electronics

MDPI

Article

Using Codes of Output Collections for Hardware Reduction in Circuits of LUT-Based Finite State Machines

Alexander Barkalov [1,2], Larysa Titarenko [1,3], Kazimierz Krzywicki [4] and Kamil Mielcarek [1,*]

[1] Institute of Metrology, Electronics and Computer Science, University of Zielona Góra, ul. Licealna 9, 65-417 Zielona Góra, Poland; a.barkalov@imei.uz.zgora.pl (A.B.); l.titarenko@imei.uz.zgora.pl (L.T.)
[2] Department of Computer Science and Information Technology, Vasyl Stus' Donetsk National University (in Vinnytsia), 600-richya Str. 21, 21021 Vinnytsia, Ukraine
[3] Department of Infocommunication Engineering, Faculty of Infocommunications, Kharkiv National University of Radio Electronics, Nauky Avenue 14, 61166 Kharkiv, Ukraine
[4] Department of Technology, The Jacob of Paradies University, ul. Teatralna 25, 66-400 Gorzów Wielkopolski, Poland; kkrzywicki@ajp.edu.pl
* Correspondence: k.mielcarek@imei.uz.zgora.pl

Abstract: A method is proposed which aims to reduce the hardware in FPGA-based circuits of Mealy finite state machines (FSMs). The proposed method is a type of structural decomposition method. Its main goal is the reducing the number of look-up table (LUT) elements in FSM circuits compared to the three-block FSM circuit. The main idea of the proposed method is the using codes of collections of FSM outputs for replacing the FSM inputs and state variables. The interstate transitions are defined using collections of outputs generated in two adjacent cycles of synchronization. One, of output collection codes, is kept into a register. To optimize block-generating FSM outputs, a new type of state codes is proposed. A state is encoded as an element of some class of states. This approach allows both the number of logic levels and inter-level interconnections in LUT-based FSM circuit to be diminished. An example of an LUT-based Mealy FSM circuit with the proposed method applied is shown. Moreover, the results of our research are represented. The research was conducted using the CAD tool Vivado by Xilinx. The experiments prove that the proposed approach allows the reduction of hardware compared with such known methods as Auto and One-hot of Vivado, and JEDI. Moreover, the proposed approach gives better results than a method based on the simultaneous replacement of inputs and encoding collections of outputs. Compared to circuits of the three-block FSMs, the LUT counts are reduced by an average of 10.07% without significant reduction in the value of operating frequency. The gain in LUT counts increases with the increasing the numbers of FSM states and inputs.

Keywords: Mealy FSM; FPGA; LUT count; synthesis; collection of outputs

Citation: Barkalov, A.; Titarenko, L.; Krzywicki, K.; Mielcarek, K. Using Codes of Output Collections for Hardware Reduction in Circuits of LUT-Based Finite State Machines. *Electronics* **2022**, *11*, 2050. https://doi.org/10.3390/electronics11132050

Academic Editors: Paris Kitsos, Dah-Jye Lee and Spyridon Nikolaidis

Received: 28 May 2022
Accepted: 27 June 2022
Published: 29 June 2022

1. Introduction

Since the 1950s, the model of Mealy finite state machine (FSM) [1] has been widely used in the design of sequential circuits [2–4]. Now, this model is used, for example, to set the behaviour of such sequential blocks as: (1) control devices of digital systems [5,6]; (2) serial communication and display protocols [7]; (3) various software tools of embedded systems [8]; (4) control-dominated systems [9]; (5) different systems in robotics [10] (6) hardware–software interfaces of embedded systems [3]; (7) the activation functions for deep neutral networks [11,12] and so on. Currently, research related to finite state machines is actively developing [9,13,14]. This justifies the choice of this model as an object of our current research.

To improve the quality of FSM-based blocks, it is necessary to improve such characteristics of corresponding FSM circuits as chip areas occupied by them, operating frequency and power dissipation. Due to this, there is a continuous interest in developing synthesis

methods leading to optimization of these characteristics. As a rule, the less chip area is occupied by an FSM circuit, the less power it consumes [15,16]. Thus, it is very important to reduce the chip area occupied by an FSM circuit.

Today, a lot of digital systems are implemented using field programmable gate arrays (FPGAs) [17]. For example, FPGAs are widely used for implementing hardware accelerators [18]. In [19], around 1700 examples of various applications of FPGAs in a wide variety of digital systems are listed. Taking into account such popularity of FPGAs, we chose these chips as a platform for implementing Mealy FSMs circuits. Practically from the beginning of the FPGA era, the largest manufacturer of FPGA chips is Xilinx [20]. This explains why we focus our current research on solutions of Xilinx. We discuss FSM circuits implemented using such internal resources of an FPGA chip as look-up table (LUT) elements, programmable flip-flops, programmable interconnects, synchronization tree, and programmable input–outputs.

To optimize the basic characteristics of FSM circuits, the methods of structural decomposition (SD) can be used [21]. These methods allow structuring an LUT-based FSM circuit and presenting it as a composition of several large logical blocks. Each block is represented by a system of Boolean functions (SBF) having unique arguments [22]. In [23], we propose an FSM design method based on simultaneously applying two methods of SD. These methods are: (1) the replacement of FSM inputs [5] and (2) the encoding of collections of FSM outputs [5]. To apply these methods, it is necessary to generate two SBFs having two sets of additional variables. To implement circuits for these SBFs, it is necessary to use some chip resources. There are three logic levels in FSM circuits based on [23]. In this article, we propose a method which allows the exclusion of a block generating the additional variables replacing the FSM inputs. We propose to replace FSM inputs by the same variables which encode the collections of FSM outputs.

The main contribution of this paper is a novel design method aimed at reducing the LUT count in circuits of the three-block FPGA-based Mealy FSMs [23]. The proposed method is based on: (1) using the same additional variables for producing both input memory functions (IMFs) and FSM outputs and (2) encoding of the FSM state using class-state codes (CSCs) proposed in this paper. Saving on the number of elements is achieved by reducing both the number of additional arguments and state variables compared to [23].

The further text of the paper includes five sections. Section 2 is devoted to the background of FPGA-based Mealy FSMs. Section 3 includes the discussion of the state-of-the-art. The main idea of the proposed method is shown in Section 4. Section 5 shows an example of FSM circuit synthesis. The results of experiments and their analysis can be found in Section 6. A short conclusion is given in Section 7.

2. Background of Designing LUT-Based Mealy FSMs

The design process starts from formal representation of interstate transitions. This can be done using various tools [24]. Very often, the behaviour is defined using either state transitions graphs (STGs) or state transitions tables (STTs) [4]. We also use these tools in our paper. There are various formal methods using which it is possible to obtain SBFs representing an FSM logic circuit [4]. These SBFs define dependencies between FSM outputs and IMFs on the one hand, and FSM inputs and state variables on the other hand.

The FSM inputs form a set $X = \{x_1, \ldots, x_L\}$, the FSM outputs form a set $Y = \{y_1, \ldots, y_N\}$, and the FSM states form a set $A = \{a_1, \ldots, a_M\}$. The inputs cause interstate transitions. To synthesise an FSM circuit, the states $a_m \in A$ are encoded by binary codes $K(a_m)$ having R bits. The r-th bit of $K(a_m)$ corresponds to a state variable $T_r \in T$, where $T = \{T_1, \ldots, T_R\}$ is a set of state variables. The minimum number of state variables is determined as

$$R = \lceil log_2 M \rceil. \quad (1)$$

State codes based on (1) are called maximal state codes [25]. State codes are kept into a state code register (SCR) [5]. As a rule, in the case of FPGA-based FSMs, the SCR has informational inputs of D type [25,26]. The content of SCR is determined by the IMFs

forming a set $\Phi = \{D_1, \ldots, D_R\}$. A synchronization pulse *Clock* allows the entry of a state code into SCR. A single pulse *Start* allows the entry of an initial state code into SCR.

To construct SBFs determining an FSM circuit, the initial STT (or STG) should be transformed into a direct structure table (DST) [5]. An STT includes five columns [4]. These columns are: a current state a_m; a state of transition a_S; a conjunction of inputs (or their complements) X_h determining the transition from a_m into a_S; a collection of outputs (CO) Y_h generated during the h-th transition; h is a column with numbers of transitions ($h \in \{1, \ldots, H\}$). Compared to an STT, a DST includes three additional columns [5].

These columns are: the code of the current state $K(a_m)$; the code of the next state $K(a_S)$; a collection of IMFs $\Phi_h \subseteq \Phi$ necessary to load the next state code into SCR.

A DST is a base for deriving the SBFs

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

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

These SBFs determine a logic circuit of P Mealy FSM Figure 1.

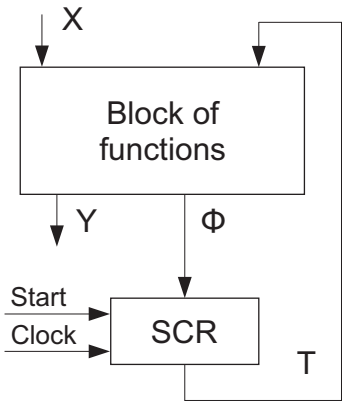
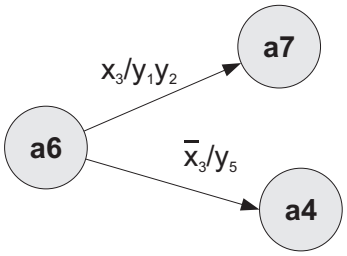


Figure 1. Structural diagram of P Mealy FSM.

In Figure 1, a block of functions implements the SBFs (2) and (3). The SCR includes R flip-flops each of which corresponds to one bit of a current state code. The meaning of pulses *Start* and *Clock* is clear.

A fragment of the STG is shown in Figure 2. It shows transitions between the current state a_6 and states of transition a_4 (the transition number $h = 12$) and a_7 (the transition number $h = 13$) of some Mealy FSM. This STG can be replaced by equivalent fragments of the STT Figure 2b and DST Figure 2c.

(a)



(b)

a_m	a_S	X_h	Y_h	h
a_6	a_7	x_3	y_1y_2	12
	a_4	$\bar{x_3}$	y_5	13

(c)

a_m	$K(a_m)$	s_S	$K(a_S)$	X_h	Y_h	Φ_h	h
a_6	101	a_7	110	x_3	y_1y_2	D_1D_2	12
		a_4	011	$\bar{x_3}$	y_5	D_2D_3	13

Figure 2. Equivalent fragments of STG (**a**), STT (**b**) and DST (**c**).

As follows from Figure 2b, the transition $\langle a_6, a_7 \rangle$ is caused by the input signal $X_{12} = x_3$. The transition is accompanied by the producing outputs $y_1, y_2 \in Y$. Row 12 of the STT Figure 2b reflects this transition. In the same manner, row 13 of the STT is filled Figure 2b. If, for example, there is $M = 7$, then using (1) gives $R = 3$ and two sets: $T = \{T_1, T_2, T_3\}$ and $\Phi = \{D_1, D_2, D_3\}$. Let the states from Figure 2a have the following codes: $K(a_4) = 011$, $K(a_6) = 101$ and $K(a_7) = 110$. These codes and corresponding IMFs are written in the rows 12 and 13 of DST Figure 2c. The row 12 determines a product term $F_{12} = T_1 \bar{T}_2 T_3 x_3$, the row 13 determines a term $F_{13} = T_1 \bar{T}_2 T_3 \bar{x}_3$. These terms enter sum-of-products (SOPs) of Boolean functions D_1, D_2, y_1, y_2 (the term F_{12}) and D_2, D_3, y_5 (the term F_{13}). All other parts of SOPs for (2) and (3) are constructed using the similar approach [27].

In this paper, we consider a case when SBFs (2) and (3) are implemented using such resources of FPGA chips as configurable logic blocks (CLBs) including LUTs, flip-flops and dedicated multiplexors [28], the programmable routing matrix, programmable input–output blocks and the synchronization tree [25,29]. Using the notation [30], we denote a LUT having I_L inputs and a single output as I_L-LUT. An I_L-LUT can implement a circuit of an arbitrary Boolean function having up to I_L arguments.

If the number of arguments exceeds the value of I_L, then it is necessary to apply various methods of functional decomposition (FD) of this Boolean function [31–34]. In this case, a resulting circuit is multi-level. As a rule, it has a complicated system of "spaghetti-type" interconnections [21].

If all LUTs have the same number of inputs, then such a logic basis is rigid. It means that in some cases, only a part of the available inputs will be used. However, in other cases, the LUTs should be combined to increase the number of inputs. To reduce the impact of interconnects on such a join, it is important to have internal fast interconnects between some LUTs. In Xilinx solutions, these CLBs are combined into slices [29,35]. For example, the SLICEL of Virtex-7 includes four 6-LUTs, eight flip-flops and 27 multiplexers [28].

In LUT-based FSMs, the SCR is hidden and distributed among LUTs implementing SOPs of functions (2). Due to it, there are only two blocks in LUT-based *P* Mealy FSM Figure 3.

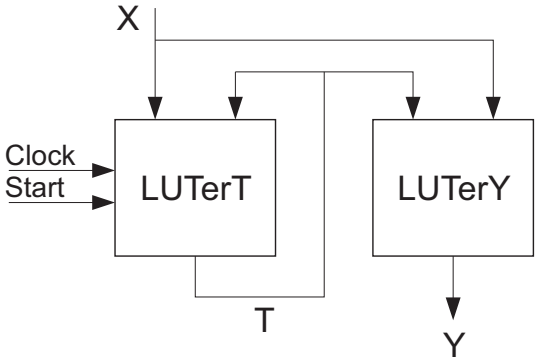


Figure 3. Structural diagram of LUT-based P Mealy FSM.

In this paper, a CLB-based block is denoted by a symbol LUTer. In *P* Mealy FSM, the LUTerT consists of CLBs generating IMFs $D_r \in \Phi$. The state variables $T_r \in T$ are kept into the distributed SCR. Due to this, the pulses *Clock* and *Start* enter the LUTerT. The outputs $y_n \in Y$ are generated by the LUTerY.

3. Related Work

If each function $\phi_k \in \Phi \cup Y$ depends on not more than I_L Boolean arguments, then there are exactly $N + R$ LUTs in the circuit of *P* Mealy FSM. This is the best possible outcome of synthesis. However, the modern LUTs have around 6 inputs [35–37]. In a CLB of Virtex-7 [36], it is possible to form either two 7-LUTs or a single 8-LUT using dedicated multiplexors. However, the total number of inputs and state variables of an FSM can significantly exceed 8 [17]. This leads to an imbalance between the characteristics of LUTs

and SBFs (2) and (3). This imbalance is a source of the necessity of improving FPGA-based design methods.

To improve area-time characteristics of CLB-based FSM circuits, it is necessary to optimize their systems of inter-slice interconnections. It is known that only 30% of power dissipation is connected with LUTs [38]. It means that around 70% of the power is dissipated on the interconnections. As shown in [38], interconnection delays are starting to play a major role in comparison with logic delays. As shown in [23], the optimization of interconnections allows the reduction of both the time of cycle and power consumption of LUT-based FSM circuits. Using either two-fold state assignment [39,40] or the extended state codes can help in the optimization of interconnections.

Each function $\phi_k \in \Phi \cup Y$ depends on $NA(\phi_k)$ arguments. If the condition

$$NA(\phi_k) \leq I_L \tag{4}$$

is violated, then there are several levels of LUTs in an FSM circuit. Various methods have been developed for improving characteristics of FSM circuits [21,25,26,30,34,41–44].

As a rule, the known optimization methods can improve either the number of LUTs or the cycle time or the power consumption [42]. Moreover, there are methods that try to optimize two or even three of these parameters. In our current research, there is proposed a method for reducing the number of LUTs of three-block circuits of Mealy FSMs [23].

The SOPs of functions $\phi_k \in \Phi \cup Y$ depend on product terms

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

These terms correspond to rows of DST. In (5), the symbol A_m stands for a conjunction of state variables corresponding to the code $K(a_m)$ of a current state written in the h-th row of DST. These conjunctions add R literals in the SOPs of functions $\phi_k \in \Phi \cup Y$.

To diminish the number of literals, various methods of state assignment are used [45–50]. These methods can be found in many academic and industrial CAD tools. The well-known academic systems are, for example, SIS [51] and ABC by Berkeley [52,53] or Sinthagate [54]. The manufactures of FPGA chips have their own CAD packages. For example, AMD (Xilinx) has the CADs Vivado [55] and Vitis [56], whereas Intel (Altera) has the package Quartus [57].

There is no a universal state assignment approach which allows achieving an optimal solution for any FSM. In [34], there are compared FSM circuits based on maximum binary codes with $R = \lceil log_2 M \rceil$ and one-hot state codes with $R = M$. As follows from the comparison, for FSMs with $M > 16$ the using one-hot codes allows FSM characteristics to be improved. However, the circuit characteristics depend strongly on the number of FSM inputs. This is due to the limited number of LUT inputs [21]. For example, the experiments [58] definitely show the following: if there is $L > 10$, then using maximum binary codes leads to FSM circuits with better characteristics than the circuits based on one-hot codes.

So, in one case, the circuits with better characteristics could be produced due to using the one-hot state codes. However, in the other case it is better to use the maximum binary codes. Therefore, it is necessary to apply several state assignment methods and to choose a method producing the best results (for a particular FSM). Taking this fact into account, we have compared the results based on our proposed approach with characteristics of FSM circuits produced using the methods JEDI [51], binary state assignment Auto and One-hot state assignment of Vivado [55] by Xilinx [35]. We chose JEDI because it is considered one of the best state-assignment approaches [51].

If condition (4) is violated, then to implement a LUT-based FSM circuit, various methods of functional decomposition should be applied [31,42,43]. To implement a circuit, an original function $\phi_k \in \Phi \cup Y$ is broken down by sub-functions for which the number of arguments does not exceed I_L. Each sub-function differs from the initial function $\phi_k \in \Phi \cup Y$ [42]. The decomposition should be executed in a way increasing the number of LUT levels of the final FSM circuit as little as possible [31]. The methods of FD are used by

both academic and industrial CAD tools dealing with FPGA-based design. Unfortunately, this approach has a serious drawback: FD-based FSM circuits have complicated systems of "spaghetti-type" interconnections [21]. This drawback is manifested in the increasing for both cycle time and power consumption of a resulting FSM circuit [59].

The methods of SD [21] can be viewed as an alternative to methods of FD. The main goal of SD-based methods is the elimination of direct connection between the variables $x_l \in X$ and $T_r \in T$, on the one hand, and functions $y_n \in Y$ and $D_r \in \Phi$, on the other hand. To achieve this goal, the block of functions (Figure 1) is represented as a composition of several logic blocks. As a rule, there are from two to four logic blocks [21]. This approach leads to the increasing the number of implemented functions. However, these new functions depend on significantly fewer arguments than functions $\phi_k \in \Phi \cup Y$.

The first known methods of SD were proposed in the mid-20th century by Prof. M. Wilkes [60]. These methods are the replacement of inputs and encoding of COs. In [23], we propose the joint use of these methods for optimization of LUT-based Mealy FSMs' circuits. The main ideas of these methods are shown below.

The first method is reduced to the replacement of the set $X = \{x_1, \ldots, x_L\}$ by a set of additional variables $B = \{b_1, \ldots, b_J\}$. This makes sense if the following condition holds: $J \ll L$. The replacement is based on the creating a system of additional functions

$$B = B(T, X). \tag{6}$$

In the case of LUT-based FSMs, these functions can be implemented with such resources of CLBs as LUTs and dedicated multiplexors [28].

The second method assumes the representing Q different COs $Y_q \subseteq Y$ by binary codes $K(Y_q)$. To do it, elements of an additional set $Z = \{z_1, \ldots, z_{RQ}\}$ are used. The minimum number of bits in the codes $K(Y_q)$ can be found as

$$R_Q = \lceil log_2 Q \rceil. \tag{7}$$

The following SBFs should be obtained to encode COs:

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

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

The SBFs (8) and (9) are implemented using LUTs. To implement the system (9), it is necessary to organize LUTs as decoders.

As shown in [23], combining these two methods is connected with introducing the following additional SBFs:

$$\Phi = \Phi(T, B); \tag{10}$$

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

The SBFs (6) and (9)–(11) determine a structural diagram of LUT-based *MPY* Mealy FSM (Figure 4).

In MPY Mealy FSM, *LUTerIR* executes the replacement of FSM inputs. Therefore, it implements SBF (6). The additional variables $b_j \in B$ enter *LUTerZT* which implements SBFs (9) and (10). The IMFs $D_r \in \Phi$ enter the state code register SCR hidden inside of *LUTerZT*. At last, *LUTerY* transforms the additional variables $z_r \in Z$ into the functions $y_n \in Y$.

We discuss a case when the logic blocks of MPY FSMs are implemented using internal resources of CLBs, inter-slice interconnections, programmable chip input–outputs and synchronization tree buffers [28]. The basic characteristics of equivalent *P* and *MPY* FSMs are compared in [23]. The research results obtained in [23] show that the joint use of discussed methods of SD leads to improving the characteristics of LUT-based Mealy FSM circuits.

In this paper, we propose to transform the CO codes into both the output functions $y_n \in Y$ and state variables $T_r \in T$. Moreover, we propose a new type of state code which allows the optimization of a circuit generating functions $z_r \in Z$.

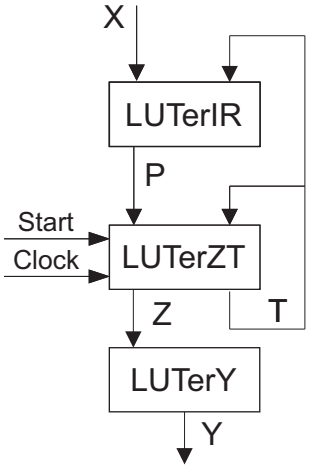


Figure 4. Structural diagram of LUT-based *MPY* Mealy FSM.

4. Main Idea of the Proposed Method

Our main idea is illustrated by Figure 5.

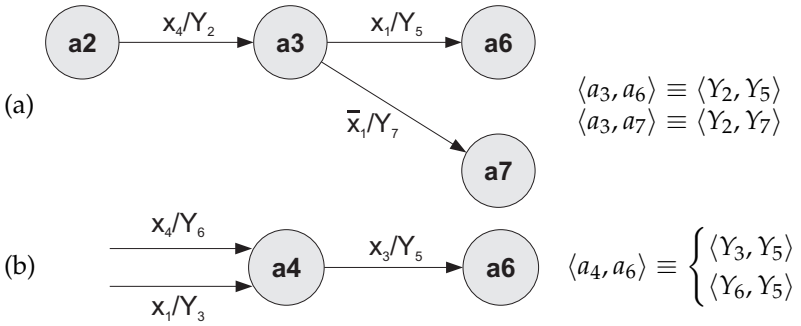


Figure 5. Replacement of transition pairs $\langle a_m, a_s \rangle$ by pairs $\langle Y_m, Y_s \rangle$.

The transition $\langle a_2, a_3 \rangle$ (Figure 5a) is caused by the input x_4. This transition is accompanied by the producing a CO Y_2. For the next instant of FSM time, this CO (we denote it as Y_m) indicates the relation $a_m = a_3$. If there is $X_h = x_1$, then there is $a_s = a_6$ and $Y_s = Y_5$. So, the transition $\langle a_3, a_6 \rangle$ can be indicated by the pair $\langle Y_2, Y_5 \rangle$. Using similar reasoning, it is possible to show that the transition $\langle a_3, a_7 \rangle$ can be indicated by the pair $\langle Y_2, Y_7 \rangle$. To show how many COs are generated during transitions to a state $a_m \in A$, we use the symbol Q_m. There is $Q_m = 1$ for the case represented by Figure 5a. The case with $Q_m > 1$ is illustrated by Figure 5b. Two COs (Y_3 and Y_6) are generated during transitions into the state a_4. So, there is $Q_4 = 2$. Now, the same transition $\langle a_4, a_6 \rangle$ is represented by two pairs, namely, $\langle Y_3, Y_5 \rangle$ and $\langle Y_6, Y_5 \rangle$.

This analysis shows that transitions $\langle a_m, a_s \rangle$ can be represented by pairs $\langle Y_m, Y_s \rangle$. Using this result of analysis, we propose a *PZ* Mealy FSM, the structural diagram of which is shown in Figure 6.

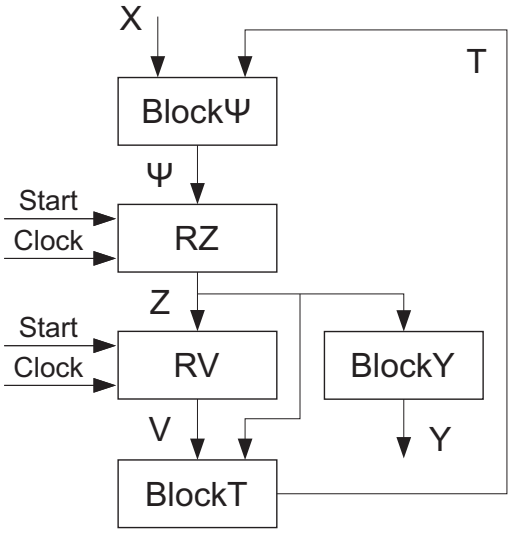


Figure 6. Structural diagram of *PZ* Mealy FSM.

There are two registers in *PZ* Mealy FSM. The register *RZ* keeps a code of CO $Y_s \subseteq Y$ represented by variables $z_r \in Z = \{z_1, \ldots, z_{RQ}\}$. The register *RV* keeps a code of CO $Y_m \subseteq Y$ represented by variables $v_r \in V = \{v_1, \ldots, v_{RQ}\}$. Obviously, these registers have R_Q *D* flip-flops each, where the value of R_Q is determined by (7). The registers are controlled by the same pulses *Clock* and *Start*. So, they can be viewed as R_Q single-bit shift registers. A *Block*Ψ generates additional variables $D_r \in \Psi = \{D_1, \ldots, D_{RQ}\}$ used to load the code $K(Y_s)$ into *RZ*. The system Ψ is represented as

$$\Psi = \Psi(T, X). \tag{12}$$

In each cycle, current codes of COs Y_m and Y_s are kept in the registers. A *BlockZ* generates FSM outputs represented by SBF (9). The contents of these registers are converted into a transition state code by a *BlockT*. To do it, the SBF

$$T = T(Z, V) \tag{13}$$

is implemented by the *BlockT*.

Such an approach allows the exclusion of FSM input variables $x_l \in X$ from both FSM output functions and IMFs. Moreover, the outputs $y_n \in Y$ are registered. So, they do not depend on possible fluctuations of inputs [21] during any cycle of FSM operation. As a rule, this stability is achieved by using additional register having *N* flip-flops controlled by an additional synchronization pulse.

We discuss a case when an FSM circuit is implemented using slices similar to ones present in Virtex-7 of Xilinx [35,36]. In this case, the number of flip-flops is twice the number of LUTs per a slice. Each pair of flip-flops can be connected to form a shift register discussed before. So, in the same SLICEL, there are resources to produce both functions (12), as well as the additional variables $z_r \in Z$ and $v_r \in V$.

If the condition (4) is violated for functions $z_r \in Z$, then there is a multi-level circuit of *Block*Ψ. To implement it, the methods of FD should be applied. To avoid the applying of SD, we propose a model of $P_C Z$ Mealy FSM. The method is based on using class-state codes proposed in this paper.

If the condition (4) is violated for functions $z_r \in Z$, then we propose to create a partition $\Pi_A = \{A^1, \ldots, A^K\}$ of the set *A*. Each class $A^k \in \Pi_A$ determines two sets. A set $X^k \subseteq X$ includes L_k FSM inputs causing transitions from states $a_m \in A^k$. A set $Z^k \subseteq Z$ consists of additional variables $z_r \in Z$ generated during these transitions. There are M_k elements in the class $A^k \in \Pi_A$.

Using ideas from the articles [39,40], we propose to encode states $a_m \in A^k$ by codes $SC(a_m)$ having R_s bits. The following formula determines the value of R_s:

$$R_s = max(\lceil log_2 M_1 \rceil, \ldots, \lceil log_2 M_K \rceil). \quad (14)$$

The partition Π_A should be created in a way that the following condition holds for each class $A^k \in \Pi_A$:

$$R_s + L_k \leq I_L. \quad (15)$$

To create a CSC, it is necessary to encode classes $A^k \in \Pi_A$ by class codes $CC(A^k)$ having R_C bits:

$$R_C = \lceil log_2 K \rceil. \quad (16)$$

Now, a state $a_m \in A^k$ is represented by its class-state code

$$CSC(a_m) = CC(A^k) * SC(a_m). \quad (17)$$

In (17), the symbol "*" stands for the concatenation of codes.

To encode the classes, we use class variables $T_r \in T_B$ where $R_C = |T_B|$. To encode the states as class elements, we use state variables $T_r \in T_A$ where $R_S = |T_A|$. These sets create a set $T = T_B \cup T_A$ having $R_T = R_C + R_S$ elements. The first R_C elements of T create codes of classes; the next R_S variables create state codes $SC(a_m)$.

Using this encoding style, we propose a structural diagram of LUT-based $P_C Z$ Mealy FSM (Figure 7).

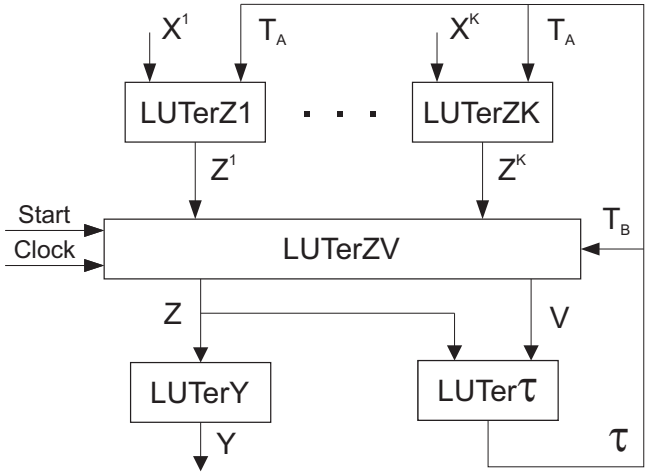


Figure 7. Structural diagram of LUT-based $P_C Z$ Mealy FSM.

In $P_C Z$ Mealy FSM, a block *LUTerk* corresponds to the class $A^k \in \Pi_A$. It implements an SBF

$$Z^k = Z^k(T_A, X^k) \quad (k \in \{1, \ldots, K\}). \quad (18)$$

A block *LUTerZV* includes CLBs and hidden distributed registers *RZ* and *RV*. It implements SBF

$$Z = (T_B, Z^1, \ldots, Z^K). \quad (19)$$

The variables $v_r \in V$ repeat the values of variables $z_r \in Z$ produced in the previous FSM operation cycle. A block *LUTerY* implements SBF (9). At last, a block *LUTerT* generates CSCs. To do it, the block implements SBF

$$T = T(Z, V). \quad (20)$$

In this paper, we propose a synthesis method for $P_C Z$-based Mealy FSMs. The synthesis process starts from an STG. The proposed method includes the following steps:

1. Constructing an STT corresponding to an initial STG.
2. Encoding of FSM states by maximum binary codes $K(a_m)$.
3. Encoding of collections of outputs $Y_q \subseteq Y$ by binary codes $K(Y_q)$.
4. Creating the SBF $Y = Y(Z)$.
5. Creating the modified direct structure table of PZ Mealy FSM.
6. Creating a table of pairs $P_g = \langle Y_i, Y_j \rangle$ corresponding to pairs $\langle a_m, X_h \rangle$.
7. Creating the partition Π_A with minimum amount of classes, K.
8. Encoding of classes and states to obtain class-state codes.
9. Creating tables representing blocks LUTer1-LUTerK and SBFs (18).
10. Creating table of $LUTerZV$ and SBF (19).
11. Creating table of $LUTer\mathcal{T}$ and SBF (20).
12. Implementing the CLB-based circuit of $P_C Z$ Mealy FSM.

5. Example of Synthesis

We use the symbol $P_C Z(S_a)$ to show that the model of $P_C Z$ Mealy FSM is used to obtain a logic circuit of an FSM S_a. This Section is devoted to the synthesis of Mealy FSM $P_C Z(S_1)$. To implement the circuit, 5-LUTs are used. We start the synthesis process from an STG (Figure 8).

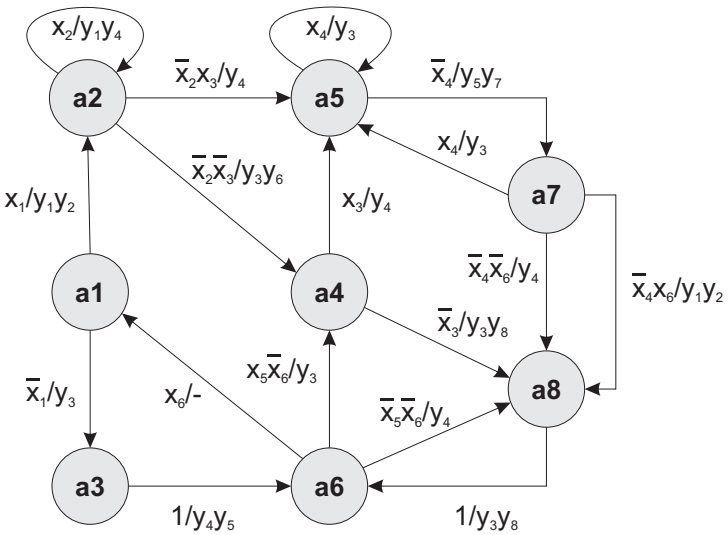


Figure 8. State transition graph of Mealy FSM S_1.

The following sets can be found from the STG (Figure 8): $A = \{a_1, \ldots, a_8\}$, $X = \{x_1, \ldots, x_6\}$ and $Y = \{y_1, \ldots, y_8\}$. So, the following characteristics characterize the FSM S_1: $M = 8$, $L = 6$, and $N = 8$. There are $H = 17$ arcs connecting the nodes of the STG (Figure 8). So, there are 17 rows in the STT (and DST) of FSM S_1.

Step 1. The transformation of an STG into an equivalent STT is executed in the trivial way [27]. As follows from Figure 3, the h-th arc of STG determines the h-th row of the corresponding STT ($h = \{1, \ldots, H\}$). The STT of Mealy FSM S_1 is represented by Table 1.

Step 2. For FSM S_1, there is $M = 8$. Using (1) gives $R = 3$. This determines the set of state variables $T = \{T_1, T_2, T_3\}$. To simplify the presentation of our method, the states are encoded in the trivial way: $K(a_1) = 000$, $K(a_2) = 001$,..., $K(a_8) = 111$.

Step 3. The analysis of Table 1 allows finding $Q = 9$ different collections $Y_q \subseteq Y$. These COs are the following: $Y_1 = \emptyset$, $Y_2 = \{y_1, y_2\}$, $Y_3 = \{y_3\}$, $Y_4 = \{y_1, y_4\}$, $Y_5 = \{y_3, y_6\}$, $Y_6 = \{y_4\}$, $Y_7 = \{y_5, y_7\}$, $Y_8 = \{y_3, y_8\}$ and $Y_9 = \{y_4, y_5\}$. Using (7) gives $R_Q = 4$ and the set $Z = \{z_1, \ldots, z_4\}$.

Table 1. State transition table of Mealy FSM S_1.

a_m	a_S	X_h	Y_h	h
a_1	a_2	x_1	y_1y_2	1
	a_3	$\bar{x_1}$	y_3	2
a_2	a_2	x_2	y_1y_4	3
	a_5	$\bar{x_2}x_3$	y_4	4
	a_4	$\bar{x_2}\bar{x_3}$	y_3y_6	5
a_3	a_6	1	y_4y_5	6
a_4	a_5	x_3	y_4	7
	a_8	$\bar{x_3}$	y_3y_8	8
a_5	a_5	x_4	y_3	9
	a_7	$\bar{x_4}$	y_5y_7	10
a_6	a_1	x_6	–	11
	a_4	$\bar{x_6}x_5$	y_3	12
	a_8	$\bar{x_6}\bar{x_5}$	y_4	13
a_7	a_5	x_4	y_3	14
	a_8	$\bar{x_4}x_6$	y_1y_2	15
	a_8	$\bar{x_4}\bar{x_6}$	y_4	16
a_8	a_6	1	y_3y_8	17

As shown in [21], COs should be encoded in a way that minimizes the number of literals in SBF (8). If the condition

$$R_Q > I_L \tag{21}$$

holds, then such an approach could minimize the LUT count for *LUTerY* [21]. If (21) is violated, this method of encoding reduces the number of interconnections [21]. This reduces chip areas occupied by LUT-based FSM circuits [23].

To encode COs, we use the approach proposed in [61]. The outcome of encoding is shown in Figure 9.

z_3z_4 \ z_1z_2	00	01	11	10
00	Y_1	Y_2	*	Y_7
01	Y_3	*	*	Y_5
11	*	*	*	Y_8
10	Y_6	Y_4	*	Y_9

Figure 9. The outcome of encoding of COs for FSM S_1.

Step 4. Using the codes of COs Figure 9 gives the following SBF:

$$\begin{array}{ll} y_1 = Y_2 \vee Y_4; & y_2 = Y_2 = z_2\bar{z_3}; \\ y_3 = Y_3 \vee Y_5 \vee Y_8 = z_4; & y_4 = Y_4 \vee Y_6 \vee Y_5 = z_3\bar{z_4}; \\ y_5 = Y_7 \vee Y_9 = z_1\bar{z_4}; & y_6 = Y_5 \vee Y_8 = z_1z_4; \\ y_7 = Y_7 = z_1z_3z_4; & y_8 = Y_8 = z_3z_4. \end{array} \tag{22}$$

The analysis of (22) shows that there are 15 literals in this system. So, there are 15 interconnections between the blocks *LUTerZV* and *LUTerY*. Obviously, the maximum

number of these interconnections is equal to NR_Q [21]. In the discussed case, there is $NR_Q = 32$. So, the number of interconnections is reduced by 2.13 times due to applying the approach [61].

If condition (21) is violated, then there are N LUTs in the circuit of $LUTerY$. The analysis of (22) shows that SOPs of functions y_1 and y_3 have a single literal. So, these functions are produced by LUTs of $LUTerZV$. So, there are $N - 2 = 6$ LUTs in the circuit of $LUTerY$ of FSM $P_C Z(S_1)$. Thus, the number of LUTs is reduced by 1.33 times due to applying the approach [61]. This is an upside effect of the method [61].

Step 5. The columns of a classical DST [27] are shown in Figure 3c. We have modified the traditional DST. The column Y_h is replaced by a column Z_h (Table 2). This table determines the Mealy FSM $PZ(S_1)$.

Table 2. Modified DST of Mealy FSM $PZ(S_1)$.

a_m	$K(a_m)$	a_S	$K(a_S)$	X_h	Φ_h	Z_h	h
a_1	000	a_2	001	x_1	D_3	z_2	1
		a_3	010	$\bar{x_1}$	D_2	z_4	2
a_2	001	a_2	001	x_2	D_3	z_2z_3	3
		a_5	100	$\bar{x_2}x_3$	D_1	z_3	4
		a_4	011	$\bar{x_2}\bar{x_3}$	D_2D_3	z_1z_4	5
a_3	010	a_6	101	1	D_1D_3	z_1z_3	6
a_4	011	a_5	100	x_3	D_1	z_3	7
		a_8	111	$\bar{x_3}$	$D_1D_2D_3$	$z_1z_3z_4$	8
a_5	100	a_5	100	x_4	D_1	z_4	9
		a_7	110	$\bar{x_4}$	D_1D_2	z_1	10
a_6	101	a_1	000	x_6	–	–	11
		a_4	011	$\bar{x_6}x_5$	D_2D_3	z_4	12
		a_8	111	$\bar{x_6}\bar{x_5}$	$D_1D_2D_3$	z_3	13
a_7	110	a_5	100	x_4	D_1	z_4	14
		a_8	111	$\bar{x_4}x_6$	$D_1D_2D_3$	z_2	15
		a_8	111	$\bar{x_4}\bar{x_6}$	$D_1D_2D_3$	z_3	16
a_8	111	a_6	101	1	D_1D_3	$z_1z_3z_4$	17

The column Z_h contains a variable $z_r \in Z$ if the r-th bit of $K(Y_q)$ is equal to 1 (we assume that the CO $Y_q \subseteq Y$ is written in the h-th row of STT). For example, there is the CO Y_3 in the second row of Table 1. As follows from Figure 9, there is $K(Y_3) = 0001$. Due to it, there is the symbol z_4 in the second row of Table 2. All other rows for column Z_h are filled in the same manner.

Step 6. A table of pairs $P_g = \langle Y_i, Y_j \rangle$ shows a correspondence between these pairs and the pairs $\langle a_m, X \rangle$. It includes the following columns: a_m (a current FSM state); a_s (a state of transition); Y_m and Y_s (COs produced during the transition into the state a_m and a_s, respectively); P_g (a pair $\langle Y_m, Y_s \rangle$); g (the number of a pair $P_g (g \in \{1, \ldots, G\})$). In the discussed case, there is $G = 29$. These pairs are represented by Table 3.

Table 3. Table of pairs of COs

a_m	a_S	Y_m	Y_S	g	a_m	a_S	Y_m	Y_S	g
a_1	a_2	Y_1	Y_2	1	a_5	a_5	Y_3	Y_3	15
a_1	a_3	Y_1	Y_3	2	a_5	a_7	Y_6	Y_7	16
a_2	a_2	Y_2	Y_4	3	a_5	a_7	Y_3	Y_7	17
a_2	a_2	Y_4	Y_4	4	a_6	a_4	Y_9	Y_3	18
a_2	a_5	Y_2	Y_6	5	a_6	a_4	Y_8	Y_3	19
a_2	a_5	Y_4	Y_6	6	a_6	a_8	Y_9	Y_7	20
a_2	a_4	Y_2	Y_5	7	a_6	a_8	Y_8	Y_7	21
a_2	a_4	Y_4	Y_5	8	a_6	a_1	Y_9	Y_1	22
a_3	a_6	Y_3	Y_9	9	a_6	a_1	Y_8	Y_1	23
a_4	a_5	Y_5	Y_6	10	a_7	a_5	Y_7	Y_3	24
a_4	a_5	Y_5	Y_8	11	a_7	a_8	Y_7	Y_2	25
a_4	a_8	Y_3	Y_6	12	a_7	a_8	Y_7	Y_6	26
a_4	a_8	Y_3	Y_8	13	a_8	a_6	Y_6	Y_8	27
a_5	a_5	Y_6	Y_3	14	a_8	a_6	Y_8	Y_8	28
					a_8	a_6	Y_2	Y_8	29

Step 7. In the discussed example, using the methods [39,40], the partition $\Pi_A = \{A^1, A^2\}$ can be found. There is the following distribution of states $a_m \in A$ between the classes: $A^1 = \{a_1, a_2, a_4, a_8\}$ and $A^2 = \{a_3, a_5, a_6, a_7\}$. The partition determines the following sets: $X^1 = \{x_1, x_2, x_3\}$, $X^2 = \{x_4, x_5, x_6\}$, and $Z^1 = Z^2 = Z$.

So, there is K = 2, $M_1 = M_2 = L_1 = L_2 = 3$. Using (4) gives the number of state variables $R_S = 2$. To implement the circuit of $P_C Z(S_1)$, the LUTs having $I_L = 5$ inputs are used. Because the relation (15) holds for each class $A^k \in \Pi_A$, this partition satisfies the previously discussed requirements.

Step 8. In the discussed example, there are $K = 2$ classes $A^k \in \Pi_A$. Using (16) gives $R_C = 1$ and $T_B = \{T_1\}$. Because there is $R_S = 2$, the state variables form the set $T_A = \{T_2, T_3\}$. So, there is the set $T = \{T_1, T_2, T_3\}$. The class–state codes are shown in (Figure 10).

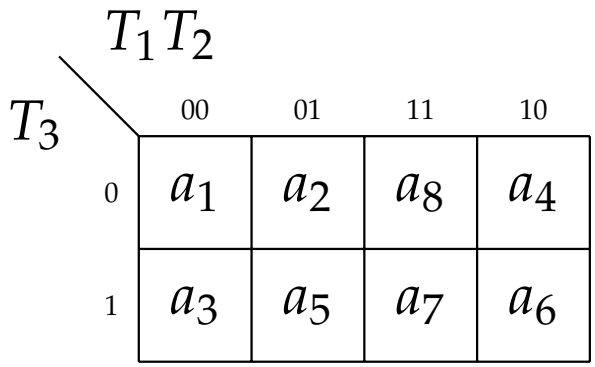


Figure 10. Outcome of encoding of states and state classes.

For example, the following codes can be found from Figure 10: $SC(a_2) = 01$, $CC(A^1) = 0$, $CSC(a_2) = 001$, $SC(a_5) = 01$, $CC(A^2) = 1$, $CSC(a_5) = 101$ and so on. These codes are used for creating SBFs (18)–(20).

Step 9. Tables of $LUTer1$–$LUTer2$ are created using the modified DST (Table 2) and state codes from Figure 10. Each table includes the columns a_m,$SC(a_m)$,X_h^1,Z_h^1,h. The $LUTerZ1$ is represented by Table 4, the $LUTerZ2$ by Table 5.

Table 4. Table of *LUTerZ*1.

a_m	$SC(a_m)$	X_h^1	Z_h^1	h
a_1	00	x_1	z_2	1
		$\bar{x_1}$	z_4	2
a_2	01	x_2	z_2z_3	3
		$\bar{x_2}x_3$	z_3	4
		$\bar{x_2}\bar{x_3}$	z_1z_4	5
a_4	10	x_3	z_3	6
		$\bar{x_3}$	$z_1z_3z_4$	7
a_8	11	1	$z_1z_3z_4$	8

Table 5. Table of *LUTerZ*2.

a_m	$SC(a_m)$	X_h^2	Z_h^2	h
a_3	00	1	z_1z_3	1
a_5	01	x_4	z_4	2
		$\bar{x_4}$	z_1	3
a_6	10	x_6	–	4
		$\bar{x_6}x_5$	z_4	5
		$\bar{x_6}\bar{x_5}$	z_3	6
a_7	11	x_4	z_4	7
		$\bar{x_4}x_6$	z_2	8
		$\bar{x_4}\bar{x_6}$	z_3	9

These tables are used for deriving SBFs (18). For example, the following equations can be derived for functions z_1^1 (from Table 4) and z_1^2 (Table 4):

$$\begin{array}{l} z_1^1 = \bar{T_2}T_3\bar{x_2}\bar{x_3} \vee T_2\bar{T_3}\bar{x_3} \vee T_2T_3; \\ z_1^2 = \bar{T_2}\bar{T_3} \vee \bar{T_2}T_3\bar{x_4}. \end{array} \quad (23)$$

Step 10. Table of *LUTerZV* includes the columns z_r (a function generated by *LUTerZV*); *LUTr*; *r* (the subscript of the corresponding function). If a partial function z_r^k appears in table of *LUTerk*, then there is 1 at the intersection of the row z_r and column k. In the discussed case, the *LUTerZV* is represented by Table 6.

Table 6. Table of *LUTerZV*.

z_r	*LUTr*		*r*
z_2	1	1	1
z_2	1	1	2
z_3	1	1	3
z_4	1	1	4

The following SBF is derived from Table 6:

$$\begin{array}{ll} z_1 = \bar{T_1}z_1^1 \vee T_1z_1^2; & z_2 = \bar{T_1}z_2^1 \vee T_1z_2^2; \\ z_3 = \bar{T_1}z_3^1 \vee T_1z_3^2; & z_4 = \bar{T_1}z_4^1 \vee T_1z_4^2; \end{array} \quad (24)$$

Step 11. The table of *LUTerT* is constructed using table of pairs of COs Table 3 and codes of COs (Figure 9). This table includes the columns Y_m, $K(Y_m)$,Y_S, $K(Y_S)$, a_s, $CSC(a_s)$,

$T(a_s)$, g. The g-th row of this table corresponds to the g-th row of table of pairs. The column $T(a_s)$ include IMFs equal to 1 to create the code $CSC(a_s)$. In the discussed case, $LUTerT$ is represented by Table 7.

Table 7. Table of $LUTerT$.

Y_m	$K(Y_m)$	Y_S	$K(Y_S)$	a_S	$CSC(a_S)$	$T(a_S)$	g
Y_1	0000	Y_2	0100	a_2	001	T_3	1
Y_1	0000	Y_3	0001	a_3	100	T_1	2
Y_2	0100	Y_4	0110	a_2	001	T_3	3
Y_4	0110	Y_4	0110	a_2	001	T_3	4
Y_2	0100	Y_6	0010	a_5	101	T_1T_3	5
Y_4	0110	Y_6	0010	a_5	101	T_1T_3	6
Y_2	0100	Y_5	1001	a_4	010	T_2	7
Y_4	0110	Y_5	1001	a_4	010	T_2	8
Y_3	0001	Y_9	1010	a_6	110	T_1T_2	9
Y_5	1001	Y_6	0010	a_5	101	T_1T_3	10
Y_5	1001	Y_8	1011	a_5	101	T_1T_3	11
Y_3	0001	Y_6	0010	a_8	011	T_2T_3	12
Y_3	0001	Y_8	1011	a_8	011	T_2T_3	13
Y_6	0010	Y_3	0001	a_5	101	T_1T_3	14
Y_3	0001	Y_3	0001	a_5	101	T_1T_3	15
Y_6	0010	Y_7	1000	a_7	111	$T_1T_2T_3$	16
Y_3	0001	Y_7	1000	a_7	111	$T_1T_2T_3$	17
Y_9	1010	Y_3	0001	a_4	010	T_2	18
Y_8	1011	Y_3	0001	a_4	010	T_2	19
Y_9	1010	Y_7	1000	a_8	011	T_2T_3	20
Y_8	1011	Y_7	1000	a_8	011	T_2T_3	21
Y_9	1010	Y_1	0000	a_1	000	–	22
Y_8	1011	Y_1	0000	a_1	000	–	23
Y_7	1000	Y_3	0001	a_5	101	T_1T_3	24
Y_7	1000	Y_2	0100	a_8	011	T_2T_3	25
Y_7	1000	Y_6	0010	a_8	011	T_2T_3	26
Y_6	0010	Y_8	1011	a_6	110	T_1T_2	27
Y_8	1011	Y_8	1011	a_6	110	T_1T2	28
Y_2	0100	Y_8	1011	a_6	110	T_1T_2	29

This table is a base for creating SBF (20). For example, the following SOP can be derived from Table 7:

$$\begin{aligned} T_1 &= E_2 \vee E_5 \vee E_6 \vee E_9 \vee E_{10} \vee E_{11} \vee E_{14} \vee E_{15} \vee E_{16} \vee E_{17} \vee E_{24} \vee e_{27} \vee E_{28} \vee E_{29} \\ &= \bar{v}_1\bar{v}_2\bar{v}_3\bar{v}_4\bar{z}_1\bar{z}_2\bar{z}_3\bar{z}_4 \vee \cdots \vee \bar{v}_1v_2\bar{v}_3\bar{4}z_1\bar{z}_2z_3z_4. \end{aligned} \tag{25}$$

Step 12. Using the obtained SBFs, we can implement the logic circuit of Mealy FSM $P_CZ(S_1)$. This circuit includes 24 LUTs having 5 inputs. The circuit is shown in Figure 11.

The first logic level of the circuit includes $2R_Q = 8$ LUTs. As follows from Table 4, there are 4 LUTs in the circuit of $LUTerZ1$ (LUT1–LUT4). As follows from Table 5, there are 4 LUTs in the circuit of $LUTerZ2$ (LUT5–LUT8).

The second level includes $R_Q = 4$ LUTs. It follows from either Table 6 or SBF (24).

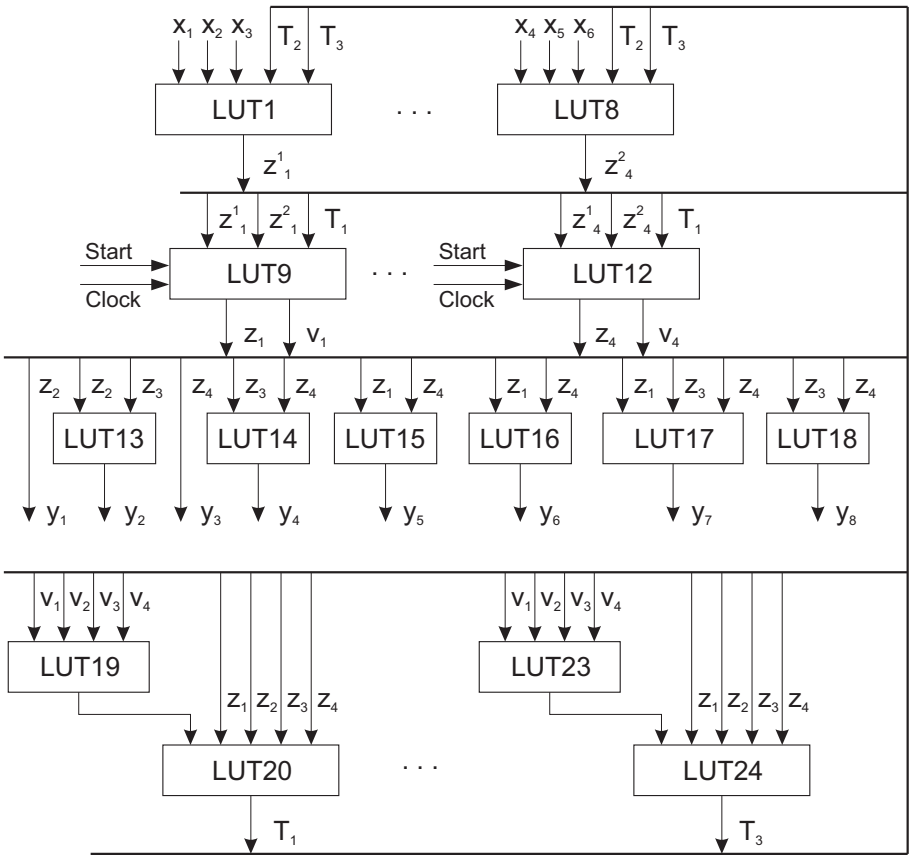


Figure 11. Logic circuit of Mealy FSM $P_C Z(S_1)$.

The third logic level includes two logic blocks (*LUTerY* and *LUTerT*) operating in parallel. As follows from SBF (22), there are 6 LUTs in the circuit of *LUTerY*. This circuit includes LUT13–LUT18.

For the discussed case, the condition

$$2R_Q > I_L \tag{26}$$

holds. Due to it, there are 2 LUTs in the circuit implementing any equation for $T_r \in T$. For example, the circuit for $T_1 \in T$ is a serial connection of LUT19 and LUT20. There are $2(R_C + R_S) = 6$ LUTs in the circuit of *LUTerT*. To improve the time characteristics of *LUTerT*. The LUT pairs (LUT19–LUT20, LUT21–LUT22, and LUT23–LUT24) can be connected using the dedicated multiplexer [28].

To obtain the LUT-based FSM circuits, the step of technology mapping [42] should be executed. To execute the technology mapping, some industrial CAD tools are used. If an FSM circuit is based on the internal resources of Virtex-7, the industrial package Vivado [55] should be used. The Vivado executes the steps of mapping, placement, routing, testing, and finding such characteristics of a circuit as the numbers of LUTs, slices, flip-flops, as well as maximum operating frequency and power consumption.

6. Experimental Results

In this Section, we show results of experiments conducted using the industrial CAD package Vivado and the library of standard benchmark (BM) FSMs [62]. In these experiments, we compared characteristics of $P_C Z$-based Mealy FSMs with characteristics of FSM circuits based on some other models. The library [62] includes 48 BMs represented by STTs in the format KISS2. These benchmarks have a wide range in such characteristics as the numbers of states, inputs, transitions and outputs. The results of research based on this library can be found in many articles, as well as the BM characteristics.

The research was conducted using a personal computer with the following characteristics: CPU—Intel Core i7 6700K 4.2@4.4 GHz; Memory—16 GB RAM 2400 MHz CL15. To implement CLB-based circuits, we used the Virtex-7 VC709 Evaluation Platform (xc7vx690tffg1761-2) [63]. The package Vivado v2019.1 (64-bit) of Xilinx [55] was used for the implementation of FSM circuits. The CLBs of this platform have 6- LUTs. We use the reports of Vivado for creating the tables with research results.

The created tables include such parameters of FSM circuits as the LUT counts and maximum operating frequencies. The following FSM models have been used in our experiments: (1) Auto of Vivado (the state codes of these FSMs have $R = \lceil log_2 M \rceil$ bits); (2) One-hot of Vivado (the state codes have $R = M$ bits); (3) JEDI; (4) MPY-based FSMs [23] and (5) $P_C Z$- based FSMs.

As in the research [23], we have divided the BMs by 5 sets denoted as $BM1$–$BM5$. Belonging to a particular set is determined by the relation between $L + R$ and I_L. In the discussed case, there is $I_L = 6$. The number of a set j is determined as

$$j = \left\lceil \frac{L + R}{I_L} \right\rceil. \tag{27}$$

The value of (27) determines a set $BMj (j \in \{1, \ldots, 5\})$. The distribution is shown in Table 8.

Table 8. Distribution of benchmarks between sets BM1–BM5.

BM1	BM2	BM3	BM4	BM5
bbtas	dk512	ex1	sand	s420
dk1	bbsse	kirkman		s510
dk27	beecount	planet		s820
dk512	cse	planet1		s832
ex3	dk14	pma		
ex5	dk15	s1		
lion	dk16	s1488		
lion9	donefile	s149		
mc	ex2	s1a		
modulo12	ex4	s208		
shiftreg	ex6	styr		
	ex7	tma		
	keyb			
	mark			
	opus			
	s2			
	s386			
	s840			
	sse			

The results of experiments are shown in Tables 9–16. The same organization is used in these tables. The table columns are marked by the names of FSM design methods. The names of benchmarks are written into the rows of these tables. Inside each table, the benchmarks are listed in alphabetical order, and sorted by ascending value of j. The rows "Total" contain results of summation of numbers for each column. The row "Percentage" contains the percentage of summarized characteristics of FSM circuits produced by other methods, respectively, to $P_C Z$-based FSMs. We use the model of Mealy P for all design methods except of MPY FSMs. The sets BMj are shown in the columns "Set".

These tables include the following information: (1) the numbers of LUTs for all BMs (Table 9); (2) the numbers of LUTs for BMs of the set BM1 (Table 10); (3) the numbers of LUTs for BMs of the set BM2 (Table 11); (4) the numbers of LUTs for BMs of sets BM3–BM5 (Table 12); (5) the maximum operating frequency for all BMs (Table 13); (6) the maximum operating frequency for BMs of the set BM1 (Table 14); (7) the maximum operating frequency

for BMs of the set BM2 (Table 15); (8) the maximum operating frequency for BMs of the sets BM3–BM5 (Table 16). The following conclusions can be made from the analysis of these tables.

Table 9. Experimental results (numbers of LUTs for BM1–BM5).

Benchmark	Auto	One-Hot	JEDI	MPY	Our Approach	Set
bbtas	5	5	5	8	8	BM1
dk17	5	12	5	8	8	BM1
dk27	3	5	4	7	7	BM1
dk512	10	10	9	12	12	BM1
ex3	9	9	9	11	11	BM1
ex5	9	9	9	10	10	BM1
lion	2	5	2	6	6	BM1
lion9	6	11	5	8	8	BM1
mc	4	7	4	6	6	BM1
modulo12	7	7	7	9	9	BM1
shiftreg	2	6	2	4	4	BM1
bbara	17	17	10	10	10	BM2
bbsse	33	37	24	26	25	BM2
beecount	19	19	14	14	14	BM2
cse	40	66	36	33	33	BM2
dk14	16	27	10	12	11	BM2
dk15	15	16	12	6	7	BM2
dk16	15	34	12	11	11	BM2
donfile	31	31	24	21	20	BM2
ex2	9	9	8	8	10	BM2
ex4	15	13	12	11	10	BM2
ex6	24	36	22	21	20	BM2
ex7	4	5	4	6	7	BM2
keyb	43	61	40	37	36	BM2
mark1	23	23	20	19	18	BM2
opus	28	28	22	21	21	BM2
s27	6	18	6	6	7	BM2
s386	26	39	22	25	24	BM2
s8	9	9	9	9	10	BM2
sse	33	37	30	26	24	BM2
ex1	70	74	53	40	34	BM3
kirkman	42	58	39	33	27	BM3
planet	131	131	88	78	68	BM3
planet1	131	131	88	78	68	BM3
pma	94	94	86	72	65	BM3
s1	65	99	61	54	48	BM3
s1488	124	131	108	89	83	BM3
s1494	126	132	110	90	78	BM3
s1a	49	81	43	38	32	BM3
s208	12	31	10	9	9	BM3
styr	93	120	81	70	59	BM3
tma	45	39	39	30	27	BM3
sand	132	132	114	99	79	BM4
s420	10	31	9	8	9	BM5
s510	48	48	32	22	19	BM5
s820	88	82	68	52	46	BM5
s832	80	79	62	50	44	BM5
Total	1808	2104	1489	1323	1202	
Percentage, %	150.42	175.04	123.88	110.07	100.00	

Table 10. Experimental results (numbers of LUTs for BMs from BM1).

Benchmark	Auto	One-Hot	JEDI	MPY	Our Approach
bbtas	5	5	5	8	8
dk17	5	12	5	8	8
dk27	3	5	4	7	7
dk512	10	10	9	12	12
ex3	9	9	9	11	11
ex5	9	9	9	10	10
lion	2	5	2	6	6
lion9	6	11	5	8	8
mc	4	7	4	6	6
modulo12	7	7	7	9	9
shiftreg	2	6	2	4	4
Total	62	86	61	89	89
Percentage, %	69.66	96.63	68.54	100.00	100.00

Table 11. Experimental results (numbers of LUTs for BM2).

Benchmark	Auto	One-Hot	JEDI	MPY	Our Approach
bbara	17	17	10	10	10
bbsse	33	37	24	26	25
beecount	19	19	14	14	14
cse	40	66	36	33	33
dk14	16	27	10	12	11
dk15	15	16	12	6	7
dk16	15	34	12	11	11
donfile	31	31	24	21	20
ex2	9	9	8	8	10
ex4	15	13	12	11	10
ex6	24	36	22	21	20
ex7	4	5	4	6	7
keyb	43	61	40	37	36
mark1	23	23	20	19	18
opus	28	28	22	21	21
s27	6	18	6	6	7
s386	26	39	22	25	24
s8	9	9	9	9	10
sse	33	37	30	26	24
Total	406	525	337	322	318
Percentage, %	127.67	165.09	105.97	101.26	100.00

Table 12. Experimental results (numbers of LUTs for BM3–BM5).

Benchmark	Auto	One-Hot	JEDI	MPY	Our Approach
ex1	70	74	53	40	34
kirkman	42	58	39	33	27
planet	131	131	88	78	68
planet1	131	131	88	78	68
pma	94	94	86	72	65
s1	65	99	61	54	48
s1488	124	131	108	89	83
s1494	126	132	110	90	78
s1a	49	81	43	38	32
s208	12	31	10	9	9
styr	93	120	81	70	59
tma	45	39	39	30	27
sand	132	132	114	99	79
s420	10	31	9	8	9
s510	48	48	32	22	19
s820	88	82	68	52	46
s832	80	79	62	50	44
Total	1340	1493	1091	912	795
Percentage, %	168.55	187.80	137.23	114.72	100.00

Table 13. Experimental results (the maximum operating frequency for BM1–BM5, MHz).

Benchmark	Auto	One-Hot	JEDI	MPY	Our Approach	Set
bbtas	204.16	204.16	206.12	200.38	200.38	BM1
dk17	199.28	167	199.39	199.87	199.87	BM1
dk27	206.02	201.9	204.18	196.65	196.65	BM1
dk512	196.27	196.27	199.75	194.17	194.17	BM1
ex3	194.86	194.86	195.76	191.22	191.22	BM1
ex5	180.25	180.25	181.16	178.06	178.06	BM1
lion	202.43	204	202.35	200.18	200.18	BM1
lion9	205.3	185.22	206.38	199.12	199.12	BM1
mc	196.66	195.47	196.87	193.17	193.17	BM1
modulo12	207	207	207.13	201.12	201.12	BM1
shiftreg	262.67	263.57	276.26	256.69	256.69	BM1
bbara	193.39	193.39	212.21	202.23	201.82	BM2
bbsse	157.06	169.12	182.34	181.23	179.22	BM2
beecount	166.61	166.61	187.32	185.14	183.29	BM2
cse	146.43	163.64	178.12	175.18	171.64	BM2
dk14	191.64	172.65	193.85	190.18	188.12	BM2
dk15	192.53	185.36	194.87	192.23	190.84	BM2
dk16	169.72	174.79	197.13	194.34	192.18	BM2
donfile	184.03	184	203.65	200.92	197.47	BM2
ex2	198.57	198.57	200.14	198.32	196.63	BM2
ex4	180.96	177.71	192.83	190.14	189.69	BM2
ex6	169.57	163.8	176.59	171.27	169.19	BM2
ex7	200.04	200.84	200.6	198.14	196.26	BM2
keyb	156.45	143.47	168.43	162.01	160.65	BM2
mark1	162.39	162.39	176.18	170.18	168.73	BM2
opus	166.2	166.2	178.32	175.29	173.68	BM2
s27	198.73	191.5	199.13	196.13	194.42	BM2
s386	168.15	173.46	179.15	176.85	175.16	BM2
s8	180.02	178.95	181.23	178.23	177.39	BM2
sse	157.06	169.12	174.63	170.12	168.14	BM2
ex1	150.94	139.76	176.87	182.34	180.01	BM3
kirkman	141.38	154	156.68	167.15	166.25	BM3
planet	132.71	132.71	187.14	189.12	188.73	BM3
planet1	132.71	132.71	187.14	189.12	188.73	BM3
pma	146.18	146.18	169.83	178.19	177.67	BM3
s1	146.41	135.85	157.16	162.23	162.12	BM3
s1488	138.5	131.94	157.18	168.32	167.54	BM3
s1494	149.39	145.75	164.34	172.27	171.09	BM3
s1a	153.37	176.4	169.17	178.21	177.42	BM3
s208	174.34	176.46	178.76	181.72	181.02	BM3
styr	137.61	129.92	145.64	161.87	160.73	BM3
tma	163.88	147.8	164.14	176.72	175.72	BM3
sand	115.97	115.97	126.82	145.68	153.49	BM4
s420	173.88	176.46	177.25	187.23	190.62	BM5
s510	177.65	177.65	181.42	187.32	189.12	BM5
s820	152	153.16	176.58	181.96	182.58	BM5
s832	145.71	153.23	173.78	186.12	188.32	BM5
Total	8127.08	8061.22	8701.97	8536.27	8508.25	
Percentage, %	95.52	94.75	102.28	100.33	100.00	

Table 14. Experimental results (the maximum operating frequency for BM1, MHz).

Benchmark	Auto	One-Hot	JEDI	MPY	Our Approach
bbtas	204.16	204.16	206.12	200.38	200.38
dk17	199.28	167	199.39	199.87	199.87
dk27	206.02	201.9	204.18	196.65	196.65
dk512	196.27	196.27	199.75	194.17	194.17
ex3	194.86	194.86	195.76	191.22	191.22
ex5	180.25	180.25	181.16	178.06	178.06
lion	202.43	204	202.35	200.18	200.18
lion9	205.3	185.22	206.38	199.12	199.12
mc	196.66	195.47	196.87	193.17	193.17
modulo12	207	207	207.13	201.12	201.12
shiftreg	262.67	263.57	276.26	256.69	256.69
Total	2254.90	2199.70	2275.35	2032.57	2032.57
Percentage, %	110.94	108.22	111.94	100.00	100.00

Table 15. Experimental results (the maximum operating frequency for BM2, MHz).

Benchmark	Auto	One-Hot	JEDI	MPY	Our Approach
bbara	193.39	193.39	212.21	202.23	201.82
bbsse	157.06	169.12	182.34	181.23	179.22
beecount	166.61	166.61	187.32	185.14	183.29
cse	146.43	163.64	178.12	175.18	171.64
dk14	191.64	172.65	193.85	190.18	188.12
dk15	192.53	185.36	194.87	192.23	190.84
dk16	169.72	174.79	197.13	194.34	192.18
donfile	184.03	184	203.65	200.92	197.47
ex2	198.57	198.57	200.14	198.32	196.63
ex4	180.96	177.71	192.83	190.14	189.69
ex6	169.57	163.8	176.59	171.27	169.19
ex7	200.04	200.84	200.6	198.14	196.26
keyb	156.45	143.47	168.43	162.01	160.65
mark1	162.39	162.39	176.18	170.18	168.73
opus	166.2	166.2	178.32	175.29	173.68
s27	198.73	191.5	199.13	196.13	194.42
s386	168.15	173.46	179.15	176.85	175.16
s8	180.02	178.95	181.23	178.23	177.39
sse	157.06	169.12	174.63	170.12	168.14
Total	3339.55	3335.57	3576.72	3508.13	3474.52
Percentage, %	96.12	96.00	102.94	100.97	100.00

Table 16. Experimental results (the maximum operating frequency for BM3-BM5, MHz).

Benchmark	Auto	One-Hot	JEDI	MPY	Our Approach
ex1	150.94	139.76	176.87	182.34	180.01
kirkman	141.38	154	156.68	167.15	166.25
planet	132.71	132.71	187.14	189.12	188.73
planet1	132.71	132.71	187.14	189.12	188.73
pma	146.18	146.18	169.83	178.19	177.67
s1	146.41	135.85	157.16	162.23	162.12
s1488	138.5	131.94	157.18	168.32	167.54
s1494	149.39	145.75	164.34	172.27	171.09
s1a	153.37	176.4	169.17	178.21	177.42
s208	174.34	176.46	178.76	181.72	181.02
styr	137.61	129.92	145.64	161.87	160.73
tma	163.88	147.8	164.14	176.72	175.72
sand	115.97	115.97	126.82	145.68	153.49
s420	173.88	176.46	177.25	187.23	190.62
s510	177.65	177.65	181.42	187.32	189.12
s820	152	153.16	176.58	181.96	182.58
s832	145.71	153.23	173.78	186.12	188.32
Total	2532.63	2525.95	2849.90	2995.57	3001.16
Percentage, %	84.39	84.17	94.96	99.81	100.00

As follows from Table 9, our approach produces FSM circuits with fewer LUTs than seen in other investigated methods. Our approach produces circuits having 50.42% less 6-LUTs than it is for equivalent Auto-based FSMs; 75.040% less 6-LUTs than it is for equivalent One-hot-based FSMs; 23.88% less 6-LUTs than it is for equivalent JEDI-based FSMs. As we expected, our approach allows circuits with better LUT counts than equivalent MPY-based FSMs to be obtained. Our approach gives 10.07% of gain. However, the analysis for different sets of benchmarks showed that sometimes our method loses, and sometimes it wins. The amount of gain (or loss) depends on each set a particular BM belongs to.

As follows from Table 10, our approach loses compared to three other investigated methods. There is the following loss: 30.34% relative to Auto-based FSMs; 3.37% relative to One-hot-based FSMs; 31.46% relative to JEDI-based FSMs. It is worth noting that there are the same LUT counts for equivalent BMs-based on both MPY and $P_C Z$ FSMs. This is easily explained. If there is $j = 1$, then $L + R \leq I_L$. In this case, LUT-based circuits of P FSMs are single-level. Therefore, there is no sense in the replacing inputs and encoding of COs. However, the encoding of COs is executed for both MPY and $P_C Z$ FSMs. Thus, their circuits include the redundant block $LUTerY$. This block consumes some chip resources; also, it adds some delay in the FSM cycle time.

Analysis of Tables 11 and 12 shows that using our approach leads to circuits with fewer LUTs compared with other investigated methods. Compared with Auto-based FSMs, there is either 27.67% win rate (set BM2) or 68.55% of gain in LUT counts (sets BM3–BM5). Compared with One-hot-based FSMs, there is either 65.09% win rate (set BM2) or 87.8% of gain in LUT counts (sets BM3–BM5). Compared with JEDI-based FSMs, there is either 5.97% of gain (set BM1) or 37.23% win rate (sets BM3–BM5). Compared with MPY-based FSMs, there is either 1.26% of gain (set BM1) or 14.72% win rate (sets BM3–BM5). So, the gain from using $P_C Z$ FSMs increases with the growth of the value $L + R$.

As follows from Table 13, our approach produces slightly faster LUT-based FSM circuits compared to Auto- and One-hot-based approaches. There is a gain of 4.48% and 5.25%, respectively. However, our approach is slightly inferior in performance compared to both JEDI-based FSMs (2.28%) and MPY-based FSMs (0.33%). The gain and loss varies depending on the value determined by the Formula (27). For the set BM1 (Table 14), our approach provides a loss relative to Auto-based FSMs (10.94%), One-hot-based FSMs (8.22%) and JEDI-based FSMs (11.94%). The same is true for MPY-based FSMs. This is explained by the existence of $LUTerY$ which is redundant for trivial FSMs. So, it does not make sense to use our approach for FSMs with $L + R \leq I_L$.

Table 15 shows results for the set BM2. As follows from Table 15, our approach produces faster circuits than both Auto- and One-hot-based FSMs (3.88% and 4% of gain, respectively). There is loss relatively to equivalent MPY-based FSMs (0.97% of loss). The JEDI-based FSMs win 2.94%. So, JEDI-based FSMs are the fastest for BMs from BM2.

As follows from Table 16, our method produces the fastest FSM circuits. There is the following gain: 15.61% compared with Auto-based FSMs; 15.83% compared with One-hot-based FSMs; 5.04% compared with JEDI-based FSMs; 0.19% compared with MPY-based FSMs. We believe that the gain compared to MPY-based FSMs is due to the fact that there are several levels of LUTs in the circuit of the block replacing FSM inputs.

So, the proposed approach allows the reduction of the LUT counts (and, therefore, the chip area occupied by FSM circuit) compared to equivalent MPY-based FSMs. At the same time, the gain in the number of LUTs grows with the increase in the total number of FSM inputs and state variables. The experimental results show that this gain in LUTs is not accompanied by the significant degradation in FSM operating frequency. Moreover, our approach produces slightly faster FSMs for rather complex FSMs (they belong to sets BM2–BM5). As follows from experimental results, $P_C Z$-based FSMs can replace other investigated models starting from simple FSMs (the set BM2).

7. Conclusions

Today, FPGA chips are widely used for implementing circuits of finite state machines representing sequential blocks of various digital systems. The increasing complexity of digital systems leads to an increase in the complexity of their sequential block circuits. In turn, this leads to an increase in the values of such FSM parameters as the numbers of inputs, outputs, transitions and states. At the same time, there is an increase in the gap between the numbers of LUT inputs on the one hand, and the summarized values of state variables and FSM inputs on the other hand. Modern LUTs have no more than six inputs. However, the number of literals in SOPs of functions representing FSM circuits significantly exceeds six. In these conditions, there is a need to apply various methods of functional decomposition for implementing LUT-based FSM circuits. As a result [42], the produced FSM circuits are multi-level and they have sophisticated systems of spaghetti-type interconnections.

As follows from [21], in many cases, the structural decomposition of LUT-based FSM circuits allows the improvement of their characteristics compared with equivalent FD-based FSM. So, as shown in [23], the three-block SD-based FSM circuits require fewer LUTs than their FD-based counterparts. However, the reducing LUT counts leads to the introduction of additional functions. To implement these functions, some FPGA chip internal resources are used. This is the main drawback of this approach.

It is known that the number of interconnections in a circuit is directly proportional to the LUT count. Interconnects have a significant impact on FSM performance and power consumption. Therefore, it is important to reduce the number of LUTs in the circuits of implemented blocks of digital systems. Modern very powerful FPGA chips are quite expensive. Many digital system designers may simply not have enough funds to purchase such expensive chips. Therefore, reducing the number of LUTs can make it possible to replace a more expensive chip with a cheaper one, where the number of elements will be sufficient to implement a system with optimized sequential blocks.

In this article, we propose to use the codes of collections of FSM outputs for generating both output functions and state variables. To do this, it is necessary to use two registers which keep these codes. The proposed method results in two-level FSM circuits which require fewer LUTs than their counterparts based on the approach [23]. Our approach gives an average a gain in the LUT counts around 10.07%. Note that the payoff in the number of LUTs increases with increasing complexity of FSMs. Moreover, the proposed two-block FSMs have practically the same cycle times as their three-block counterparts. It is very important that reducing the number of LUTs for the proposed method does not lead to performance degradation. We think that the proposed approach has enough positive qualities to be used for the implementation of LUT-based FSM circuits.

Author Contributions: Conceptualization, A.B., L.T., K.K. and K.M.; methodology, A.B., L.T., K.K. and K.M.; formal analysis, A.B., L.T., K.K. and K.M.; writing—original draft preparation, A.B., L.T., K.K. and K.M.; supervision, A.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available in the article.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

BM	standard benchmark
CLB	configurable logic block
CO	collection of outputs
CSC	composite state code
DST	direct structure table
FD	functional decomposition
FPGA	field-programmable gate array
FSM	finite state machine
IMF	input memory function
LUT	look-up table
SBF	systems of Boolean functions
SCR	state code register
SD	structural decomposition
SOP	sum-of-products
STG	state transitions graph
STT	state transition table

References

1. Glushkov, V. *Synthesis of Digital Automata*; FTD-MT, Translation Division, Foreign Technology Division: Wright-Patterson AIR Force Base, OH, USA, 1965; p. 487.
2. Baranov, S. *Logic and System Design of Digital Systems*; TUT Press: Tallinn, Estonia, 2008; p. 276.
3. Gajski, D.D.; Abdi, S.; Gerstlauer, A.; Schirner, G. *Embedded System Design: Modeling, Synthesis and Verification*, 1st ed.; Springer Publishing Company, Incorporated: Berlin/Heidelberg, Germany, 2009.
4. De Micheli, G. *Synthesis and Optimization of Digital Circuits*; McGraw–Hill: New York, NY, USA, 1994; p. 578.
5. Baranov, S. *Logic Synthesis of Control Automata*; Kluwer Academic Publishers: Norwell, MA, USA, 1994; p. 312.
6. Czerwinski, R.; Kania, D. Finite State Machine Logic Synthesis for Complex Programmable Logic Devices. In *Lecture Notes in Electrical Engineering*; Springer: Berlin/Heidelberg, Germany, 2013; Volume 231, p. 172. [CrossRef]
7. Gazi, O.; Arli, A. *State Machines Using VHDL: FPGA Implementation of Serial Communication and Display Protocols*; Springer: Berlin/Heidelberg, Germany, 2021. [CrossRef]
8. Koo, B.; Bae, J.; Kim, S.; Park, K.; Kim, H. Test Case Generation Method for Increasing Software Reliability in Safety-Critical Embedded Systems. *Electronics* **2020**, *9*, 797. [CrossRef]
9. Baranov, S. *High Level Synthesis of Digital Systems*; Amazon Publishing: Seattle, WA, USA, 2018; p. 207.
10. Zhao, X.; He, Y.; Chen, X.; Liu, Z. Human-Robot Collaborative Assembly Based on Eye-Hand and a Finite State Machine in a Virtual Environment. *Appl. Sci.* **2021**, *11*, 5754. [CrossRef]
11. Li, P.; Lilja, D.J.; Qian, W.; Riedel, M.D.; Bazargan, K. Logical Computation on Stochastic Bit Streams with Linear Finite-State Machines. *IEEE Trans. Comput.* **2014**, *63*, 1474–1486. [CrossRef]
12. Xie, Y.; Liao, S.; Yuan, B.; Wang, Y.; Wang, Z. Fully-Parallel Area-Efficient Deep Neural Network Design Using Stochastic Computing. *IEEE Trans. Circuits Syst. II Express Briefs* **2017**, *64*, 1382–1386. [CrossRef]
13. Bollig, B.; Fortin, M.; Gastin, P. Communicating finite-state machines, first-order logic, and star-free propositional dynamic logic. *J. Comput. Syst. Sci.* **2021**, *115*, 22–53. [CrossRef]
14. Cassel, S.; Howar, F.; Jonsson, B.; Steffen, B. Active Learning for Extended Finite State Machines. *Form. Asp. Comput.* **2016**, *28*, 233–263. [CrossRef]
15. Jóźwiak, L.; Ślusarczyk, A.; Chojnacki, A. Fast and compact sequential circuits for the FPGA-based reconfigurable systems. *J. Syst. Archit.* **2003**, *49*, 227–246. [CrossRef]
16. Islam, M.M.; Hossain, M.; Shahjalal, M.; Hasan, M.K.; Jang, Y.M. Area-Time Efficient Hardware Implementation of Modular Multiplication for Elliptic Curve Cryptography. *IEEE Access* **2020**, *8*, 73898–73906. [CrossRef]
17. Maruyama, T.; Yamaguchi, Y.; Osana, Y. Programmable Logic Devices (PLDs) in Practical Applications. In *Principles and Structures of FPGAs*; Amano, H., Ed.; Springer: Singapore, 2018; pp. 179–206. [CrossRef]
18. Skliarova, I.; Sklyarov, V. *FPGA-Based Hardware Accelerators*; Lecture Notes in Electrical Engineering; Springer: Berlin/Heidelberg, Germany, 2019; p. 245. [CrossRef]
19. Ruiz-Rosero, J.; Ramirez-Gonzalez, G.; Khanna, R. Field Programmable Gate Array Applications—A Scientometric Review. *Computation* **2019**, *7*, 63. [CrossRef]
20. Trimberg, S. Three ages of FPGA: A Retrospective on the First Thirty Years of FPGA Technology. *IEEE Proc.* **2015**, *103*, 318–331. [CrossRef]
21. Barkalov, A.; Titarenko, L.; Krzywicki, K. Structural Decomposition in FSM Design: Roots, Evolution, Current State—A Review. *Electronics* **2021**, *10*, 1174. [CrossRef]

22. Barkalov, A.; Titarenko, L.; Krzywicki, K.; Saburova, S. Improving the Characteristics of Multi-Level LUT-Based Mealy FSMs. *Electronics* **2020**, *9*, 1859. [CrossRef]
23. Barkalov, A.; Titarenko, L.; Krzywicki, K. Reducing LUT Count for FPGA-Based Mealy FSMs. *Appl. Sci.* **2020**, *10*, 5115. [CrossRef]
24. Grout, I. *Digital Systems Design with FPGAs and CPLDs*; Elsevier Science: Amsterdam, The Netherlands, 2011; p. 718.
25. Kubica, M.; Kania, D.; Kulisz, J. A Technology Mapping of FSMs Based on a Graph of Excitations and Outputs. *IEEE Access* **2019**, *7*, 16123–16131. [CrossRef]
26. Skliarova, I.; Sklyarov, V.; Sudnitson, A. *Design of FPGA-Based Circuits Using Hierarchical Finite State Machines*; TUT Press: Tallinn, Estonia, 2012.
27. Baranov, S. *Finite State Machines and Algorithmic State Machines*; Amazon Publishing: Seattle, WA, USA 2018; p. 185.
28. Chapman, K. *Multiplexer Design Techniques for Datapath Performance with Minimized Routing Resources*; Xilinx: San Jose, CA, USA, 2014; pp. 1–32. Available online: https://www.xilinx.com/support/documentation/application_notes/xapp522-mux-design-techniques.pdf (accessed on 8 January 2022).
29. Trimberger, S. *Field-Programmable Gate Array Technology*; Springer US: New York, NY, USA, 2012.
30. Mishchenko, A.; Brayton, R.; Jiang, J.H.R.; Jang, S. Scalable Don't-Care-Based Logic Optimization and Resynthesis. *ACM Trans. Reconfigurable Technol. Syst.* **2011**, *4*, 1–23. [CrossRef]
31. Scholl, C. *Functional Decomposition with Application to FPGA Synthesis*; Kluwer Academic Publishers: Boston, MA, USA, 2001.
32. Kubica, M.; Kania, D. Technology mapping oriented to adaptive logic modules. *Bull. Pol. Acad. Sci. Tech. Sci.* **2019**, *67*, 947–956.
33. Mishchenko, A.; Chatterjee, S.; Brayton, R. Improvements to technology mapping for LUT-based FPGAs. *Comput.-Aided Des. Integr. Circuits Syst.* **2007**, *26*, 240–253. [CrossRef]
34. Khatri, S.; Gulati, K. (Eds.) *Advanced Techniques in Logic Synthesis, Optimizations and Applications*; Springer: New York, NY, USA; Dordrecht, The Netherlands; London, UK, 2011; p. 425. [CrossRef]
35. Xilinx. FPGA. Available online: https://www.xilinx.com/products/silicon-devices/fpga.html (accessed on 7 January 2022).
36. Soloviev, V. *Architecture of the FILM of the Firm Xilinx: CPLD and FPGA of the 7th Series*; Hotline-Telecom: Moscow, Russia, 2016; p. 392. (In Russian)
37. Altera. Cyclone IV Device Handbook. Available online: http://www.altera.com/literature/hb/cyclone-iv/cyclone4-handbook.pdf (accessed on 6 January 2022).
38. Feng, W.; Greene, J.; Mishchenko, A. Improving FPGA Performance with a S44 LUT Structure. In Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, New York, NY, USA, 25–27 February 2018; pp. 61–66. [CrossRef]
39. Barkalov, O.; Titarenko, L.; Mielcarek, K. Hardware reduction for LUT-based Mealy FSMs. *Int. J. Appl. Math. Comput. Sci.* **2018**, *28*, 595–607. [CrossRef]
40. Barkalov, O.; Titarenko, L.; Mielcarek, K. Improving characteristics of LUT-based Mealy FSMs. *Int. J. Appl. Math. Comput. Sci.* **2020**, *30*, 745–759. [CrossRef]
41. Senhadji-Navarro, R.; Garcia-Vargas, I. Methodology for Distributed-ROM-Based Implementation of Finite State Machines. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2020**, *40*, 2411–2415. [CrossRef]
42. Kubica, M.; Opara, A.; Kania, D. *Technology Mapping for LUT-Based FPGA*; Springer: Berlin/Heidelberg, Germany, 2021. [CrossRef]
43. Kubica, M.; Kania, D. Decomposition of multi-output functions oriented to configurability of logic blocks. *Bull. Pol. Acad. Sci. Tech. Sci.* **2017**, *65*, 317–331. [CrossRef]
44. Salauyou, V.; Ostapczuk, M. State Assignment of Finite-State Machines by Using the Values of Output Variables. In *Theory and Applications of Dependable Computer Systems*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 543–553. [CrossRef]
45. Solov'ev, V.V. Implementation of finite-state machines based on programmable logic ICs with the help of the merged model of Mealy and Moore machines. *J. Commun. Technol. Electron.* **2013**, *58*, 172–177. [CrossRef]
46. Park, J.; Yoo, H. Area-Efficient Differential Fault Tolerance Encoding for Finite State Machines. *Electronics* **2020**, *9*, 1110. [CrossRef]
47. Amann, R.; Baitinger, U. Optimal state chains and states codes in finite state machines. *IEEE Trans. Comput. Aided Des.* **1989**, *8*, 153–170. [CrossRef]
48. Chattopadhyay, S. Area conscious state assignment with flip-flop and output polarity selection for finite state machines synthesis—A genetic algorithm. *Comput. J.* **2005**, *48*, 443–450. [CrossRef]
49. De Micheli, G.; Brayton, R.K.; Sangiovanni-Vincentelli, A. Optimal State Assignment for Finite State Machines. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2006**, *4*, 269–285. [CrossRef]
50. El-Maleh, A.H. A probabilistic pairwise swap search state assignment algorithm for sequential circuit optimization. *Integr. VLSI J.* **2017**, *56*, 32–43. [CrossRef]
51. Sentowich, E.; Singh, K.; Lavango, L.; Moon, C.; Murgai, R.; Saldanha, A.; Savoj, H.; P, P.S.; Bryton, R.; Sangiovanni-Vincentelli, A. *SIS: A System for Sequential Circuit Synthesis*; Technical Report; University of California: Berkely, CA, USA, 1992.
52. ABC System. 2022. Available online: https://people.eecs.berkeley.edu/~alanmi/abc/ (accessed on 1 January 2022).
53. Brayton, R.; Mishchenko, A. ABC: An Academic Industrial-Strength Verification Tool. In *Computer Aided Verification*; Touili, T., Cook, B., Jackson, P., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 24–40. [CrossRef]
54. Baranov, S. *From Algorithm to Digital System: HSL and RTL tool Sinthagate in Digital System Design*; Amazon Publishing: Seattle, WA, USA, 2020; p. 76.

55. Xilinx. Vivado Design Suite User Guide: Synthesis; UG901 (v2019.1). 2022. Available online: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug901-vivado-synthesis.pdf (accessed on 2 January 2022).
56. Xilinx. Vitis Platform. Available online: https://www.xilinx.com/products/design-tools/vitis/vitis-platform.html (accessed on 3 January 2022).
57. Quartus Prime. 2022. Available online: https://www.intel.pl/content/www/pl/pl/software/programmable/quartus-prime/overview.html (accessed on 4 January 2022.)
58. Sklyarov, V. *Synthesis and Implementation of RAM-Based Finite State Machines in FPGAs*; Springer: Berlin/Heidelberg, Germany, 2000; Volume 1896, pp. 718–728. [CrossRef]
59. Tiwari, A.; Tomko, K. Saving power by mapping finite-state machines into Embedded Memory Blocks in FPGAs. In Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, Paris, France, 16–20 February 2004; pp. 916–921.
60. Wilkes, M.V.; Stringer, J.B. Micro-programming and the design of the control circuits in an electronic digital computer. *Math. Proc. Camb. Philos. Soc.* **1953**, *49*, 230–238. [CrossRef]
61. Achasova, S. *Synthesis Algorithms for Automata with PLAs*; M: Soviet Radio: Moscow, Russia, 1987; p. 135. (In Russian)
62. McElvain, K. Lgsynth93 Benchmark Set: Version 4.0. 1993. Available online: https://people.engr.ncsu.edu/brglez/CBL/benchmarks/LGSynth93/LGSynth93.tar (accessed on 29 June 2022).
63. Xilinx. VC709 Evaluation Board for the Virtex-7 FPGA. Available online: https://www.xilinx.com/support/documentation/boards_and_kits/vc709/ug887-vc709-eval-board-v7-fpga.pdf (accessed on 5 January 2022).