subseteq X$ (inputs written in the column X_h^i), $Y^i \subseteq Y$ (outputs written in the column Y_h^i) and $\Phi^i \subseteq \Phi$ (input memory functions written in the column Φ_h^i).

Each state $a_m \in A$ has two state codes. The code $K(a_m)$ identifies the state a_m as an element of the set A. The code $C_0(a_m)$ identifies the state a_m as an element of the set A_0, the code $C_R(a_m)$ as an element of the set A_R.

Each subtable ST_i corresponds to a block LUTeri. From (6) it follows that it is sufficient to have only a single LUT having S_L inputs for implementing any function $D_r \in \Phi^i$ and $y_n \in Y^i (i \in \{1, \dots, I\})$.

Based on this preliminary information, we propose the structural diagram of Mealy FSM U_2 (Fig. 2).

The block LUTeri implements functions

$$Y^i = Y^i(\mathcal{T}^i, X^i) \quad (i \in \{1, \dots, I\}), \tag{8}$$

$$\Phi^i = \Phi^i(\mathcal{T}^i, X^i) \quad (i \in \{1, \dots, I\}). \tag{9}$$

The block LUTer0 implements functions

$$Y^0 = Y^0(B, X^0), \tag{10}$$

$$\Phi^0 = \Phi^0(B, X^0). \tag{11}$$

In (8) and (9), the symbol τ^i stands for the subset of $\mathcal{T}$ whose variables are used to create codes $C_R(a_m)$, where $a_m \in A^i$.

The block LUTerOR generates outputs $y_n \in Y$ and state variables $T_r \in T$. This block includes the distributed register RG keeping state codes $K(a_m)$. As a result, pulses $Start$ and $Clock$ enter the LUTerOR.

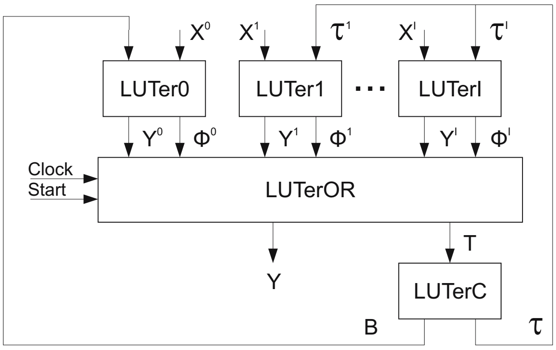

Fig. 2. Structural diagram of Mealy FSM U_2.

The block LUTerC transforms state codes $K(a_m)$ into state codes $C_0(a_m)$ and $C_R(a_m)$. As a result, variables $\tau_r \in \mathcal{T}$ and $\beta_r \in B$ are generated. This means that the LUTerC implements the functions:

$$\mathcal{T}_r = \mathcal{T}_r(T) \quad (r \in \{1, \dots, R_A\}), \tag{12}$$

$$\beta_r = \beta_r(T) \quad (r \in \{1, \dots, R_0\}). \tag{13}$$

At each time instant, only a single LUTeri is "active." It means that there are ones on some outputs of this block. There are only zeros on outputs of other blocks. These blocks are "idle." Use the codes $C_r(a_m)$ with all zeros to show that a block is idle. This explains the presence of 1 in (7)

The analysis of FSM U_2 (Fig. 2) shows that its circuit has the following specifics. Firstly, it has exactly three levels of logic blocks. Secondly, each level of blocks has its own unique input and output variables. The variables $x_e \in X$, $\tau_r \in \mathcal{T}$ and $\beta_r \in B$ enter only blocks of the first level. Their outputs enter only the block LUTerOR. The same is true for the pulses $Start$ and $Clock$. The variables $T_r \in T$ (outputs of LUTerOR) enter only the third level of the FSM circuit. In addition, state variables $\tau_r \in \mathcal{T}^i$ enter only the block LUTeri ($i = \overline{1, I}$). Thus, the proposed approach leads to LUT-based FSM circuits with regular systems of interconnections.

The condition (6) is violated for states $a_m \in A_0$. Therefore, the circuit of block LUTer0 includes more than a single level of logic. To implement the circuit of LUTer0, it is necessary to apply methods of functional decomposition.

Let the symbol $U_i(S_j)$ mean that: (i) an FSM S_j is represented by an STT and (ii) the model U_i is used to synthesize an FSM circuit. In this article, we propose a design method for Mealy FSM $U_2(S_j)$. The method includes the following steps:

1. Finding the set A from the initial STT. Partitioning the set A into sets A_0 and A_R using the value of S_L and sets $X(a_m) \subseteq X$.
2. Encoding of states $a_m \in A$ by codes $K(a_m)$.
3. Constructing the structure table of Mealy FSM S_j. This step is executed using the rules from (Baranov, 1994).
4. Constructing the partition Π_A for the set A_R.
5. Executing the state encoding for states $a_m \in A^i$ ($i \in \{1, \dots, I\}$).
6. Constructing subtables ST_i for states $a_m \in A^i$ ($i \in \{1, \dots, I\}$). The subtables are extracted from the structure table of the Mealy FSM.
7. Deriving systems (8) and (9) from subtables ST_i representing blocks LUTer1–LUTerI.

8. Design of the block LUTer0:
    (a) Executing the state encoding for states $a_m \in A_0$.
    (b) Constructing a subtable ST_0 using states $a_m \in A_0$ and the structure table of Mealy FSM.
    (c) Deriving systems (10) and (11) from the subtable ST_0.
    (d) Implementing the circuit of LUTer0.
9. Constructing the systems representing LUTerOR.
10. Constructing the table of LUTerC and deriving the systems (12) and (13).
11. Implementing the FSM circuit with particular LUTs.

We discuss this method in detail in Sections 5 and 6. We use LUTs with $S_L = 5$. In Section 5 we show how to use this method for the synthesis of Mealy FSM $U_2(S_1)$. In Section 6 we discuss how to construct the partition Π_A with a minimum number of classes.

5. Example of synthesis

As can be seen from the STT (Table 1), the set A includes $M = 10$ elements. Transitions from states $a_1, \ldots, a_9 \in A$ depend on up to 3 inputs. Transitions from the state $a_{10} \in A$ depend on 5 inputs. Because $S_L = 5$, the set A_R includes all states except the state a_{10}. It gives the sets $A_R = \{a_1, \ldots, a_9\}$ and $A_0 = \{a_{10}\}$.

Using (1) gives $R = 4$, $T = \{T_1, \ldots, T_4\}$ and $\Phi = \{D_1, \ldots, D_4\}$. Encode the states $a_m \in A$ in the trivial way: $K(a_1) = 0000$, $K(a_2) = 0001, \ldots, K(a_{10}) = 1001$. Now, we can construct the structure table of FSM S_1 (Table 2). To this end, we use Table 1 and the rules from (Baranov, 1994).

Step 4 is the most important stage of the proposed design method. It determines the hardware amount in the resulting circuit. We discuss this step in the next section. In this section, we just use a partition $\Pi_A = \{A^1, A^2, A^3\}$ with classes $A^1 = \{a_1, a_2, a_4\}$, $A^2 = \{a_3, a_5, a_8\}$ and $A^3 = \{a_6, a_7, a_9\}$.

Using Table 2, the following sets could be found: $X^1 = \{x_1, x_2, x_3\}$, $Y^1 = \{y_1, \ldots, y_5\}$, $\Phi^1 = \{D_2, D_3, D_4\}$, $X^2 = \{x_3, x_4, x_7\}$, $Y^2 = \{y_3, y_5, y_6, y_7\}$, $\Phi^2 = \{D_1, \ldots, D_4\}$, $X^3 = \{x_5, x_6\}$, $Y^3 = \{y_1, y_7, y_8\}$, $\Phi^3 = \{D_2, D_3, D_4\}$.

Using (7), we can find $R_1 = R_2 = R_3 = 2$. This yields $R_0 = 6$ and $\mathcal{T} = \{\tau_1, \ldots, \tau_6\}$. Let $\mathcal{T}^1 = \{\tau_1, \tau_2\}, \mathcal{T}^2 = \{\tau_3, \tau_4\}$ and $\mathcal{T}^3 = \{\tau_5, \tau_6\}$. For each class $A^i \in \Pi_A$, the condition (6) takes place.

Obviously, there is no influence of the outcome of state encoding on the hardware amount in blocks LUTer1–LUTer3. Therefore, we can encode the states in the following way: $C_R(a_1) = C_R(a_3) = C_R(a_6) = 01$, $C_R(a_2) = C_R(a_5) = C_R(a_7) = 10$ and $C_R(a_4) = C_R(a_8) = C_R(a_9) = 11$.

To construct subtables $ST_i (i \in \{1, 2, 3\})$, we should (i) take the corresponding rows of ST and (ii) replace codes $K(a_m)$ by codes $C_R(a_m)$. For example, the LUTer1 is represented by Table 3. To construct Table 3, we use rows 1–5 and 9 of the structure table (Table 2). The superscript 1 in Table 3 means that the corresponding functions are generated by LUTer1. The subtables ST_2 and ST_3 are constructed in the same manner.

Using Table 3, the equations for functions $y^1_n \in Y^1$ and $D^1_r \in \Phi^1$ can be found. In these equations, the conjunctions A_m in (5) depend on variables $\tau_r \in \mathcal{T}^1$. For example, the following equations can be derived from Table 3: $y^1_3 = \tau_1\bar{\tau}_2 x_1 \bar{x}_2 \vee \tau_1\tau_2$; $D^1_3 = \tau_1\bar{\tau}_2$. Acting in the same manner, it is possible to find all functions (8) and (9).

The presence of Step 8 is the main difference between the proposed approach and our previous methods (Barkalov *et al.*, 2018; Barkalov *et al.*, 2020b). Let us discuss this step for a given example.

We have $A_0 = \{a_{10}\}$. This gives $R_0 = 1$ and the set $B = \{\beta_1\}$. Using Table 2, we can get sets $X^0 = \{x_3, x_7, \ldots, x_{10}\}$, $Y^0 = \{y_2, y_3, y_4, y_7, y_9, y_{10}, y_{11}\}$ and $\Phi^0 = \Phi$. The state encoding is trivial in the discussed case: $C_0(a_{10}) = 1$.

To construct the subtable ST_0, we should use rows 11-23 of the ST (Table 2). Replacing $K(a_{10}) = 1001$

Table 2. Structure table of Mealy FSM S_1.

a_m	$K(a_m)$	a_s	$K(a_s)$	X_h	Y_h	Φ_h	h
a_1	0000	a_2	0001	1	y_1y_2	D_4	1
a_2	0001	a_3	0010	x_1x_2	y_1	D_3	2
		a_3	0010	$x_1\bar{x}_2$	y_2y_3	D_3	3
		a_3	0010	$\bar{x}_1x_3$	y_3y_4	D_3	4
		a_3	0010	$\bar{x}_1\bar{x}_3$	y_5	D_3	5
a_3	0010	a_4	0011	x_3	y_6y_7	D_3D_4	6
		a_4	0011	$\bar{x}_3x_4$	y_3y_5	D_3D_4	7
		a_3	0010	$\bar{x}_3\bar{x}_4$	y_5	D_3	8
a_4	0011	a_5	0100	1	y_2y_3	D_2	9
a_5	0100	a_6	0101	x_7	y_3y_6	D_2D_4	10
		a_3	0010	$\bar{x}_7$	y_5	D_3	11
a_6	0101	a_7	0110	x_5	y_1y_7	D_2D_3	12
		a_7	0110	$\bar{x}_5$	y_8	D_2D_3	13
a_7	0110	a_8	0111	x_6	y_7y_8	$D_2D_3D_4$	14
		a_7	0110	$\bar{x}_6$	y_8	D_2D_3	15
a_8	0111	a_9	1000	1	y_7	D_1	16
a_9	1000	a_{10}	1001	1	y_1	D_1D_4	17
a_{10}	1001	a_3	0010	$x_3x_7x_8x_9x_{10}$	y_3y_4	D_3	18
		a_1	0000	$x_3x_7x_8x_9\bar{x}_{10}$	y_7y_9	–	19
		a_5	0100	$x_3x_7x_8\bar{x}_9$	$y_{10}y_{11}$	D_2	20
		a_8	0111	$x_3x_7\bar{x}_8$	y_2y_3	$D_2D_3D_4$	21
		a_9	1000	$x_3\bar{x}_7$	y_{11}	D_1	22
		a_1	0000	$\bar{x}_3$	–	–	23

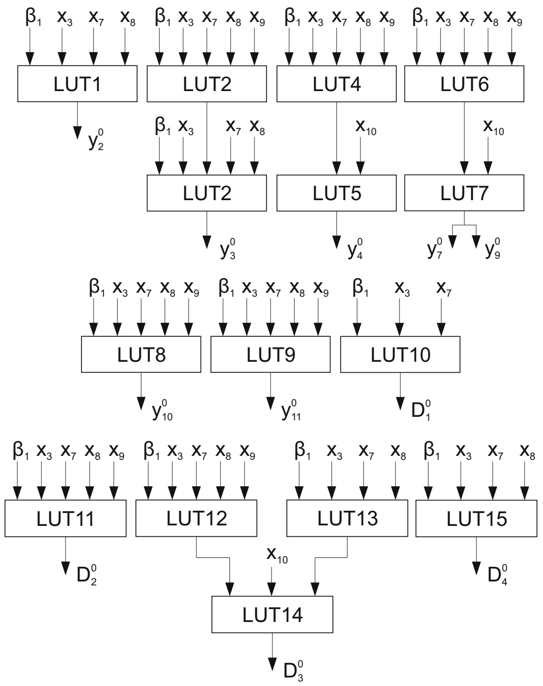

Fig. 3. Logic circuit of LUTer0.

by $C_0(a_{10}) = 1$ gives ST_0 (Table 4). We hope that the connection between Tables 2 and 4 is transparent.

Using Table 4, we can derive the following system of Boolean functions:

$$\begin{aligned}
y_2^0 &= \beta_1 x_3 x_7 \bar{x}_8,\\
y_3^0 &= \beta_1 x_3 x_7 x_8 x_9 x_{10} \vee \beta_1 x_3 x_7 \bar{x}_8,\\
y_4^0 &= \beta_1 x_3 x_7 x_8 x_9 x_{10},\\
y_7^0 &= \beta_1 x_3 x_7 x_8 x_9 \bar{x_{10}} = y_9^0,\\
y_{10}^0 &= \beta_1 x_3 x_7 x_8 \bar{x}_9,\\
y_{11}^0 &= \beta_1 x_3 x_7 x_8 \bar{x}_9 \vee \beta_1 x_3 \bar{x}_7,\\
D_1^0 &= \beta_1 x_3 \bar{x}_7,\\
D_2^0 &= \beta_1 x_3 x_7 x_8 \bar{x}_9 \vee \beta_1 x_3 x_7 \bar{x}_8,\\
D_3^0 &= \beta_1 x_3 x_7 x_8 x_9 x_{10} \vee \beta_1 x_3 x_7 \bar{x}_8,\\
D_4^0 &= \beta_1 x_3 x_7 \bar{x}_8.
\end{aligned} \tag{14}$$

We have $S_L = 5$ for our example. There are 6 arguments in terms corresponding to rows 1 and 2 of Table 4. Thus, it is necessary more than a single LUT to implement circuits for functions from columns Y^0 and Φ^0 written in rows 1 and 2. This means that two levels of logic are necessary in circuits for functions $y_3^0, y_4^0, y_7^0, y_9^0$ and D_3^0. Only a single LUT is sufficient to implement each of functions $y_2^0, y_{10}^0, y_{11}^0, D_1^0, D_2^0$ and D_4^0.

To implement multi-level circuits, the methods of functional decomposition should be used. This results in the following transformed equations of the system (14):

$$\begin{aligned}
y_3^0 &= (\beta_1 x_3 x_7 x_8 x_9) x_{10} \vee \beta_1 x_3 x_7 \bar{x}_8,\\
y_4^0 &= (\beta_1 x_3 x_7 x_8 x_9) x_{10},\\
y_7^0 = y_9^0 &= (\beta_1 x_3 x_7 x_8 x_9) \bar{x_{10}},\\
D_3^0 &= (\beta_1 x_3 x_7 x_8 x_9) x_{10} \vee (\beta_1 x_3 x_7 x_8).
\end{aligned} \tag{15}$$

The circuit of LUTer0 is shown in Fig. 3. It includes 15 LUTs with $S_L = 5$. The circuit is based on (14) and (15).

As can be seen from Fig. 2, functions generated by LUTer0 are used as arguments of functions generated by LUTerOR. But functions y_9–y_{11} are generated only by LUTer0. This means that the circuit of LUTerOR does not contain LUTs with outputs y_9–y_{11}. Only outputs y_1–y_8 are generated by LUTerOR.

To find equations representing LUTerOR, it is necessary to analyse sets Y^i and Φ^i. For example, we have the relation $y_1 \notin Y^0 \cup Y^2$, so that $y_1 = y_1^1 \vee y_1^3$. Next, $D_1 \notin \Phi^1 \cup \Phi^3$. Consequently, $D_1 = D_1^0 \vee D_1^2$.

Acting in the same way, it is possible to find equations for all functions generated by LUTerOR. If a function D_r is generated by some LUTeri, then pulses $Clock$ and $Start$ should be connected with the corresponding LUT of LUTeri.

The table of LUTerC has M rows. It includes columns $a_m, K(a_m), C_0(a_m), C_R(a_m), B_m, \tau_m, m$. The meaning of these columns is clear from Table 5.

Using Table 5, we can find equations (12) and (13). For example, we can find that $\beta_1 = T_1 T_4$, and $\tau_1 = \bar{T}_1 \bar{T}_2 T_4$. To get these equations, we use some rules of minimizing Boolean functions (Micheli, 1994).

To implement an FSM circuit, it is necessary to use standard CAD tools (Altera, 2020; Xilinx, 2015). They

Table 3. Table ST_1 of Mealy FSM $U_2(S_1)$.

a_m	$C_R(a_m)$	a_s	$K(a_s)$	X_h^1	Y_h^1	Φ_h^1	h
a_1	01	a_2	0001	1	$y_1^1 y_2^1$	D_4^1	1
a_2	10	a_3	0010	$x_1 x_2$	y_1^1	D_3^1	2
		a_3	0010	$x_1 \bar{x}_2$	$y_2^1 y_3^1$	D_3^1	3
		a_3	0010	$\bar{x}_1 x_3$	$y_3^1 y_4^1$	D_3^1	4
		a_3	0010	$\bar{x}_1 \bar{x}_3$	y_5^1	D_3^1	5
a_4	11	a_5	0100	1	$y_2^1 y_3^1$	D_2^1	6

Table 4. Table ST_0 of Mealy FSM $U_2(S_1)$.

a_m	$C_0(a_m)$	a_s	$K(a_s)$	X_h^0	Y_h^0	Φ_h^0	h
a_{10}	1	a_3	0010	$x_3 x_7 x_8 x_9 x_{10}$	$y_3^0 y_4^0$	D_3^0	1
		a_1	0000	$x_3 x_7 x_8 x_9 \bar{x_{10}}$	$y_7^0 y_9^0$	–	2
		a_5	0100	$x_3 x_7 x_8 \bar{x}_9$	$y_{10}^0 y_{11}^0$	D_2^0	3
		a_8	0111	$x_3 x_7 \bar{x}_8$	$y_2^0 y_3^0$	$D_2^0 D_3^0 D_4^0$	4
		a_9	1000	$x_3 \bar{x}_7$	y_{11}^0	D_1^0	5
		a_1	0000	$\bar{x}_3$	–	–	6

Table 5. Table of LUTerC for Mealy FSM $U_2(S_1)$

a_m	$K(a_m)$	$C_0(a_m)$	$C_R(a_m)$			B_m	τ_m	m
a_1	0000	0	01	00	00	–	τ_2	1
a_2	0001	0	10	00	00	–	τ_1	2
a_3	0010	0	00	01	00	–	τ_4	3
a_4	0011	0	11	00	00	–	$\tau_1\tau_2$	4
a_5	0100	0	00	10	00	–	τ_3	5
a_6	0101	0	00	00	01	–	τ_6	6
a_7	0110	0	00	00	10	–	τ_5	7
a_8	0111	0	00	11	00	–	$\tau_3\tau_4$	8
a_9	1000	0	00	00	11	–	$\tau_5\tau_6$	9
a_{10}	1001	1	00	00	00	β_1	–	10

form bit-streams for each LUT. Also, they execute the technology mapping of FSM circuit. We do not discuss this step for our example.

6. Constructing partition for a set of states

We should find a partition Π_A of the set A_R with a minimum number of blocks I and such that restriction (6) takes place for each block $A^i \in \Pi_A$. To solve this problem, we propose a simple sequential algorithm for finding a partition Π_A.

Each state $a_m \in A$ is characterized by two sets. The set $X(a_m)$ includes input variables determining transitions from state $a_m \in A$. The set $Y(a_m)$ includes outputs generated during transitions from state $a_m \in A$. If $a_m \in A^i$, then $X(a_m) \subseteq X^i$ and $Y(a_m) \subseteq Y^i$.

We use two evaluations to find the partition. The first of them determines how many new input variables will be added to the set X^i due to including the state a_m into the class $A^i \in \Pi_A$. The second of them determines the number of outputs common for both sets $Y(a_m)$ and Y^i. Let us denote these evaluations by the symbols $N(a_m, X^i)$ and $N(a_m, Y^i)$, respectively. They are calculated as follows:

$$N(a_m, X^i) = |X(a_m) \setminus X^i|, \tag{16}$$

$$N(a_m, Y^i) = |Y(a_m) \cap Y^i|. \tag{17}$$

In (16), the symbol "\" means the subtraction of sets.

Each block $A^i \in \Pi_A$ is generated in two stages. At the first stage, we take the state $a_m \in A^*$ as a basic element (BE) of A^i. Here A^* is a set of states which were not distributed after forming the block $A^{i-1} \in \Pi_A$. The BE should satisfy to the following relation:

$$|X(a_m)| = \max |X(a_j)|, \quad a_j \in A^* \setminus \{a_m\}. \tag{18}$$

If condition (18) takes place for states a_m and a_s, we will choose the state a_m if $m > s$.

The second stage is a multistep one. At each step, the next state is successively added to the block A^i in accordance with the rules given below. The process of forming block A^i is terminated when all states are already distributed among the blocks or when it is not possible to include any state in A^i without violation of (6).

There are the following rules for including the next successive state in A^i. Let A^* include all unallocated states $a_m \in A$. Choose all states $a_m \in A^*$ whose inclusion into A^i does not violate the restriction (6). Place them in a set $P(A^i)$. Select a state $a_m \in P(A^i)$ with the minimum value of evaluation (16). It is the first rule.

If there are more than one such state, then choose a state having maximum value of evaluation (17). If more than one state has such a property, then one of them is included into A^i. Next, all elements are eliminated from $P(A^i)$. It is the second rule.

Let us discuss an example of forming the partition Π_A for a Mealy FSM represented by an STT (Table 1). The process is shown in Table 6. We assume that $S_L = 5$. Hence, the following pairs $\langle L_i, R_i \rangle$ are possible: $\langle 0, 5\rangle, \langle 1, 4\rangle, \langle 2, 3\rangle, \langle 3, 2\rangle, \langle 4, 1\rangle$.

Let us explain the columns of Table 6. The column a_m contains states of the FSM. There are numbers of input variables in column $|X(a_m)|$. The columns $BE_i \, (i = 1, 2, 3)$ contain basic elements for Step i. The symbol "I" stands for $N(a_m, X^i)$, the symbol "II" for $N(a_m, Y^i)$. The sign $\oplus$ means that the state in the corresponding row is included in the set A^i. The sign "–" means that $a_m \notin A^*$, where a_m is the state from the corresponding row. There are states $a_m \in A^i$ in the row A^i. They are shown in the order of selecting.

As can be seen from Table 6, the process of selection includes 9 steps. As a result, we obtain the following partition $\Pi_A = \{A^1, A^2, A^3\}$ with $I = 3$ blocks: $A^1 = \{a_1, a_2, a_4\}, A^2 = \{a_3, a_5, a_8\}, A^3 = \{a_6, a_8, a_9\}$. As can be seen, this partition is the same as that used in Section 5.

Using the first rule allows decreasing the number of the same variables $x_e \in X$ in different blocks $A^i \in \Pi_A$. In turn, this leads to a decrease in the number of LUTs compared with the situation when the inputs $x_e \in X$ are duplicated in different blocks $A^i \in \Pi_A$. This also leads to further regularization of the system of interconnections.

Using the second rule allows decreasing the number of the same variables $y_n \in Y$ in different blocks $A^i \in \Pi_A$. It could lead to a decrease in the number of LUTs compared with the situation when the outputs $y_n \in Y$ are duplicated in different blocks $A^i \in \Pi_A$. This also reduces the number of interconnections between blocks LUTeri $(i = \overline{1, I})$ and the block LUTerOR. Obviously, the fewer such interblock connections, the more likely is that there is only a single level of LUTs in the circuit of LUTerOR.

Table 6. Forming of partition Π_A.

a_m	$X(a_m)$	BE_1	I/II		BE_2	I/II		BE_3	I/II	
			1	2		1	2		1	2
a_1	0		0/2$\oplus$	–		–	–		–	–
a_2	3	$\oplus$	–	–		–	–		–	–
a_3	2		1/2	1/2	$\oplus$	–	–		–	–
a_4	0		0/2	0/2$\oplus$		–	–		–	–
a_5	1		1/2	1/2		1/3 $\oplus$	–		–	–
a_6	1		1/1	1/1		1/1	1/1	$\oplus$	–	–
a_7	1		1/0	1/0		1/1	1/1		1/2$\oplus$	–
a_8	0		0/0	0/0		0/1	0/1$\oplus$		–	–
a_9	0		0/1	0/1		0/0	0/0		0/0	0/0$\oplus$
A^i		a_2	a_1	a_4	a_3	a_5	a_8	a_6	a_8	a_9

7. Experimental results

To investigate the efficiency of the proposed method, we use standard benchmarks from the LGSynth93 library (LGSynth93, 1993). It includes 48 benchmarks appearing in the practice of FSM design. These benchmarks are Mealy FSMs presented in the KISS2 format.

To work with these benchmarks, we used the CAD tool named K2F. It translates the KISS2 file into a VHDL model of an FSM. To synthesize and simulate the FSM, we use the Active-HDL environment. To get the FSM circuit, we use Xilinx CAD tool Vivado 2019.1 (Vivado, 2020). The investigation path used in our system is shown in Fig. 4.

The target platform was the FPGA device Xilinx Virtex-7 (XC7VX690tffg1761-2, Virtex-7 VC709 Evaluation platform). It includes LUTs having $S_L = 6$.

We compared our approach with four other methods: (i) Auto of Vivado 2019.1, (ii) One-hot of Vivado 2019.1, (iii) JEDI, (iv) DEMAIN. In all these cases, the model of U_1 (Fig. 1) is used. The results of experiments are shown in Table 7 (for the number of LUTs), Table 8 (for the operating frequency), and Table 9 (for the consumed energy).

All tables are organized in the same order. Their rows are marked with the names of benchmarks, the columns with design methods. The rows "Total" include results of summation for the corresponding values. We have included the summarized characteristics of U_2 as 100%. The rows "Percentage" show the percentage of the summarized characteristics with respect to the benchmarks synthesized as U_2.

As can be seen from Table 7, the proposed method allows minimizing the number of LUTs in U_2-based circuits in comparison with other investigated methods. There is the following savings: (i) 34.7% regarding Auto, (ii) 58.2% regarding to the One-hot, (iii) 12.8% regarding the JEDI-based FSMs and (iv) 20.4% regarding DEMAIN.

The following conclusion can be made. Our approach gives better results for FSMs having more than 15 states. If $M < 15$, then JEDI-based FSMs require fewer LUTs. DEMAIN sometimes produces better circuits than JEDI (for rather simple FSMs).

In all investigated cases, our approach produces FSM circuits having exactly three levels of logic. This is because

$$R \leq S_L = 6 \tag{19}$$

for all benchmarks from LGSynth93 (LGSynth93, 1993).

Our approach allows obtaining FSM circuits with more regular interconnections than for other investigated methods. Accordingly, U_2-based FSMs yield better results for both the operating frequency (Table 8) and the power consumption (Table 9).

As can be seen from Table 8, our approach gives the following gain in the operating frequency: (i) 12% in comparison with both Auto and DEMAIN, (ii) 11% in comparison with One-hot, (iii) 5% in comparison with JEDI-based FSMs. Table 8 demonstrates that the following gains in the consumed energy: (i) 45.7% in comparison with Auto, (ii) 52.7% in comparison with One-hot, (iii) 12.4% in comparison with JEDI and (iv) 16.8% in comparison with DEMAIN.

Let us point out that reducing power consumption, in our case, is not associated with additional overhead costs. The known methods in this area are connected to: (i) the representation of the initial FSM as a network of interacting automata (Chow *et al.*, 1996; Liu *et al.*, 2005), or (ii) the special state assignment (Benini and De Micheli, 1995; Benini *et al.*, 2001; Agrawal *et al.*, 2019), or (iii) the clock gating (Nag *et al.*, 2018) or (iv)

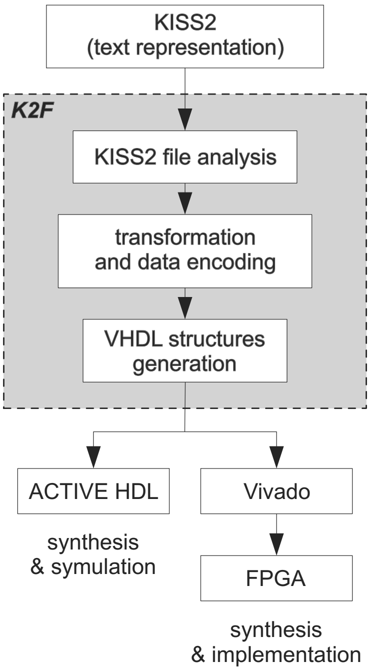

Fig. 4. Typical investigation path based on the K2F tool.

the power gating (Pradhan *et al.*, 2011; Choudhury and Pradhan, 2012; Benini *et al.*, 2000). We do not discuss these methods in detail. Note only that they are associated with the use of additional circuits and the introduction of a delay in the clock cycle. Our method is free from these drawbacks.

As can be seen from Tables 8 and 9, our approach gives better results for FSM, with $M > 15$. For simpler FSMs, better results are produced either by JEDI or DEMAIN. Of course, all these conditions are valid only for the benchmarks (LGSynth93, 1993) and the device XC7VX690tffy1761-2. It is almost impossible to make similar conclusions for the general case. However, it follows from our experiments that our approach gives good results for the cases when (i) the condition (19) takes place and (ii) the number of FSM states exceeds 15.

This conclusion is supported by comparison of our approach with some other methods when the Virtex 5 family of Xilinx is used. In this case, we used the benchmarks (LGSynth93, 1993), the device XC5VLX30FF324 and the Xilinx ISE 14.1 package (Xilinx, 2020b) instead of Vivado. We compared our approach with U_1-based FSMs implemented using: (i) Auto of ISE, (ii) One-hot of ISE, (iii) JEDI and (iv) DEMAIN.

As it is for Vivado-based research, our approach allows minimizing the number of LUTs in FSM circuits in comparison with other investigated methods. We got the following savings: (i) 23% in comparison with Auto, (ii) 29% in comparison with One-hot, (iii) 9% in comparison with JEDI and (iv) 14% in comparison with DEMAIN. Again our approach gives better results for FSMs having $M > 15$. The same is true for the operating frequency.

Next, we compared FSMs U_2 and PY Mealy FSMs based on two-fold state assignment and encoding of collections of outputs (Barkalov *et al.*, 2018).

The comparison was performed using the CAD tool Vivado 2019.1, standard benchmarks (LGSynth93, 1993) and the FPGA device XC7VX690tffy1761-2 by Xilinx (Virtex 7). The results of comparison showed that both approaches lead to FSM circuits with better characteristics than for Auto, One-hot, JEDI and DEMAIN. At the same time, the PY-based FSMs had better characteristics for numbers of LUTs and consumed energy. However, the U_2-based Mealy FSMs had better performance.

To investigate the effect of the number of inputs (S_L) on the efficiency of the proposed method, we use the FPGA chips of Virtex-4 (Xilinx, 2010). To get the FSM circuits, we use Xilinx CAD tool ISE 14.1 (Xilinx, 2020b). The chip XC4VLX40FF668-12 was used at this stage. It includes LUTs having $S_L = 4$.

We compared our approach with three other methods: Auto of ISE 14.1, One-hot of ISE 14.1, PY Mealy FSM (Barkalov *et al.*, 2018). We used the model U_1 (Fig. 1) to get the results for Auto and One-hot. The following conclusion can be made on the base of these investigations.

The method of Barkalov *et al.* (2018) cannot be used for the most complex benchmarks of the LGsynth93 library (ex1, keyb, kirkman, planet, planet1, s1488, s1494, s208, sand, s420, s510, s820, s832). This means that the condition (6) is violated for these benchmarks. Our approach produced circuits for all benchmarks. Therefore, the proposed approach allows using two-fold state assignment (Barkalov *et al.*, 2020b) for any Mealy FSM. The proposed method is free from the limitations inherent to the method (Barkalov *et al.*, 2020b).

As it is for circuits based on LUTs with $S_L = 6$, our approach allows minimizing the number of LUTs with $S_L = 4$ in comparison with other investigated methods. But if $S_L = 4$, then one-hot-based FSMs and U_2-based FSMs have almost the same number of LUTs for complex benchmarks having more than either 15 states or 10 inputs. The library LGSynth includes 23 benchmarks with such characteristics.

We think that this phenomenon is related to the following: the more states and inputs an FSM has, the more states the class A_0 includes. We use the one-hot state assignment for states $a_m \in A_0$. Therefore, as the ratio of $|A_0|$ to M grows, more states have one-hot codes.

8. Conclusion

The paper presents an original approach targeting FPGA-based Mealy FSMs. The proposed design method leads to FSM circuits having exactly three levels of logic and regular interconnections between these levels. It is based on the representation of an FSM as two interconnected parts. The first of these parts is synthesized based on the states for which $|X(a_m)| < S_L$. The second part is synthesized based on states for which $|X(a_m)| \geq S_L$. This representation allows overcoming the main drawback of the methods (Barkalov *et al.*, 2020b), which cannot be used if there is at least a single state violating the condition (6). In consequence, the proposed method can be applied to arbitrary Mealy FSMs.

The experiments clearly show that our approach leads to a reduction in the number of LUTs in comparison with circuits obtained by Xilinx Vivado 2019.1, JEDI-based FSMs and DEMAIN. It is also worth pointing that it allows obtaining higher operating frequency and consuming less power than for FSMs designed using the above-mentioned methods. Thus, our approach allows improving all three main characteristics of FSM circuits.

In conclusion, it should be noted that our approach gives good results for rather complex FSMs having more than 15 internal states. The best results can be achieved if the number of LUTers for the first level does not exceed the number of LUT inputs.

Note that the smaller the ratio $|A_0|/M$, the better the characteristics of U_2-based FSMs (compared with the characteristics of FSM circuits based on other investigated methods).

There are two directions for our future research. The first direction is related to the research of the applicability of our approach to FSMs implemented with FPGAs by Intel (Altera). Next, we will try to use this approach for improving characteristics of LUT-based Moore FSMs.

Table 7. Experimental results (the number of LUTs).

Benchmark	Auto	One-Hot	JEDI	DEMAIN	U_2
bbara	17	17	10	9	11
bbsse	33	37	24	26	22
bbtas	5	5	5	5	5
beecount	19	19	14	16	12
cse	40	66	36	38	32
dk14	10	27	10	12	12
dk15	5	16	5	6	6
dk16	15	34	12	14	10
dk17	5	12	5	6	5
dk27	3	5	4	4	6
dk512	10	10	9	10	8
donfile	31	31	22	26	19
ex1	70	74	53	57	42
ex2	9	9	8	9	8
ex3	9	9	9	9	8
ex4	15	13	12	13	10
ex5	9	9	9	9	8
ex6	24	36	22	23	20
ex7	4	5	4	4	4
keyb	43	61	40	42	37
kirkman	42	58	39	41	35
lion	2	5	2	2	2
lion9	6	11	5	5	5
mark1	23	23	20	21	18
mc	4	7	4	5	4
modulo12	7	7	7	7	7
opus	28	28	22	26	23
planet	131	131	88	94	80
planet1	131	131	88	94	80
pma	94	94	86	91	78
s1	65	99	61	64	57
s1488	124	131	108	112	92
s1494	126	132	110	117	94
s1a	49	81	43	54	41
s208	12	31	10	11	9
s27	6	18	6	6	6
s386	26	39	22	25	18
s420	10	31	9	10	8
s510	48	48	32	39	29
s8	9	9	9	9	10
s820	88	82	68	76	58
s832	80	79	62	70	54
sand	132	132	114	121	101
shiftreg	2	6	2	2	4
sse	33	37	30	32	26
styr	93	120	81	88	73
tma	45	39	39	41	33
Total	1792	2104	1480	1601	1330
Percentage	134,7%	158,2%	112,8%	120,4%	100%

References

ABC System (2020). `https://people.eecs.berkeley.edu/~alanmi/abc/`.

Agrawal, R., Borowczak, M. and Vemuri, R. (2019). A state encoding methodology for side-channel security vs. power trade-off exploration, *32nd International Conference on VLSI Design and 18th International Conference on Embedded Systems (VLSID), Delhi, India* pp. 70–75.

Altera (2020). *Cyclone IV Device Handbook*, `http://www.altera.com/literature/hb/cyclone-iv/cyclone4-handbook.pdf`.

Ardakani, A., Leduc-Primeau, F., Onizawa, N., Hanyu, T. and Gross, W.J. (2017). VLSI implementation of deep neural network using integral stochastic computing, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **25**(10): 2688–2699.

Baranov, S. (1994). *Logic Synthesis of Control Automata*, Kluwer, Boston, MA.

Baranov, S. (2008). *Logic and System Design of Digital Systems*, TUT Press, Tallinn.

Barkalov, A.A. and Barkalov Jr., A.A. (2005). Design of Mealy finite-state machines with the transformation of object codes, *International Journal of Applied Mathematics and Computer Science* **15**(1): 151–158.

Barkalov, A. and Titarenko, L. (2009). *Logic Synthesis for FSM-based Control Units*, Springer, Berlin.

Barkalov, A., Titarenko, L., Kołopieńczyk, M., Mielcarek, K. and Bazydło, G. (2015). *Logic Synthesis for FPGA-Based Finite State Machines*, Springer, Cham.

Barkalov, A., Titarenko, L., Mazurkiewicz, M. and Krzywicki, K. (2020a). Encoding of terms in EMB-based Mealy FSMs, *Applied Sciences* **10**(8): 21.

Barkalov, A., Titarenko, L., Mielcarek, K. and Chmielewski, S. (2020b). *Logic Synthesis for FPGA-Based Control Units—Structural Decomposition in Logic Design*, Springer, Berlin.

Barkalov, O., Titarenko, L. and Mielcarek, K. (2018). Hardware reduction for LUT-based Mealy FSMs, *International Journal of Applied Mathematics and Computer Science* **28**(3): 595–607, DOI: 10.2478/amcs-2018-0046.

Benini, L., Bogliolo, A. and Micheli, G. (2000). A survey of design techniques for system-level dynamic power management, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **8**(3): 299–316.

Table 8. Experimental results (the consumed power, Watts).

Benchmark	Auto	One-Hot	JEDI	DEMAIN	U_2
bbara	0,569	0,569	0,488	0,482	0,499
bbsse	2,22	1,206	1,713	1,824	1,622
bbtas	0,533	0,533	0,533	0,533	0,582
beecount	1,631	1,631	1,021	1,236	0,935
cse	0,958	1,019	0,891	0,911	0,783
dk14	2,959	3,33	2,952	2,998	3,002
dk15	1,403	1,905	1,399	1,402	1,412
dk16	2,967	2,742	2, 512	2,715	2,435
dk17	1,901	1,935	1,891	1,938	1,907
dk27	1,168	0,854	1,158	1,161	1,172
dk512	1,496	1,496	1,345	1,498	1,265
donfile	0,709	0,709	0,603	0,638	0,578
ex1	4,102	2,968	2,342	2,416	1,928
ex2	0,368	0,386	0,342	0,365	0,367
ex3	0,391	0,391	0,391	0,394	0,374
ex4	1,562	1,241	1,187	1,198	1,123
ex5	0,387	0,387	0,385	0,383	0,326
ex6	2,269	3,85	2,242	2,258	2,175
ex7	0,992	1,181	0,994	0,996	0,998
keyb	1,093	1,071	1,075	1,082	0,996
kirkman	1,693	1,844	1,439	1,498	1,327
lion	0,542	0,629	0,547	0,544	0,549
lion9	0,733	0,97	0,728	0,73	0,784
mark1	1,445	1,445	1,227	1,301	1,187
mc	0,447	0,561	0,443	0,492	0,462
modulo12	0,559	0,559	0,563	0,532	0,548
opus	1,344	1,344	1,283	1,334	1,221
planet	4,122	4,122	2,456	3,002	2,328
planet1	4,122	4,122	2,456	3,002	2,238
pma	1,37	1,37	1,253	1,361	1,003
s1	2,685	3,13	2,518	2,612	2,348
s1488	3,982	4,096	3,548	3,629	2,083
s1494	3,079	3,178	2,982	3,011	2,658
s1a	1,322	2,01	1,208	1,602	1,085
s208	1,367	2,82	1,249	1,302	1,257
s27	0,756	1,95	0,765	0,769	0,764
s386	1,251	1,393	1,121	1,187	1,098
s420	1,337	2,82	1,286	1,334	1,292
s510	1,543	1,543	1,091	1,218	1,002
s8	0,736	0,805	0,732	0,734	0,882
s820	2,054	1,801	1,463	1,612	1,143
s832	2,096	2,087	1,828	1,512	1,232
sand	1,149	1,149	0,988	1,017	0,817
shiftreg	0,523	0,603	0,512	0,503	0,712
sse	1,22	1,296	1,089	1,193	1,007
styr	4,044	4,771	3,187	3,612	2,932
tma	1,589	1,314	1,321	1,427	1,118
Total	85,479	89,585	65,935	68,498	58,646
Percentage	145,7%	152,7%	112,4%	116,8%	100%

Benini, L. and De Micheli, G. (1995). State assignment for low power dissipation, *IEEE Journal of Solid-State Circuits* **30**(3): 258–268.

Benini, L., De Micheli, G. and Macii, E. (2001). Designing low-power circuits: Practical recipes, *IEEE Circuits and Systems Magazine* **1**(1): 6–25.

Borowik, G. (2018). Optimization on the complementation procedure towards efficient implementation of the index generation function, *International Journal of Applied Mathematics and Computer Science* **28**(4): 803–815, DOI: 10.2478/amcs-2018-0061.

Brayton, R. and Mishchenko, A. (2010). ABC: An academic industrial-strength verification tool, *in* T. Touili *et al.* (Eds), *Computer Aided Verification*, Springer, Berlin/Heidelberg, pp. 24–40.

Brown, B.D. and Card, H.C. (2001). Stochastic neural computation. I: Computational elements, *IEEE Transactions on Computers* **50**(9): 891–905.

Choudhury, P. and Pradhan, S. (2012). Power modeling of power gated FSM and its low power realization by simultaneous partitioning and state encoding using genetic algorithm, *in* H. Rahaman *et al.* (Eds), *Progress in VLSI Design and Test*, Springer, Berlin/Heidelberg, pp. 19–29.

Chow, S., Ho, Y.-C., Hwang, T. and Liu, C. (1996). Low power realization of finite state machines—A decomposition approach, *ACM Transactions on Design Automation of Electronic Systems* **1**(3): 315–340.

Cong, J. and Yan, K. (2000). Synthesis for FPGAs with embedded memory blocks, *Proceedings of the 2000 ACM/SIGDA Eighth International Symposium on Field Programmable Gate Arrays, FPGA'00, Monterey, CA, USA*, pp. 75–82.

Czerwiński, R. and Kania, D. (2013). *Finite State Machine Logic Synthesis for Complex Programmable Logic Devices*, Springer, Berlin.

Das, N. and Priya, P.A. (2018). FPGA implementation of reconfigurable finite state machine with input multiplexing architecture using Hungarian method, *International Journal of Reconfigurable Computing* **2018**: 1–15.

Gajski, D. D., Abdi, S., Gerstlauer, A. and Schirner, G. (2009). *Embedded System Design: Modeling, Synthesis and Verification*, Springer, Berlin.

Garcia-Vargas, I. and Senhadji-Navarro, R. (2015). Finite state machines with input multiplexing: A performance study, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **34**: 867–871.

Garcia-Vargas, I., Senhadji-Navarro, R., Jiménez-Moreno, G., Civit-Balcells, A. and Guerra-Gutierrez, P. (2007). ROM-based finite state machine implementation in low cost FPGAs, *IEEE International Symposium on Industrial Electronics ISIE 2007, Vigo, Spain*, pp. 2342–2347.

Glaser, J., Damm, M., Haase, J. and Grimm, C. (2011). TR-FSM: Transition-based reconfigurable finite state machine, *ACM Transactions on Reconfigurable Technology and Systems* **4**(3): 23:1–23:14.

Grout, I. (2008). *Digital Systems Design with FPGAs and CPLDs*, Elsevier, Oxford.

Kam, T., Villa, T., Brayton, R. and Sangiovanni-Vincentelli, A. (2010). *A Synthesis of Finite State Machines: Functional Optimization*, Springer, Boston, MA.

Khatri, S. and Gulati, K. (Eds) (2011). *Advanced Techniques in Logic Synthesis, Optimizations and Applications*, Springer, New York, NY.

Kołopieńczyk, M., Titarenko, L. and Barkalov, A. (2017). Design of EMB-based Moore FSMs, *Journal of Circuits, Systems, and Computers* **26**(7): 1–23.

Kubatova, H. and Becvar, M. (2002). FEL-Code: FSM internal state encoding method, *Proceedings of the 5th International Workshop on Boolean Problems, Freiberg, Germany,* pp. 109–114.

Kubica, M. and Kania, D. (2017). Area-oriented technology mapping for LUT-based logic blocks, *International Journal of Applied Mathematics and Computer Science* **27**(1): 207–222, DOI: 10.1515/amcs-2017-0015.

Kubica, M., Kania, D. and Kulisz, J. (2019). A technology mapping of FSMs based on a graph of excitations and outputs, *IEEE Access* **7**: 16123–16131.

LGSynth93 (1993). Benchmark suite, *International Workshop on Logic Synthesis, Tahoe City, CA, USA,* `https://people.engr.ncsu.edu/brglez/CBL/benchmarks/LGSynth93/LGSynth93.tar`.

Li, J., Ren, A., Li, Z., Ding, C., Yuan, B., Qiu, Q. and Wang, Y. (2017). Towards acceleration of deep convolutional neural networks using stochastic computing, *22nd Asia and South Pacific Design Automation Conference, ASP-DAC, Chiba/Tokyo, Japan*, pp. 115–120.

Li, P., Lilja, D.J., Qian, W., Riedel, M.D. and Bazargan, K. (2014). Logical computation on stochastic bit streams with linear finite-state machines, *IEEE Transactions on Computers* **63**(6): 1474–1486.

Liu, B., Cai, Y., Zhou, Q., Bian, J. and Hong, X. (2005). FSM decomposition for power gating design automation in sequential circuits, *6th International Conference on ASIC, Shanghai, China*, Vol. 2, pp. 944–947.

Machado, L. and Cortadella, J. (2020). Support-reducing decomposition for FPGA mapping, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **39**(1): 213–224.

Maxfield, C. (2004). *The Design Warrior's Guide to FPGAs*, Academic Press, Orlando, FL.

Michalski, T. and Kokosiński, Z. (2016). Functional decomposition of combinational logic circuits with PKmin, *Technical Transactions: Electrical Engineering* **113**(2-E): 191–202.

Micheli, G.D. (1994). *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, New York, NY.

Mishchenko, A. and Brayton, R. (2006). Scalable logic synthesis using a simple circuit structure, `https://people.eecs.berkeley.edu/~brayton/publications/2006/iwls06_sls.pdf`.

Table 9. Experimental results (the operating frequency, MHz).

Benchmark	Auto	One-Hot	JEDI	DEMAIN	U_2
bbara	193,39	193,39	212,21	198,46	210,37
bbsse	157,06	169,12	182,34	178,91	198,65
bbtas	204,16	204,16	206,12	208,32	200,38
beecount	166,61	166,61	187,32	184,21	201,43
cse	146,43	163,64	178,12	174,19	206,56
dk14	191,64	172,65	193,85	187,32	186,53
dk15	192,53	185,36	194,87	188,54	189,14
dk16	169,72	174,79	197,13	189,83	211,52
dk17	199,28	167	199,39	172,19	199,87
dk27	206,02	201,9	204,18	205,1	196,65
dk512	196,27	196,27	199,75	197,49	208,17
donfile	184,03	184	203,65	194,83	231,63
ex1	150,94	139,76	176,87	186,14	212,93
ex2	198,57	198,57	200,14	199,75	201,34
ex3	194,86	194,86	195,76	193,43	201,12
ex4	180,96	177,71	192,83	178,14	197,76
ex5	180,25	180,25	181,16	181,76	182,01
ex6	169,57	163,8	176,59	174,12	198,65
ex7	200,04	200,84	200,6	200,32	200,69
keyb	156,45	143,47	168,43	157,16	187,48
kirkman	141,38	154	156,68	143,76	174,73
lion	202,43	204	202,35	201,32	200,18
lion9	205,3	185,22	206,38	205,86	207,13
mark1	162,39	162,39	176,18	169,65	189,58
mc	196,66	195,47	196,87	192,53	196,12
modulo12	207	207	207,13	207,37	208,12
opus	166,2	166,2	178,32	168,79	177,84
planet	132,71	132,71	187,14	185,73	193,49
planet1	132,71	132,71	187,14	185,73	193,49
pma	146,18	146,18	169,83	153,57	184,45
s1	146,41	135,85	157,16	149,17	170,19
s1488	138,5	131,94	157,18	153,12	187,95
s1494	149,39	145,75	164,34	159,42	186,22
s1a	153,37	176,4	169,17	158,12	178,84
s208	174,34	176,46	178,76	172,87	196,37
s27	198,73	191,5	199,13	198,43	198,76
s386	168,15	173,46	179,15	169,21	182,63
s420	173,88	176,46	177,25	172,87	181,62
s510	177,65	177,65	198,32	183,18	209,36
s8	180,02	178,95	181,23	180,39	178,32
s820	152	153,16	176,58	166,29	192,14
s832	145,71	153,23	173,78	160,03	192,87
sand	115,97	115,97	126,82	120,63	163,18
shiftreg	262,67	263,57	276,26	276,14	256,69
sse	157,06	169,12	174,63	169,69	189,64
styr	137,61	129,92	145,64	138,83	178,65
tma	163,88	147,8	164,14	168,19	181,22
Total	8126,95	8173,06	8719,07	8103,27	9172,65
Percentage	88%	89%	95%	88%	100%

Mishchenko, A. and Brayton, R. (2007). SAT-based logic optimization and resynthesis, `https://people.eecs.berkeley.edu/~alanmi/publications/2007/tech07_imfs.pdf`.

Mishchenko, A., Brayton, R., Jiang, J.-H.R. and Jang, S. (2011). Scalable don't-care-based logic optimization and resynthesis, *ACM Transactions on Reconfigurable Technology and Systems* **4**(4): 23.

Nag, A., Das, S. and Pradhan, S. (2018). Low power FSM synthesis based on automated power and clock gating technique, *Journal of Circuits, Systems and Computers* **28**(5), Article ID 1920003.

Nowicka, M., Łuba, T. and Rawski, M. (1999). FPGA-based decomposition of Boolean functions: Algorithms and implementation, *6th International Conference on Advanced Computer Systems, Szczecin, Poland*, pp. 502–509.

Opara, A. and Kania, D. (2010). Decomposition-based logic synthesis for PAL-based CPLDs, *International Journal of Applied Mathematics and Computer Science* **20**(2): 367–384, DOI: 10.2478/v10006-010-0027-1.

Opara, A., Kubica, M. and Kania, D. (2019). Methods of improving time efficiency of decomposition dedicated at FPGA structures and using BDD in the process of cyber-physical synthesis, *IEEE Access* **7**: 20619–20631.

PKmin (2020). `http://www.pk.edu.pl/~zk/PKmin/PKmin_pomoc-help.zip`.

Pradhan, S., Kumar, M. and Chattopadhyay, S. (2011). Low power finite state machine synthesis using power-gating, *Integration* **44**(3): 175–184.

Rafla, N.I. and Gauba, I. (2010). A reconfigurable pattern matching hardware implementation using on-chip RAM-based FSM, *53rd IEEE International Midwest Symposium on Circuits and Systems, Boise, ID, USA*, pp. 49–52.

Rawski, M., Jozwiak, L., Nowicka M. and Luba T. (1997). Non-disjoint decomposition of Boolean functions and its application in FPGA-oriented technology mapping, *Proceedings of the 23rd EUROMICRO Conference: New Frontiers of Information Technology, Budapest, Hungary*, pp. 24–30.

Rawski, M., Selvaraj, H. and Łuba, T. (2005). An application of functional decomposition in ROM-based FSM implementation in FPGA devices, *Journal of System Architecture* **51**(6–7): 423–434.

Rawski, M., Tomaszewicz, P., Borowski, G. and Łuba, T. (2011). Logic synthesis method of digital circuits designed for implementation with embedded memory blocks on FPGAs, *in* M. Adamski *et al.* (Eds), *Design of Digital Systems and Devices (LNEE 79)*, Springer, Berlin, pp. 121–144.

Scholl, C. (2001). *Functional Decomposition with Application to FPGA Synthesis*, Kluwer, Boston, MA.

Sentowich, E., Singh, K., Lavango, L., Moon, C., Murgai, R., Saldanha, A., Savoj, H., Stephan, P., Bryton, R. and Sangiovanni-Vincentelli, A. (1992). SIS: A system for sequential circuit synthesis, *Technical report*, University of California, Berkely, CA.

Sklyarov, V. (2000). Synthesis and implementation of RAM-based finite state machines in FPGAs, *Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing, Villach, Austria*, pp. 718–728.

Sklyarov, V., Skliarova, I., Barkalov, A. and Titarenko, L. (2014). *Synthesis and Optimization of FPGA-Based Systems*, Springer, Berlin.

Sutter, G., Todorovich, E., López-Buedo, S. and Boemo, E. (2002). Low-power FSMs in FPGA: Encoding alternatives, *Integrated Circuit Design. Power and Timing Modeling, Optimization and Simulation, Seville, Spain*, pp. 363–370.

Testa, E., Amaru, L., Soeken, M., Mishchenko, A., Vuillod, P., Luo, J., Casares, C., Gaillardon, P. and Micheli, G.D. (2019). Scalable Boolean methods in a modern synthesis flow, *Design, Automation Test in Europe Conference Exhibition (DATE), Florence, Italy*, pp. 1643–1648.

Tiwari, A. and Tomko, K. (2004). Saving power by mapping finite-state machines into embedded memory blocks in FPGAs, *Proceedings of the Conference on Design, Automation and Test in Europe*, Vol. 2, pp. 916–921.

Vivado (2020). `https://www.xilinx.com/products/design-tools/vivado.html`.

Wu, X., Pedram, M. and Wang, L. (2000). Multi-code state assignment for low-power design, *IEEE Proceedings on Circuits, Devices and Systems* **147**(5): 271–275.

Xie, Y., Liao, S., Yuan, B., Wang, Y. and Wang, Z. (2017). Fully-parallel area-efficient deep neural network design using stochastic computing, *IEEE Transactions on Circuits and Systems II: Express Briefs* **64**(12): 1382–1386.

Xilinx (2010). *Virtex-4 Family Overview*, `http://www.xilinx.com/support/documentation/data_sheets/ds112.pdf`.

Xilinx (2015). *Virtex-5 Family Overview*, `http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf`.

Xilinx (2020a). `http://www.xilinx.com`.

Xilinx (2020b). *ISE Foundation*, `https://www.xilinx.com/products/design-tools/ise-design-suite.html`.

Alexander A. Barkalov worked in Donetsk National Technical University (DNTU) from 1976 till 1996 as a tutor. He cooperated actively with the Kiev Institute of Cybernetics (IC) named after Victor Glushkov. He obtained his PhD in computer science in 1995 from the IC. From 1996 till 2003 he worked as a professor of DNTU. Since 2003 he has been working as a professor in the Department of Computer, Electrical and Control Engineering of the University of Zielona Góra.

Larysa Titarenko obtained her PhD degree (telecommunications) in 2005 from the Kharkov National University of Radioelectronics (KNURE). Till 2003 she worked as a professor of the KNURE. Since 2005 she has been working as a professor in the Department of Computer, Electrical and Control Engineering of the University of Zielona Góra.

Kamil Mielcarek received his MSc degree in computer engineering from the Technical University of Zielona Góra, Poland, in 1995 and his PhD degree in computer science from the University of Zielona Góra, Poland, in 2010. Since 2001 he has been a lecturer there. His current interests include methods of synthesis and optimization of control units in field-programmable logic devices, hardware description languages, perfect graphs and Petri nets, algorithmic theory and safety of UNIX and network systems.

Received: 21 February 2020
Revised: 20 May 2020
Re-revised: 21 July 2020
Accepted: 4 August 2020