sciendo

Int. J. Appl. Math. Comput. Sci., 2022, Vol. 32, No. 3, 479–494
DOI: 10.34768/amcs-2022-0035

amcs

IMPROVING THE LUT COUNT FOR MEALY FSMs WITH TRANSFORMATION OF OUTPUT COLLECTIONS

ALEXANDER BARKALOV [a], LARYSA TITARENKO [a], MAŁGORZATA MAZURKIEWICZ [b,*]

[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}@imei.uz.zgora.pl

[b] Institute of Control and Computation Engineering
University of Zielona Góra
ul. Szafrana 2, 65-516 Zielona Góra, Poland
e-mail: m.mazurkiewicz@issi.uz.zgora.pl

A method is proposed which aims at reducing the number of LUTs in the circuits of FPGA-based Mealy finite state machines (FSMs) with transformation of collections of outputs into state codes. The reduction is achieved due to the use of two-component state codes. Such an approach allows reducing the number of state variables compared with FSMs based on extended codes. There are exactly three levels of LUTs in the resulting FSM circuit. Each partial function is represented by a single-LUT circuit. The proposed method is illustrated with an example of synthesis. The experiments were conducted using standard benchmarks. They show that the proposed method produces FSM circuits with significantly smaller LUT counts compared with those produced by other investigated methods (Auto and One-hot of Vivado, JEDI, and transformation of output collection codes into extended state codes). The LUT count is decreased by, on average, from 9.86% to 59.64%. The improvement of the LUT count is accompanied by a slightly improved performance. The maximum operating frequency is increased, on average, from 2.74% to 12.93%. The advantages of the proposed method become more pronounced with increasing values of FSM inputs and state variables.

Keywords: Mealy FSM, FPGA, LUT, synthesis, state codes.

1. Introduction

A lot of various sequential blocks can be found in modern digital systems (Marwedel, 2018; Sklyarov *et al*., 2014; Borowczak and Vemuri, 2013). Very often, these blocks are represented using the model of a Mealy finite state machine (FSM) (Baranov, 1994; Micheli, 1994). In this paper, we discuss a case when Mealy FSM circuits are implemented using look-up table (LUT) elements of field-programmable gate arrays (FPGAs) (Altera, 2021; Xilinx, 2021). Nowadays, a huge number is of various digital systems are implemented on the base of FPGAs (Ruiz-Rosero *et al*., 2019). Due to that, the development of effective implementation methods is of great scientific and practical interest.

One of the central problems associated with LUT-based FSM synthesis is the issue of reducing the chip area occupied by an FSM circuit (Kubica and Kania, 2017; Barkalov *et al*., 2018). In scientific papers, the chip area of LUT-based circuits is estimated as the LUT count (Islam *et al*., 2020). As a rule, area reduction also reduces power consumption (Barkalov *et al*., 2021b). At the same time, it is important that area reduction leads to the smallest possible increase in the delay of the signal propagation time (Barkalov and Barkalov Jr., 2005; Maxfield, 2008; Sklyarov *et al*., 2014; Tiwari and Tomko, 2004). In this paper, we propose a method of LUT count reduction which does not reduce the maximum operating frequency. We discuss a case where an FPGA-based FSM circuit is implemented using slices consisting of LUTs, programmable flip-flops and multiplexers (Trimberg, 2015; Machado and Cortadella, 2020).

*Corresponding author

To be a flexible design tool, modern LUTs have a very limited number of inputs (Altera, 2021; Atmel, 2021; Xilinx, 2021). However, this flexibility also has a gloomy side, namely, the need for decomposition of functions representing an FSM logic circuit (Scholl, 2001; Solovjev and Czyzy, 1999). The circuit based on the functional decomposition has a lot of logic levels and very complex systems of spaghetti-type interconnections (Barkalov *et al*., 2020a). Accordingly, it is very important to develop FSM design methods with a minimum negative impact of a limited number of LUT inputs. This can be done using methods of structural decomposition (Barkalov *et al.*, 2021b; 2021a).

The main contribution of this paper is a novel method of reducing LUT counts in logic circuits of FPGA-based Mealy FSMs. The method is based on a transformation of collections of FSM outputs into two-component state codes proposed in this paper. It can be viewed as an improvement of the approach of Barkalov *et al*. (2020a), where the collections of outputs are transformed into extended state codes. This transformation requires a special block consuming some chip resources. The proposed method is based on creating a partition of states such that a partial function for each block is represented by a single-LUT circuit (Barkalov, *et al.*, 2018; 2020b). The main difference between our method and the known FSM synthesis techniques is the development of a new approach for representing state codes.

The experiments conducted using standard benchmarks show that the proposed method allows improving LUT counts in Mealy FSM circuits compared with the circuits produced by other investigated methods. Two-component state codes have practically the same number of state variables as maximum binary state codes. This leads to a decrease in the number of feedback signals compared with equivalent FSMs based on extended state codes. Due to that, our approach leads to a minor increase in the FSM performance.

The rest of the paper is organized as follows. Section 2 includes the basic information connected with LUT-based design of Mealy FSM circuits. A brief analysis of the related works is provided in Section 3. Section 4 is a central part of the paper, where the proposed method is discussed. An example of synthesis is shown in Section 5. Section 6 includes results of experiments with standard benchmarks. The paper is ended with a short conclusion.

2. Background of LUT-based FSMs

A Mealy FSM is represented by a vector having six components. These components include: (1) a set of inputs $X = \{x_1, \dots, x_L\}$, (2) a set of outputs $Y = \{y_1, \dots, y_N\}$, (3) a set of states $A = \{a_1, \dots, a_M\}$, (4) a function of transitions, (5) a function of output, (6) an initial state $a_1 \in A$. An FSM can be defined using a lot of tools. For example, it can be represented with state transition tables (Micheli, 1994; Minns and Elliot, 2008), binary decision diagrams (Kubica *et al.*, 2017; 2019), and-inverter graphs (Brayton and Mishchenko, 2010), or graph schemes of algorithms (Baranov, 2008). In this article, we use state transition graphs (STGs) to represent Mealy FSMs. An example of the STG representing a Mealy FSM A_o is shown in Fig. 1.

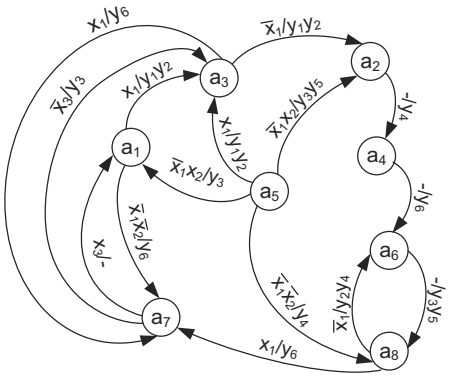

Fig. 1. State transition graph of Mealy FSM A_o.

The FSM states are represented by the nodes of the STG. For example, there are $M = 8$ states in the FSM A_o. The initial state $a_1 \in A$ is clearly highlighted on the STG. The arcs of the STG represent interstate transitions. There are $H = 15$ arcs in the discussed example. An arc is directed from a current state $a_m \in A$ to a state of transition $a_s \in A$. The h-th arc is marked with a two-component pair. The first component is an input signal X_h causing the corresponding transition. An input signal X_h is a conjunction of some inputs $x_l \in X$ (or their compliments). The second component is a collection of outputs (CO) $Y_h \subseteq Y$ generated during the h-th transition. The outputs of a particular FSM are combined into Q different COs.

The analysis of the STG (Fig. 1) gives the following sets: $X = \{x_1, x_2, x_3\}$, $Y = \{y_1, \dots, y_6\}$, and $A = \{a_1, \dots, a_8\}$. Thus, we have $L = 3$, $N = 6$, and $M = 8$.

To design an FSM circuit starting from an STG, it is necessary to transform the STG into systems of Boolean functions (SBFs) representing the circuit. To get these SBFs, we should execute state assignment. During this step, states $a_m \subseteq A$ are represented by binary codes $K(a_m)$ having R bits. The minimum value of R is determined as

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

The formula (1) determines the so-called maximum binary state codes (Sutter *et al*., 2002; Das and Panchanathan, 2018).

The r-th state code bit corresponds to a variable $T_r \subseteq T$, where $T = \{T_1, \dots, T_R\}$ is a set of state

variables. A special register, RG, keeps state codes. There are R flip-flops in the RG. Each flip-flop has inputs D (informational inputs), Start and Clock. The pulse Start loads the code $K(a_1)$ of the initial state into the RG (Skliarova *et al.*, 2012). Each transition is reduced to replacing the RG contents. A transition from $a_m \in A$ into $a_s \in A$ is associated with the replacement of $K(a_m)$ by $K(a_s)$. This can be done using the synchronization pulse Clock and input memory functions (IMFs) creating the set $\Phi = \{D_1, \ldots, D_R\}$.

The FSM logic circuit is represented by the following SBFs:

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

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

The SBF (2) represents the function of transitions, the SBF (3) is the function of output. The systems (2)–(3) determine the structural diagram of the P Mealy FSM shown in Fig. 2.

There are two blocks in the P Mealy FSM. The block of functions (BF) implements the SBFs (2) and (3). The register, RG, keeps state codes. The RG is controlled by the pulses Start and Clock. In this paper, we discuss a case where configurable logic blocks (CLBs) of the FPGA are used for implementing an FSM circuit. To design a circuit, we use CLBs including LUTs, programmable flip-flops, and multiplexers. The flip-flops form a distributed state code register RG (Sklyarov *et al.*, 2014).

An LUT has S_L inputs and a single output (Xilinx, 2021). Thus, a LUT can implement an arbitrary Boolean function having up to S_L arguments. The value of this function can be registered using the internal flip-flop. The CLBs form slices. The largest manufacturer of FPGA chips is the Xilinx company (Xilinx, 2021). Because of this, our current research focuses on the Virtex-7 family by Xilinx (Xilinx, 2021). A slice of Virtex-7 (Xilinx, 2021) includes four 6-LUTs, eight flip-flops and a lot of multiplexers. Three of these multiplexers are used for creating either two 7-LUTs or a single 8-LUT. Such an approach allows getting 7-LUTs and 8-LUT as fast as the initial 6-LUTs (Chapman, 2014). If it is necessary to create a LUT having more than eight inputs, then resources of several CLBs should be used.

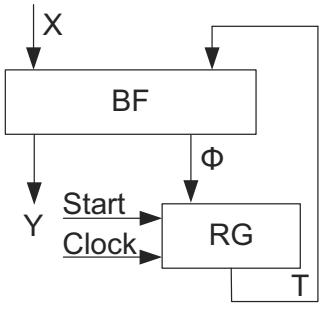

Fig. 2. Structural diagram of the P Mealy FSM.

For average FSMs (Baranov, 1994), the SBFs (2) and (3) can include around 50–70 literals (Machado and Cortadella, 2020). Because a LUT has not enough inputs, it is necessary to transform these SBFs. The transformation is associated with using various methods of functional decomposition (FD) (Kubica *et al.*, 2021). Using FD-based methods results in obtaining FSM circuits with a lot of logic levels and "spaghetti-type" interconnections (Barkalov *et al.*, 2021b). In this case, FSM circuits are rather slow because several CLBs are used to implement some of the functions (2) and (3) (Sasao and Mishchenko, 2009).

The LUT-based synthesis assumes solving some optimization problems (Kubica and Kania, 2017). It is highly desirable that the FSM circuit consists of the minimum possible number of LUTs, has the maximum operating frequency and provides the minimum power consumption. As the level of integration increases, the influence of the interconnect system on the characteristics of the circuit increases. As mentioned by Feng *et al.* (2018), now the interconnection delay dominates the logic delay. Also, it is known that the interconnections are responsible for consuming up to 70% of power (Amano, 2018).

To optimize the system of interconnections, it is necessary to minimize the number of literals in sum-of-products (SOPs) of functions representing an FSM circuit (Barkalov *et al.*, 2021b). This can be done using various methods of state assignment (Kubica *et al.*, 2021) or structural decomposition (Barkalov *et al.*, 2020c; 2021b). A method proposed in this paper belongs to the second group.

3. Related work

Structural decomposition (SD) is associated with a transformation of the structural diagram (Fig. 2) (Barkalov *et al.*, 2021b; 2021a; Senhadji-Navaro *et al.*, 2015). This means that the P FSM is represented by some composition of several interrelated logic blocks (Barkalov *et al.*, 2021a). Each block has its unique system of input and output variables. The outputs of blocks are inputs of other blocks. Each of these blocks is represented by an SOP having significantly fewer literals than SOPs of SBFs (2)–(3). As shown by Barkalov *et al.* (2021b), SD produces FSM circuits with fewer LUT counts compared with equivalent P FSMs.

One of SD-based methods is a transformation of collections of outputs (COs) $Y_q \subseteq Y$ into FSM states (Barkalov *et al.*, 2020c; 2021b). This approach leads to the so-called $P_Y Y$ FSMs (Barkalov *et al.*, 2020c). The subscript "Y" means that COs are transformed into states. The capital "Y" means that COs are transformed into

Table 1. State transition table of FSM A_o.

a_m	a_s	X_h	Y_h	h	Y_q	Δ_j
	a_3	x_1	y_1y_2	1	Y_2	Δ_3
a_1	a_5	$\bar{x_1}x_2$	y_3	2	Y_3	Δ_5
	a_7	$\bar{x_1}\bar{x_2}$	y_6	3	Y_4	Δ_7
a_2	a_4	1	y_4	4	Y_5	Δ_8
a_3	a_7	x_1	y_6	5	Y_4	Δ_7
	a_2	$\bar{x_1}$	y_1y_2	6	Y_2	Δ_2
a_4	a_6	1	y_6	7	Y_4	Δ_6
	a_3	x_1	y_1y_2	8	Y_2	Δ_3
a_5	a_2	$\bar{x_1}x_2$	y_3y_5	9	Y_6	Δ_{10}
	a_8	$\bar{x_1}\bar{x_2}$	y_4	10	Y_5	Δ_9
a_6	a_8	1	y_3y_5	11	Y_6	Δ_{10}
a_7	a_1	x_3	–	12	Y_1	Δ_1
	a_3	$\bar{x_3}$	y_3	13	Y_3	Δ_4
a_8	a_7	x_1	y_6	14	Y_4	Δ_7
	a_6	$\bar{x_1}$	y_3y_4	15	Y_7	Δ_{12}

outputs $y_n \in Y$.

To explain this method, we transform the STG (Fig. 1) into a state transition table (STT). An STT is a list of transitions $\langle a_m, a_s \rangle$ for a given STG (Micheli, 1994). An STT has the following columns (Micheli, 1994): a_m, a_s, X_h, Y_h, h. Accordingly, each row shows (i) the current state a_m from which the h-th arc comes out, (ii) the state of transition a_s which the h-th arc enters, (iii) the input signal X_h written above the arc, (iv) the CO $Y_h \subseteq Y$ written above the arc. There are H rows in an STT. For example, the STT of FSM A_o (Table 1) has $H = 15$ rows.

We add two additional columns in Table 1. We will explain their meaning a bit later.

There are $Q = 7$ different COs in Table 1. They are the following: $Y_1 = \emptyset$, $Y_2 = \{y_1, y_2\}$, $Y_3 = \{y_3\}$, $Y_4 = \{y_6\}$, $Y_5 = \{y_4\}$, $Y_6 = \{y_3, y_5\}$ and $Y_7 = \{y_2, y_4\}$. The symbols Y_q for COs are shown in the column Y_q.

As follows from row 12, if $Y_h = Y_1$, then $a_s = a_1$. But COs $Y_q \subseteq Y$ do not always uniquely identify the states of transition. For example, the CO Y_2 is generated during the transitions into states a_2 (row 6) and a_3 (row 1). Therefore, an additional identifier is needed to select a specific state of transition based on a CO Y_q. These identifiers form a set $I = \{I_1, \ldots, I_G\}$.

To find the value of G, we should create sets $A(Y_q)$. A set $A(Y_q) \subseteq A$ includes states of transition for the arcs marked by $Y_q \subseteq Y$. There are M_q elements in the set $A(Y_q)$ ($q \in \{1, \ldots, Q\}$). The value of G is determined as

$$G = \max(M_1, \ldots, M_Q). \tag{4}$$

For FSM A_o, we have the sets $A(Y_1) = \{a_1\}$, $A(Y_2) = \{a_2, a_3\}$, $A(Y_3) = \{a_3, a_5\}$, $A(Y_4) = \{a_6, a_7\}$, $A(Y_5) = \{a_4, a_8\}$, $A(Y_6) = \{a_2, a_8\}$, and $A(Y_7) = \{a_6\}$. Using (4) gives $G = 2$ and $I = \{I_1, I_2\}$. A pair $\Delta_j = \langle Y_q, I_g \rangle$ determines a state $a_s \in A$. There are $J = 12$ pairs shown in the column Δ_j of Table 1. All these pairs are shown it Table 2.

Table 2 uses information from Table 1. The identifiers are distributed in the following order: the smaller the state number for a given pair, the smaller the number of the identifier.

To design the circuit of P_YY FSM, it is necessary to encode states, COs and identifiers (Barkalov *et al.*, 2020c). Each CO $Y_q \subseteq Y$ is encoded by binary code $K(Y_q)$ having

$$R_Y = \lceil \log_2 Q \rceil \tag{5}$$

bits. Identifier codes $K(I_g)$ have R_I bits, where

$$R_I = \lceil \log_2 G \rceil. \tag{6}$$

The variables $z_r \in Z$ encode the COs, where $|Z| = R_Y$. The variables $v_r \in V$ encode the identifiers, where $|V| = R_I$.

The following SBFs represent a P_YY FSM:

$$Z = Z(T, X), \tag{7}$$

$$V = V(T, X), \tag{8}$$

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

$$\Phi = \Phi(Z, V). \tag{10}$$

The structural diagram of P_YY FSM is shown in Fig. 3.

In the P_YY FSM, the block of functions (BF) implements the SBFs (7) and (8). The block of outputs (BO) implements the SBF (9), the block of input memory functions (BIMF) implements the SBF (10). In LUT-based P_YY FSMs, the RG is distributed among LUTs of the BIMF.

Denote by $\mathrm{NA}(f_i)$ the number of arguments for a function f_i. If the condition

$$\mathrm{NA}(f_i) \le S_L \tag{11}$$

holds for functions $f_i \in Z \cup V \cup \Phi$, then a circuit of P_YY FSM includes fewer LUTs compared with an equivalent P FSM (Barkalov *et al.*, 2020c). If this condition is violated, then either the BF or BIMF or both are represented by multi-level LUT-based circuits.

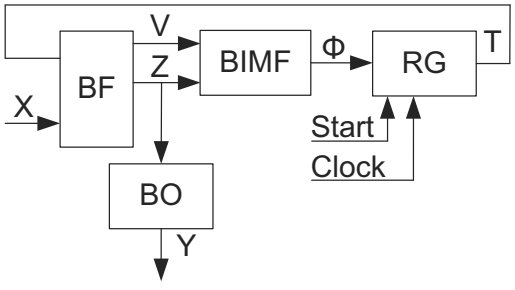

Fig. 3. Structural diagram of P_YY Mealy FSM.

Table 2. Table of transformations of COs for $P_Y Y$ FSM A_o.

Y_q	Y_1	Y_2		Y_3		Y_4		Y_5		Y_6		Y_7
a_m	a_1	a_2	a_3	a_3	a_5	a_6	a_7	a_4	a_8	a_2	a_8	a_6
I_g	I_1	I_1	I_2	I_1	I_2	I_1	I_2	I_1	I_2	I_1	I_2	I_1
Δ_j	Δ_1	Δ_2	Δ_3	Δ_4	Δ_5	Δ_6	Δ_7	Δ_8	Δ_9	Δ_{10}	Δ_{11}	Δ_{12}

If (11) is violated, then the number of logic levels can be reduced using the approach of Barkalov *et al*. (2020a). It is based on extended state codes (ESCs).

To use this approach, it is necessary to form a partition $\Pi_A = \{A^1, \dots, A^K\}$ for the set A. Each class $A^k \in \Pi_A$ determines sets $X^k \subseteq X$, $Z^k \subseteq Z$ and $V^k \subseteq V$. The set X^k includes L_k inputs $x_l \in X$ determining transitions from states $a_m \in A^k$. The sets $Z^k \subseteq Z$ and $V^k \subseteq V$ include additional variables generated during the transitions from states $a_m \in A^k$.

If $|A^k| = M_k$, then R_k variables encode states $a_m \in A^k$ by codes $C(a_m)$, where

$$R_k = \lceil \log_2(M_k + 1) \rceil. \tag{12}$$

In the work of Barkalov *et al*. (2018), an algorithm is proposed which allows finding the partition Π_A with the minimum number of classes, K. The following condition holds for each class $A^k \in \Pi_A$:

$$R_k + L_k \le S_L. \tag{13}$$

Codes $K(a_m)$ are represented as concatenations of $K(Y_q)$ and $K(I_g)$. These codes are transformed into extended state codes. States $a_m \in A^k$ are encoded by state variables $\tau_r \in T^k$, where $|T^k| = R_k$. There are R_E variables in ESCs:

$$R_E = R_1 + R_2 + \dots + R_K. \tag{14}$$

These variables form a set $T = T^1 \cup T^2 \cup \dots \cup T^K$.

In the work of Barkalov *et al*. (2020a), a structural diagram of $P_{YE}Y$ Mealy FSM (Fig. 4) is proposed. It consists of K blocks of partial functions (BPF), a BF, a BO and a block of ESC (BESC).

A block BPFk implements SBFs,

$$Z^k = Z^k(T^k, X^k), \tag{15}$$

$$V^k = V^k(T^k, X^k). \tag{16}$$

The BF implements functions,

$$Z = Z(Z^1, \dots, Z^K), \tag{17}$$

$$V = V(V^1, \dots, V^K). \tag{18}$$

The BESC implements an SBF,

$$T = T(Z, V). \tag{19}$$

The experimental results of Barkalov *et al*. (2020a) show that $P_{YE}Y$ FSM circuits include, on average, 8.84% more LUTs than the equivalent $P_Y Y$ FSM. But the approach (Barkalov *et al*., 2020a) allows increasing the maximum operating frequency by 12.57%, on average. In consequence, compared with $P_Y Y$ FSM, the approach of Barkalov *et al*. (2020a) has both strong and weak sides.

In our opinion, the increase in the LUT count is due to two factors. Firstly, because of the relation $R \ll R_E$, the BESC consist of a significantly larger number of LUTs than the BIMF. Secondly, due to (12), each class $A^k \in \Pi_A$ has less than the maximum possible number of states. This number is equal to 2^{R_k} states. If each class included 2^{R_k} states, then the number of classes (K) could be reduced. In turn, this would lead to a decrease in the number of LUTs in circuits of BPF1–BPFK. Such an approach is proposed in the current paper.

4. Main idea of the proposed method

In (12), the value of M_k is incremented to take into account the relation $a_m \notin A^k$. As a result, only a single block BPFk is active; all other blocks have zeros at their outputs. This allows representing the functions (17) and (18) as disjunctions of eponymous outputs of BPF1–BPFK. We propose to eliminate 1 from (12). This

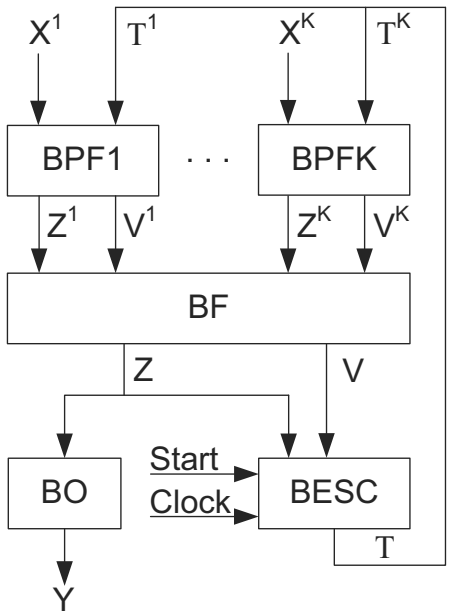

Fig. 4. Structural diagram of $P_{YE}Y$ Mealy FSM.

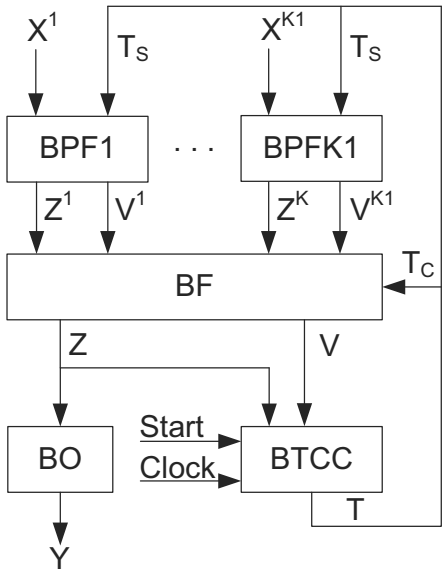

Fig. 5. Structural diagram of $P_{YC}Y$ Mealy FSM.

can be done due to the following approach.

Let us form the partition $\Pi_B = \{B^1, \ldots, B^{K1}\}$ for a set A. Each class includes M_k states. Each class determines a set X^k having L_k elements. We propose to encode the states $a_m \in B^k$ by codes $\mathrm{CS}(a_m)$ having NR_k bits, where

$$\mathrm{NR}_k = \lceil \log_2(M_k) \rceil. \tag{20}$$

The following relation should take place:

$$\mathrm{NR}_k + L_k \leq S_L. \tag{21}$$

To select a particular class $B^k \in \Pi_B$, we encode the classes by binary codes $K(B^k)$. These codes have R_C bits:

$$R_C = \lceil \log_2 K1 \rceil. \tag{22}$$

Now, a state $a_m \in A$ is represented by a two-component code (TCC) $TC(a_m)$:

$$\mathrm{TC}(a_m) = K(B^k) * \mathrm{CS}(a_m). \tag{23}$$

In (23), we assume that $a_m \in B^k$. The symbol "*" stands for the concatenation of codes.

To encode states, the variables $T_r \in T$ are used, where $T = T_C \cup T_S$. The first R_C elements of T form a set T_C used for encoding of the classes. The last R_S elements form a set T_S used for encoding the states. The value of R_S is determined as

$$R_S = \max(\mathrm{NR}_1, \ldots, \mathrm{NR}_{K1}). \tag{24}$$

Using TCCs allows transforming a $P_{YE}Y$ FSM into a $P_{YC}Y$ FSM (Fig. 5).

In the $P_{YC}Y$ FSM, block BPFk ($k = \overline{1, K1}$) implements SBFs,

$$Z^k = Z^k(T_S, X^k), \tag{25}$$

$$V^k = V^k(T_S, X^k). \tag{26}$$

To chose a particular partial function, a multiplexer should be used. These multiplexers create the BF implementing SBFs,

$$Z = Z(T_C, Z^1, \ldots, Z^{K1}), \tag{27}$$

$$V = V(T_C, V^1, \ldots, V^{K1}). \tag{28}$$

The BO implements the SBF (9). The block of TCC (BTCC) implements an SBF,

$$T = T(Z, V). \tag{29}$$

This block includes a distributed RG having

$$R_{\mathrm{TC}} = R_C + R_S \tag{30}$$

flip-flops.

Hardware reduction compared with $P_{YE}Y$ FSMs is possible if the following relations take places:

$$K1 < K, \tag{31}$$

$$R_{\mathrm{TC}} < R_E. \tag{32}$$

Our analysis of the library LGSynth93 (1993) shows that the relations (31) and (32) hold for the vast majority of benchmarks.

In this article, we propose a synthesis method for $P_{YC}Y$ Mealy FSMs. The synthesis starts from an STG. It includes the following steps:

1. Creating an STT of a P Mealy FSM.
2. Representing states $a_m \in A$ by pairs $\langle Y_q, I_g \rangle$.
3. Encoding of COs $Y_q \subseteq Y$ in a way optimising the SBF (9).
4. Encoding of identifiers $I_g \in I$.
5. Forming the partition Π_B.
6. Encoding states $a_m \in B^k$ and classes $B^k \in \Pi_B$.
7. Creating a direct structure table of $P_{YC}Y$ FSM.
8. Creating the SBFs (25) and (26).
9. Creating the table of BF and deriving the SBFs (27) and (28).
10. Creating the table of the BTCC.
11. Implementing an FSM circuit with resources of a particular FPGA chip.

To explain this method, we discuss an example of synthesis for $P_{YC}Y$ FSM A_o. We use LUTs with $S_L = 4$ to implement the FSM circuit.

5. Example of synthesis

For FSM A_o, the STT is already created. It is given in Table 1. There are $Q = 9$ COs in the discussed case. They are discussed in Section 3. There are $J = 12$ pairs and $G = 2$ identifiers shown in Table 2. These pairs are the following: $\Delta_1 = \langle Y_1, I_1\rangle$, $\Delta_2 = \langle Y_2, I_1\rangle$, $\Delta_3 = \langle Y_2, I_2\rangle$, $\Delta_4 = \langle Y_3, I_1\rangle$, $\Delta_5 = \langle Y_3, I_2\rangle$, $\Delta_5 = \langle Y_4, I_1\rangle$, $\Delta_7 = \langle Y_4, I_2\rangle$, $\Delta_8 = \langle Y_5, I_1\rangle$, $\Delta_9 = \langle Y_5, I_2\rangle$, $\Delta_{10} = \langle Y_6, I_1\rangle$, $\Delta_{11} = \langle Y_6, I_2\rangle$, and $\Delta_{12} = \langle Y_7, I_1\rangle$. These pairs are used to form the table of the BTCC.

Using (5) gives $R_Y = 3$ and $Z = \{z_1, z_2, z_3\}$. To optimize the FSM circuit, it is necessary to minimize the number of literals in the SBF (9). It can be done using the method of Achasova (1987). There is an outcome of encoding shown in Fig. 6.

Using contents of COs and the codes (Fig. 6), we can get the following SBF:

$$\begin{aligned} y_1 &= Y_2 = \bar{z_1} z_2 \bar{z_3}, & y_4 &= Y_5 \vee Y_7 = \bar{z_1} z_3, \\ y_2 &= Y_2 \vee Y_7 = \bar{z_1} z_2, & y_5 &= Y_6 = z_1 z_3, \\ y_3 &= Y_3 \vee Y_6 = z_1 \bar{z_2}, & y_6 &= Y_4 = z_1 z_2. \end{aligned} \tag{33}$$

The SBF (33) represents the BO of $P_{YC}Y$ FSM A_o.

We have $G = 2$. Using (6) gives $R_I = 1$ and $V = \{v_1\}$. We propose to encode the identifiers in the following way: $K(I_1) = 0$ and $K(I_2) = 1$.

Using the methods of Barkalov *et al.* (2018; 2020c) gives the partition $\Pi_B = \{B^1, B^2\}$ with $B^1 = \{a_1, a_3, a_5, a_8\}$ and $B^2 = \{a_2, a_4, a_6, a_7\}$. These classes define the sets $X^1 = \{x_1, x_2\}$ and $X^2 = \{x_3\}$. We have $M_1 = M_2 = 4$. Using (20) gives $\mathrm{NR}_1 = \mathrm{NR}_2 = 2$. From (24) it follows that $R_S = 2$.

Obviously, the condition (21) holds for $S_L = 4$. Thus, the model of the $P_{YC}Y$ FSM can be used for FSM A_o. The outcome of state assignment does not affect the number of LUTs in BPF1–BPF2. The same is true for codes $K(B^k)$. Therefore, we encode states $a_m \in B^k$ and classes $B^k \in \Pi_B$ in a trivial way.

We have $K1 = 2$. Using (22) gives $R_C = 1$ and $T_C = \{T_1\}$. Because $R_S = 2$, we can get the sets $T_S = \{T_2, T_3\}$ and $T = \{T_1, T_2, T_3\}$. The codes $CS(a_m)$ and $K(B^k)$ are shown in Fig. 7.

Using Fig. 7 gives two-component state codes. For example, we get $\mathrm{TC}(a_3) = K(B^1) * \mathrm{CS}(a_3) = 001$.

A DST represents blocks BPF1–BPF$K1$. It is based on the initial STT and contents of pairs Δ_j. This table includes columns a_m, X_h, h of the initial STT. Also, it includes the columns $\mathrm{CS}(a_m)$, Z_h, V_h. In the discussed case, it is Table 3.

We have $\Delta_3 = \langle Y_2, I_2\rangle$ in the first row of Table 1. From Fig. 6, we have $K(Y_2) = 010$. Also, we have $K(I_2) = 1$. Consequently, we have z_2^1 in the column Z_h and v_1^1 in the column V_h in the first row of Table 3. This row contains $\mathrm{CS}(a_1) = 00$ in the column $\mathrm{CS}(a_m)$. All other rows of Table 3 are filled in the same order. If $a_m \in B^k$, then all functions in the columns Z_h, V_h have the superscript k ($k \in \{1, \ldots, K1\}$).

Using Table 3 gives the SBFs (25) and (26). For example, the following SOPs can be derived from Table 3 (after minimization):

$$\begin{aligned} z_1^1 &= \bar{T_2}\bar{T_3}\bar{x_1} \vee \bar{T_2}T_3x_1 \vee T_2\bar{T_3}\bar{x_1}x_2 \vee T_2\bar{T_3} \vee \\ &\vee \ T_2T_3x_1, \\ v_1^1 &= \bar{T_2}\bar{T_3} \vee \bar{T_2}T_3x_1 \vee T_2\bar{T_3}x_2 \vee T_2\bar{T_3}\bar{x_2} \vee T_2T_3x_1. \end{aligned} \tag{34}$$

The table of BF includes the columns Function, BPF1, …, BPF$K1$. If some function is generated by BPFk, then we have 1 at the intersection of the corresponding row and the column BPFk. In the discussed case, this is Table 4.

The following SBFs are derived from Table 4:

$$\begin{aligned} z_1 &= \bar{T_1} z_1^1 \vee T_1 z_1^2, \\ z_2 &= \bar{T_1} z_2^1 \vee T_1 z_2^2, \\ z_3 &= \bar{T_1} z_3^1 \vee T_1 z_3^2, \end{aligned} \tag{35}$$

$$v_1 = \bar{T_1} v_1^1. \tag{36}$$

The SBFs (35) and (36) correspond to the SBFs (27) and (28), respectively. They represent multiplexers with

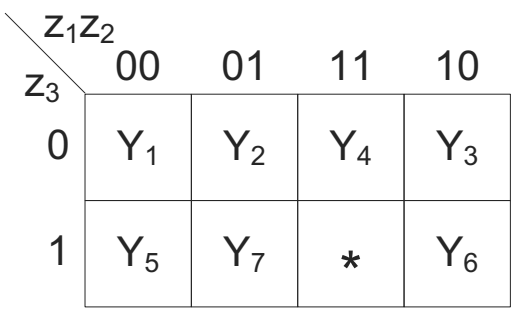

Fig. 6. Codes $K(Y_q)$ for FSM A_o.

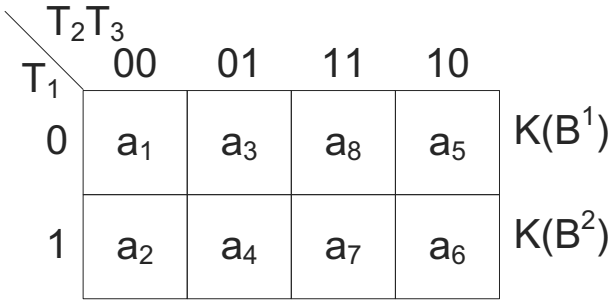

Fig. 7. Two-component state codes of Mealy $P_{YC}Y$ FSM A_o.

Table 3. Direct structure table of $P_{YC}Y$ FSM A_o.

a_m	CS(a_m)	X_h	Z_h	V_h	h
		x_1	z_2^1	v_1^1	1
a_1	00	$\bar{x_1}x_2$	z_1^1	v_1^1	2
		$\bar{x_1}\bar{x_2}$	$z_1^1 z_2^1$	v_1^1	3
a_2	00	1	z_3^2	–	4
a_3	01	x_1	$z_1^1 z_2^1$	v_1^1	5
		$\bar{x_1}$	z_2^1	–	6
a_4	01	1	$z_1^2 z_2^2$	–	7
		x_1	z_2^1	v_1^1	8
a_5	10	$\bar{x_1}x_2$	$z_1^1 z_3^1$	–	9
		$\bar{x_1}\bar{x_2}$	z_3^1	v_1^1	10
a_6	10	1	$z_1^2 z_3^2$	–	11
a_7	11	x_3	–	–	12
		$\bar{x_3}$	z_1^2	–	13
a_8	11	x_1	$z_1^1 z_2^1$	v_1^1	14
		$\bar{x_1}$	$z_2^1 z_2^1$	–	15

Table 4. Table of BF for $P_{YC}Y$ FSM A_o.

Function	BPF1	BPF2
z_1	1	1
z_2	1	1
z_3	1	1
v_1	1	0

control inputs connected with $T_1 \in T_C$. Using a single slice of Virtex-7, we can get fast multiplexers having up to four inputs (Chapman, 2014).

The table of the BTCC represents the SBF (28). This table has the following columns: j (the number of a pair Δ_j), $K(Y_q)$, $K(I_g)$ (codes of a CO Y_q and identifier $I_g \in I$ from the pair Δ_j), a_m (a state represented by Δ_j), $TC(a_m)$, Φ_j (IMFs generated for loading $TC(a_m)$ into the RG). In the discussed case, the BTCC is represented by Table 5.

For example, the following SBF can be derived from Table 5 (after minimization):

$$D_1 = z_1\bar{v_1} \vee z_1 z_2 \vee z_3 v_1. \tag{37}$$

To get the SBF (37), we used "don't" care assignments 1110 and 1111.

The last step of the proposed method is technology mapping (Wolf, 2004; Barkalov *et al.*, 2021b). It is executed using various CAD tools such as Vivado by Xilinx (Vivado, 2021). We do not discuss this step. We only estimate the LUT count for the circuit of $P_{YC}Y$ Mealy FSM A_o.

As follows from Table 4, there are four LUTs in the circuit of BPF1 and three LUTs in BPF2. So, there are $2R_Y + R_I = 7$ LUTs on the first level of logic.

As follows from (35), there are $R_Y + R_I = 4$ LUTs in the circuit of BF. These LUTs form the second level of logic.

As follows from (33), there are $N = 6$ LUTs in the circuit of BO. From Table 5 it can be deduced that there are $R_{T_C} = 3$ LUTs in the circuit of the BTCC. In consequence, there are nine LUTs on the third level of logic.

That means that the circuit of $P_{YC}Y$ FSM A_o includes $3R_Y+2R_I+N+R_{TC} = 20$ LUTs with $S_L = 4$. In the case of $P_{YE}Y$ FSM A_o, we get $K = 3$. The blocks BO and BF have the same amount of LUTs (10) as is the case with $P_{YC}Y$ FSM A_o. But there are six LUTs in BESC and 11 LUTs in blocks BPF1–BPF3. As a result, there are 27 LUTs in the circuit of $P_{YE}Y$ FSM A_o.

In the discussed case, the replacement of $P_{YE}Y$ model by $P_{YC}Y$ FSM gives 35% of gain in LUTs. As our experiments show, such a replacement gives the gain practically for 78.8% of benchmarks (LGSynth93, 1993).

6. Experimental results

In this section, we compare the proposed approach ($P_{YC}Y$ Mealy FSMs) with some known methods including $P_{YE}Y$ FSMs. In the experiments, we use the library of standard benchmarks LGSynth93 (LGSynth93, 1993). This library includes 48 benchmarks of different complexness (different numbers of inputs, outputs and states). These benchmarks are used very often by different researchers to compare the basic characteristics of FSM circuits obtained with different design methods. The known format KISS2 is used to represent the benchmarks. The characteristics of benchmarks (numbers of inputs, outputs, states, state variables and interstate transitions) are shown in Table 6. We will discuss the meaning of the column "Class" further on.

As follows from Table 6, benchmarks have a wide range of different quantitative characteristics. The number of states varies from 5 to 172, the number of inputs is in the range from 1 to 19 (the same is true for outputs), the number of transitions is in the range from 11 to 370. As noted in LGSynth93 (1993), most of these benchmarks are taken from practice, that is, they correspond to real FSMs.

To conduct the experiments, as in our previous works (Barkalov *et al.*, 2020a; 2021a; 2021b), we use a personal computer having the following characteristics: CPU: Intel Core i7 6700K 4.2@4.4 GHz, memory: 16 GB RAM 2400 MHz CL15. For implementing LUT-based FSM circuits, the Virtex-7 VC709 Evaluation Platform (xc7vx690tffg1761-2) (Xilinx, 2020) is used. The FPGA chip of this platform includes slices having four LUTs with six inputs. The internal multiplexers of the slices can be used for implementing fast multiplexers from 4:1 to 16:1.

The step of circuit implementation was executed by the CAD tool Vivado v2019.1 (64-bit) (Vivado, 2021). To compare equivalent Mealy FSMs, we used the values of LUT counts and maximum operating frequencies.

Table 5. Table of the BTCC of $P_{YC}Y$ FSM A_o.

j	$K(Y_q)$			$K(I_g)$	a_m	TC(a_m)			Φ_j
	z_1	z_2	z_3	V_1		T_1	T_2	T_3	
1	0	0	0	0	a_1	0	0	0	–
2	0	1	0	0	a_2	1	0	0	D_1
3	0	1	0	1	a_3	1	0	1	D_3
4	1	0	0	0	a_3	0	0	1	D_3
5	1	0	0	1	a_5	0	1	0	D_2
6	1	1	0	0	a_6	1	1	0	D_1D_2
7	1	1	0	1	a_7	1	1	1	D_1D_2
8	0	0	1	0	a_4	1	0	1	D_1D_3
9	0	0	1	1	a_8	0	1	1	D_2D_3
10	1	0	1	0	a_2	1	0	0	D_1
11	1	0	1	1	a_8	0	1	1	D_2D_3
12	0	1	1	0	a_6	1	1	0	D_1D_2

These values were taken from the reports of Vivado. VHDL-based FSM models were used as input data for Vivado. To execute the transformation of KISS2-based files into VHDL-based FSM models, we used our CAD tool K2F (Barkalov *et al*., 2020c).

We compared the area and time characteristics of $P_{YC}Y$ Mealy FSMs with the characteristics of circuits obtained with four other approaches. Three of them are P. Mealy FSMs based on the following state assignment methods: (i) Auto of Vivado (it uses binary state codes), (ii) One-hot of Vivado, (iii) JEDI. It is not known on which heuristics the state assignment methods used in the Vivado system are based. The algorithm JEDI was proposed by scientists at the University of California, Berkeley. This algorithm is quite well described in its technical documentation. Thus, we can say that we compare our results with those based on algorithms proposed by both UC Berkeley and Xilinx scientists. Also, we compared our approach with $P_{YE}Y$ FSMs proposed by Barkalov *et al*. (2020a).

Our previous research (Barkalov *et al.*, 2021b;2020a) shows that both the LUT count and the maximum operating frequency depend strongly on the relation between the value of $L + R$, on the one hand, and the number of inputs S_L, on the other. Consequently, we divided the benchmarks into five classes. They belong to the class of trivial FSMs (Class 0) if $R + L \leq 6$. The benchmarks from the class of simple FSMs (Class 1) are characterized by $R + L \leq 12$. The benchmarks create the class of average FSMs (Class 2) if $R + L \leq 18$. The class of big FSMs (Class 3) consists of the benchmarks with $R + L \leq 24$. At last, the most complex class of very big FSMs (Class 4) includes the benchmarks with $R + L > 24$. As Barkalov *et al*. (2021b) show, the larger the class number, the bigger the gain from using methods of structural decomposition.

Using relations between $L + R$ and S_L leads to the following classes of benchmarks. Class 0 consists of the benchmarks *bbtas*, *dk17*, *dk27*, *dk512*, *ex3*, *ex5*, *lion*, *lion9*, *mc*, *modulo12*, and *shiftreg*. Class 1 includes the benchmarks *bbara*, *bbsse*, *beecount*, *cse*, *dk14*, *dk15*, *dk16*, *donfile*, *ex2*, *ex4*, *ex6*, *ex7*, *keyb*, *mark1*, *opus*, *s27*, *s386*, *s840*, and *sse*. Class 2 contains the benchmarks *ex1*, *kirkman*, *planet*, *planet1*, *pma*, *s1*, *s1488*, *s1494*, *s1a*, *s208*, *styr*, and *tma*. Class 3 is created by a single benchmark *sand*. At last, benchmarks *s420*, *s510*, *s820*, and *s832* form Class 4. The class numbers are shown in the column "Class" (Table 6).

The results of the experiments are shown in Tables 7 (LUT counts) and 8 (maximum operating frequency, MHz). We organize these tables in the following manner. The table columns are marked by the names of investigated methods. The column "Class" shows the class of a particular benchmark. The names of the benchmarks are shown in the table rows. The benchmarks in the tables are shown in ascending order of the class number. There are results of summation of values from columns in the row "Total". The row "Percentage" includes the percentage of summarized characteristics of FSM circuits produced by other $P_{YC}Y$-based FSM methods. We marked in bold the best results in Tables 7 and 8. Analysis of Tables 7 and 8 leads to the following conclusions.

As follows from Table 7, the circuits of $P_{YC}Y$-based FSMs require the smallest number of LUTs compared with the other investigated methods. There is the following gain in the LUT counts: (i) 37.18% compared with Auto-Based FSMs, (ii) 59.64% compared with FSMs based on One-hot state assignment, (iii) 12.97% compared with JEDI-based FSMs, (iv) 9.86% compared with $P_{YE}Y$-based FSMs.

The analysis of LUT counts for different classes of benchmarks shows the following. For Class 0, our approach yields 1.83% of gain regarding equivalent

Table 6. Characteristics of benchmarks used in the experiments.

Benchmark	Inputs, L	Outputs, N	State variables + inputs, $R+L$	States/ state variables, M/R	Transitions, H	Class
bbara	4	2	8	12/4	60	1
bbsse	7	7	12	26/5	56	1
bbtas	2	2	6	9/4	24	0
beecount	3	4	7	10/4	28	1
cse	7	7	12	32/5	91	1
dk14	3	5	8	26/5	56	1
dk15	3	5	8	17/5	32	1
dk16	2	3	9	75/7	108	1
dk17	2	3	6	16/4	32	0
dk27	1	2	5	10/4	14	0
dk512	1	3	6	24/5	15	0
donfile	2	1	7	24/5	96	1
ex1	9	19	16	80/7	138	2
ex2	2	2	7	25/5	72	1
ex3	2	2	6	14/4	36	0
ex4	6	9	11	18/5	21	1
ex5	2	2	6	16/4	32	0
ex6	5	8	9	14/4	34	1
ex7	2	2	12	17/5	36	1
keyb	7	7	12	22/5	170	1
kirkman	12	6	18	48/6	370	2
lion	2	1	5	5/3	11	0
lion9	2	1	6	11/4	25	0
mark1	5	16	10	22/5	22	1
mc	3	5	6	8/3	10	0
modulo12	1	1	5	12/4	24	0
opus	5	6	10	18/5	22	1
planet	7	19	14	86/7	115	2
planet1	7	19	14	86/7	115	2
pma	8	8	14	49/6	73	2
s1	8	7	14	54/6	106	2
s1488	8	19	15	112/7	251	2
s1494	8	19	15	118/7	250	2
s1a	8	6	15	86/7	107	2
s208	11	2	17	37/6	153	2
s27	4	1	8	11/4	34	1
s386	7	7	12	23/5	64	1
s420	19	2	27	137/8	137	4
s510	19	7	27	172/8	77	4
s8	4	1	8	15/4	20	1
s820	18	19	25	78/7	232	4
s832	18	19	25	76/7	245	4
sand	11	9	18	88/7	184	3
shiftreg	1	1	5	16/4	16	0
sse	7	7	12	26/5	56	1
styr	9	10	16	67/7	166	2
tma	7	9	13	63/6	44	2

$P_{YE}Y$-based FSMs. At the same time, our approach loses compared to all other models. This result can be explained by the fact that, for Class 0, each function from the SBFs (2) and (3) is represented by a circuit consisting of a single LUT. Therefore, there is no point in using methods of structural decomposition for FSMs of this class.

But starting from simple FSMs, our approach provides better LUT counts regarding all investigated models. Interestingly, for Classes 1–4, the gain related to $P_{YE}Y$-based FSMs is almost the same. It is 11.31% for simple FSMs, and 10.32% for other classes. We think this is due to the difference in the length of the state codes, for states from classes $A^k \in \Pi_A$ and $B^k \in \Pi_B$. In the case of extended state codes the number of state variables is determined by (12). In the case of two-component state codes this number is determined by (20). This follows from the comparison of the formulae (12) and (20) that for equivalent FSMs the partition Π_B includes fewer classes than the partition Π_A. We assume that for different classes of FSMs (average, big, and complex) there is approximately the same ratio between the parameters K and $K1$. For equivalent $P_{YC}Y$ and $P_{YE}Y$ this phenomenon determines approximately the same difference in the number of LUTs.

Starting from simple FSMs, the gain from the application of our method increases relatively to equivalent Auto-, One-hot-, and JEDI-based FSMs. This is connected with the following. We think that the most important benchmark characteristic is the difference between the number of LUT inputs (S_L) and the total number of FSM inputs (L) and the minimum number of state variables (R) determined by Eqn. (1). The greater the difference $(L+R)-S_L$, the greater the gain that can be expected from the application of our approach. Obviously, the difference $(L + R) - S_L$ increases as the benchmark class number increases. This explains the increase in gain from the application of two-component state codes as the complexity of benchmarks increases.

In addition to the total number of FSM inputs (L), the LUT counts of FSM circuits are influenced by the branching of the STG. There is a direct relationship between $H(a_m)$ and $L(a_m)$, where $H(a_m)$ is the number of transitions from a state $a_m \in A$ and $L(a_m)t$ is the number of inputs determining these transitions (Achasova, 1987): $H(a_m) = L(a_m) + 1$. Theoretically, a situation is possible when the value of $L(a_m)$ exceeds the number of LUT inputs. In such a case, as follows from (21), our approach cannot be used. However, Sklyarov (2000) notes that for real FSMs the value of $L(a_m)$ does not exceed 3. A similar condition is met for benchmarks from the library LGSynth93. Thus, starting from simple FSMs (Class 1), our method allows improving LUT counts.

As follows from Table 8, our approach leads to the fastest FSM circuits (circuits with the highest operating frequency). There is the following gain in operating frequency: (i) 12.22% compared with Auto-Based FSMs, (ii) 12.93% compared with FSMs based on one-hot state assignment, (iii) 5.83% compared with JEDI-based FSMs, (iv) 2.74% compared with $P_{YE}Y$-based FSMs.

For all classes, our approach provides slightly better time characteristics regarding equivalent $P_{YE}Y$-based FSMs. It has a very small gain (0.57%) for FSMs belonging to Class 0. For Class 1, there is a gain equal to 2.04%. For more complex FSMs, there is a gain equal to 4.09%. Therefore, the gain increases slightly as the value of $L + R$ increases. We think that this gain is mainly due to a decrease in the number of interconnections, which is a consequence of the fulfilment of the condition (32).

Also, we compared $P_{YC}Y$-based FSMs with P_YY FSMs. As follows from the work of Barkalov *et al.* (2020a), it is necessary for 1320 LUTs to implement the circuits for all benchmarks (LGSynth93, 1993). Using our approach, it is necessary to have 1318 LUTs. Thus, the LUT-based circuits of $P_{YC}Y$-based FSMs have practically the same LUT counts as equivalent P_YY FSMs. There is 7873.36 MHz in the row "Total" for P_YY FSMs (LGSynth93, 1993). There is 9258.84 MHz in the row "Total" for $P_{YC}Y$-based FSMs (Table 8). Therefore, our approach yields a 15% increase in frequency compared with equivalent P_YY FSMs.

The data shown in Tables 7 and 8 are obtained when using the directive "Default" at the stage of project optimization (opt_design) by Vivado. In this case, Vivado uses default settings during the logic design stage. But we also compared the implementation results using the directive "ExploreArea." This led to a slight decrease in the values of LUT counts in FSM circuits based on the methods Auto, One-hot and JEDI (up to 2.8%). At the same time, the use of this directive did not improve LUT counts for the circuits of both $P_{YC}Y$ and $P_{YE}Y$ FSMs. For Auto-, One-hot-, and JEDI-based FSMs, the optimization of LUT counts led to a slight decrease in performance (up to 4.6%). At the same time, the performance did not change for the circuits of both $P_{YC}Y$ and $P_{YE}Y$ FSMs. The main goal of the proposed method is to reduce the number of LUTs in FSM circuits. Consequently, we did not study the influence of other Vivado modes on the FSM characteristics.

As follows from Tables 7 and 8, using two-component state codes allows obtaining FSM circuits with better area (LUT count) and time (maximum operating frequency) characteristics compared with the equivalent $P_{YE}Y$ FSMs based on extended state codes. Note that the proposed method makes it possible to reduce the number of LUTs related to the equivalent $P_{YE}Y$ FSMs starting from the benchmarks of Class 1. Also, our approach produces better results compared with results obtained using various methods of state assignment (Auto, One-hot, and JEDI) and functional decomposition for P FSMs. If we compare our approach with P_YY

Table 7. Results of the experiments (LUT count).

Benchmark	P-Auto	P-One-hot	P-JEDI	$P_{YE}Y$	$P_{YC}Y$	Class
bbtas	**5**	**5**	**5**	9	9	0
dk17	5	12	**5**	10	10	0
dk27	**3**	5	**4**	9	9	0
dk512	10	10	**9**	14	12	0
ex3	**9**	**9**	**9**	14	14	0
ex5	**9**	**9**	**9**	12	12	0
lion	**2**	5	**2**	8	8	0
lion9	6	11	**5**	10	10	0
mc	**4**	7	**4**	8	8	0
modulo12	**7**	**7**	**7**	11	11	0
shiftreg	**2**	6	**2**	6	6	0
bbara	17	17	**10**	14	13	1
bbsse	33	37	**24**	29	26	1
beecount	19	19	**14**	16	15	1
cse	40	66	36	35	**31**	1
dk14	16	27	**10**	14	13	1
dk15	15	16	12	11	**10**	1
dk16	15	34	**12**	13	**12**	1
donfile	31	31	24	24	**21**	1
ex2	9	9	**8**	10	9	1
ex4	15	13	12	13	**11**	1
ex6	24	36	22	23	**21**	1
ex7	**4**	5	**4**	8	7	1
keyb	43	61	40	40	**37**	1
mark1	23	23	20	21	**19**	1
opus	28	28	22	23	**21**	1
s27	**6**	18	**6**	8	7	1
s386	26	39	22	22	**19**	1
s8	**9**	**9**	**9**	11	10	1
sse	33	37	30	29	**25**	1
ex1	70	74	53	44	**39**	2
kirkman	42	58	39	35	**32**	2
planet	131	131	88	82	**76**	2
planet1	131	131	88	82	**76**	2
pma	94	94	86	76	**71**	2
s1	65	99	61	58	**52**	2
s1488	124	131	108	93	**88**	2
s1494	126	132	110	94	**89**	2
s1a	49	81	43	42	**36**	2
s208	12	31	**10**	11	**10**	2
styr	93	120	81	78	**68**	2
tma	45	39	39	34	**28**	2
sand	132	132	114	103	**94**	3
s420	10	31	**9**	10	**9**	4
s510	48	48	32	23	**19**	4
s820	88	82	68	56	**49**	4
s832	80	79	62	52	**46**	4
Total	1808	2104	1489	1448	**1318**	
Percentage	137,18	159,64	112,97	109,86	100,00	

FSMs, then both models require the same number of LUTs. However, our approach allows obtaining FSM circuits with significantly better performance. Therefore, we think that our new approach can be used in CAD tools targeting LUT-based FSM synthesis.

7. Conclusion

Nowadays, a wide variety of digital systems are implemented using various FPGA chips. One of the important problems associated with FPGA-based design is that of reducing the chip area occupied by the digital system circuit. At the same time, it is desirable that the reduction in the area is not associated with a sharp decrease in performance. This fully applies to LUT-based synthesis of FSM circuits. To improve the area, it is necessary to reduce the numbers of literals in sum-of-products of Boolean functions representing an FSM circuit. In many scientific works, the required chip area for a given circuit is estimated as a LUT count (Islam *et al*., 2020).

Structural decomposition (Barkalov *et al*., 2021b) is one of the efficient ways to diminish LUT counts in an FSM logic circuit. All methods of structural decomposition lead to a change in the FSM structural diagram compared with single-level P Mealy FSMs. Of particular interest are methods of structural decomposition that can simultaneously reduce the LUT count and increase the maximum operating frequency. One of such methods is proposed in this paper.

The method is aimed at improving the characteristics of LUT-based $P_{YE}Y$ Mealy FSMs with the transformation of collections of outputs into extended state codes. Our new approach is based on using two-component state codes. One of the components is a code of some class including a particular state. The second part represents this state as the class element. Due to this approach, we simplified the code transformer used in $P_{YE}Y$ Mealy FSMs. Also, the number of state variables is significantly reduced compared with equivalent $P_{YE}Y$ Mealy FSMs. As a result, compared with equivalent $P_{YE}Y$ Mealy FSMs, our approach gives an improvement in both the LUT count (on average, by 9.86%) and maximum operating frequency (on average, by 2.74%).

The results of experiments show that the proposed approach allows reducing LUT counts in FSM circuits. Moreover, this improvement is associated with a slight increase in FSM performance. Thus, the proposed method improves two main characteristics of FSM circuits. We think that it has a rather good potential to be used in LUT-based FSM design.

References

Achasova, S. (1987). *Synthesis Algorithms for Automata with PLAs*, M: Soviet Radio, Moscow.

Altera (2021). Corporate website, `http://www.altera.com`, (currently: Intel Corporation, `https://www.intel.com/content/www/us/en/products/programmable.html`).

Amano, H. (2018). *Principles and Structures of FPGAs*, Springer Singapore, Singapore.

Atmel (2021). Corporate website, `http://www.atmel.com`, (currently: Microchip Technology, `https://www.microchip.com/`).

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

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., Titarenko, L. and Krzywicki, K. (2021b). Structural decomposition in FSM design: Roots, evolution, current state—A review, *Electronics* **10**(10): 1–44.

Barkalov, A., Titarenko, L., Krzywicki, K. and Saburova, S. (2020a). Improving the characteristics of multi-level LUT-based Mealy FSMs, *Electronics* **9**(11): 1–34.

Barkalov, A., Titarenko, L., Mazurkiewicz, M. and Krzywicki, K. (2021a). Improving LUT count of FPGA-based sequential blocks, *Bulletin of the Polish Academy of Sciences: Technical Sciences* **69**(2): 1–12, DOI: 10.24425/bpasts.2021.136728.

Barkalov, A., 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.

Barkalov, A., Titarenko, L. and Mielcarek, K. (2020b). Improving characteristics of LUT-based Mealy FSMs, *International Journal of Applied Mathematics and Computer Science* **30**(4): 745–759, DOI: 10.34768/amcs-2020-0055.

Barkalov, A., Titarenko, L., Mielcarek, K. and Chmielewski, S. (2020c). *Logic Synthesis for FPGA-Based Control Units - Structural Decomposition in Logic Design*, Lecture Notes in Electrical Engineering, Vol. 636, Springer, Berlin, DOI: 10.1007/978-3-030-38295-7.

Borowczak, M. and Vemuri, R. (2013). Secure controllers: Requirements of S*FSM, *2013 IEEE 56th International Midwest Symposium on Circuits and Systems (MWSCAS), Columbus, USA*, pp. 553–557, DOI: 10.1109/MWSCAS.2013.6674708.

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.

Table 8. Results of experiments (maximum operating frequency, MHz).

Benchmark	P-Auto	P-One-hot	P-JEDI	$P_{YE}Y$	$P_{YC}Y$	Class
bbtas	204,16	204,16	**206,12**	201,47	202,31	0
dk17	199,28	167,00	199,39	172,99	173,14	0
dk27	**206,02**	201,9	204,18	190,32	191,12	0
dk512	196,27	196,27	**199,75**	187,45	188,52	0
ex3	194,86	194,86	**195,76**	187,26	188,35	0
ex5	180,25	180,25	**181,16**	162,56	163,19	0
lion	202,43	**204,00**	202,35	195,73	196,17	0
lion9	205,3	185,22	**206,38**	183,45	184,82	0
mc	196,66	195,47	**196,87**	182,95	183,42	0
modulo12	207,00	207,00	**207,13**	201,74	203,01	0
shiftreg	262,67	263,57	276,26	253,72	**257,64**	0
bbara	193,39	193,39	**212,21**	210,21	**214,47**	1
bbsse	157,06	169,12	182,34	193,43	**197,36**	1
beecount	166,61	166,61	187,32	194,47	**198,72**	1
cse	146,43	163,64	178,12	182,62	**186,53**	1
dk14	191,64	172,65	193,85	201,39	**205,06**	1
dk15	192,53	185,36	194,87	206,74	**210,41**	1
dk16	169,72	174,79	197,13	**199,14**	**203,19**	1
donfile	184,03	184	203,65	206,83	**210,36**	1
ex2	198,57	198,57	**200,14**	196,58	199,89	1
ex4	180,96	177,71	192,83	196,18	**200,32**	1
ex6	169,57	163,8	176,59	187,53	**191,85**	1
ex7	200,04	200,84	200,6	204,16	**208,73**	1
keyb	156,45	143,47	168,43	178,59	**182,92**	1
mark1	162,39	162,39	176,18	182,37	**186,72**	1
opus	166,2	166,2	178,32	186,34	**190,42**	1
s27	198,73	191,5	199,13	201,26	**205,42**	1
s386	168,15	173,46	179,15	192,34	**196,38**	1
s8	180,02	178,95	181,23	191,32	**195,26**	1
sse	157,06	169,12	174,63	171,18	**175,41**	1
kirkman	141,38	154,00	156,68	184,62	**192,36**	2
ex1	150,94	139,76	176,87	180,72	**187,14**	2
planet	132,71	132,71	187,14	212,45	**220,65**	2
planet1	132,71	132,71	187,14	212,45	**220,65**	2
pma	146,18	146,18	169,83	192,43	**200,38**	2
s1	146,41	135,85	**157,16**	145,32	153,98	2
s1488	138,5	131,94	157,18	182,14	**190,42**	2
s1494	149,39	145,75	164,34	186,49	**194,73**	2
s1a	153,37	176,4	169,17	188,92	**192,14**	2
s208	174,34	176,46	178,76	192,15	**200,37**	2
styr	137,61	129,92	145,64	164,52	**172,47**	2
tma	163,88	147,8	164,14	182,72	**191,03**	2
sand	115,97	115,97	126,82	143,14	**155,42**	3
s420	173,88	176,46	177,25	218,62	**234,21**	4
s510	177,65	177,65	198,32	221,19	**237,12**	4
s820	152,00	153,16	176,58	195,73	**210,34**	4
s832	145,71	153,23	173,78	199,18	**214,32**	4
Total	8127,08	8061,22	8718,87	9005,11	**9258,84**	
Percentage	87,78	87,07	94,17	97,26	100,00	

Chapman, K. (2014). Multiplexer design techniques for datapath performance with minimized routing resources, *Xilinx Application Note 522 (v1.2)*, https://www.xilinx.com/support/documentation/application_notes/xapp522-mux-design-techniques.pdf.

Das, N. and Panchanathan, A. (2018). FPGA implementation of reconfigurable finite state machine with input multiplexing architecture using Hungarian method, *International Journal of Reconfigurable Computing* **2018**, Article ID: 6831901, DOI: 10.1155/2018/6831901.

Feng, W., Greene, J. and Mishchenko, A. (2018). Improving FPGA performance with a S44 LUT structure, *FPGA'18: Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, USA*, pp. 61–66, DOI: 10.1145/3174243.3174272.

Islam, M.M., Hossain, M.S., Shahjalal, M., Hasan, M.K. and Jang, Y.M. (2020). Area-time efficient hardware implementation of modular multiplication for elliptic curve cryptography, *IEEE Access* **8**: 73898–73906.

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**: 16123–161131, DOI: 10.1109/ACCESS.2019.2895206.

Kubica, M., Opara, A. and Kania, D. (2017). Logic synthesis for FPGAs based on cutting of BDD, *Microprocessors and Microsystems* **52**(C): 173–187, DOI: 10.1016/j.micpro.2017.06.010.

Kubica, M., Opara, A. and Kania, D. (2021). *Technology Mapping for LUT-Based FPGA*, Lecture Notes in Electrical Engineering, Vol. 13, Springer International Publishing, Cham.

LGSynth93 (1993). Benchmark suite, https://people.engr.ncsu.edu/brglez/CBL/benchmarks/LGSynth93/LGSynth93.tar.

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, DOI: 10.1109/TCAD.2018.2878187.

Marwedel, P. (2018). *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things*, 3rd Edn, Springer International Publishing, Cham, DOI: 10.1007/978-3-319-56045-8.

Maxfield, C. (2008). *FPGAs: Instant Access*, Newnes, Burlington.

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

Minns, P. and Elliot, I. (2008). *FSM-based Digital Design Using Verilog HDL*, John Wiley and Sons, Chichester.

Ruiz-Rosero, J., Ramirez-Gonzalez, G. and Khanna, R. (2019). Field programmable gate array applications—A scientometric review, *Computation* **7**(4): 63, DOI: 10.3390/computation7040063.

Sasao, T. and Mishchenko, A. (2009). LUTMIN: FPGA logic synthesis with MUX-based and cascade realizations, *International Workshop on Logic Synthesis, Berkeley, USA,* pp. 310–316.

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

Senhadji-Navaro, R. and Garcia-Vargas, I. (2015). High-speed and area-efficient reconfigurable multiplexer bank for RAM-based finite state machine implementations, *Journal of Circuits, Systems and Computers* **24**(07): 1550101:1–1550101:15, DOI: 10.1142/S0218126615501017.

Skliarova, I., Sklyarov, V. and Sudnitson, A. (2012). *Design of FPGA-based Circuits using Hierarchical Finite State Machines*, TUT Press, Tallinn.

Sklyarov, V. (2000). Synthesis and implementation of RAM-based finite state machines in FPGAs, *in* R.W. Hartenstein and H. Grünbacher (Eds), *Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing*, Springer, Berlin/Heidelberg, pp. 718–727.

Sklyarov, V., Skliarova, I., Barkalov, A. and Titarenko, L. (2014). *Synthesis and Optimization of FPGA-Based Systems*, Lecture Notes in Electrical Engineering, Vol. 294, Springer-Verlag, Berlin.

Solovjev, V. and Czyzy, M. (1999). Refined CPLD macrocells architecture for effective FSM implementation, *Proceedings of the 25th EUROMICRO Conference, Milan, Italy*, Vol. 1, pp. 102–109.

Sutter, G., Todorovich, E., López-Buedo, S. and Boemo, E. (2002). Low-power FSMs in FPGA: Encoding alternatives, *in* B. Hochet *et al.* (Eds), *Integrated Circuit Design: Power and Timing Modeling, Optimization and Simulation*, Springer-Verlag, Berlin/Heidelberg, pp. 363–370.

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, Paris, France,* Vol. 2, pp. 916–921.

Trimberg, S. (2015). Three ages of FPGA: A retrospective on the first thirty years of FPGA Technology, *IEEE Proceedings* **103**(3): 318–331.

Vivado (2021). CAD tools website, https://www.xilinx.com/products/design-tools/vivado.html.

Wolf, W. (2004). *FPGA-Based System Design*, Prentice Hall PTR, Upper Saddle River.

Xilinx (2020). *VC709 Evaluation Board for the Virtex-7 FPGA*, https://www.xilinx.com/support/documentation/boards_and_kits/vc709/ug887-vc709-eval-board-v7-fpga.pdf.

Xilinx (2021). Corporate website, http://www.xilinx.com.

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

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 a professor at the Department of Computer, Electrical and Control Engineering of the University of Zielona Góra, Poland.

Małgorzata Mazurkiewicz received her MSc degree from the Technical University of Zielona Góra in 1999 and her PhD degree in 2007 from the University of Zielona Góra, both in computer science. Since then, she has been an assistant professor at the Faculty of Computer, Electrical and Control Engineering, University of Zielona Góra.

Received: 9 December 2021
Revised: 22 February 2022
Re-revised: 18 March 2022
Accepted: 23 March 2022