Int. J. Appl. Math. Comput. Sci., 2024, Vol. 34, No. 1, 167–178
DOI: 10.61822/amcs-2024-0012

REDUCING THE NUMBER OF LUTs FOR MEALY FSMs WITH STATE TRANSFORMATION

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

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

In many digital systems, various sequential blocks are used. This paper is devoted to the case where the model of a Mealy finite state machine (FSM) represents the behaviour of a sequential block. The chip area occupied by an FSM circuit is one of the most important characteristics used in logic synthesis. In this paper, a method is proposed which aims at reducing LUT counts for FPGA-based Mealy FSMs with transformation of state codes into FSM outputs. This is done using the combined state codes. Such an approach allows excluding a block of transformation of binary state codes into extended state codes. The proposed method leads to LUT-based Mealy FSM circuits having exactly three levels of logic blocks. Under certain conditions, each function for any logic level is represented by a circuit including a single LUT. The proposed approach is illustrated with an example of synthesis. The results of experiments conducted using standard benchmarks show that the proposed method produces LUT-based FSM circuits with significantly smaller LUT counts than is the case for circuits produced by other investigated methods (Auto and One-hot of Vivado, JEDI, and transformation of binary codes into extended state codes). The LUT count is decreased by an average of 17.96 to 91.8%. Moreover, if some conditions are met, the decrease in the LUT count is accompanied with a slight improvement in the operating frequency compared with circuits based on extended state codes. The advantages of the proposed method multiply with increasing the numbers of FSM inputs and states.

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

1. Introduction

Modern digital systems (Marwedel, 2018; Sklyarov *et al.*, 2014) include a lot of sequential blocks, starting with rather simple binary counters and ending with very sophisticated digital controllers (Gajski *et al.*, 2009; Borowczak and Vemuri, 2013). Very often, the behaviour of a sequential block is represented by the model of Mealy finite state machine (FSM) (Baranov, 1994; Micheli, 1994). In this paper, we discuss the case where Mealy FSMs circuits are implemented using internal resources of field-programmable gate arrays (FPGAs) (Intel, 2023; AMD, 2023a). This case is of great practical interest since a huge number of various projects are implemented on the basis of FPGAs (Ruiz-Rosero *et al.*, 2019).

Several optimization problems arise when implementing FPGA-based FSM circuits (Kubica *et al.*, 2021; Kubica and Kania, 2017; Barkalov *et al.*, 2018). The main issues are a decrease in the chip area occupied by an FSM circuit, an increase in performance (the maximum operating frequency), and reducing the power consumption (Barkalov and Barkalov Jr., 2005; Sklyarov *et al.*, 2014; Tiwari and Tomko, 2004). As a rule, reducing the area also reduces the power consumption (Barkalov *et al.*, 2021). At the same time, it is very important that a reduction in the area does not lead to a significant decrease in the performance (Kubica *et al.*, 2019; Maxfield, 2008). In this paper, we propose a method of area reduction which does not lead to a reduction in the maximum operating frequency. We discuss the case where an FPGA-based FSM circuit is implemented using look-up table (LUT) elements (Trimberger, 2015; Machado and Cortadella, 2020). We use the approach from the paper by Islam *et al.* (2020), where the chip area is estimated as an LUT count.

[*] Corresponding author

Modern LUTs have rather small amounts of inputs

(Intel, 2023; Microchip, 2023; AMD, 2023a). As a result, the methods of functional decomposition (Scholl, 2001; Solovjev and Czyzy, 1999) should be used to represent an FSM circuit by a network of interconnected LUTs. These circuits have many levels of logic and complex systems of spaghetti-type interconnections (Barkalov *et al.*, 2020a). Therefore, it is very important to develop FSM design methods, where the negative impact of a limited number of LUT inputs is minimized. The methods of structural decomposition (Barkalov *et al.*, 2021) are aimed at solving this problem.

The main contribution of this paper is a novel method of reducing LUT counts in logic circuits of Mealy FSMs. The method is based on a transformation of combined state codes, proposed in this paper, into FSM outputs. It is an improvement of the approach of Barkalov *et al.* (2022), where the transformation of maximum binary state codes into extended state codes is connected with the presence of a special block. This block generates extended state codes using codes of collections of outputs and identifiers. Thus, the main difference between our technique and the known FSM synthesis methods is the development of a new approach for representing state codes. As follows from our experiments, the proposed representation of state codes allows improving LUT counts in Mealy FSM circuits compared with the circuits produced by other investigated methods. In addition, due to the resulting decrease in the number of interconnections among the LUTs of an FSM circuit, our approach leads to a slight increase in the FSM performance.

The rest of the paper is organized as follows. Section 2 includes the basic information on LUT-based design of FSM circuits. The relative works are discussed in Section 3. The main idea of the proposed method is discussed in Section 4. Section 5 shows an example of synthesis. Section 6 includes experimental results. A concise conclusion ends the paper.

2. Synthesis of LUT-based FSMs

A Mealy FSM is represented using three sets and two functions (Baranov, 1994; Micheli, 1994). A set $A = \{a_1, \ldots, a_M\}$ includes internal states, a set $X = \{x_1, \ldots, x_L\}$ consists of FSM inputs, a set $Y = \{y_1, \ldots, y_N\}$ includes FSM outputs. A function of transition sets the dependence of the states of transitions on the current states and FSM inputs. A function of output shows the dependence of FSM outputs on the current states and FSM inputs. The state $a_1 \in A$ is an initial state.

An FSM can be represented by state transition tables (Micheli, 1994; Minns and Elliot, 2008), binary decision diagrams (Anceau, 1986; Milik, 2016; Kubica *et al.*, 2019), and-inverter graphs (Brayton and Mishchenko, 2010) and graph-schemes of algorithms (Baranov, 2008).

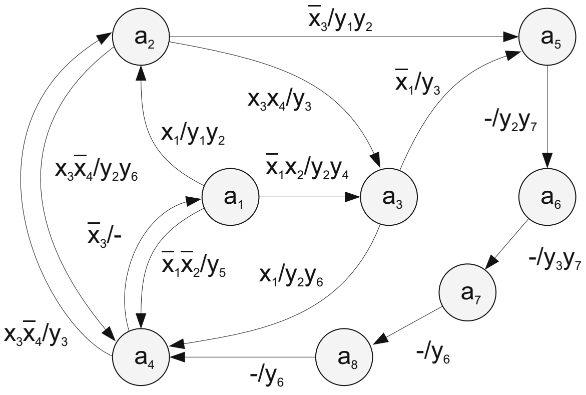


Fig. 1. State transition graph of Mealy FSM S_1.

In this paper, FSMs are represented by state transition graphs (STGs) (Micheli, 1994; Senhadji-Navarro and Garcia-Vargas, 2018). An example of an STG for an FSM S_1 is shown in Fig. 1.

STG nodes correspond to FSM states. As follows from Fig. 1, there are $M = 8$ states in FSM S_1. STG arcs correspond to interstate transitions. There are $H = 15$ arcs in the STG (Fig. 1). An arc is directed from a current state $a_m \in A$ to a state of transition $a_s \in A$. The arc number h ($h \in \{1, \ldots, H\}$) is marked by a pair $\langle X_h, Y_h \rangle$. The symbol X_h stands for a conjunction of FSM inputs (or their compliments) causing the h-th transition. The symbol Y_h stands for a collection of outputs (CO) generated during the h-th transition.

The analysis of Fig. 1 allows obtaining the sets $X = \{x_1, \ldots, x_4\}$ and $Y = \{y_1, \ldots, y_7\}$. This gives the values $L = 4$ and $N = 7$. Obviously, the state $a_1 \in A$ is an initial state of S_1.

An FSM circuit is represented by some systems of Boolean functions (SBFs) (Baranov, 1994; Micheli, 1994). To get these SBFs, the step of state assignment should be executed. During this step, states $a_m \in A$ are represented by R-bit codes $K(a_m)$. The minimum value of R is determined as

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

The formula (1) determines maximal binary state codes (Sutter *et al.*, 2002).

The bits of $K(a_m)$ correspond to state variables forming the set $T = \{T_1, \ldots, T_R\}$. State codes are kept in a state register (RG) consisting of R flip-flops. To load the code $K(a_1)$ into the RG, a special pulse $Start$ is used. To change the RG contents, input memory functions (IMFs) are used. There are R IMFs forming the set Φ. As a rule, D flip-flops are used in state registers (Skliarova *et al.*, 2012). In our paper, we use this approach. Thus, there is a set $\Phi = \{D_1, \ldots, D_R\}$. The synchronization pulse $Clock$ allows changing the contents of the RG.

Using state codes allows transforming an STG into a

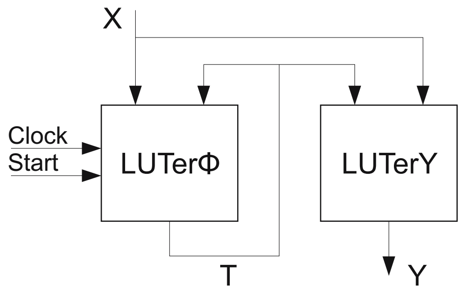


Fig. 2. Structural diagram of P Mealy FSMs.

direct structure table (DST) (Baranov, 1994; Klimovich and Solov'ev, 2012). The following SBFs are derived from a DST:

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

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

The system (2) corresponds to the function of output, the system (3) corresponds to the function of transitions.

In this paper, we discuss the case where the SBFs (2) and (3) are implemented using LUT-based configurable logic blocks (CLBs). A CLB includes LUTs, flip-flops, multiplexers and internal connections (Chapman, 2014). The flip-flops can be used as a hidden distributed state register (Sklyarov *et al.*, 2014). In our previous work (Barkalov *et al.*, 2018), we employed the symbol LUTer to define a logic block consisting of LUTs. Using this symbol, we can show a structural diagram of P Mealy FSM (Fig. 2).

In the P FSM, the LUTerΦ implements the SBF (3). The outputs of LUTs generating IMFs $D_r \in \Phi$ are connected with the inputs of flip-flops. Therefore, there is a distributed RG inside LUTerΦ. LUTerY implements the SBF (2).

The largest manufacturer of FPGAs is the company AMD (Xilinx) (AMD, 2023a). Hence our research is focused on Xilinx FPGAs from the Virtex-7 family (AMD, 2019). A LUT of Virtex-7 has $S_L = 6$ inputs (AMD, 2023a). A slice of Virtex-7 consists of four 6-LUTs and eight flip-flops. Using internal multiplexers (MX) allows creating either two 7-LUTs or a single 8-LUT (Chapman, 2014).

As follows from the results of Baranov (1994; 2008), a sum-of-products (SOP) of functions $f_i \in Y \cup \Phi$ can include up to 30–40 literals. Thus, there is an obvious contradiction between the extremely limited value of S_L and the very large number of arguments in functions $f_i \in Y \cup \Phi$. This contradiction leads to multi-level circuits of LUTerΦ and LUTerY (Barkalov *et al.*, 2020a). It is known (Wolf, 2004) that multi-level circuits have worse characteristics than their single-level counterparts. Therefore, it is necessary to decrease the number of logic levels in circuits of LUT-based Mealy FSMs.

3. Related work

The number of literals in each function representing an FSM circuit can be reduced by using various methods of structural decomposition (SD) (Barkalov *et al.*, 2021; Senhadji-Navaro *et al.*, 2015). The main idea of SD is the following. An FSM circuit is represented as a composition of logic blocks. Each block has its unique systems of inputs and outputs. SD assumes an increase in the number of logic levels of an FSM circuit. However, each of these blocks is represented by an SOP that has much fewer literals than the SOPs of the functions (2)–(3). Numerous studies (Barkalov *et al.*, 2021) have shown that this approach can significantly reduce the number of LUTs compared with that for P FSMs.

One of such methods is discussed by Barkalov *et al.* (2022). It is a method of transformation of states into COs $Y_q \subseteq Y$. A particular CO is represented using states and identifiers $I_j \in SI = \{I_1, \ldots, I_J\}$. Thus, each CO $Y_q \subseteq Y$ is represented by a pair $P_g = \langle a_m, I_j \rangle$ ($g \in \{1, \ldots, G\}$).

The need for identifiers is explained as follows. For example, there are two transitions into state $a_5 \in A$. A CO Y_3 is generated during the first one, a CO Y_7 is generated in the second case. To distinguish these COs, two identifiers (I_1, I_2) are necessary. Now, these COs can be represented as pairs $P_1 = \langle a_5, I_1 \rangle$ and $P_2 = \langle a_5, I_2 \rangle$. For example, the pair P_1 determines Y_3 and P_2 determines Y_7. If N_m different COs are generated during transitions into a state $a_m \in A$ then N_m identifiers are necessary to distinguish these COs. The number of identifiers for a particular FSM is determined as

$$J = \max(N_1, \ldots, N_M). \tag{4}$$

To synthesize an FSM circuit, the identifiers should be encoded by binary codes $K(I_j)$. These include

$$R_V = \lceil \log_2 J \rceil \tag{5}$$

additional variables. We use elements of the set $V = \{v_1, \ldots, v_{R_V}\}$ to encode the identifiers $I_j \in SI$.

Now, codes $K(Y_q)$ are represented as

$$K(Y_q) = K(a_m) * K(I_j), \tag{6}$$

where * is a sign of concatenation. The codes (6) are transformed into one-hot codes of outputs. Such an approach leads to P_AY Mealy FSM (Barkalov *et al.*, 2020b). Its LUT-based structural diagram is shown in Fig. 3.

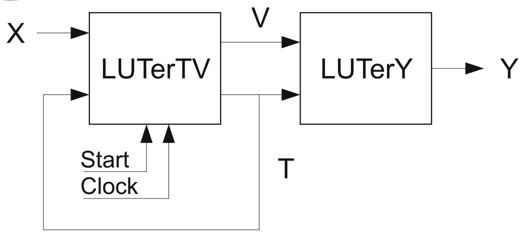


Fig. 3. Structural diagram of P_AY FSM.

In P_AY FSM, LUTerTV implements the SBFs (3) and

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

while LUTerY implements the SBF

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

Obviously, variables $v_r \in V$ replace inputs $x_e \in X$ in (8). As a rule, the following condition holds (Barkalov *et al.*, 2020b):

$$R_V \ll L. \tag{9}$$

Thus the SOPs of the functions (8) are much simpler than their counterparts for the functions (2). As a result, there are fewer LUTs in the circuit of P_AY FSM compared with the equivalent P Mealy FSM (Barkalov *et al.*, 2020b).

A SOP of function $f_i \in V \cup \Phi$ includes $NA(f_i)$ arguments. If the condition

$$NA(f_i) > S_L \tag{10}$$

holds, then the circuit of LUTerTV is multi-level. To decrease the number of LUTs in LUTerTV, it is possible to use the twofold state assignment (Barkalov *et al.*, 2018; 2020c). Such an approach is proposed by Barkalov *et al.* (2022).

The method of Barkalov *et al.* (2018) is based on finding a partition $\Pi_A = \{A^1, \ldots, A^K\}$ of the set A. Each class $A^k \in \Pi_A$ is characterized by sets X^k and $\mathcal{T}^k$. The set $X^k \subseteq X$ includes inputs $x_e \in X$ determining transitions from states $a_m \in A^k$. The set $\mathcal{T}^k \subseteq \mathcal{T}$ includes R_k state variables encoding states $a_m \in A^k$ by codes $C(a_m)$. The value of R_k is determined as

$$R_k = \lceil \log_2(|A^k| + 1) \rceil. \tag{11}$$

The value of $M_k = |A^k|$ is increased by 1 to take into account the relation $a_m \notin A^k$.

Each state $a_m \in A$ has two codes: $K(a_m)$ and $C(a_m)$. There are R_0 state variables in the set $\mathcal{T} = \mathcal{T}^1 \cup \mathcal{T}^2 \cup \cdots \cup \mathcal{T}^K$.

Each class $A^k \in \Pi_A$ determines two SBFs:

$$\Phi^k = \Phi^k(\mathcal{T}^k, X^k), \tag{12}$$

$$V^k = V^k(\mathcal{T}^k, X^k). \tag{13}$$

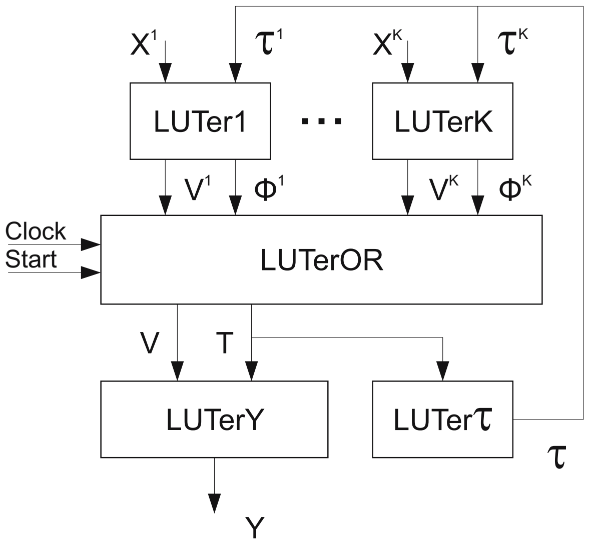


Fig. 4. Structural diagram of $P_{AT}Y$ Mealy FSM.

We name the functions (12)–(13) partial functions. To obtain the functions (3) and (7), the disjunctions of partial functions $D_r^k \in \Phi^k$ and $v_r^k \in V^k$ should be found. This approach determines a $P_{AT}Y$ Mealy FSM (Fig. 4).

In the P_{AT} FSM, LUTerk implements the partial functions (12) and (13), LUTerOR generates state variables $T_r \in T$ and additional variables $v_r \in V$, LUTerY implements the SBF (8). To create codes $C(a_m)$, LUTer$\mathcal{T}$ implements the following SBF:

$$\mathcal{T} = \mathcal{T}(T). \tag{14}$$

Some experiments (Barkalov *et al.*, 2022) show that $P_{AT}Y$ FSMs have better characteristics than the equivalent P_AY FSMs and FSMs based on various methods of state assignment. The following methods of state assignment were analyzed: Auto and One-hot of Vivado (Vivado, 2023) and JEDI (Sentowich *et al.*, 1992a; 1992b). The replacement of other models by $P_{AT}Y$ FSMs allows reduction of the LUT counts (from 11.9% to 62.6%) and increasing the maximum operating frequency (from 11.7% to 18.4%).

Our analysis shows that $P_{AT}Y$ FSMs have one serious drawback. Namely, the transformation $K(a_m) \to C(a_m)$ requires an additional block LUTer$\mathcal{T}$, which consumes some internal resources of a chip (LUTs and interconnections). Consequently, it would be useful to eliminate LUTer$\mathcal{T}$. Such an approach is proposed in the present paper.

4. Main idea of the proposed method

The proposed method can be applied if the condition (10) holds for a P_AY FSM. In this case, we suggest to find a partition $\Pi_B = \{B^1, \ldots, B^I\}$ of the set A. As for a

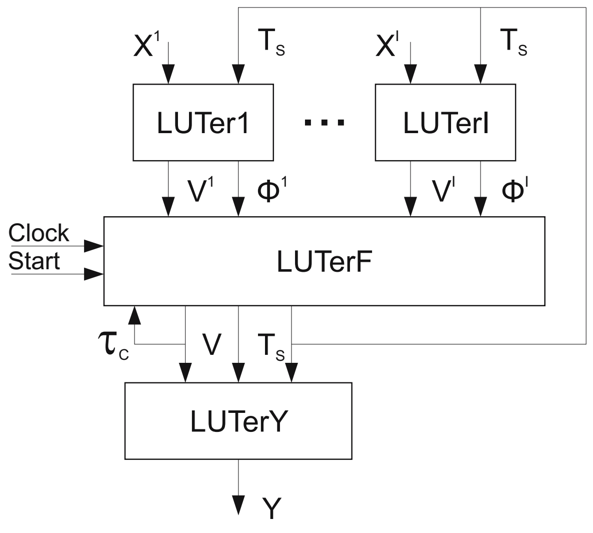


Fig. 5. Structural diagram of $P_{AC}Y$ Mealy FSM.

class $A^k \in \Pi_A$, each class $B^i \in \Pi_B$ defines a set $X^i \subseteq X$. The set X^i includes L_i inputs determining transitions from states $a_m \in B^i$.

If $|B^i| = M_i$, then it is enough to use

$$R_i = \lceil \log_2 M_i \rceil \tag{15}$$

state variables to encode states $a_m \in B^i$ by codes $C(a_m)$.

As in the case of Π_A, the following condition should hold for each class $B^i \in \Pi_B$:

$$R_i + L_i \leq S_L. \tag{16}$$

In this case, each partial function is implemented by a single LUT. It is enough to use

$$R_S = \max(R_1, \dots, R_I) \tag{17}$$

state variables to encode any state from any class $B^i \in \Pi_B$. These variables form a set $T_S = \{T_1, \dots, T_{RS}\}$.

Obviously, any state $a_m \in A$ should have its unique code. To reach this goal, we propose to encode a class $B^i \in \Pi_B$ with a binary code $C(B_i)$. There are

$$R_C = \lceil \log_2 I \rceil \tag{18}$$

bits in the codes $C(B^i)$. To encode classes, we use the elements of a set $\mathcal{T}_c = \{\tau_1, \dots, \tau_{RC}\}$.

Using codes $C(B^i)$ and $C(a_m)$, where $a_m \in B^i$, gives a combined state code (CSC),

$$CC(a_m) = C(B^i) * C(a_m). \tag{19}$$

In (19), the symbol * stands for the concatenation of codes. A $P_{AC}Y$ FSM based on (19) has the structural diagram shown in Fig. 5.

In $P_{AC}Y$ FSM, LUTeri implements the SBFs

$$\Phi^i = \Phi^i(T_S, X^i), \tag{20}$$

$$V^i = V^i(T_S, X^i), \tag{21}$$

while LUTerF implements the SBFs

$$\mathcal{T}_{CC} = \mathcal{T}_{CC}(\mathcal{T}_C, \Phi^1, \dots, \Phi^I), \tag{22}$$

$$V = V(\mathcal{T}_C, V^1, \dots, V^I). \tag{23}$$

The set $\mathcal{T}_{CC}$ includes class variables $\tau_r \in \mathcal{T}_C$ and state variables $T_r \in T_S$: $\mathcal{T}_{CC} = \mathcal{T}_C \cup T_S$. The block LUTerY implements FSM outputs represented as

$$Y = Y(\mathcal{T}_{CC}, V). \tag{24}$$

The set of IMFs includes R_{CC} elements:

$$R_{CC} = R_C + R_S. \tag{25}$$

Our analysis of standard benchmarks (LGSynth93, 1993) shows that the following relations hold:

$$2R < R_0 < 4R, \tag{26}$$

$$R \leq R_{CC} \leq R + 1, \tag{27}$$

$$R_C \leq 4. \tag{28}$$

Accordingly, $P_{AC}Y$ FSMs have the following advantages compared with the equivalent $P_{AT}Y$ FSMs:

1. There is no LUTer$\mathcal{T}$ in $P_{AC}Y$ FSMs. If there is the same LUT count for parts of circuit generating the SBFs (8), (12), (13) and (20)–(24), then $P_{AC}Y$ FSMs require fewer LUTs than the equivalent $P_{AT}Y$ FSMs.

2. $P_{AC}Y$ FSMs have fewer feedback signals between the RG and LUTs of the first level of logic. This allows simplification of the interconnects compared with $P_{AT}Y$ FSMs.

3. There is no special code to show that $a_m \notin B^i$. Hence the following condition can occur:

$$I < K. \tag{29}$$

Reducing the number of blocks LUTer of the first logic level can lead to a decrease in the LUT counts for $P_{AC}Y$ FSMs.

In this paper, we propose the following method for synthesis of $P_{AC}Y$ FSMs:

1. Constructing the partition Π_B with a minimum number of classes I.

2. Representing outputs $y_n \in Y$ by pairs $\langle a_m, I_j \rangle$.

3. Encoding states, classes $B^i \in \Pi_B$ and identifiers in a way optimizing the SBF (24).

4. Constructing the SBF (24) representing LUTerY.

5. Constructing tables representing blocks LUTeri on the basis of the DST of $P_{AC}Y$ FSM.

6. Constructing the SBFs (20) and (21).

7. Constructing a table of LUTerF.

8. Constructing the SBFs (22) and (23).

9. Implementing the FSM circuit using internal resources of a particular FPGA chip.

We use the record $Model(S_a)$ to show that a particular model is employed for synthesis of the logic circuit for an FSM S_a. In the next section, we discuss an example of the synthesis of Mealy FSM $P_{AC}Y(S_1)$.

5. Example of synthesis

To design the logic circuit of $P_{AC}Y(S_1)$, we use LUTs having $S_L = 4$. As follows from (16), each class $B^i \in \Pi_B$ should satisfy the condition (Barkalov *et al.*, 2018)

$$L_i + R_i \leq 4. \tag{30}$$

The value of R_i is determined by (15).

This step is very important (Barkalov *et al.*, 2018). Its outcome significantly affects the LUT count of the resulting circuit. We use the methods of (Barcalov *et al.*, 2018; 2020c) to find a partition Π_B with a minimum number of classes.

There is a set $A = \{a_1, \dots, a_8\}$ in the discussed case. Using the method of Barkalov *et al.* (2018) gives a partition $\Pi_B = \{B^1, B^2\}$, where $B^1 = \{a_1, a_3, a_5, a_6\}$, $B^2 = \{a_2, a_4, a_7, a_8\}$. These classes determine sets $X^1 = \{x_1, x_2\}$ and $X^2 = \{x_3, x_4\}$ with $L_1 = L_2 = 2$. Using (15) and (17) gives $R_1 = R_2 = R_S = 2$. Obviously, the condition (30) holds for these classes. There are $I = 2$ classes in the partition Π_B.

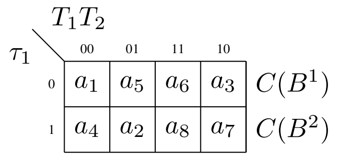
The analysis of an STG (Fig. 1) allows representing FSM outputs by pairs $\langle a_m, I_j \rangle$. There are $I = 2$, $G = 11$, $SI = \{I_1, I_2\}$. The representation is shown in Table 1. Let us discuss it below.

In the given case, there are $Q = 8$ COs $Y_q \subseteq Y$: $Y_1 = \emptyset$, $Y_2 = \{y_1, y_2\}$, $Y_3 = \{y_2, y_4\}$, $Y_4 = \{y_5\}$, $Y_5 = \{y_3\}$, $Y_6 = \{y_2, y_6\}$, $Y_7 = \{y_3, y_7\}$ and $Y_8 = \{y_2\}$. These COs are shown in the row Y_q of Table 1. The last row of Table 1 includes pairs P_1–P_{11}. Their contents follow from rows a_m and I_j. For example, there are pairs $P_1 = \langle a_2, I_1 \rangle$, $P_2 = \langle a_5, I_1 \rangle$, and so on. To distribute the identifiers among the pairs P_q, we use the approach of Barkalov *et al.* (2022).

A code $CC(a_m)$ is represented by a conjunction A_m of class variables $\tau_r \in \mathcal{T}_c$ and state variables $T_r \in T_S$. An identifier I_j is represented by a conjunction CI_j of variables $v_r \in V$. A pair $P_g = \langle a_m, I_j \rangle$ is represented by a conjunction $CP_g = A_m \cdot CI_j$. Using these conjunctions, we can form the SOPs of the functions (24).

Table 1. Representation of outputs for FSM S_1.

Y_q	y_1y_2		y_2y_4		y_5	y_3			y_2y_6	y_3y_7	y_2
a_m	a_2	a_5	a_3	a_6	a_4	a_3	a_5	a_2	a_4	a_7	a_8
I_j	I_1	I_1	I_1	I_1	I_1	I_2	I_2	I_2	I_2	I_1	I_1
P_g	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}



$\tau_1 \backslash T_1T_2$	00	01	11	10	
0	a_1	a_5	a_6	a_3	$C(B^1)$
1	a_4	a_2	a_8	a_7	$C(B^2)$

Fig. 6. Combined state codes of $P_{AC}Y(S_1)$.

Because of (16), each partial function is implemented as a single LUT circuit. In consequence, there is no influence of codes $C(B^i)$ and $C(a_m)$ on the LUT count. Therefore, classes and states should be encoded in a way optimizing the SBF (24).

We use the following rules for encoding of classes and states:

1. The classes with a maximum number of common outputs $y_n \in Y$ must have adjacent codes (with a Hamming distance equal to 1).

2. If states $a_m \in B^i$ and $a_s \in B^j$ are parts of pairs with the same outputs $y_n \in Y$, then these states should have adjacent codes.

The identifiers are encoded using the approach by Barkalov *et al.* (2022). It is the following: the more often an identifier appears in Table 1, the more zeros its code includes.

In the discussed case, there are sets $\mathcal{T}_c = \{\tau_1\}$ and $T_s = \{T_1, T_2\}$. Using (5) gives $R_V = 1$ and $V = \{v_1\}$. We can encode the classes in the following way: $C(B^1) = 0$ and $C(B^2) = 1$. The codes of states are shown in Fig. 6. The identifier I_1 (resp. I_2) appears 7 (resp. 4) times in Table 1. Thus, there is $K(I_1) = 0$ and $K(I_2) = 1$.

The following condition holds for this example:

$$R_{cc} + R_V \leq S_L. \tag{31}$$

Because of it, each function (8) is represented by a single LUT. In this case, the proposed approach for encoding $B^i \in \Pi_B$ and $a_m \in B^i$ allows reducing the number of interconnections between the blocks LUTerF and LUTerY.

Using codes $C(B^i)$, $C(a_m)$, and $K(I_j)$ leads, for example, to the following SOPs:

$$\begin{aligned} y_1 &= A_2 \cdot CI_1 \vee A_5 \cdot CI_1 \\ &= \tau_1 \bar{T}_1 T_2 \bar{v}_1 \vee \bar{\tau}_1 \bar{T}_1 T_2 \bar{v}_1, \\ y_3 &= A_3 \cdot CI_2 \vee A_5 \cdot CI_2 \vee A_2 \cdot CI_2 \vee A_7 \cdot CI_2 \\ &= \bar{T}_1 T_2 v_1 \vee T_1 \bar{T}_2 v_1. \end{aligned} \tag{32}$$

Table 2. Direct structure table of $P_{AC}Y(S_1)$.

a_m	$C(a_m)$	a_s	$CC(a_s)$	X_h	I_h	Φ_h	V_h	h
a_1	00	a_2	101	x_1	I_1	D_1D_3	–	1
		a_3	010	$\bar{x}_1x_2$	I_1	D_2	–	2
		a_4	100	$\bar{x}_1\bar{x}_2$	I_1	D_1	–	3
a_2	01	a_3	010	x_3x_4	I_2	D_2	v_1	4
		a_4	100	$x_3\bar{x}_4$	I_1	D_1	–	5
		a_5	001	$\bar{x}_3$	I_1	D_3	–	6
a_3	10	a_4	100	x_1	I_2	D_1	v_1	7
		a_5	001	$\bar{x}_1$	I_2	D_3	v_1	8
a_4	00	a_5	001	x_3x_4	I_1	D_3	v_1	9
		a_2	101	$x_3\bar{x}_4$	I_2	D_1D_3	–	10
		a_1	000	$\bar{x}_3$	–	–	–	11
a_5	01	a_6	011	1	I_1	D_2D_3	–	12
a_6	11	a_7	110	1	–	D_1D_2	–	13
a_7	10	a_8	111	1	–	$D_1D_2D_3$	–	14
a_8	11	a_4	100	1	I_2	D_1	v_1	15

All other SOPs for LUTerY can be obtained in the same way.

Blocks LUTer1–LUTerI generate the functions (20) and (21). Thus, in a DST of $P_{AC}Y$ FSM, each CO $Y_q \subseteq Y$ should be replaced by a pair $\langle a_m, I_j\rangle$. For example, the CO $Y_2 = \{y_1, y_2\}$ is generated during the transition $\langle a_1, a_2\rangle$ caused by x_1. As follows from Table 1, this situation is represented by a pair $\langle a_2, I_1\rangle = P_1$. Thus, in the DST of $P_{AC}Y(S_1)$, the identifier I_1 should replace CO Y_2. All rows of the DST are filled in the same way.

Thus, there are the following columns in the DST: a_m (a current state), $C(a_m)$, a_s (a next state), $CC(a_s)$, X_h (an input signal determining the transition $\langle a_m, a_s\rangle$, I_h (an identifier replacing the CO $Y_q \subseteq Y$), Φ_h (a collection of IMFs equal to 1 to load the $CC(a_s)$ into RG), V_h (a conjunction corresponding to $K(I_i)$), h (the number of transitions). There are $H = 15$ arcs in the STG (Fig. 1). Accordingly, DST of $P_{AC}Y(S_1)$ has 15 rows (Table 2).

The tables of LUTer1–LUTer2 are constructed in a trivial way. For example, the table of LUTer1 includes the rows of Table 2 with the transitions from states a_1,a_3,a_5,$a_6 \in B^1$. We do not show these tables in our example. The SBFs (20) and (21) can be derived from Table 2.

For example, the following SOPs can be obtained from Table 2 (after minimization):

$$\begin{aligned} D_3^1 &= \bar{T}_1\bar{T}_2x_1 \vee T_1\bar{T}_2\bar{x}_1 \vee \bar{T}_1T_2, \\ D_3^2 &= \bar{T}_1T_2\bar{x}_3 \vee \bar{T}_1\bar{T}_2x_3 \vee T_1\bar{T}_2, \\ v_1^1 &= T_1\bar{T}_2, \\ v_1^2 &= \bar{T}_1T_2x_3x_4 \vee \bar{T}_1\bar{T}_2x_3\bar{x}_4. \end{aligned} \tag{33}$$

The LUTerF includes up to $R_{cc} + R_V$ multiplexers implementing the functions (22) and (23), Its table conveys the rows marked by symbols $D_1, \dots, D_{R_{cc}}$, $v_1, \dots, v_{R_V}$. There are $I + 1$ columns in a table of LUTerF. The last I columns contain the numbers of blocks of LUTeri. If a partial function is generated by LUTeri, then there is 1 on the intersection of the corresponding function and the column i.

Analysis of Table 2 shows that all $R_{cc} + R_V$ partial functions are generated by each LUTeri ($i = \overline{1, 2}$). Hence, the SBFs (22) and (23) are constructed in a trivial way. They are the following:

$$\begin{aligned} D_1 &= \bar{\tau}_1D_1^1 \vee \tau_1D_1^2, \\ D_2 &= \bar{\tau}_1D_2^1 \vee \tau_1D_2^2, \\ D_3 &= \bar{\tau}_1D_3^1 \vee \tau_1D_3^2, \\ v_1 &= \bar{\tau}_1v_1^1 \vee \tau_1v_1^2. \end{aligned} \tag{34}$$

The variable $\tau_1 \in \mathcal{T}_c$ is connected with a control input of an MX. The data inputs of each MX are connected with the corresponding outputs of LUTer1–LUTer2.

Using the obtained SBFs, we can create the logic circuit of FSM $P_{AC}Y(S_1)$. Let us estimate the LUT count of this circuit.

As follows from (34), in total, there are 12 LUTs in the circuit of LUTer1, LUTer2 and LUTerF. Because the condition (26) holds, there are $N = 7$ LUTs in the circuit of LUTerY. Consequently, there are 19 LUTs in the circuit of $P_{AC}Y(S_1)$. Also, this circuit has three levels of LUTs. We do not show this circuit.

To compare the approach by Barkalov *et al.* (2022) with our new method, we synthesized the logic circuit of $P_{AT}Y(S_1)$. If $S_L = 4$, then $K = 3 > I = 2$. Consequently, there is a partition $\Pi_A = \{A^1, A^2, A^3\}$, where $A^1 = \{a_1, a_3, a_5\}$, $A^2 = \{a_2, a_4, a_6\}$, $A^3 = \{a_7, a_8\}$. This means that the first level of FSM circuit includes three blocks. There are 14 LUTs in the circuits of LUTer1–LUTer3. There is $R_1 = R_2 = R_3 = 2$, but the LUTerTV consists of five LUTs. Thus, there are seven LUTs in LUTerY and six LUTs in LUTer$\mathcal{T}$. In consequently, there are 32 LUTs in the circuit of $P_{AT}Y(S_1)$.

As can be seen, the transition from FSM $P_{AT}Y(S_1)$ to FSM $P_{AC}Y(S_1)$ allows us to reduce the LUT count by 1.68 times. Obviously, to implement the circuits of LUT-based FSMs, we should use various methods of technology mapping (Kubica *et al.*, 2021; Krishnamoorthy and Tessier, 2003; Kubica and Kania, 2017; Ling *et al.*, 2005; Zgheib and Ouaiss, 2015). This can be done using standard CAD packages such as, for example, Vivado by AMD (Xilinx) (Vivado, 2023). We do not show the results of implementation for our example. In the next section, we present the results of experiments conducted with standard benchmarks (LGSynth93, 1993) and CAD tool Vivado.

6. Experimental results

This section includes experimental results. We compare the proposed approach ($P_{AC}Y$ Mealy FSMs) with some known methods including $P_{AT}Y$ FSMs. In the experiments, we use the library of standard benchmarks LGSynth93 (LGSynth93, 1993). Here there are 48 benchmarks of different complexness. The benchmarks are represented in the format KISS2. These benchmarks have a wide range of basic characteristics (numbers of states, inputs, and outputs). The characteristics of these benchmarks can be found in many articles and books, (e.g., Barkalov *et al.*, 2022; 2020b). Thus we do not show them in this paper. The benchmarks are used very often by different researchers to compare the basic characteristics of FSM circuits obtained with different design methods.

The experiments were conducted using a personal computer with the following characteristics: Intel Core i7 6700 K 4.2@4.4 GHz, 16 GB RAM 2400 MHz CL 15 Memory. To implement LUT-based FSM circuits, we used the Virtex-7 VC709 Evaluation Platform (xc7vx690tffg1761-2) (AMD, 2023b). 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 technology mapping was executed by the CAD tool Vivado v2019.1 (64-bit) (Vivado, 2023). The reports of Vivado give us the results of the experiments (LUT counts and maximum operating frequencies). The process of circuit implementation starts from the transformation of KISS2-based files into VHDL-based FSM models. To execute the transformation, our CAD tool K2F (Barkalov *et al.*, 2020b) was used.

We compared area and time characteristics of FSM circuits obtained using the proposed method with four other approaches. Three of them are Mealy FSMs based on (i) Auto of Vivado (the value of R is selected from the interval $(\lceil \log_2 M \rceil, M)$), (ii) One-hot of Vivado, (iii) JEDI. The model of P Mealy FSM was used in these three cases. Also, we compared our approach with $P_{AT}Y$ FSM, proposed by Barkalov *et al.* (2022).

Our previous research (Barkalov *et al.*, 2021; 2022) shows that both the LUT count and maximum operating frequency depend strongly on the relation between the sum $L + R$, where the value of R is determined by (1), on the one hand, and the number of inputs S_L, on the other. Thus, we divided the benchmarks into five classes. The benchmarks create the class of trivial FSMs (Class 0) if $R + L \leq 6$, the class of simple FSMs (Class 1) if $6 < R + L \leq 12$, the class of average FSMs (Class 2) if $12 < R + L \leq 18$, the class of big FSMs (Class 3) if $18 < R + L \leq 24$, the class of very big FSMs (Class 4) if $R + L > 24$. As previous research (Barkalov *et al.*, 2021) shows, the larger the class number, the bigger the gain from using methods of structural decomposition.

For the library the LGSynth, the benchmarks are divided as follows. 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, the benchmarks $s420$, $s510$, $s820$, and $s832$ form Class 4.

The results of experiments are shown in Table 3 (the LUT counts) and Table 4 (the maximum operating frequency). These tables are organized in the same manner. The columns are marked by the names of the investigated methods. The names of the benchmarks are shown in the rows. There are results of summation of values from the columns in the row “Total”. The row “%” includes the percentage of summarized characteristics of FSM circuits produced by other methods with present to $P_{AC}Y$-based FSMs. The following conclusions can be made on the basis of the experimental results.

As follows from Table 3, the circuits of $P_{AC}Y$-based FSMs use a minimum number of LUTs compared with the other investigated methods. There is the following gain in the LUT counts: (i) 64.81% compared with Auto-based FSMs; (ii) 91.8% compared with FSMs based on one-hot state assignment; (iii) 35.73% compared with JEDI-based FSMs; (iv) 17.96% compared with $P_{AT}Y$-based FSMs.

As follows from Table 4, our approach allows obtaining FSM circuits with the highest operating frequency. There is the following gain in the operating frequency: (i) 10.49% compared with Auto-based FSMs, (ii) 11.21% compared with FSMs based on one-hot state assignment, (iii) 3.07% compared with JEDI-based FSMs, (iv) 2.17% compared with $P_{AT}Y$-based FSMs.

We think that the improvement in performance related to $P_{AT}Y$-based FSMs is associated with a decrease in the number of interconnects. This number is decreased due to the following factors. Firstly, as follows from (26)–(28), there is a significant reduction in the number of state variables used as feedback to the blocks of the first logic level (for $P_{AC}Y$ FSMs, the first level includes blocks LUTer1–LUTerI) related to the equivalent $P_{AT}Y$-based FSMs. Secondly, the following condition holds for the equivalent benchmark FSMs (LGSynth93, 1993):

$$I \leq K. \tag{35}$$

The consequence of the validity of (35) is a decrease in the number of connections between the blocks of the first and second level compared with this number for the equivalent $P_{AT}Y$-based FSMs.

As mentioned by Feng *et al.* (2018), for the modern nanoelectronics “...wire delay has come to dominate logic delay.” Our approach allows reducing the number of interconnections. Therefore, this reduction of interconnections is responsible for the increase in the value of the maximum operating frequency.

As follows from Tables 3 and 4, using the combined state codes allows obtaining FSM circuits with better area (the LUT count) and time (the maximum operating frequency) characteristics compared with the equivalent $P_{AT}Y$ FSMs. Also, our approach produces better results compared with those obtained using the known methods

Table 3. Results of the experiments (the LUT count).

Benchmark	Auto	One-Hot	JEDI	TCAD21	Our appr.	Class
bbtas	5	5	5	10	10	0
dk17	5	12	5	10	10	0
dk27	3	5	4	10	10	0
dk512	10	10	9	15	15	0
ex3	9	9	9	16	14	0
ex5	9	9	9	12	12	0
lion	2	5	2	10	10	0
lion9	6	11	5	11	11	0
mc	4	7	4	9	8	0
modulo12	7	7	7	11	11	0
shiftreg	2	6	2	8	8	0
bbara	17	17	10	11	10	1
bbsse	33	37	24	26	24	1
beecount	19	19	14	14	13	1
cse	40	66	36	33	32	1
dk14	16	27	10	12	11	1
dk15	15	16	12	8	8	1
dk16	15	34	12	11	10	1
donfile	31	31	24	21	19	1
ex2	9	9	8	8	9	1
ex4	15	13	12	11	10	1
ex6	24	36	22	21	19	1
ex7	4	5	4	6	6	1
keyb	43	61	40	37	34	1
mark1	23	23	20	18	16	1
opus	28	28	22	19	17	1
s27	6	18	6	6	6	1
s386	26	39	22	20	18	1
s8	9	9	9	9	8	1
sse	33	37	30	24	22	1
ex1	70	74	53	38	34	2
kirkman	42	58	39	31	25	2
planet	131	131	88	74	61	2
planet1	131	131	88	74	61	2
pma	94	94	86	68	56	2
s1	65	99	61	51	41	2
s1488	124	131	108	82	67	2
s1494	126	132	110	89	62	2
s1a	49	81	43	32	23	2
s208	12	31	10	9	9	2
styr	93	120	81	68	57	2
tma	45	39	39	28	22	2
sand	132	132	114	91	75	3
s420	10	31	9	8	8	4
s510	48	48	32	20	16	4
s820	88	82	68	48	35	4
s832	80	79	62	46	34	4
Total	1808	2104	1489	1294	1097	
%	164,81	191,80	135,73	117,96	100,00	

Table 4. Results of the experiments (the maximum operating frequency, MHz).

Benchmark	Auto	One-Hot	JEDI	TCAD21	Our appr.	Class
bbtas	204,16	204,16	206,12	190,38	191,38	0
dk17	199,28	167,00	199,39	181,14	182,01	0
dk27	206,02	201,9	204,18	190,32	190,98	0
dk512	196,27	196,27	199,75	188,32	189,07	0
ex3	194,86	194,86	195,76	188,22	188,09	0
ex5	180,25	180,25	181,16	172,45	173,01	0
lion	202,43	204,00	202,35	196,73	197,01	0
lion9	205,3	185,22	206,38	189,49	190,32	0
mc	196,66	195,47	196,87	181,03	181,79	0
modulo12	207,00	207,00	207,13	199,79	200,01	0
shiftreg	262,67	263,57	276,26	246,14	247,08	0
bbara	193,39	193,39	212,21	186,24	188,12	1
bbsse	157,06	169,12	182,34	159,73	164,26	1
beecount	166,61	166,61	187,32	178,66	185,75	1
cse	146,43	163,64	178,12	168,23	173,42	1
dk14	191,64	172,65	193,85	179,18	184,29	1
dk15	192,53	185,36	194,87	187,37	191,21	1
dk16	169,72	174,79	197,13	189,16	193,87	1
donfile	184,03	184	203,65	199,92	203,75	1
ex2	198,57	198,57	200,14	196,58	200,32	1
ex4	180,96	177,71	192,83	184,52	187,56	1
ex6	169,57	163,8	176,59	169,45	172,45	1
ex7	200,04	200,84	200,6	194,36	197,23	1
keyb	156,45	143,47	168,43	152,59	154,18	1
mark1	162,39	162,39	176,18	169,42	172,08	1
opus	166,2	166,2	178,32	170,07	172,14	1
s27	198,73	191,5	199,13	189,18	192,43	1
s386	168,15	173,46	179,15	162,68	165,14	1
s8	180,02	178,95	181,23	172,12	174,18	1
sse	157,06	169,12	174,63	161,67	164,21	1
ex1	150,94	139,76	176,87	190,76	194,78	2
kirkman	141,38	154,00	156,68	186,62	192,47	2
planet	132,71	132,71	187,14	210,37	218,42	2
planet1	132,71	132,71	187,14	210,37	218,42	2
pma	146,18	146,18	169,83	202,43	209,73	2
s1	146,41	135,85	157,16	188,72	194,23	2
s1488	138,5	131,94	157,18	199,62	205,38	2
s1494	149,39	145,75	164,34	201,34	207,14	2
s1a	153,37	176,4	169,17	199,83	206,18	2
s208	174,34	176,46	178,76	212,72	218,59	2
styr	137,61	129,92	145,64	178,62	182,19	2
tma	163,88	147,8	164,14	198,82	203,26	2
sand	115,97	115,97	126,82	154,56	165,12	3
s420	173,88	176,46	177,25	212,62	221,34	4
s510	177,65	177,65	198,32	226,13	243,01	4
s820	152,00	153,16	176,58	205,14	214,37	4
s832	145,71	153,23	173,78	208,62	217,45	4
Total	8127,08	8061,22	8718,87	8882,43	9079,42	
%	89,51	88,79	96,03	97,83	100,00	

of state assignment (Auto, One-hot, and JEDI) and functional decomposition. In consequence, the proposed approach has a rather good potential and can be used in CAD tools targeting LUT-based FSM synthesis.

7. Conclusion

A characteristic trend of our time is the widespread application of various VLSI chips for implementing digital systems. An important problem associated with VLSI-based design is that of reducing the area occupied by the digital system circuit. This fully applies to the LUT-based synthesis of FSM circuits. In this case, it is necessary to reduce the numbers of arguments in the sum-of-products of Boolean functions representing an FSM circuit. As a rule, the required chip area is estimated as an LUT count of a particular circuit (Islam *et al.*, 2020). Thus, to reduce the required chip area, it is necessary to diminish the LUT count in an FSM logic circuit.

Structural decomposition (Barkalov *et al.*, 2021) is one of the efficient ways to solve this problem. Various methods of structural decomposition are connected with a change in the FSM structural diagram compared with single-level P Mealy FSMs. Of particular interest are methods that can simultaneously reduce the LUT count and increase the maximum operating frequency. Such a method is proposed in this paper.

The developed approach is aimed at improving the characteristics of LUT-based $P_{AT}Y$ Mealy FSMs with the transformation of collections of outputs into extended state codes. Our method is based on using combined state codes having two parts. One is a code of some class including a particular state. The other represents this state as a class element. Due to this approach, we eliminated the code transformer used in $P_{AT}Y$ Mealy FSMs. Also, the number of state variables is significantly reduced compared with equivalent $P_{AT}Y$ Mealy FSMs. As a result, we have the improvement in important characteristics such as the LUT count (on the average, by 17.96%) and maximum operating frequency (on the average, by 2.17%).

The results of our experiments show that the proposed approach allows a reduction in LUT counts in FSM circuits. Moreover, this improvement does not lead to a decrease in the FSM performance. Consequently, this method improves two main characteristics of FSM circuits. We think that it can be used in CAD tools aimed at LUT-based implementations of Mealy FSMs.

References

AMD (2023a). Corporate website, `http://www.amd.com`, (formerly Xilinx).

AMD (2023b). *VC709 Evaluation Board for the Virtex-7 FPGA*, AMD, San Jose, `https://www.xilinx.com/support/documentation/boards_and_kits/vc709/ug887-vc709-eval-board-v7-fpga.pdf`.

AMD (2019). *Virtex-7 Family Overview*, AMD, San Jose, `http://www.xilinx.com/support/documentation/data_sheets/ds183_Virtex_7_Data_Sheet.pdf`.

Anceau, F. (1986). *The Architecture of Microprocessors*, Addison-Wesley, Workingham.

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

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

Barkalov, A.A., Titarenko, L. and Mielcarek, K. (2022). Reducing LUT count for Mealy FSMs with transformation of states, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **41**(5): 1400–1411.

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. (2021). Structural decomposition in FSM design: Roots, evolution, current state—A review, *Electronics* **10**(10): 44.

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

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

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.

Barkalov, O., Titarenko, L. and Mielcarek, K. (2020c). 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.

Borowczak, M. and Vemuri, R. (2013). Secure controllers: Requirements of S*FSM, *Midwest Symposium on Circuits and Systems, Washington DC, USA*, pp. 553–557.

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.

Chapman, K. (2014). Multiplexer design techniques for datapath performance with minimized routing resources, *Xilinx All Programmable*, `https://docs.xilinx.com/v/u/en-US/xapp522-mux-design-techniques`.

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

Gajski, D.D., Abdi, S., Gerstlauer, A. and Schirner, G. (2009). *Embedded System Design: Modeling, Synthesis and Verification*, 1st Edn, Springer, Berlin.

Intel (2023). Corporate website, `http://www.intel.com`, (formerly Altera).

Islam, M.M., Hossain, M., 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.

Klimovich, A.S. and Solov'ev, V.V. (2012). Minimization of Mealy finite-state machines by internal states gluing, *International Journal of Computer and Systems Sciences* **51**(2): 244–255, DOI: 10.1134/S1064230712010091.

Krishnamoorthy, S. and Tessier, R. (2003). Technology mapping algorithms for hybrid FPGAs containing lookup tables and PLAs, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **22**(5): 545–559.

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.

Kubica, M., Opara, A. and Kania, D. (2021). *Technology Mapping for LUT-Based FPGA*, Springer, Cham.

LGSynth93 (1993). Benchmark suite, `https://ddd.fit.cvut.cz/www/prj/Benchmarks/`.

Ling, A., Singh, D. and Brown, S. (2005). FPGA technology mapping: A study of optimality, *Proceedings of the 42nd Annual Design Automation Conference, Anaheim, USA*, pp. 427– 432.

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.

Marwedel, P. (2018). *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things*, 3rd Edn, Springer, Cham.

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

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

Microchip (2023). Corporate website, `http://www.microchip.com`, (formerly Atmel).

Milik, A. (2016). On hardware synthesis and implementation of PLC programs in FPGAs, *Microprocors and Microsystems* **44**(C): 2–16, DOI: 10.1016/j.micpro.2016.02.003.

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

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

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

Senhadji-Navarro, R. and Garcia-Vargas, I. (2018). High-performance architecture for binary-tree-based finite state machines, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **37**(4): 796–805.

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

Sentowich, E., Singh, K., Lavango, L., Moon, C., Murgai, R., Saldanha, A., Savoj, H., Savoj, H., Stephan, P., Bryton, R. and Sangiovanni-Vincentelli, A. (1992b). SIS: A system for sequential circuit synthesis, *Proceedings of the International Conference of Computer Design (ICCD'92), Berkeley, USA*, pp. 328–333.

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

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

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, *Design, Automation and Test in Europe Conference and Exhibition, Paris, France*, Vol. 2, pp. 916–921.

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

Vivado (2023). Design tools documentation, `https://www.xilinx.com/products/design-tools/vivado.html`.

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

Zgheib, G. and Ouaiss, I. (2015). Enhanced technology mapping for FPGAs with exploration of cell configurations, *Journal of Circuits, Systems and Computers* **24**(3): 1550039.

Alexander A. Barkalov worked at Donetsk National Technical University (DNTU) from 1976 till 1996 as a tutor. He then cooperated actively with the Kiev Institute of Cybernetics (IC) named after Victor Glushkov. He obtained his degree of a doctor of technical sciences (informatics) 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 at the Faculty of Computer, Electrical and Control Engineering of the University of Zielona Góra.

Larysa Titarenko obtained her degree of a doctor of technical sciences (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 at the Faculty 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 University of Zielona Góra, Poland, in 2010. Since 2001 he has been a lecturer at the University of Zielona Góra. 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, as well as algorithmic theory and safety of UNIX and network systems.

Received: 28 April 2023
Revised: 16 October 2023
Accepted: 14 November 2023