IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 9 January 2024, accepted 1 March 2024, date of publication 11 March 2024, date of current version 25 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3376472

RESEARCH ARTICLE

Hardware Reduction for FSMs With Extended State Codes

ALEXANDER BARKALOV[1], LARYSA TITARENKO[1], KAMIL MIELCAREK[1], AND MAŁGORZATA MAZURKIEWICZ[2]

[1]Institute of Metrology, Electronics and Computer Science, Faculty of Computer, Electrical and Control Engineering, University of Zielona Góra, 65-417 Zielona Góra, Poland

[2]Institute of Control and Computation Engineering, Faculty of Computer, Electrical and Control Engineering, University of Zielona Góra, 65-417 Zielona Góra, Poland

Corresponding author: Kamil Mielcarek (K.Mielcarek@imei.uz.zgora.pl)

ABSTRACT A method is proposed for reducing chip area occupied by logic circuits of FPGA-based Mealy finite state machines (FSMs). The proposed method aims at optimization of FSM circuits implemented with look-up table (LUT) elements of FPGA chip. The proposed method combines positive features of such state assignment methods as extended (ESCs) and composite (CSCs) state codes. The method is based on finding a partition of the set of internal states by classes of compatible states. To reduce LUT count, we propose a special kind of states codes named mixed state codes. These codes include two parts. The first part includes the maximum binary codes of states as elements of some partition class. The second part consists of the code of corresponding partition class. Unlike CSCs, the proposed approach allows obtaining partial state codes having different lengths. Unlike ESCs, all partial state codes consist of the same variables. Using mixed state codes allows obtaining LUT-based FSM circuits with exactly two levels of logic. The first level generates partial Boolean functions. The second level combines them into FSM outputs and input memory functions. If some conditions hold, then both levels consist of single-LUT circuits. Example of FSM synthesis based on mixed state codes is shown. The conducted experiments prove that the proposed approach allows reducing hardware compared with FSMs based on JEDI, one-hot state codes, CSCs and ESCs. On average, the proposed method reduces the value of LUT count by 6.21% compared to CSC-based FSMs and 22.21% compared to ESC-based counterparts. The advantages of the proposed approach grow with the growth of FSM complexness. An additional positive effect of the proposed method is a slight increase in the operating frequency.

INDEX TERMS Mealy FSM, FPGA, LUT count, synthesis, extended state codes, composite state codes, mixed state codes.

I. INTRODUCTION

Now, various digital systems are widely used practically in all spheres of human activity [1], [2], [3]. These systems include a lot of combinational and sequential devices [3], [4]. It is very important to improve the basic characteristics of digital systems (price, hardware amount, performance, power consumption) [4]. In this article, we propose a method aimed at reducing the area of sequential devices. We use the model of Mealy finite state machine (FSM) [5] to represent the sequential devices.

This model is viewed as a starting point of the sequential device design process [6]. This has led to the emergence of a significant number of FSM-based design methods [7]. As a rule, all these methods are associated with the solution of some optimization problems [8], [9]. In the design process, such problems are solved as reducing the chip area occupied by the resulting FSM circuit, increasing its operating frequency and reducing the consumed electrical power. If there is no single criterion for the optimality of the FSM circuit, then the designers strive to achieve a balance

The associate editor coordinating the review of this manuscript and approving it for publication was Abdallah Kassem.

of its main characteristics [10]. Many studies show that the chip area occupied by an FSM circuit has a decisive influence on both the performance and power consumption [11]. Due to it, our current article is devoted specifically to the problem of reducing the required chip area. The proposed method focuses on Mealy FSM circuits implemented with field-programmable gate arrays (FPGAs) [12], [13], [14]. The proposed method combines approaches related to the use of extended state codes (ESCs) [15] and composite state codes (CSCs) [16].

In our study, we focus on the FPGA chips of the company AMD Xilinx [17]. This choice is due to two facts. First, FPGAs are widespread in modern digital design [12]. This determines our choice of FPGA as a basis for the implementation of FSM circuits. Secondly, the company AMD Xilinx is the largest producer of FPGA chips [17]. This determines the choice of their chips Virtex 7 [18] in our current research. To evaluate the required chip area, we use the widely used approach described in [10]. In this case, the area is estimated as a number of look-up table (LUT) elements (LUT count) in an FSM circuit.

The main goal of the proposed method is the reduction of LUT count in Mealy FSM circuits. We analysed positive and negative features of extended state codes [15] and composite state codes [16]. Both approaches are based on finding minimum possible number of classes of compatible states. In ESC-based FSMs, the total number of code bits (extended code length) exceeds significantly their minimum possible number inherent in maximum binary state codes (MBCs). This leads to increasing the number of flip-flops in FSM state memory. In addition, this increase leads to a complication of the interconnect system. But for ESCs, partial state codes can have different lengths for different classes. This reduces the number of classes. In CSC-based FSMs, the same state variables create partial state codes for all classes. This reduces the total length of state codes compared to equivalent ESC-based FSMs. But due to it, CSC-based FSMs require more classes than their ESC-based counterparts. This can lead to reducing performance of CSC-based FSMs compared to ESC-based FSMs. In our current article, we propose a method eliminating the mentioned drawbacks and combining together positive features of both CSC- and ESC-based FSMs.

The main contribution of this paper is a **novel design method aimed at reducing the LUT counts in circuits of two-level FPGA-based Mealy FSMs compared to equivalent FSMs with either extended or composite state codes.** The resulting FSMs combine positive features of ESC-based FSMs (the different partial state code length for different classes) and CSC-based FSMs (the minimum possible length of state codes). Combination of these features leads to so-called mixed state codes (MSCs) proposed in our current article. The proposed method of state assignment provides a variable length of partial state codes for different classes (minimizing the number of classes) and minimizing the overall bitness of state codes.

The rest of the article is organized as follows. The essential information about LUT-based Mealy FSMs is discussed in Section II. The Section III includes a short analysis of related works. The main idea of the proposed method is shown in Section IV. An example of MSC-based FSM synthesis is shown in Section V. Section VI includes the experimental results and their analysis. The short summary is shown in Section VII.

II. ESSENTIAL INFORMATION CONCERNING LUT-BASED FSM DESIGN

A Mealy FSM may be represented by a state transition graph (STG) [19]. From an STG, three sets could be derived. They are: 1) a set of states $A = \{a_1, \dots, a_M\}$, 2) a set of inputs $X = \{x_1, \dots, x_L\}$ and 3) a set of outputs $Y = \{y_1, \dots, y_N\}$. After the step of state assignment [5], it is possible to get two systems of Boolean functions (SBFs) [19]. They represent the dependences of FSM outputs and input memory functions (IMFs) on FSM inputs and state variables. The state variables form a set $T = \{T_1, \dots, T_R\}$ Each state variable corresponds to a single bit of state code $C(a_m)$. The minimum code length corresponds to so-called maximum binary state codes (MBCs) [20]. The length of MBCs is determined by the following formula:

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

The state codes $C(a_m)$ are kept into the state code register (RG). In the case of FPGA-based FSMs, the RG includes the flip-flops of D type [21]. To change the content of RG, the input memory functions are used. The IMFs form a set $D = \{D_1, \dots, D_R\}$.

An FSM circuit is represented by two SBFs:

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

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

The SBFs (2)–(3) represent so-called P Mealy FSM [5]. It includes two logic blocks and the state code register. One of these blocks generates functions from SBF (2). The second block generates IMFs (2). These functions enter the RG to set a next state code. In the beginning, the RG is cleared using a pulse Start. The resulting code consists of zeros; it corresponds to the initial state $a_1 \in A$. The values of IMFs determine the code of a next state. This code is written into RG using a pulse Clock. In this article, we do not show the architecture of P Mealy FSM due to its triviality.

To get the systems (2)–(3), the initial STG should be transformed into a direct structure table (DST) [5]. Also, a DST can be obtained using a state transition table (STT) [19]. Both STG and STT are interchangeable. At the same time, there are elementary procedures for converting one form to another [19]. In our paper, for representing a Mealy FSM, we use an STG.

An STT includes the following columns [5]: a current state a_m; a next state a_S; an input signal X_h which is a conjunction of inputs (or their complements) determining

a particular interstate transition $\langle a_m, a_s \rangle$; a collection of outputs Y_h generated during the transition $\langle a_m, a_s \rangle$; h is a column including the numbers of interstate transitions ($h \in \{1, \ldots, H\}$). The number of STT rows is equal to the number of arcs in the equivalent STG.

To transform an STT into the equivalent DST, it is necessary to assign state codes. This is a state assignment step [19]. This step is reduced to the replacement of states $a_m \in A$ by their binary codes $C(a_m)$. The current state codes are placed into the column $C(a_m)$ of DST, the next state codes are placed into the column $C(a_s)$. In the case of D flip-flops, the values of IMFs are created on the base of the next state codes. These functions are placed into the column D_h of DST.

In this paper, we consider a case when the circuit of P Mealy FSM is implemented using internal resources of FPGA chips. A lot of configurable logic blocks (CLB) could be found in chips produced by AMD Xilinx [17]. Each CLB includes LUTs, dedicated multiplexors, flip-flops and tools of fast internal interconnect [22]. All these components are programmable. The logic functions are implemented using LUTs. The number of LUT inputs could be increased with the help of dedicated multiplexers. Each bit of state codes corresponds to a flip-flop's output. To get an FSM circuit, it is necessary to connect CLBs by tools of a programmable routing matrix [22].

A LUT has S_L inputs and a single output. Any Boolean function depending on up to S_L variables is generated using only a single LUT [20]. The value of S_L is rather small [14]. For example, there is $S_L = 6$ for basic LUTs of the Virtex-7 family [18]. But the functions (2)–(3) can be very complex. It is quite possible that the numbers of arguments of sum-of-products (SOPs) of these functions exceed significantly the value of S_L. To generate such complex functions, it is necessary to represent each of them as a composition of partial Boolean functions (PBFs). Each PBF has no more than S_L arguments. To optimize the resulting compositions, it is necessary to apply various methods of functional decomposition (FD) [23]. Although these methods allow implementing an FSM circuit, they have one negative quality: FD-based FSMs are represented by multi-level circuits with complicated systems of ''spaghetti-type'' interconnections [20].

In LUT-based FSMs, the flip-flops of RG are distributed among LUTs generating functions (2). Due to it, the RG is hidden inside the slices generating IMFs. So, LUT-based P Mealy FSMs have only two blocks. In the further material of this article, we denote a block consisting of LUTs as a LUTer.

In the best case, there is exactly a single level of LUTs in the circuit of P FSM. This is possible if each function $f_i \in D \cup Y$ depends on no more than S_L arguments. However, for sufficiently complex FSMs, the number of LUT inputs is not enough to implement single-level FSM circuits. So, rather complex FSMs are represented by multi-level LUT-based circuits. To improve the basic characteristics of LUT-based FSM circuits (LUT count, maximum operating frequency, power consumption), it is necessary to improve the existing design methods.

In addition to reducing the number of LUTs, it is very important to optimize the system of connections between different CLBs of an FSM circuit. As shown in [11], the interconnections are responsible for: 1) around 70% of power consumption and 2) the major part of FSM circuit latency time. This can be done using such methods of structural decomposition as: 1) twofold state assignment [24], [25], 2) extended state assignment [15] or 3) composite state assignment [16].

III. RELATED WORK

Over the past fifty years, a huge number of methods have been developed to optimize the characteristics of FSM circuits. Since the 1980s, most of the developed methods have focused on FPGA-based FSMs [20], [26], [27], [28]. These methods should be applied if the number of arguments $NA(f_i)$ exceeds the number of LUT address inputs (at least for a single function $f_i \in D \cup Y$) [20]. As a rule, the known design methods can improve only a single characteristic of an FSM circuit (either the LUT count or the latency time or the value of power consumption) [29], [30], [31], [32], [33]. There are also methods that attempt to balance two or three characteristics of the circuit [34], [35], [36], [37], [38]. The main goal of the method proposed in this article is reducing the chip area occupied by a LUT-based FSM circuit. As it is done in [10], we use the LUT count of a circuit to evaluate the occupied chip area.

Now, there are various methods of state assignment improving the LUT counts. These methods reduce the number of arguments in functions $f_i \in D \cup Y$ [20]. The reduction can be done by lengthening the state codes. The method of one-hot assignment is very popular in FPGA-based design [20]. In this case, the number of code bits is equal to the number of states, M. Sometimes, for optimization, it is enough to increase the number of code bits by only 2 or 3, as it is possible when using the algorithm JEDI [39]. So, the number of code bits can differ from $\lceil log_2 M \rceil$ to M. As the examples of CAD tools, we can note such systems as SIS [39], ABC by Berkeley [40], Vivado [41] by AMD Xilinx [17], Quartus [42] by Intel (Altera) [43].

Currently, there is no single best method which always gives the best results (for any FSM). For example, in [44], there are compared FSM circuits based on maximum binary and one-hot state codes (OHCs). As follows from research results reported in [44], the one-hot codes improve FSM characteristics for rather complex FSMs (when there is $M > 16$). However, there are other important factors influencing the circuit characteristics. These factors are: 1) the number of FSM inputs and 2) the distribution of FSM inputs between the states. For example, the results shown in [45] prove that using OHCs loose against MBCs if $L > 10$.

From this brief analysis, it is clear that the characteristics of LUT-based circuits depend on the total number of inputs and

state variables, $L+R$. Obviously, this value determines which state assignment method should be used for a particular FSM. To be sure which method should be applied, it is necessary to apply both MBCs and OHCs. Due to it, we have investigated the efficiency of these both methods in our research. As a base for comparison with our proposed method, we use the algorithm JEDI [39], the MBC-based method Auto and the OHC-based method One-hot of Vivado [41] by AMD Xilinx [17]. Our choice of Vivado is due to its focus on AMD Xilinx FPGA chips. Our choice of JEDI is determined by the fact that JEDI is one of the best MBC-based methods [8].

To optimize the characteristics of FSM circuits, various methods of structural decomposition (SD) [20] can be used. The main idea of SD-based methods is the elimination of direct dependence between FSM inputs $x_l \in X$ and state variables $T_r \in T$, on the one hand, and outputs $y_n \in Y$ and IMFs $D_r \in D$, on the other hand. Due to it, an FSM circuit is represented as a composition of big blocks having their own systems of input and output variables. Each block implements some system of PBFs. In this article, we propose a new SD-based FSM design method. This method combines features of methods based on extended state codes [15] and composite state codes [16].We use the term "mixed state assignment" to define the proposed method. As a result of applying this method, the states are encoded by the so-called mixed state codes. To understand the proposed method, we should analyse its predecessors discussed in articles [15], [16].

Like both ESC- and CSC-based design methods, the proposed method is based on finding a partition $\Pi_A = \{A^1, \ldots, A^{K1}\}$ of the set A by classes of compatible states [24], [25].There are M_k states in the class $A^k \in \Pi_A$. Including a state $a_m \in A$ in a class $A^k \in \Pi_A$ can change the value of NA_k, where NA_k is a number of variables on which transitions from the states already included into this class depend. A state $a_m \in A$ is compatible with states from class $A^k \in \Pi_A$ if the inclusion of this state in this class does not lead to the condition

$$NA_k > S_L. \tag{4}$$

The value of NA_k is a result of summation the numbers L_k and R_k. The value of L_k is equal to the number of FSM inputs on which transitions from the states $a_m \in A^k$ depend. These inputs create a set $X^k \subseteq X$. The value of R_k is equal to the minimum number of bits required for encoding of the states $a_m \in A^k$. Each class $A^k \in \Pi_A$ determines sets $Y^k \subseteq Y$ (a set of outputs generated during the transitions from states $a_m \in A^k$) and $D^k \subseteq D$ (a set of IMFs generated during the transitions from states $a_m \in A^k$).

Each class corresponds to some sub-table of the initial STT. The number of FSM inputs in this sub-table is equal to L_k. This is a feature common to all methods discussed below. But there are different approaches for determining the value of R_k. Now we will analyse the features of ESC- and CSC-based FSMs.

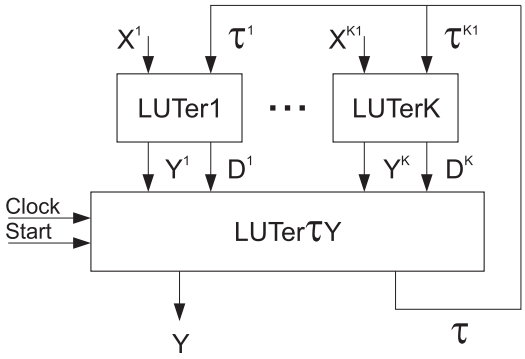

FIGURE 1. Architecture of P_E Mealy FSM.

In the case of ESC-based FSMs, the value of R_k is determined as

$$R_K = \lceil log_2(M_k + 1) \rceil \quad (k \in \{1, \ldots, K1\}). \tag{5}$$

In (5), the number 1 is added to M_k to take into account the relation $a_m \notin A^k$.

To encode states for all classes of Π_A, it is necessary R_E variables, where

$$R_E = R_1 + \ldots + R_{K1}. \tag{6}$$

To encode states, the variables $\tau_r \in \tau$ are used where $\mathcal{T} = \{\tau_1, \ldots, \tau_{RE}\}$. To encode states $a_m \in A^k$, we use a set $\tau^k \subset \tau$. The variables $\tau_r \in \mathcal{T}^k$ form partial state codes $PC(a_m)$ where $a_m \in A^k$. There are $K1$ fields in the extended state codes $CE(a_m)$. The k-th field corresponds to the set $A^k \in \Pi_A$. For a state $a_m \in A^k$, only some of variables $\tau_r \in \mathcal{T}^k$ are non-zero. All other fields include only zeroes.

The following sets correspond to each class $A^k \in \Pi_A$: X^k, Y^k, D^k, $\mathcal{T}^k$. Using a sub-table of DST which represents transitions from states $a_m \in A^k$, the following systems of PBFs can be obtained:

$$D^k = D^k(\mathcal{T}^k, X^k); \tag{7}$$

$$Y^k = Y^k(\mathcal{T}^k, X^k). \tag{8}$$

In [15], architecture of Mealy FSM based on ESCs is proposed. We denote this FSM by the symbol P_E (Fig. 1).

In P_E Mealy FSM, a block $LUTerk$ implements SBFs (7)–(8). These partial functions enter a block $LUTer\mathcal{T}Y$. The functions (8) are transformed into outputs $y_n \in Y$. The partial IMFs enter informational inputs of flip-flops from the corresponding CLBs. These flip-flops create the hidden distributed state code register RG. If there is Start = 1, then all flip-flops are zeroed. This corresponds to the code $C(a_1)$. A particular edge of the synchronization pulse Clock allows changing the contents of RG.

The PBFs (7)–(8) enter a block $LUTer\mathcal{T}Y$. This block transforms PBFs into final SBFs

$$D = D(D^1, \ldots, D^{K1}); \tag{9}$$

$$Y = Y(Y^1, \ldots, Y^{K1}). \tag{10}$$

The functions (9)–(10) are just disjunctions of corresponding PBFs.

A positive feature of ESC-based approach is the ability to choose codes of different lengths for states from different classes. This flexibility allows including an arbitrary number of states in any particular class. The only condition for including a state in a class $A^k \in \Pi_A$ is the condition

$$R_k + L_k \leq S_L. \tag{11}$$

If condition (11) holds, then each PBF from SBFs (7)–(8) is generated by a single LUT. Obviously, a state $a_m \in A$ cannot be included in a class $A^k \in \Pi_A$ if the inclusion leads to the violation of condition (11).

This feature is positive, because it keeps the number of classes to a minimum. This allows optimizing the number of LUTs for the first logic level of the circuit. However, the property of possible variability of the length of state codes has a negative side. Namely: the total number of bits of codes determined by (6) can significantly exceed the minimum required value represented by (1). This leads to a significant number of flip-flops and partial IMFs in the circuit, which increases the LUT count for the first circuit level. It also increases the requirements for the synchronization tree, which in turn leads to an increase in power consumption [46].

All noted shortcomings are not inherent in CSC-based FSMs proposed in [16]. In this case, all state codes have the same length. This method is also based on creating the partition $\Pi_A = \{A^1, \ldots, A^{K2}\}$ of the set A by classes of compatible states [25]. The states $a_m \in A^k$ are encoded by partial codes $PC(a_m)$. For each class, there is the same number of bits (R_S).

In [16], we did not show the approach used for finding the value of R_S. In [16], we used the following approach. Obviously, the value of R_S should not exceed the value $S_L - 1$. We changed R_S from 1 to $S_L - 1$ and implemented the FSM circuit. Then we chose a value of R_S that minimized the total number of LUTs in the FSM circuit. This value was taken as the number of bits used to encode the states of each class. Thus, the number of partial code bits is the same for all classes:

$$R_S = \lceil log_2\, max(M_1, \ldots, M_{K2}) \rceil. \tag{12}$$

Obviously, in the case of CSCs, the maximum amount of states for any class $A^k \in \Pi_A$ is equal to 2^{R_S}. To encode states, the state variables are used. They create a set $S = \{S_1, \ldots, S_{RS}\}$.

To distinguish the classes, they are encoded by binary codes $C(A^k)$. These codes include R_C bits, where

$$R_C = \lceil log_2\, K2 \rceil. \tag{13}$$

To encode classes, the class variables are used. They create a set $V = \{v_1, \ldots, v_{RC}\}$.

To create composite state codes $CC(a_m)$, the variables from the set $T = V \cup S$ are used. Each code $CC(a_m)$ is represented by a concatenation of class code $C(A^k)$ and partial state code $PC(a_m)$ [16]. So, the total number of code bits (R_{CC}) is equal to the result of summation R_C and R_S:

$$R_{CC} = R_C + R_S. \tag{14}$$

As shown in [16], the value of R_{CC} does not exceed the value of $R + 2$.

For CSC-based FSMs, each class $A^k \in \Pi_A$ determines the sets X^k, Y^k, D^k. These sets determine functions of LUTs creating the first level of FSM circuit. In [16], we propose architecture of P_C FSM based on composite state codes. This architecture is shown in Fig. 2.

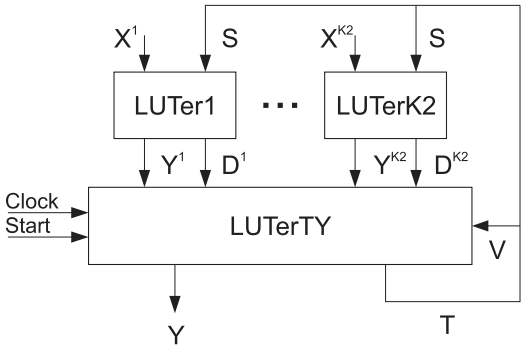

FIGURE 2. Architecture of LUT-based P_C FSM.

In P_C FSM, a block *LUTerk* generates the following PBFs:

$$D^k = D^k(S, X^k); \tag{15}$$

$$Y^k = Y^k(S, X^k). \tag{16}$$

A block *LUTerTY* transforms these PBFs into final values represented by the following SBFs:

$$D = D(V, D^1, \ldots, D^{K2}); \tag{17}$$

$$Y = Y(V, Y^1, \ldots, Y^{K2}). \tag{18}$$

To create SBFs (17)–(18), it is necessary to multiply each PBF by a conjunction V_k of class variables. The conjunctions V_k are determined by the class codes $C(A^k)$.

The block *LUTerTY* includes a hidden register. This register is controlled by pulses Clock and Start. There are R_{CC} flip-flops inside of *LUTerTY*. Their outputs correspond to class and state variables from the set $T = T \cup S$.

A positive feature of CSC-based approach is practically minimum amount of used flip-flops forming a register RG. Compared to equivalent ESC-based FSMs, this reduces: 1) the requirements for the synchronization tree and 2) the number of LUTs generating IMFs. The disadvantage of CSC-based approach is the constant number of bits (R_S) used to encode the states within each class $A^k \in \Pi_A$. This leads to an increase in the number of classes K2 in relation to K1. If for some class the difference $\Delta_k = S_L - L_k - R_S$ is positive, then this class could be supplemented with Δ_k states and, possibly, reduce the total number of classes, K_2. In turn, reducing the number of classes can reduce the number of LUTs and their levels in the resulting FSM circuit.

Thus, each of the considered approaches (P_E and P_C FSMs) has its positive and negative qualities. In this article, we propose an approach which allows combining the positive qualities of the discussed FSMs and reducing their inherent disadvantages.

IV. MAIN IDEA OF THE PROPOSED METHOD

The proposed method is based on finding a partition $\Pi_A = \{A^1, \ldots, A^K\}$ of the set A by K classes of compatible states. We denote a resulting FSM by the symbol P_{EC}. The subscript EC emphasizes that the proposed model combines the positive properties of P_E and P_C FSMs. As a result, the states are encoded by so-called mixed state codes $EC(a_m)$ proposed in this article. Each MSC consists of a class code $C(A^k)$ and partial state code $PC(a_m)$.

Our approach has the following peculiarities. Unlike partial state codes used in P_C FSMs, the partial codes used in P_{EC} FSMs can have different lengths, which is inherent in P_E FSMs. Now, the number of variables forming partial codes for states $a_m \in A^k$ is determined as

$$R_k = \lceil log_2 M_k \rceil. \tag{19}$$

As opposed to CSC-based FSMs [16], the length of the partial codes is defined as

$$R_S = max(R_1, \ldots, R_K). \tag{20}$$

So, the total number of variables in the set S is equal to R_S. However, for a particular class A^k, only a part of these variables may be used. This part forms a set Sk. Now, each class $A^k \in \Pi_A$ is represented by the following systems of PBFs:

$$D^k = D^k(Sk, X^k); \tag{21}$$
$$Y^k = Y^k(Sk, X^k). \tag{22}$$

To encode classes, we use variables $v_r \in V$. The set V includes R_V elements where

$$R_V = \lceil log_2 K \rceil. \tag{23}$$

Totally, the proposed mixed codes have R_{EC} bits, where

$$R_{EC} = R_V + R_S. \tag{24}$$

The set D^k includes partial functions $D^k_1, \ldots, D^k_{REC}$, the set Y^k includes partial output functions $y^k_1, \ldots, y^k_N$.

The SBFs (21)–(22) are transformed into FSM outputs $y_n \in Y$ and IMFs $D_r \in D$. The transformation is represented by the following SBFs:

$$D_r = V_1 D^1_r + \ldots + V_K D^K_r; \tag{25}$$
$$y_n = V_1 y^1_n + \ldots + V_K y^K_n. \tag{26}$$

The following relations exist in (25)–(26): $r \in \{1, \ldots, R_{EC}\}$ and $n \in \{1, \ldots, N\}$. As follows from (25)–(26), the functions $D_r \in D$ and $y_n \in Y$ are multiplexer functions. To generate these functions, it is necessary to use $N + R_{EC}$ multiplexers. The partial functions (21)–(22) are used as data inputs of these multiplexers. Each multiplexer selects a particular PBF using the class variables $v_r \in V$.

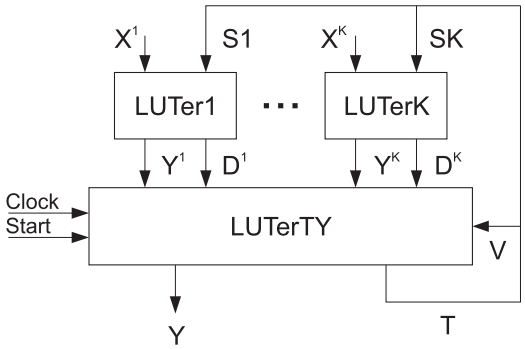

FIGURE 3. Architecture of P_{EC} FSM.

So, SBFs (21)–(22) determines a block of partial functions. The SBFs (25)–(26) determine a block of final values. This block should include flip-flops keeping mixed state codes. So, the proposed P_{EC} FSM has an architecture shown in Fig. 3.

In P_{EC} FSMs, the blocks *LUTer*1–*LUTerK* form the first circuit level. A block *LUTerk* generates PBFs (21)–(22). The block *LUTerTY* forms the final values of functions (25)–(26). This is the second circuit level. The block *LUTerTY* includes the hidden register RG consisting of R_{CE} flip-flops. The RG is controlled by pulses *Start* and *Clock*.

In the common case, the following conditions hold:

$$K \leq K1; \tag{27}$$
$$K < K2; \tag{28}$$
$$R_C < R_{EC} \ll R_E. \tag{29}$$

As follows from relations (27)–(28), our approach provides the minimum possible amount of classes. This can result in reducing the numbers of LUTs and their levels in the blocks generating functions $y_n \in Y$ and $D_r \in D$. As follows from (29), the proposed approach provides practically the same amount of flip-flops in equivalent CSC- and MSC-based FSMs. So, these FSMs can have practically the same values of power consumption. Also, the registers of P_{EC} FSMs have significantly fewer flip-flops than equivalent ESC-based FSMs. Due to it, we can expect that the proposed FSMs provide better values of power consumption than ESC-based FSMs. The experiments reported in Section VI show that our approach allows improving the basic characteristics of LUT-based circuits of Mealy FSMs.

In this paper, we propose a synthesis method for P_{EC} Mealy FSMs. The proposed method produces the logic circuits of LUT-based FSMs. We start the synthesis process from an FSM state transition graph. The proposed method includes the following steps:

1) Transforming initial STG into equivalent STT.
2) Constructing the partition Π_A with minimum possible cardinality number.
3) Creating sets of class and state variables ($v_r \in V$, $s_r \in S$).

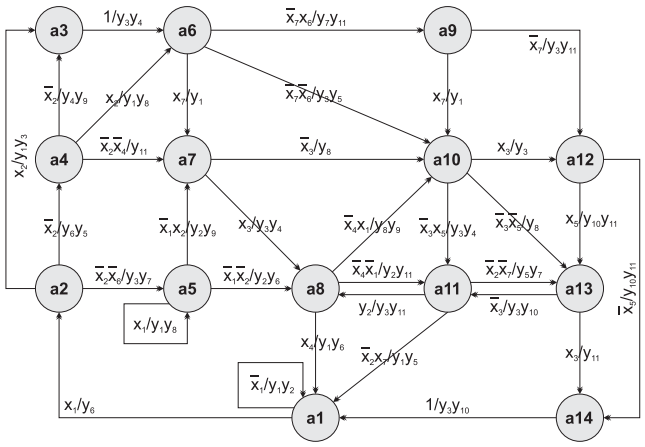

FIGURE 4. State transition graph of Mealy FSM F1.

4) Encoding of FSM states by mixed state codes $EC(a_m)$.
5) Creating tables of blocks $LUTer1$–$LUTerK$.
6) Deriving SBFs (21)–(22) for each block of PBFs.
7) Creating table of block $LUTerTY$.
8) Constructing SBFs (25)–(26) representing $LUTerTY$.
9) Implementing the LUT-based circuit of P_{EC} Mealy FSM using internal resources of a particular FPGA chip.

The second step is executed using the approach proposed in [24]. This is a greedy algorithm creating the classes of compatible states with maximum possible cardinality numbers. This results in minimizing the value of K. During this step, the classes are created in a way minimizing the number of shared outputs. This leads to reducing the LUT count in the circuit generating SBF (26). Any multiplexer from the second level of an FSM circuit is implemented by a single LUT if the following condition takes place:

$$R_V + K \leq S_L. \tag{30}$$

If condition (30) is violated, then the multiplexers could be implemented as single-level circuits. This is possible, if the number of partial functions for a given function does not exceed the value $S_L - R_V$.

V. EXAMPLE OF SYNTHESIS

We use the symbol $P_i(Fj)$ to show that the model of P_i Mealy FSM ($i \in \{E, C, EC\}$) is used to implement the circuit of an FSM Fj. In this Section, we show how to design the circuit of Mealy FSM $P_{EC}(F1)$. The circuit should be implemented using 5-LUTs (LUTs with $S_L = 5$). The synthesis process starts from an STG shown in Fig. 4.

Step 1: To transform an STG into the equivalent STT, it is necessary to transform each arc of STG into a line of STT [5]. There are the following columns in the STT [5]: a_m (a current FSM state); a_S (a next FSM state); X_h (the input signal written above the h-th arc of STG); Y_h (the collection of outputs $y_n \in Y$ written above the h-th arc of STG); h (the number of transition where $h \in \{1, \ldots, H\}$). The input

TABLE 1. State transition table of FSM F1.

a_m	a_s	X_h	Y_h	h
a_1	a_2	x_1	y_6	1
	a_1	$\bar{x_1}$	y_1y_2	2
a_2	a_3	x_2	y_1y_3	3
	a_4	$\bar{x_2}x_6$	y_5	4
	a_5	$\bar{x_2}\bar{x_6}$	y_3y_7	5
a_3	a_6	1	y_3y_4	6
a_4	a_6	x_2	y_1y_8	7
	a_3	$\bar{x_2}x_4$	y_9	8
	a_7	$\bar{x_2}\bar{x_4}$	y_{11}	9
a_5	a_5	x_1	y_1y_8	10
	a_7	$\bar{x_1}x_2$	y_2y_9	11
	a_8	$\bar{x_1}\bar{x_2}$	y_2y_6	12
a_6	a_7	x_7	y_1	13
	a_9	$\bar{x_7}x_6$	y_7y_{11}	14
	a_{10}	$\bar{x_7}\bar{x_6}$	y_3y_5	15
a_7	a_8	x_3	y_3y_4	16
	a_{10}	$\bar{x_3}$	y_8	17
a_8	a_1	x_4	y_1y_6	18
	a_{10}	$\bar{x_4}x_1$	y_1y_6	19
	a_{11}	$\bar{x_4}\bar{x_1}$	y_2y_{11}	20
a_9	a_{10}	x_7	y_1	21
	a_{12}	$\bar{x_7}$	y_3y_{11}	22
a_{10}	a_{12}	x_3	y_3	23
	a_{11}	$\bar{x_3}x_5$	y_3y_4	24
	a_{13}	$\bar{x_3}\bar{x_5}$	y_8	25
a_{11}	a_8	x_2	y_3y_{11}	26
	a_1	$\bar{x_2}x_7$	y_1y_5	27
	a_{13}	$\bar{x_2}\bar{x_7}$	y_5y_7	28
a_{12}	a_{13}	x_5	y_3y_8	29
	a_4	$\bar{x_5}$	$y_{10}y_{11}$	30
a_{13}	a_{14}	x_3	y_{11}	31
	a_{11}	$\bar{x_3}$	y_3y_{10}	32
a_{14}	a_1	1	y_3y_{10}	33

signals are conjunctions of some FSM inputs $x_l \in X$ (or their compliments). For arc number h, the vertex from which the arc emerges corresponds to the current state, and the vertex that the arc enters corresponds to the next state (state of transition).

There are $H = 33$ arcs in the STG (Fig. 4). So, it should be 33 lines in the equivalent STT. The following can be found from analysis of Fig. 4: 1) FSM transitions depend on $L = 7$ variables $x_l \in X$; 2) during these transitions, $N = 11$ outputs $y_n \in Y$ are generated; 3) there are $M = 14$ nodes in STG (Fig. 4). This determines the sets $X = \{x_1, \ldots, x_7\}$, $Y = \{y_1, \ldots, y_{11}\}$ and $A = \{a_1, \ldots, a_{14}\}$.

The transition from STG to STT is executed in the trivial way [5]. In the discussed case, the STT is represented by Table 1.

Table 1 includes H = 33 lines. We hope there is a transparent connection between the STG (Fig. 4) and STT (Table 1).

Step 2: Using the greedy approach [24], Table 1 and 5-LUTs, we have obtain the partition $\Pi_A = \{A^1, A^2, A^3\}$. So, there is $K = 3$. There are the following classes $A^k \in \Pi_A$: $A^1 = \{a_3, a_7, a_{10}, a_{12}, a_{13}, a_{14}\}$, $A^2 = \{a_1, a_4, a_5, a_8\}$, and $A^3 = \{a_2, a_6, a_9, a_{11}\}$. So, there is $M_1 = 6$ and

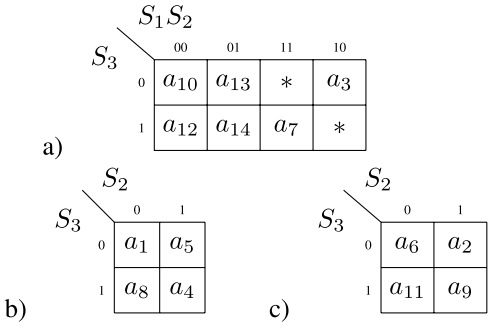

FIGURE 5. Outcome of state assignment for FSM $P_{CE}(F1)$.

$M_2 = M_3 = 4$. These classes determines the sets $X^1 = \{x_3, x_5\}$, $Y^1 = \{y_3, y_4, y_8, y_{10}, y_{11}\}$, $X^2 = \{x_1, x_2, x_4\}$, $Y^2 = \{y_1, y_2, y_6, y_8, y_9, y_{11}\}$, $X^3 = \{x_2, x_6, x_7\}$ and $Y^3 = \{y_1, y_3, y_5, y_7, y_{11}\}$. This gives $L_1 = 2, L_2 = L_3 = 3$.

Step 3: To execute this step, it is necessary to find the values of R_V and R_S. This is done using the outcome of the previous step. Using (19) gives $R_1 = 3$ and $R_2 = R_3 = 2$. Using (20) gives $R_S = 3$. So, the partial codes are created using the state variables $s_r \in S = \{s_1, s_2, s_3\}$. The states $a_m \in A^1$ are encoded using all elements of the set S. Other states are encoded using only variables $s_2, s_3 \in S$. Using (23) gives $R_V = 2$. This determines the set $V = \{v_1, v_2\}$.

Using (24) gives the value $R_{EC} = 5$. The set T includes 5 elements: $T = \{v_1, v_2, s_1, s_2, s_3\}$. These variables are represented by the outputs of flip-flops distributed inside *LUTerTY*.

Step 4: To encode states, we use the approach proposed in the article [25]. It allows minimizing the number of partial IMFs. One of the possible outcomes is shown in Fig. 5. It includes three Karnaugh maps with partial codes for each class $A^k \in \Pi_A$.

To encode classes $A^k \in \Pi_A$, we should use the code with all zeros to encode the class which includes the initial state $a_1 \in A$. In the discussed case, this is the class A^2. So, we choose the following class codes for our example: $C(A^1) = 01$, $C(A^2) = 00$ and $C(A^3) = 10$.

Using class and partial state codes gives the mixed state codes $EC(a_m)$. For example, the following codes can be obtained: $EC(a_1) = 0000$ (it follows from Fig. 5b and $C(A^2) = 00$), $EC(a_2) = 1010$ (it follows from Fig. 5c and $C(A^3) = 10$), $EC(a_3) = 01100$ (it follows from Fig. 5a and $C(A^1) = 01$) and so on. The tables of *LUTer*1–*LUTer*3 are based on the partial and mixed state codes.

Step 5: Each block *LUTerk* is represented by its direct structure table. A DST includes all columns from the equivalent STT and three additional columns. The additional columns contain: partial state codes of current states (the column $PC(a_m)$); mixed codes of next states (the column $EC(a_s)$); partial IMFs $D_r^k \in D^k$ required for loading the value of $EC(a_s)$ into RG (the column D_h). The *LUTer*1 is represented by Table 2. The table is based on 11 lines of

TABLE 2. Table of *LUTer*1.

a_m	$PC(a_m)$	a_s	$EC(a_s)$	X_h	Y_h	D_h	h
a_3	100	a_6	10000	1	y_3y_4	D_1	1
a_7	101	a_8	00001	x_3	y_3y_4	D_5	2
		a_{10}	01000	$\bar{x_3}$	y_8	D_2	3
a_{10}	000	a_{12}	01001	x_3	y_3	D_2D_5	4
		a_{11}	10001	$\bar{x_3}x_5$	y_3y_4	D_1D_5	5
		a_{13}	01010	$\bar{x_3}\bar{x_5}$	y_8	D_2D_4	6
a_{12}	001	a_{13}	01010	x_5	y_3y_8	D_2D_4	7
		a_4	01011	$\bar{x_5}$	$y_{10}y_{11}$	$D_2D_4D_5$	8
a_{13}	010	a_{14}	01011	x_3	y_{11}	$D_2D_4D_5$	9
		a_{11}	10001	$\bar{x_3}$	y_3y_{10}	D_1D_5	10
a_{14}	011	a_1	00000	1	y_3y_{10}	–	11

TABLE 3. Table of *LUTer*2.

a_m	$PC(a_m)$	a_s	$EC(a_s)$	X_h	Y_h	D_h	h
a_1	00	a_2	10010	x_1	y_6	D_1D_4	1
		a_1	00000	$\bar{x_1}$	y_1y_2	–	2
a_4	11	a_6	10000	x_2	y_1y_8	D_1	3
		a_3	01000	$\bar{x_2}x_4$	y_9	D_2D_3	4
		a_7	01101	$\bar{x_2}\bar{x_4}$	y_{11}	$D_2D_3D_5$	5
a_5	10	a_5	00010	x_1	y_1y_8	D_4	6
		a_7	01101	$\bar{x_1}x_2$	y_2y_9	$D_2D_3D_5$	7
		a_8	00001	$\bar{x_1}\bar{x_2}$	y_2y_6	D_5	8
a_8	01	a_1	00000	x_4	y_1y_6	–	9
		a_{10}	01000	$\bar{x_4}x_1$	y_1y_6	D_2	10
		a_{11}	10001	$\bar{x_4}\bar{x_1}$	y_2y_{11}	D_1D_5	11

Table 1 (lines 6, 16, 17, 23–25, 29–32). The *LUTer*2 is represented by Table 3. The table is based on 11 lines of Table 1 (lines 1, 2, 7–12, 18–20). The *LUTer*3 is represented by Table 4. The table is based on 11 lines of Table 1 (lines 3–5, 13–15, 21–22, 26–28).

Step 6: The partial functions are represented by SOPs (21)–(22). All these SOPs are created in the same way. Bellow, we show partial SOPs for functions D_1 and y_1.

Using Table 2, we can derive the following SOPs:

$$D_1^1 = s_1\bar{s_2}\bar{s_3} \vee \bar{s_1}\bar{s_2}\bar{s_3}\bar{x_3}\bar{x_5} \vee \bar{s_1}s_2\bar{s_3}\bar{x_3};$$
$$y_1^1 = 0. \tag{31}$$

Using Table 3, we can derive the following SOPs:

$$D_1^2 = \bar{s_2}\bar{s_3}x_1 \vee s_2s_3x_2 \vee \bar{s_2}s_3\bar{x_4}\bar{x_1};$$
$$y_1^2 = \bar{s_2}\bar{s_3}\bar{x_1} \vee s_2\bar{s_3}x_1 \vee \bar{s_2}s_3x_4. \tag{32}$$

Using Table 4, we can derive the following SOPs:

$$D_1^3 = \bar{s_2}\bar{s_3}\bar{x_7}x_6;$$
$$y_1^3 = s_2\bar{s_3}x_2 \vee \bar{s_2}\bar{s_3}x_7 \vee s_2s_3x_7 \vee s_2\bar{s_3}\bar{x_2}x_7. \tag{33}$$

We hope there is a transparent connection between Tables 2–4, on the one hand, and formulae (31)–(33), on the other hand. There is $y_1^1 = 0$, because there is no symbol y_1 in Table 2.

Step 7: The table of *LUTerTY* is constructed on the base of tables of partial functions (in the discussed example we should analyze Table 2–Table 4). This table includes a column

TABLE 4. Table of *LUTer*3.

a_m	$PC(a_m)$	a_s	$EC(a_s)$	X_h	Y_h	D_h	h
a_2	10	a_3	01100	x_2	y_1y_3	D_2D_3	1
		a_4	00011	$\bar{x_2}x_6$	y_5	D_4D_5	2
		a_5	00010	$\bar{x_2}\bar{x_6}$	y_3y_7	D_4	3
a_6	00	a_7	01101	x_7	y_1	$D_2D_3D_5$	4
		a_9	10011	$\bar{x_7}x_6$	y_7y_{11}	$D_1D_4D_5$	5
		a_{10}	01000	$\bar{x_7}\bar{x_6}$	y_3y_5	D_2	6
a_9	11	a_{10}	01000	x_7	y_1	D_2	7
		a_{12}	01001	$\bar{x_7}$	y_3y_{11}	D_2D_5	8
a_{11}	10	a_8	00001	x_2	y_3y_{11}	D_5	9
		a_1	00000	$\bar{x_2}x_7$	y_1y_5	–	10
		a_{13}	01010	$\bar{x_2}\bar{x_7}$	y_5y_7	D_2D_4	11

TABLE 5. Table of *LUTerTY*.

Function	1	2	3	Function	1	2	3
D_1	+	+	+	y_4	+	−	−
D_2	+	+	+	y_5	−	−	+
D_3	−	+	+	y_6	−	+	−
D_4	+	+	+	y_7	−	−	+
D_5	+	+	+	y_8	+	+	−
y_1	−	+	+	y_9	−	+	−
y_2	−	+	−	y_{10}	+	−	−
y_3	+	−	+	y_{11}	+	+	+

with functions $D_r \in D$ and $y_n \in Y$. Other columns contain number of blocks (1, 2,. . ., K). In the discussed case, this block is represented by Table 5.

There are symbols "+" and "−" in Table 5. For example, if some partial function $D_r^k \neq 0$, then there is the symbol "+" at the intersection of row D_r and column k. Obviously, if some partial function $y_n^k \neq 0$, then there is the symbol "+" at the intersection of row y_n and column k.

Step 8: The circuit of *LUTerTY* is represented by SBFs (25)–(26). They are constructed in the following way. If there is the symbol "+" at the intersection of row D_r and column k, then a SOP of function $D_r \in D$ includes the term $V_k \cdot D_r^k$. In this term, the member V_k is a conjunction corresponding to $C(A^k)$. Obviously, if there is the symbol "+" at the intersection of row y_n and column k, then a SOP of function $y_n \in Y$ includes the term $V_k \cdot y_n^k$.

In the discussed case, there is $V_1 = \bar{v_1}v_2$, $V_2 = \bar{v_1}\bar{v_2}$ and $V_3 = v_1\bar{v_2}$. Using these conjunctions and systems (31)–(33), we can create SOPs of full functions $D_1 \in D$ and $y_1 \in Y$:

$$D_1 = \bar{v_1}v_2D_1^1 \vee \bar{v_1}\bar{v_2}D_1^2 \vee v_1\bar{v_2}D_1^3;$$
$$y_1 = \bar{v_1}\bar{v_2}y_1^2 \vee v_1\bar{v_2}y_1^3. \quad (34)$$

All other SOPs (25)–(26) are created in the same way.

Step 9: Now we can estimate the value of LUT count for each block of FSM $P_{EC}(F1)$. To do it, we should use Table 5.

As follows from the column "1" of Table 5, LUTs of *LUTer*1 generate 4 partial IMFs and 5 partial outputs. So, this block includes 9 LUTs. Also, the following relations can be found from this column: $y_4^1 = y_4$ and $y_{10}^1 = y_{10}$ This means that only 7 outputs of *LUTer*1 are connected with LUTs of the second logic level.

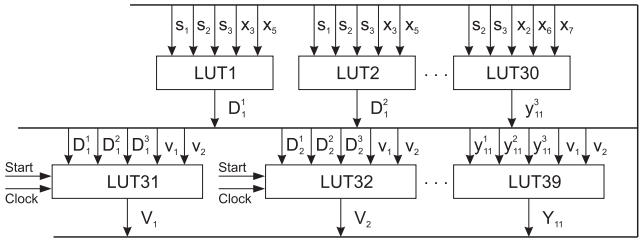

FIGURE 6. Logic circuit of FSM $P_{EC}(F1)$.

As follows from the column "2" of Table 5, LUTs of *LUTer*2 generate 5 partial IMFs and 6 partial outputs. So, this block includes 11 LUTs. Also, the following relations can be found from this column: $y_2^2 = y_2$, $y_6^2 = y_6$ and $y_9^2 = y_9$. This means that only 8 outputs of *LUTer*2 are connected with LUTs of the second logic level.

As follows from the column "3" of Table 5, LUTs of *LUTer*3 generate 5 partial IMFs and 5 partial outputs. So, this block includes 10 LUTs. Also, the following relations can be found from this column: $y_5^3 = y_5$ and $y_7^3 = y_7$. This means that only 8 outputs of *LUTer*3 are connected with LUTs of the second logic level.

The conducted analysis shows that there are totally 30 LUTs at the first circuit logic level. The outputs of 23 LUTs are connected with the block *LUTerTY* of FSM $P_{EC}(F1)$.

Obviously, some function should be generated by *LUTerTY* if there are at least two symbols "+" in the corresponding row of Table 5. As follows from Table 5, there are 9 such rows. For the discussed case, there is $R_V = 2$ and K = 3. So, the condition (30) holds. This means that each function is represented by a single-LUT circuit. So, there are 9 LUTs in the circuit of *LUTerTY*.

Thus, there are 39 LUTs and 5 flip-flops in the circuit of FSM $P_{EC}(F1)$. The circuit has two levels of logic. All circuits are implemented without the dedicated multiplexers. This circuit is shown in Fig. 6.

Now, we compare LUT counts for $P_E(F1)$ and $P_C(F1)$ FSM. As it is in the discussed example, we use 5-LUTs for implementing FSM circuits.

In the case of $P_E(F1)$, using the greedy algorithm from [24] gives the partition of the set A with $K1 = 4$ classes. These classes are the following: $A^1 = \{a_1, a_5, a_8\}$, $A^2 = \{a_2, a_3, a_4\}$, $A^3 = \{a_6, a_9, a_{11}\}$ and $A^4 = \{a_7, a_{10}, a_{12}, a_{13}, a_{14}\}$. Using (5) gives the following lengths of partial state codes: $R_1 = R_2 = R_3 = 2$, $R_4 = 3$. Using (6) gives $R_E = 9$. So, it is necessary 9 flip-flops if extended state codes are used. So, even for this simple example, the number of required flip-flops for $P_E(F1)$ is almost twice that for $P_{EC}(F1)$. This leads to an almost doubling of the number of feedback signals. As follows from [20], the more interconnections, the greater the power consumption and signal propagation time is. We have found that it is necessary: 1) 13 LUTs in the circuit of *LUTer*1; 2) 14 LUTs in the circuit of *LUTer*2; 3) 11 LUTs in the circuit of *LUTer*3; 4) 13 LUTs in the circuit of *LUTer*4. So, totally, there

are 51 LUTs generating PBFs. Also, it is necessary 17 LUTs in the circuit of $LUTer\mathcal{T}Y$. Because there is $K1 < S_L = 5$, the circuit of $LUTer\mathcal{T}Y$ has a single level. To summarize: the circuit contains 68 LUTs and has two logic levels.

In the case of $P_C(F1)$, using the greedy algorithm from [24] gives the partition of the set A with $K2 = 4$ classes. These classes are the following: $A^1 = \{a_7, a_{10}, a_{12}, a_{13}\}$, $A^2 = \{a_1, a_4, a_5, a_8\}$, $A^3 = \{a_2, a_6, a_9, a_{11}\}$ and $A^4 = \{a_3, a_{14}\}$. Using (5) gives the following lengths of partial state codes: $R_1 = R_2 = R_3 = 2$, $R_4 = 3$. Using (12) gives $R_S = 2$. Using (13) gives $R_C = 2$. So, it is necessary 4 flip-flops if composite state codes are used. So, the number of required flip-flops for $P_C(F1)$ is practically the same as it is for $P_{EC}(F1)$. We have found that it is necessary: 1) 9 LUTs in the circuit of $LUTer1$; 2) 10 LUTs in the circuit of $LUTer2$; 3) 9 LUTs in the circuit of $LUTer3$; 4) 7 LUTs in the circuit of $LUTer4$. So, there are 35 LUTs generating PBFs. Five outputs $y_n \in Y$ are generated by LUTs of $LUTer1$–$LUTer4$. So, only 6 LUTs are necessary to implement the rest of outputs. These LUTs are included into $LUTerTY$. Each function $f_i \in D \cup Y$ can be represented as a disjunction of four partial functions. Thus, the value of $R_Y + K2 = 6$ exceeds the value $S_L = 5$. As a result, it is necessary 8 LUTs to generate IMFs (17). Moreover, this part of $LUTerTY$ has two logic levels. So, is necessary 14 LUTs in the circuit of $LUTerTY$. To summarize: the circuit of $P_C(F1)$ FSM contains 49 LUTs and has three levels of them.

For the discussed example, the comparison of three different approaches of state assignment allows making the following conclusion. The proposed method allows obtaining the FSM circuit with a minimum number of LUTs (39 LUTs). Our approach allows saving 43% of LUTs compared to $P_E(F1)$ FSMs and 21% of LUTs compared to $P_C(F1)$ FSMs. The number of flip-flops for $P_{EC}(F1)$ is only one more than the minimum possible number, which is ensured by using the composite state codes. Our approach allows obtaining the fastest solution because: 1) ESC-based FSM has more interconnections and 2) CSC-based FSM has three levels of LUTs.

To get the electrical circuit of Mealy FSM $P_{EC}(F1)$, it is necessary to execute the step of technology mapping [26], [47], [48]. This is connected with using the sophisticated CAD tools. In the case of circuits implemented with internal resources of Virtex-7, the industrial package Vivado [41] should be used. Unfortunately, Vivado can design circuits of digital devices using basic LUTs with $S_L = 6$. So, using 5-LUTs, we cannot design and analyse the circuit of Mealy FSM $P_{EC}(F1)$ automatically. In the next Section, the results of experiments are shown.

VI. EXPERIMENTAL RESULTS

We conducted a lot of experiments to compare the basic characteristics of LUT-based P_{EC} Mealy FSMs with characteristics of FSM circuits based on some other models. The benchmark FSMs from the library [49] are used for the experiments. De facto, the library includes 48 benchmarks represented by their state transition tables. The tables are represented by KISS2-based files. The basic characteristics of benchmarks (the values of parameters M, L, and N) have a wide range. Due to it, these benchmarks are used by a lot of researchers as a base for comparison different FSM design methods. The characteristics of benchmark FSMs are represented by Table 6.

TABLE 6. Characteristics of benchmark Mealy FSMs [49].

Benchmark	L	N	R+L	M/R	H	Type
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
mCC	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

We execute the experiments using a personal computer with the following characteristics: CPU: Intel Core i7 6700K 4.2@4.4GHz, Memory: 16GB RAM 2400MHz CL15. Also, we use the Virtex-7 VC709 Evaluation Platform

(xc7vx690tffg1761-2) [50] and CAD tool Vivado v2019.1 (64-bit) [41]. There is $S_L = 6$ for FPGAs of Virtex-7. We use reports of Vivado to get the results of experiments. To enter Vivado, we use the CAD tool K2F [20]. This tool allows creating VHDL codes on the base of files represented in KISS2 format.

In our experiments, we have checked three basic characteristics of FSM circuits. They are the LUT counts (it allows to estimate the chip areas occupied by FSM circuits), performance (the maximum operating frequencies), and area-time products. To estimate the area, we use the LUT counts taken from reports of Vivado. The performance is represented by the maximum operating frequency which is achievable for each benchmark FSM. The area-time products are calculated as results of multiplication of the LUT counts by the maximum latency times obtained directly from Vivado reports. In our experiments, we use five FSM models. These models are P FSMs based on: 1) maximum binary state codes (JEDI); 2) one-hot state codes (One-hot by Vivado); 3) extended state codes (P_E-based FSMs); 4) composite state codes (P_C-based FSMs); 5) mixed state codes (P_{EC}-based FSMs proposed in this article).

As in our previous research [15], [16], the benchmarks are divided by five types. To do it, we use the relation between the values of $R + L$ and S_L. For basic LUTs of Virtex-7, there is $S_L = 6$. So, we use $S_L = 6$ to determine the types of benchmarks. The FSMs are trivial (type 0), if the result of summation of R and L does not exceed 6. The FSMs are simple (type 1), if the result of summation does not exceed 12. The FSMs are average (type 2), if the result of summation does not exceed 18. The FSMs are big (type 3), if the result of summation does not exceed 24. Otherwise, the benchmarks FSMs are very big (type 4). For very big FSMs, the following relation takes place: $R + L > 24$. As shown in the paper [16], there is a direct dependence between the degree of improving the basic characteristics of FSM circuits with help of SD-based methods and the type number.

In the discussed case, there is the following distribution of benchmarks. The FSMs represented by the benchmarks *bbtas*, *dk*17, *dk*27, *dk*512, *ex*3, *ex*5, *lion*, *lion*9, *mc*, *modulo*12, and *shiftreg* have the type 0. The FSMs represented by the benchmarks *bbara*, *bbsse*, *beecount*, *cse*, *dk*14, *dk*15, *dk*16, *donfile*, *ex*2, *ex*4, *ex*6, *ex*7, *keyb*, *mark*1, *opus*, *s*27, *s*386, *s*840, and *sse* have the type 1. The FSMs represented by the benchmarks *ex*1, *kirkman*, *planet*, *planet*1, *pma*, *s*1, *s*1488, *s*1494, *s*1*a*, *s*208, *styr*, and *tma* have type 2. There is a single FSM *sand* which has the type 3. Four remaining FSMs (*s*420, *s*510, *s*820, and *s*832) have the type 4.

The results of experiments are shown in Table 7–Table 11. Three tables are organized in the same manner (Table 7, Table 9 and Table 11). For these tables, the columns are marked by the names of investigated methods (JEDI, One-hot, P_C, P_{EC}). The names of benchmarks are written in the table rows. In these tables, we show results of experiments for type 0 benchmarks first, then for type 1, and so on. Parts of tables with the same type of benchmarks can be considered as sub-tables. Within each sub-table, benchmarks are shown in the alphabetical order. In the rows ''Total'', we place results of summation of values for corresponding columns. The row ''Percentage'' includes the percentage of summarized characteristics of FSM circuits produced by other methods respectively to P_{EC} FSMs. We use the model of P Mealy FSM as a starting point for methods JEDI and One-hot. Two tables (Table 8 and Table 9) include only summarized values of LUT counts and maximum operation frequencies for investigated methods.

Now, we will analyse the experimental results taken from the reports produced by Vivado (or calculated by us using these reports). The following information can be found in these tables: 1) the LUT counts for all benchmarks (Table 7); 2) the LUT counts for various types of benchmarks (Table 8); 3) the maximum operating frequencies for all benchmarks (Table 9); 4) the maximum operating frequencies for various types of benchmarks (Table 10); 5) the area-time products for all benchmarks (Table 11). The data for two last tables are calculated using the values of cycle times obtained from Vivado reports.

As follows from Table 7, the P_{EC}-based FSMs require fewer LUTs than it is for other investigated methods. Our approach produces circuits having 27.15% less 6-LUTs than it is for equivalent JEDI-based FSMs; 79.57% less 6-LUTs than it is for equivalent One-hot-based FSMs; 6.21% less 6-LUTs than it is for equivalent CSC-based FSMs; 22.21% less 6-LUTs than it is for equivalent ESC-based FSMs. While developing our method, we hoped that P_{EC}-based FSMs will require fewer LUTs in comparison with both equivalent P_E- and P_C-based FSMs. As follows from the last column of Table 7, our assumptions turn out to be correct.

To compare the LUT counts for different types of benchmarks, we have calculated these values for type 0, type 1 and types 2–4. This information is represented by Table 8.

As follows from Table 8, for the FSMs of type 0, practically all methods produce FSM circuits with the same LUT counts. This is explained by the fact that for these benchmarks the following condition takes place:

$$R + L \leq S_L. \tag{35}$$

If (35) holds, then all FSM outputs and IMFs are generated by single-LUT circuits. Obviously, there is no need in optimization if the condition (35) holds. Due to it, the JEDI-based optimization cannot produce circuits better than they are produced by other investigated methods. The negative outcome of one-hot assignment (loss in 39.39%) can be explained by the fact that there is $R = M$ for OHC-based FSMs Due to it, OHC-based FSMs require more LUTs for generating IMFs than it is for other models.

Starting from simple FSMs, P_{EC}-based FSMs require fewer LUTs than it is for other investigated methods. Our approach produces circuits having 12.33% less 6-LUTs than it is for equivalent JEDI-based FSMs; 75% less 6-LUTs

TABLE 7. Experimental results (LUT counts).

Benchmark	JEDI	One Hot	PC FSM	PE FSM	PEC FSM	Type
bbtas	5	5	5	5	5	0
dk17	5	12	5	5	5	0
dk27	4	5	4	4	4	0
dk512	9	10	9	9	9	0
ex3	9	9	9	9	9	0
ex5	10	13	10	10	10	0
lion	4	5	4	4	4	0
lion9	5	11	5	5	5	0
mc	4	7	4	4	4	0
modulo12	7	9	7	7	7	0
shiftreg	4	6	4	4	4	0
bbara	10	17	11	13	10	1
bbsse	24	37	21	26	20	1
beecount	14	19	12	14	12	1
cse	36	66	31	34	29	1
dk14	10	27	11	12	10	1
dk15	12	16	7	9	9	1
dk16	12	34	9	11	9	1
donfile	24	31	19	21	19	1
ex2	8	9	8	9	8	1
ex4	12	13	10	11	10	1
ex6	22	36	20	22	21	1
ex7	4	5	4	6	4	1
keyb	40	61	36	38	36	1
mark1	20	23	18	20	18	1
opus	22	28	23	25	22	1
s27	6	18	7	8	8	1
s386	22	39	20	22	20	1
s8	9	9	9	12	9	1
sse	30	37	27	29	26	1
ex1	53	74	37	44	34	2
kirkman	39	58	31	37	29	2
planet	88	131	72	87	66	2
planet1	88	131	72	87	66	2
pma	86	94	70	80	64	2
s1	61	99	51	61	48	2
s1488	108	131	83	96	77	2
s1494	110	132	82	94	78	2
s1a	43	81	38	47	35	2
s208	10	31	9	11	9	2
styr	81	120	65	79	61	2
tma	39	39	31	36	29	2
sand	114	132	97	108	89	3
s420	9	31	9	10	9	4
s510	32	48	28	31	24	4
s820	68	82	54	59	48	4
s832	62	79	50	61	43	4
Total:	1494	2110	1248	1436	1175	
Percentage:	127,15	179,57	106,21	122,21	100,00	

than it is for equivalent One-hot-based FSMs; 1% less 6-LUTs than it is for equivalent CSC-based FSMs; 14% less 6-LUTs than it is for equivalent ESC-based FSMs. We explain this gain by the fact that slightly fewer partial functions are formed for the proposed method than for equivalent CSC-based FSMs. The advantage with respect to ESC-based FSMs is explained by the fact that the latter require significantly more variables to generate partial state codes.

TABLE 8. Results for benchmarks of different types (LUT count).

Methods	JEDI	One Hot	PC FSM	PE FSM	PEC FSM	Type
Total:	66	92	66	66	66	0
Percentage:	100	139,39	100	100	100	
Total:	337	525	303	342	300	1
Percentage:	112,33	175	101	114	100	
Total:	1091	1493	879	1028	809	2,3,4
Percentage:	134,86	184,55	108,65	127,07	100	

The same facts also explain the growing advantage of our method connected with the growth of the complexity of the FSMs. As a result, our approach produces circuits having 34.86% less 6-LUTs than it is for equivalent JEDI-based FSMs; 84.55% less 6-LUTs than it is for equivalent One-hot-based FSMs; 8.65% less 6-LUTs than it is for equivalent CSC-based FSMs; 27.07% less 6-LUTs than it is for equivalent ESC-based FSMs. As follows from Table 8, the gain from the use of mixed state codes increases as the complexity of FSM grows (an increase in the number of states, inputs and outputs).

As follows from Table 9, our approach produces faster LUT-based FSM circuits relative to other investigated methods. The average win is from 2.36% (compared with P_C-based FSMs) to 22.39% relative to OHC-based FSMs. This fact is explained in the same way as it is for the gain in LUT counts. To compare frequencies for different types of FSMs, we created Table 10.

The following conclusions can be made on the base of Table 10. For trivial FSMs (type 0), the one-hot state assignment produces circuits which are slower than circuits produced using other state assignment methods. This is explained by the fact that, due to the violation of condition (35), these circuits have more interconnections compared to circuits generated by other methods. It is known [11] that interconnections significantly affect the performance of LUT-based circuits. All other models have the same performance because in fact they degenerate into P FSMs with maximum binary state codes.

Starting from simple FSMs (type 1), the advantage of the proposed method begins to appear. For simple FSMs, our method provides gain relative to JEDI-based FSMs (22.46%), One-hot-based FSMs (27.69%), and P_E-based FSMs (3.88%). It is interesting that there is the equal performance for P_{EC} and P_C FSMs. Apparently, these two models have the same state code length. As a consequence, they have the same number of logic levels (and, therefore, the same values of maximum operating frequencies). For other types of FSMs (2, 3 and 4), our method allows producing the fastest circuits. There is the following gain in FSM performance: 18.27.6% compared with JEDI-based FSMs; 28.33% compared with OHC-based FSMs; 7.41% compared with CSC-based FSMs; 12.87% compared with ESC-based FSMs.

So, the proposed approach allows increasing performance in comparison with other methods under study.

TABLE 9. Experimental results (the maximum operating frequency for all benchmarks, MHz).

Benchmark	JEDI	One Hot	PC FSM	PE FSM	PEC FSM	Type
bbtas	206,12	204,16	206,12	206,12	206,12	0
dk17	199,39	167,00	199,39	199,39	199,39	0
dk27	204,18	201,90	204,18	204,18	204,18	0
dk512	199,75	196,27	199,75	199,75	199,75	0
ex3	195,76	194,86	195,76	195,76	195,76	0
ex5	181,16	180,25	181,16	181,16	181,16	0
lion	202,35	204,00	202,35	202,35	202,35	0
lion9	206,38	185,22	206,38	206,38	206,38	0
mc	196,87	195,47	196,87	196,87	196,87	0
modulo12	207,13	207,00	207,13	207,13	207,13	0
shiftreg	276,26	263,57	276,26	276,26	276,26	0
bbara	212,21	193,39	262,22	252,44	262,22	1
bbsse	182,34	169,12	248,07	238,38	248,07	1
beecount	187,32	166,61	249,13	241,72	249,13	1
cse	178,12	163,64	252,43	247,86	252,43	1
dk14	193,85	172,65	233,63	223,84	233,63	1
dk15	194,87	185,36	236,02	226,97	236,02	1
dk16	197,13	174,79	264,93	253,82	264,93	1
donfile	203,65	184,00	258,11	248,19	258,11	1
ex2	200,14	198,57	251,45	241,61	251,45	1
ex4	192,83	177,71	247,14	237,31	247,14	1
ex6	176,59	163,80	248,21	238,35	248,21	1
ex7	200,60	200,84	250,14	240,83	250,14	1
keyb	168,43	143,47	235,01	224,98	235,01	1
mark1	176,18	162,39	237,76	227,47	237,76	1
opus	178,32	166,20	223,26	213,40	223,26	1
s27	199,13	191,50	248,04	238,53	248,04	1
s386	179,15	173,46	228,63	218,87	228,63	1
s8	181,23	178,95	223,04	213,65	223,04	1
sse	174,63	169,12	215,53	205,41	215,53	1
ex1	176,87	139,76	191,01	176,82	201,12	2
kirkman	156,68	154,00	182,19	177,15	199,17	2
planet	187,14	132,71	198,78	190,54	208,12	2
planet1	187,14	132,71	198,78	190,54	208,12	2
pma	169,83	146,18	184,17	179,83	192,23	2
s1	157,16	135,85	186,09	177,47	198,41	2
s1488	157,18	131,94	188,03	174,31	197,14	2
s1494	164,34	145,75	187,24	172,05	193,01	2
s1a	169,17	176,40	189,17	169,53	199,32	2
s208	178,76	176,46	188,42	179,28	201,70	2
styr	145,64	129,92	187,53	172,24	198,19	2
tma	164,14	147,80	189,18	178,59	199,09	2
sand	126,82	115,97	149,65	141,14	163,12	3
s420	117,25	106,46	133,14	129,32	154,02	4
s510	118,32	107,65	137,22	127,76	162,17	4
s820	116,58	103,16	129,26	118,15	157,13	4
s832	113,78	103,23	133,31	124,22	157,31	4
Total:	8458,87	7821,22	9841,27	9487,92	10077,47	
Percentage:	83,94	77,61	97,66	94,15	100,00	

TABLE 10. Results for benchmarks of different types (maximum operating frequency, MHz).

Methods	JEDI	One Hot	PC FSM	PE FSM	PEC FSM	Type
Total:	2275,35	2199,7	2275,35	2275,35	2275,35	0
Percentage:	100	96,68	100	100	100	
Total:	3576,72	3335,57	4612,75	4433,63	4612,25	1
Percentage:	77,54	72,31	100	96,12	100	
Total:	2606,8	2285,95	2953,17	2778,94	3189,37	2,3,4
Percentage:	81,73	71,67	92,59	87,13	100	

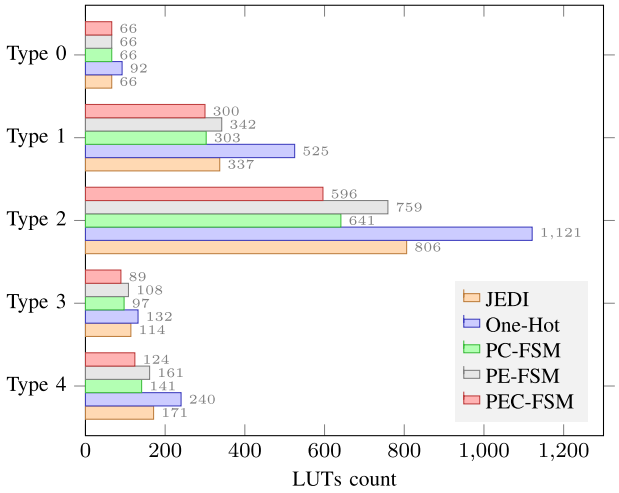

FIGURE 7. Total numbers of LUTs for each type of benchmarks.

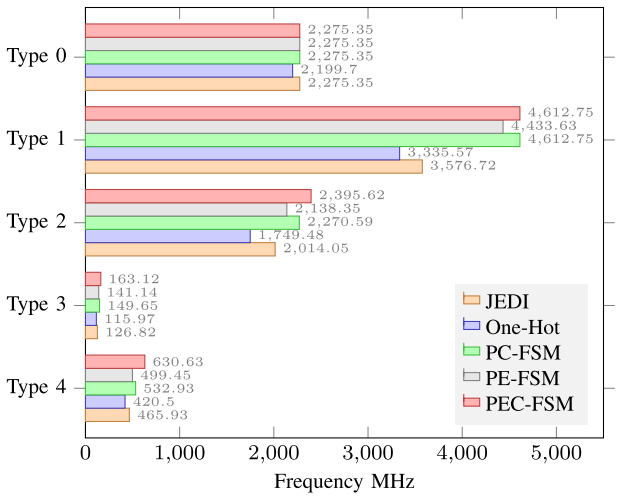

FIGURE 8. Total values of maximum operating frequencies for each type of benchmarks.

Moreover, the gain increases as the complexity of the FSMs increases.

To better visualize the results of experiments, we presented them in the form of diagrams (bar charts). These charts show the total number of LUTs (Fig. 7) and the total maximum operating frequency (Fig. 8).

To construct the diagram (Fig. 7), we used data from Table 7. The table includes the sums of LUTs required for each FSM model within each group of benchmarks (benchmarks of the same type). To construct the diagram (Fig. 8), we used data from Table 9. The table includes the sums of maximum operating frequencies for each FSM model within each group of benchmarks (types 0–4). We hope that these diagrams provide a better understanding of the specifics of the proposed state assignment method.

The main goal of the proposed method was to reduce the number of LUTs (the chip area occupied by FSM circuit) compared to this value for equivalent P_C and P_E FSMs.

TABLE 11. Experimental results (the area-time products for all benchmarks).

Benchmark	JEDI	One Hot	PC FSM	PE FSM	PEC FSM	Type
bbtas	24,26	24,49	24,26	24,26	24,26	0
dk17	25,08	71,86	25,08	25,08	25,08	0
dk27	19,59	24,76	19,59	19,59	19,59	0
dk512	45,06	50,95	45,06	45,06	45,06	0
ex3	45,97	46,19	45,97	45,97	45,97	0
ex5	55,20	72,12	55,20	55,20	55,20	0
lion	19,77	24,51	19,77	19,77	19,77	0
lion9	24,23	59,39	24,23	24,23	24,23	0
mc	20,32	35,81	20,32	20,32	20,32	0
modulo12	33,80	43,48	33,80	33,80	33,80	0
shiftreg	14,48	22,76	14,48	14,48	14,48	0
bbara	47,12	87,91	41,95	51,50	38,14	1
bbsse	131,62	218,78	84,65	109,07	80,62	1
beecount	74,74	114,04	48,17	57,92	48,17	1
cse	202,11	403,32	122,81	137,17	114,88	1
dk14	51,59	156,39	47,08	53,61	42,80	1
dk15	61,58	86,32	29,66	39,65	38,13	1
dk16	60,87	194,52	33,97	43,34	33,97	1
donfile	117,85	168,48	73,61	84,61	73,61	1
ex2	39,97	45,32	31,82	37,25	31,82	1
ex4	62,23	73,15	40,46	46,35	40,46	1
ex6	124,58	219,78	80,58	92,30	84,61	1
ex7	19,94	24,90	15,99	24,91	15,99	1
keyb	237,49	425,18	153,18	168,90	153,18	1
mark1	113,52	141,63	75,71	87,92	75,71	1
opus	123,37	168,47	103,02	117,15	98,54	1
s27	30,13	93,99	28,22	33,54	32,25	1
s386	122,80	224,84	87,48	100,52	87,48	1
s8	49,66	50,29	40,35	56,17	40,35	1
sse	171,79	218,78	125,27	141,18	120,63	1
ex1	299,66	529,48	193,71	248,84	169,05	2
kirkman	248,91	376,62	170,15	208,86	145,60	2
planet	470,24	987,11	362,21	456,60	317,12	2
planet1	470,24	987,11	362,21	456,60	317,12	2
pma	506,39	643,04	380,08	444,86	332,93	2
s1	388,14	728,74	274,06	343,72	241,92	2
s1488	687,11	992,88	441,42	550,74	390,59	2
s1494	669,34	905,66	437,94	546,35	404,12	2
s1a	254,18	459,18	200,88	277,24	175,60	2
s208	55,94	175,68	47,77	61,36	44,62	2
styr	556,17	923,65	346,61	458,66	307,79	2
tma	237,60	263,87	163,87	201,58	145,66	2
sand	898,91	1138,23	648,18	765,20	545,61	3
s420	76,76	291,19	67,60	77,33	58,43	4
s510	270,45	445,89	204,05	242,64	147,99	4
s820	583,29	794,88	417,76	499,37	305,48	4
s832	544,91	765,28	375,07	491,06	273,35	4
Total:	9388,96	15000,91	6685,28	8141,82	5902,09	
Percentage:	159,08	254,16	113,27	137,95	100,00	

The results of experiments show that this goal has been achieved. In addition, starting from average FSMs (type 2), our approach simultaneously allows increasing the maximum operating frequency. Due to it, our approach produces FSM circuits with the best values of area-time products. The corresponding values are shown in Table 11.

As follows from Table 11, our approach provides the following average gain: 1)59.08% regarding JEDI-based FSMs; 2)154.16% regarding OHC- based FSMs; 3) 13.27% regarding CSC-based FSMs and 4) 32.05% regarding *ESC*-based FSMs. We did not show it, but the gain obtained by our approach increases with the increasing the FSM type. It is known [10], that the smaller the value of the product of the number of LUTs in circuit 1 and its cycle time compared to this product for circuit 2, the circuit 1 either requires fewer LUTs or has a higher performance than circuit 2. Both of these phenomena are inherent in our proposed method.

So, the results of our experiments show that the proposed approach can be used instead of other models starting from simple FSMs (type 1). Our approach allows improving LUT counts, maximum operating frequency, and area-time products compared with other investigated design methods. We think that our approach has rather good potential and can be used in CAD systems targeting FPGA-based Mealy FSMs.

VII. CONCLUSION

Very often, modern digital systems are implemented using FPGA chips. Current FPGAs are very complicated devices having up to 7 billion transistors [17], [43]. Due to it, a single chip can be used for implementing rather complex circuits of various digital systems. As the complexity of these systems increases, the contradiction between a significant number of system inputs and a very small number of LUT inputs increases, too. Modern LUTs have around 6 inputs. Obviously, this value is rather small compared with numbers of literals in SBFs representing FSM circuits. This leads to the need for using different methods of functional decomposition in LUT-based FSM design. It is known [20] that the functional decomposition leads to multi-level LUT-based FSM circuits having very complicated systems of spaghetti-type interconnections.

To improve the characteristics of LUT-based FSM circuits compared with their counterparts based on functional decomposition, various methods of structural decomposition can be applied [20]. As follows from our previous research [15], [16], the characteristics of LUT-based Mealy FSM circuits can be improved using either extended or composite state codes. These approaches have both positive and negative sides.

In our current paper, we propose a new method of state assignment, namely, the assignment based on mixed state codes. This approach allows combining positive features of the methods based on either extended or composite state codes. Also, the proposed method is free from the disadvantages inherent in these two methods. As it is for CSCs, mixed state code is represented by a concatenation of a class code and the code of a state as an element of this class. However, unlike CSCs, the mixed state codes can have different lengths of partial state codes (this is a positive feature borrowed from ESCs). The proposed approach preserves the flexibility inherent in ESCs and the almost minimum number of variables inherent in CSCs. This approach leads to two-level FSM circuits which require fewer LUTs than their counterparts based on other state assignment methods. Moreover, the MSC-based FSMs have slightly better performance as their ESC- and CSC-based counterparts.

REFERENCES

[1] S. C. Suh, U. J. Tanik, and J. N. Carbone, *Applied Cyber-Physical Systems*. New York, NY, USA: Springer, 2013.

[2] L. Ashford and S. S. Arunkumar, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, 2nd ed. Cambridge, MA, USA: MIT Press, 2016.

[3] P. Marwedel, *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things*, 3rd ed. New York, NY, USA: Springer, 2018.

[4] D. D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner, *Embedded System Design: Modeling, Synthesis and Verification*, 1st ed. New York, NY, USA: Springer, 2009.

[5] S. Baranov, *Finite State Machines and Algorithmic State Machines*. Seattle, WA, USA: Amazon, Jan. 2018.

[6] I. Grout, *Digital Systems Design With FPGAs and CPLDs*. Amsterdam, The Netherlands: Elsevier Science, 2011. [Online]. Available: https://books.google.pl/books?id=vggmNXdzayYC

[7] M. Kubica, A. Opara, and D. Kania, *Technology Mapping for LUT-Based FPGA*. Cham, Switzerland: Springer, Jan. 2021.

[8] M. Kubica and D. Kania, "Technology mapping oriented to adaptive logic modules," *Bull. Polish Acad. Sci. Tech. Sci.*, vol. 67, no. 5, pp. 947–956, 2019.

[9] A. Mishchenko, S. Chatterjee, and R. K. Brayton, "Improvements to technology mapping for LUT-based FPGAs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 2, pp. 240–253, Feb. 2007.

[10] M. M. Islam, M. S. Hossain, M. Shahjalal, M. K. Hasan, and Y. M. Jang, "Area-time efficient hardware implementation of modular multiplication for elliptic curve cryptography," *IEEE Access*, vol. 8, pp. 73898–73906, 2020.

[11] W. Feng, J. Greene, and A. Mishchenko, "Improving FPGA performance with a S44 LUT structure," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays* New York, NY, USA: Association for Computing Machinery, Feb. 2018, pp. 61–66, doi: 10.1145/3174243.3174272.

[12] J. Ruiz-Rosero, G. Ramirez-Gonzalez, and R. Khanna, "Field programmable gate array applications—A scientometric review," *Computation*, vol. 7, no. 4, p. 63, Nov. 2019.

[13] M. Amagasaki and Y. Shibata, *FPGA Structure*, Sep. 2018, pp. 47–86.

[14] I. Kuon, R. Tessier, and J. Rose, "FPGA architecture: Survey and shallenges—found trends," *Electr. Design Autom.*, no. 2, pp. 135–253, 2008.

[15] A. Barkalov, L. Titarenko, K. Krzywicki, and S. Saburova, "Improving characteristics of LUT-based mealy FSMs with twofold state assignment," *Electronics*, vol. 10, no. 8, p. 901, Apr. 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/8/901

[16] A. Barkalov, L. Titarenko, and K. Krzywicki, "Improving characteristics of LUT-based sequential blocks for cyber-physical systems," *Energies*, vol. 15, no. 7, p. 2636, Apr. 2022. [Online]. Available: https://www.mdpi.com/1996-1073/15/7/2636

[17] *Xilinx*. Accessed: Jan. 2024. [Online]. Available: http://www.xilinx.com

[18] Xilinx Corporation. (2019). *Virtex-7 Family Overview*. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds183_Virtex_7_Data_Sheet.pdf

[19] G. De Micheli, *Synthesis and Optimization of Digital Circuits*. New York, NY, USA: McGraw-Hill, 1994.

[20] A. Barkalov, L. Titarenko, and K. Krzywicki, "Structural decomposition in FSM design: Roots, evolution, current state—A review," *Electronics*, vol. 10, no. 10, p. 1174, May 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/10/1174

[21] M. Kubica, D. Kania, and J. Kulisz, "A technology mapping of FSMs based on a graph of excitations and outputs," *IEEE Access*, vol. 7, pp. 16123–16131, 2019.

[22] K. Chapman, *Multiplexer Design Techniques for Datapath Performance With Minimized Routing Resources*. Xilinx All Programmable, 2014, Accessed: Jan. 2024.

[23] C. Scholl, *Functional Decomposition With Application to FPGA Synthesis*. Boston, MA, USA: Kluwer Academic, 2001.

[24] A. Barkalov, L. Titarenko, and K. Mielcarek, "Hardware reduction for LUT–based mealy FSMs," *Int. J. Appl. Math. Comput. Sci.*, vol. 28, no. 3, pp. 595–607, Sep. 2018.

[25] A. Barkalov, L. Titarenko, and K. Mielcarek, "Improving characteristics of LUT-based mealy FSMs," *Int. J. Appl. Math. Comput. Science*, vol. 30, no. 4, pp. 745–759, 2020.

[26] M. Kubica and D. Kania, "Area–oriented technology mapping for LUT–based logic blocks," *Int. J. Appl. Math. Comput. Sci.*, vol. 27, no. 1, pp. 207–222, Mar. 2017.

[27] L. Machado and J. Cortadella, "Support-reducing decomposition for FPGA mapping," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 1, pp. 213–224, Jan. 2020. [Online]. Available: https://www.cs.upc.edu/~jordicf/gavina/BIB/files/iwls2018_FPGA.pdf

[28] R. Senhadji-Navarro and I. Garcia-Vargas, "High-performance architecture for binary-tree-based finite state machines," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 4, pp. 796–805, Apr. 2018.

[29] S. Chattopadhyay, "Area conscious state assignment with flip-flop and output polarity selection for finite state machine synthesis—A genetic algorithm approach," *Comput. J.*, vol. 48, no. 4, pp. 443–450, May 2005.

[30] C. Chen, J. Zhao, and M. Ahmadi, "A semi-gray encoding algorithm for low-power state assignment," in *Proc. Int. Symp. Circuits Syst. (ISCAS)*, vol. 5, 2003, pp. 389–392.

[31] G. De Micheli, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Optimal state assignment for finite state machines," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. CAD-4, no. 3, pp. 269–285, Jul. 1985, doi: 10.1109/TCAD.1985.1270123.

[32] Y. Tao, Y. Zhang, Q. Wang, and J. Cao, "MPGA: An evolutionary state assignment for dynamic and leakage power reduction in FSM synthesis," *IET Comput. Digit. Techn.*, vol. 12, no. 3, pp. 111–120, May 2018.

[33] G. Venkataraman, S. M. Reddy, and I. Pomeranz, "GALLOP: Genetic algorithm based low power FSM synthesis by simultaneous partitioning and state assignment," in *Proc. 16th Int. Conf. VLSI Design, Proceedings.*, 2003, pp. 533–538.

[34] R. Agrawal, M. Borowczak, and R. Vemuri, "A state encoding methodology for side-channel security vs. power trade-off exploration," in *Proc. 32nd Int. Conf. VLSI Design 18th Int. Conf. Embedded Syst. (VLSID)*, Jan. 2019, pp. 70–75.

[35] A. H. El-Maleh, "A probabilistic pairwise swap search state assignment algorithm for sequential circuit optimization," *Integration*, vol. 56, pp. 32–43, Jan. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167926016300384

[36] A. H. El-Maleh, "Finite state machine-based fault tolerance technique with enhanced area and power of synthesised sequential circuits," *IET Comput. Digit. Techn.*, vol. 11, no. 4, pp. 159–164, Jul. 2017.

[37] A. H. El-Maleh, "A probabilistic Tabu search state assignment algorithm for area and power optimization of sequential circuits," *Arabian J. for Sci. Eng.*, vol. 45, no. 8, pp. 6273–6285, Aug. 2020, doi: 10.1007/s13369-020-04697-y.

[38] S. Park, S. Cho, S. Yang, and M. Ciesielski, "A new state assignment technique for testing and low power," in *Proc. 41st Annu. Design Autom. Conf.* New York, NY, USA: Association for Computing Machinery, Jun. 2004, pp. 510–513.

[39] E. Sentowich, K. Singh, L. Lavango, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. S. P, R. Bryton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," in *Proc. Int. Conf. Comput. Design (ICCD)*, 1992, pp. 328–333.

[40] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Computer Aided Verification*, T. Touili, B. Cook, and P. Jackson, Eds. Berlin, Germany: Springer, 2010, pp. 24–40.

[41] *Vivado*. Accessed: Jan. 2024. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado.html

[42] *QuartusPrime*. Accessed: Jan. 2024. [Online]. Available: https://www.intel.pl/content/www/pl/pl/software/programmable/quartus-prime/overview.html

[43] *Altera*. Accessed: Jan. 2024. [Online]. Available: http://www.altera.com

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

[45] I. Skliarova, V. Sklyarov, and A. Sudnitson, *Design of FPGA-Based Circuits Using Hierarchical Finite State Machines*. Tallinn, Estonia: TUT Press, 2012.

[46] A. Opara, M. Kubica, and D. Kania, "Decomposition approaches for power reduction," *IEEE Access*, vol. 11, pp. 29417–29429, 2023.

[47] R. Senhadji-Navarro and I. Garcia-Vargas, "Mapping outputs and states encoding bits to outputs using multiplexers in finite state machine implementations," *Electronics*, vol. 12, no. 3, p. 502, Jan. 2023. [Online]. Available: https://www.mdpi.com/2079-9292/12/3/502

[48] R. Senhadji-Navarro and I. Garcia-Vargas, "Mapping arbitrary logic functions onto carry chains in FPGAs," *Electronics*, vol. 11, no. 1, p. 27, Dec. 2021. [Online]. Available: https://www.mdpi.com/2079-9292/11/1/27
[49] K. McElvain. (1993). *Lgsynth93 Benchmark Set. Version 4.0*. Accessed: Feb. 2018. [Online]. Available: https://people.engr.ncsu.edu/brglez/CBL/benchmarks/LGSynth93/LGSynth93.tar
[50] Xilinx. (2024). *VC709 Evaluation Board for the Virtex-7 FPGA*. Accessed: Jan. 2024. [Online]. Available: https://www.xilinx.com/support/documentation/boards_and_kits/vc709/ug887-vc709-eval-board-v7-fpga.pdf

ALEXANDER BARKALOV received the M.Sc. degree in computer engineering from Donetsk Polytechnical Institute (currently Donetsk National Technical University), Ukraine, in 1976, the Ph.D. degree in computer science from the Leningrad Institute of Fine Mechanics and Optics, Russia, in 1983, and the Doctor of Technical Sciences degree in computer science from the Institute of Cybernetics (Kiev), in 1995. Since 2003, he has been a Professor of computer engineering with the Institute of Informatics and Electronics, University of Zielona Góra, Poland. His current research interests include the theory of digital automata, especially the methods of synthesis and optimization of control units implemented with field-programmable logic devices.

LARYSA TITARENKO received the M.Sc., Ph.D., and Doctor of Technical Sciences degrees in telecommunications from Kharkiv National University of Radioelectronics, Ukraine, in 1993, 1996, and 2005, respectively. Since 2007, she has been a Professor of telecommunications with the Institute of Informatics and Electronics, University of Zielona Góra, Poland. She has taken part in several research projects sponsored by the Ministry of Science and Higher Education of Ukraine, from 1993 to 2005. Her current research interests include the theory of telecommunication systems, the theory of antennas, and the theory of digital automata and its applications.

KAMIL MIELCAREK received the M.Sc. degree in computer engineering from the Technical University of Zielona Góra, Poland, in 1995, and the Ph.D. degree in computer science from the University of Zielona Góra, Poland, in 2010. Since 2001, he has been a Lecturer with the University of Zielona Góra. His current interests include the methods of logic synthesis and optimization of control units in FPGA logic devices, VLSI-based FSMs, hardware description languages, perfect graphs, petri nets, algorithmic theory, and the safety of UNIX and network systems.

MAŁGORZATA MAZURKIEWICZ received the M.Sc. degree in computer science from the Technical University of Zielona Góra, Poland, in 1999, and the Ph.D. degree in computer science from the University of Zielona Góra, Poland, in 2007. Since 2007, she has been an Assistant Professor with the Faculty of Computer, Electrical and Control Engineering, University of Zielona Góra. Her research interests include the methods of digital circuit synthesis, design, and PLC programming.