Improving Characteristics of LUT-Based Mealy FSMs with Twofold State Assignment

Alexander Barkalov 1,2, Larysa Titarenko 1,3, Kazimierz Krzywicki 4,* and Svetlana Saburova 3

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 Mathematics and Information Technology, Vasyl’ Stus Donetsk National University, 600-Richya Str. 21, 21021 Vinnytsia, Ukraine
3 Department of Infocommunication Engineering, Faculty of Infocommuncations, Kharkiv National University of Radio Electronics, Nauky Avenue 14, 61166 Kharkiv, Ukraine; sabsvet@gmail.com
4 Department of Technology, The Jacob of Paradies University, ul. Teatralna 25, 66-400 Gorzów Wielkopolski, Poland
* Correspondence: kkrzywicki@ajp.edu.pl

Abstract: Practically, any digital system includes sequential blocks. This article is devoted to a case when sequential blocks are represented by models of Mealy finite state machines (FSMs). The performance (maximum operating frequency) is one of the most important characteristics of an FSM circuit. In this article, a method is proposed which aims at increasing the operating frequency of LUT-based Mealy FSMs with twofold state assignment. This is done using only extended state codes. Such an approach allows excluding a block of transformation of binary state codes into extended state codes. The proposed approach leads to LUT-based Mealy FSM circuits having two levels of logic blocks. Each function for any logic level is represented by a circuit including a single LUT. The proposed method is illustrated by an example of synthesis. The results of experiments conducted with standard benchmarks show that the proposed approach produces LUT-based circuits with significantly higher operating frequency than it is for circuits produced by other investigated methods (Auto and One-hot of Vivado, JEDI, twofold state assignment). The performance is increased by an average of 15.9 to 25.49 percent. These improvements are accompanied by a small growth of the numbers of LUTs compared with circuits based on twofold state assignment. Our approach provides the best area-time products compared with other investigated methods. The advantages of the proposed approach increase as the number of FSM inputs and states increases.

Keywords: mealy FSM; FPGA; LUT; synthesis; twofold state assignment; extended state codes

1. Introduction

Practically, any digital system includes various sequential blocks [1–3]. To specify the behaviour of a sequential block, it is necessary to use some formal model. In many cases, the behaviour of sequential blocks is specified by Mealy finite state machines (FSMs) [4–7]. Three main characteristics determine the quality of an FSM circuit: the chip area occupied by the circuit, performance (either minimum propagation time or maximum operating frequency), and the consumption of power. These characteristics are strongly interconnected [8]. As a rule, the occupied chip area significantly affects other characteristics of an FSM circuit [9,10]. Various methods of structural decomposition can be used for optimizing the size of the occupied chip area [11]. These methods have one serious drawback: they lead to multi-level FSM circuits with a significant decrease in performance compared with equivalent single-level circuits.

If a multi-level FSM circuit does not provide the required operating frequency, then it is necessary to reduce the number of levels. This reducing must be performed in a way that increases as little as possible the occupied chip area. One of these methods is proposed...
in [12]. This is a method of twofold state encoding. The method is aimed at Mealy FSMs implemented with field programmable gate arrays (FPGAs) [13–15].

Now, FPGAs are widely used for implementing various digital systems [16–18]. Due to this, we chose the FPGA-based Mealy FSMs as a research object. Our current article considers Mealy FSM circuits implemented using three main components of FPGA chips. These components are look-up table (LUT) elements, programmable flip-flops and interconnections of FPGAs. We focus our current research on solutions of Xilinx which is the largest manufacturer of FPGA chips [19,20]. So, we consider ways of increasing the performance of LUT-based Mealy FSMs.

A LUT is a logic block having $N_{LUT}$ inputs and a single output [14,21]. A single LUT may implement an arbitrary Boolean function having up to $N_{LUT}$ arguments. Unfortunately, the value of $N_{LUT}$ is very small [20]. If a Boolean function depends on more than $N_{LUT}$ variables, then it is necessary to apply methods of functional decomposition of this function [19,22]. It is known, the applying functional decomposition leads to multi-level FSM circuits with “spaghetti-type” interconnections [23,24].

FSM circuits are represented by systems of Boolean functions (SBFs). To implement an LUT-based FSM circuit, it is necessary to transform an initial SBF into a network of LUTs of a particular FPGA chip. This is a step of technology mapping [25,26]. The outcome of technology mapping tremendously affects all main characteristics of FSM circuits [20,27,28]. To implement an LUT-based FSM circuit, it is necessary to use LUTs, programmable flip-flops, programmable interconnections, circuits of synchronization, and input-output blocks.

The article [26] notes that time delays of the interconnection system are starting to play a major role in comparison with LUT delays. Furthermore, more than 70% of the power dissipation is due to the interconnections [29]. So, the optimization of interconnections leads to increasing the performance and reducing the power consumption of LUT-based FSM circuits. This can be done, for example, using the twofold state assignment [12].

Obviously, increasing the number of LUT inputs leads to a decrease in both the number of LUTs and their levels in FSM circuits. However, as shown in [30,31], for the foreseeable future, it is very difficult to expect an increase in the number of LUT inputs. Basically, modern LUTs have no more than 6 inputs [14,21]. An increase in the number of inputs leads to an imbalance for the main characteristics of an LUT circuit. However, this state of affairs leads to an imbalance between an increasing number of arguments in SBFs representing FSM circuits and a fairly small number of LUT inputs. However, the increasing complexity of modern digital systems is accompanied by an increase in the number of arguments in functions representing FSM circuits. This imbalance is the source of the need to improve synthesis methods of LUT-based FSMs.

Our current article is devoted to synthesis of multi-level LUT-based circuits of Mealy FSMs obtained using the method of twofold state assignment [12]. The twofold state assignment improves characteristics of LUT-based Mealy FSMs compared with their counterparts based on functional decomposition [32]. However, applying twofold state assignment leads to FSM circuits with three levels of LUTs. These circuits are significantly slower than their counterparts with fewer logic levels. If performance is the dominant quality factor, then the number of levels in FSM circuit should be reduced. It is extremely important that the increase in operating frequency is accompanied by as small an increase in the number of LUTs as possible (compared with the original three-level FSM circuit).

Recently, we have published works [11,12,32] where a method of twofold state assignment have been proposed. In these works, each FSM state has two codes. The first of them represents a state as an element of the set of FSM states. The second code is an extended state code (ESC). The ESC represents a state as an element of some class of partition of the set of states. This approach requires using an additional block generating ESCs. This block introduces an additional level of logic to the FSM circuit. In our current article, we propose a method which allows eliminating this block. The elimination of this additional block is the main feature that distinguishes our current work from works [11,12,32]. This
approach leads to an increase in the value of FSM operating frequency compared with FSMs based on methods [11,12,32].

The main contribution of this paper is a novel design method aimed at increasing the operating frequency of LUT-based Mealy FSMs with twofold state assignment. The main idea of the proposed approach is to use only extended state codes. This approach leads to a completely new structural diagram of the LUT-based Mealy FSM. This diagram does not include a block transforming binary state codes into extended state codes. Our current research shows that this approach leads to FSM circuits having practically the same amount of LUTs as FSM circuits based on the twofold state assignment. The experimental results show that this method allows increasing the maximum operating frequency of LUT-based FSMs compared with equivalent FSMs obtained using some known methods of FSM design.

The further text of the article includes six sections. The second section shows the background of single-level LUT-based Mealy FSMs. The third section discusses the state-of-the-art in the synthesis of LUT-based FSMs. The fourth section represents the main idea of the proposed method. The fifth section includes an example of FSM synthesis using the extended state codes. The results of experiments are shown and discussed in the sixth section of the article. A short conclusion ends the article.

2. Single-Level LUT-Based Mealy FSMs

FPGAs manufactured by Xilinx include a lot of configurable logic blocks (CLB) [1,15], such as embedded memory blocks, digital signal processors, even microprocessors. The CLBs are connected using a programmable routing matrix [33]. In this paper, we consider CLBs including LUTs, multiplexers and programmable flip-flops. A LUT has \( N_{LUT} \) inputs and a single output. Networks of LUTs implement systems of Boolean functions representing an FSM circuit.

A LUT can implement an arbitrary Boolean function which depends on up to \( N_{LUT} \) arguments. A combinational output of a LUT may be connected with a flip-flop. Usually, D flip-flops are used to organize an FSM state register (SRG) [34,35]. The value of a function may be loaded into a flip-flop using the pulse of synchronization \( Clk \). It is possible to make output of flip-flop equal to zero using the pulse \( Reset \). To select the type of a CLB output (combinational or registered), a programmable multiplexer is used.

In practical digital systems, an SBF representing an FSM circuit may depend on up to 50–70 literals [4,33]. At the same time, modern LUTs have no more than 6 inputs. So, there is a contradiction between a large number of arguments and a very small number of LUT inputs. This leads to the need for functional decomposition (FD) of SBFs representing an FSM circuit [36,37]. However, FD-based circuits have a lot of logic levels and complex systems of “spaghetti-type” interconnections [38,39].

A Mealy FSM is represented by a six-component vector \(< I, O, S, \delta, \lambda, s_1 > \) [4,33]. These components are the following: \( I = \{i_1, \ldots, i_l\} \) is a set of inputs, \( O = \{o_1, \ldots, o_N\} \) is a set of outputs, \( S = \{s_1, \ldots, s_M\} \) is a set of internal states, a function of transitions \( \delta \), a function of outputs \( \lambda \), and an initial state \( s_1 \). An FSM can be represented using various tools, such as state transition graphs [4], binary decision diagrams [40,41], and-inverter graphs [25,26], graph-schemes of algorithms [33], state transition tables (STTs) [4].

Very often, STGs represent FSMs. An STG is a directed graph which vertices correspond to FSM states and edges correspond to interstate transitions [4]. Each edge is directed from the current state \( s_m \in S \) to the state of transition \( s_T \in S \). There are \( H \) edges in an STG. The \( h \)-th edge is marked by the pair \(< I_h, O_h > \), where \( I_h \) is a conjunction of inputs causing a transition \( < s_m, s_T > \), \( O_h \) is a collection of outputs generated during the transition \( < s_m, s_T > \). Consider a Mealy FSM \( A_0 \) represented by the STG shown in Figure 1.
Figure 1. State transition graph of Mealy FSM $A_0$.

The following characteristics of FSM $A_0$ can be found from Figure 1: $L = 5$ inputs, $N = 7$ outputs, $M = 6$ states, and $H = 15$ interstate transitions. Furthermore, the STG (Figure 1) defines functions of transitions and outputs of FSM $A_0$.

An STG can be transformed into an STT. Each edge of an STG corresponds to a single row of an STT. For example, FSM $A_0$ can be represented by its STT (Table 1) obtained from the STG (Figure 1).

Table 1. STT of Mealy FSM $A_0$.

| $s_m$ | $s_T$ | $I_h$ | $O_h$ | $h$ |
|-------|-------|-------|-------|-----|
| $s_1$ | $s_2$ | $i_1$ | $o_1$ | 1   |
|       | $s_2$ | $i_1$ | $o_2$ | 2   |
|       | $s_3$ | $i_1$ | $o_3$ | 3   |
| $s_2$ | $s_3$ | $i_4$ | $o_4$ | 4   |
|       | $s_4$ | $i_4$ | $o_5$ | 5   |
| $s_3$ | $s_1$ | $i_2$ | $o_1$ | 6   |
|       | $s_5$ | $i_2$ | $o_2$ | 7   |
|       | $s_5$ | $i_2$ | $o_7$ | 8   |
| $s_4$ | $s_2$ | $i_3$ | $o_6$ | 9   |
|       | $s_4$ | $i_3$ | $o_4$ | 10  |
|       | $s_6$ | $i_2$ | $o_1$ | 11  |
| $s_5$ | $s_6$ | $1$   | $o_3$ | 12  |
| $s_6$ | $s_1$ | $i_4$ | $-1$  | 13  |
|       | $s_4$ | $i_4$ | $o_3$ | 14  |
|       | $s_5$ | $i_4$ | $o_5$ | 15  |

There are five columns in an STT [4]. The first four of them are the following: a current state $s_m$; a state of transition $s_T$; a conjunction of inputs $I_h$; a collection of outputs $O_h$. The column $h$ includes the numbers of interstate transitions ($h \in \{1, \ldots, H\}$). For example, the following functions are determined by the fourth row of Table 1: $\delta(s_2, i_4) = s_3$ and $\lambda(s_2, i_4) = o_3$. For a row with the unconditional transition, there is $I_h = 1$. In Table 1, this is the row 12.

Let us encode FSM states $s_m \in S$ by binary codes $K(s_m)$ having $R_s$ bits where

$$R_s = \lceil \log_2 M \rceil.$$
To encode states, we use state variables from the set $T = \{T_1, \ldots, T_R\}$. These variables are kept into the SRG. To change the contents of SRG, the special input memory functions (IMFs) are used. They form a set $D = \{D_1, \ldots, D_R\}$. Now, we can transform an STT into a direct structure table (DST). A DST is an extension of the STT with the three following columns: $K(s_m)$, $K(s_T)$, and $D_b$ [1]. The first of them contains a code of a current state, the second includes a code of a state of transition, and there is a collection of IMFs in the third of them. This collection includes IMFs equal to 1 to load the code $K(s_T)$ into SRG.

Using a DST, we can derive two following SBFs representing an FSM circuit:

$$D = D(T, I);$$  \hspace{1cm} (2)

$$O = O(T, I).$$  \hspace{1cm} (3)

The SBF (2) represents the function of transitions, the SBF (3) represents the function of outputs. Using the terminology from [42], we can state that SBFs (2) and (3) represent a structural diagram of $P$ Mealy FSM (Figure 2).

![Figure 2. Structural diagram of $P$ Mealy FSM.](image)

In $P$ Mealy FSMs, the block of transition logic is determined by SBF (2), the block of output logic is specified by SBF (3). The inputs of register SRG are connected with outputs of the block of transition logic. In each cycle of FSM operation, the SRG contains a current state code. The pulse of synchronization $Clk$ allows changing the contents of SRG. To load the zero code of the initial state $s_1 \in S$ into SRG, it is necessary to generate the pulse $Reset$. We discuss a case when both blocks are implemented using LUT-based CLBs. In this case, the flip-flops of SRG are distributed among LUTs of the block of transition logic.

The analysis of systems (2) and (3) shows that both input memory functions and FSM outputs depend on state variables and FSM inputs. This peculiarity of Mealy FSMs is used for optimizing LUT-based FSM circuits [11].

3. Optimizing Circuits of FPGA-Based Mealy FSMs

There are a significant number of methods for synthesis of LUT-based FSMs [11,12,23,24,26,32,36–38,41,43,44]. It is quite possible that the quality of FSM circuits obtained by different synthesis methods will differ significantly. As a rule, the quality of an FSM circuit is determined by a combination of three main characteristics. These characteristics are the following: (1) chip resources used for implementing the circuit; (2) the maximum performance and (3) the power consumption [45,46]. In the case of LUT-based FSMs, the following chip resources are necessary: (1) LUTs; (2) programmable flip-flops; (3) programmable interconnections; (4) the circuit of synchronization and (5) the programmable input-outputs of a chip. Obviously, the best FSM circuit consumes the minimum amount of chip resources, has the maximum possible operating frequency, and consumes the minimum power.

To improve the quality of an FSM circuit, it is necessary to solve some optimization problems [47,48]. In this article, we propose a method for improving the performance (the maximum operating frequency) of LUT-based Mealy FSMs.
The functions (2) and (3) are represented as sum-of-products (SOPs) [4]. These SOPs include product terms $F_h$ corresponding to rows of a DST. A term $F_h$ is determined as the following conjunction:

$$F_h = S_m \land I_h \quad (h \in \{1, \ldots, H\}).$$  \hspace{1cm} (4)

In (4), the symbol $S_m$ stands for a conjunction of state variables corresponding to the code of a current state $s_m \in S$ from the $h$-th row of the DST.

Each function $f_i \in D \cup O$ depends on $NL(f_i)$ literals, where a literal is either direct or compliment value of a Boolean variable [4]. Consider the following condition

$$NL(f_i) \leq NI_{LUT}.$$  \hspace{1cm} (5)

If condition (5) takes place, then there is exactly a single LUT in the circuit implementing the function $f_i \in D \cup O$. If condition (5) is violated, then the corresponding LUT-based circuit is a multi-level one.

To design multi-level LUT-based FSM circuits, various methods of functional decomposition (FD) may be used [22]. During the process of FD, an initial SOP is broken down by partial SOPs corresponding to some additional functions. This process is terminated when each partial SOP includes no more than $NI_{LUT}$ literals. Different partial SOPs of a function $f_i \in D \cup O$ may include the same inputs $i_l \in I$ or the same state variables $T_r \in T$. Thus, there is a duplication of literals in different partial SOPs of the original SOP. This phenomenon leads to a significant complication of the interconnection system. In turn, this not only complicates the placement and tracing process, but also reduces performance and increases power consumption compared with an equivalent single-level circuit [11,29].

If condition (5) is true for all functions representing an FSM circuit, then the number of LUTs in the circuit is equal to $N + RS$. This is the best possible LUT count. If condition (5) is violated, then the LUT count is equal to $RS + N(F)$, where $N(F)$ is the number of additional functions different from $f_i \in D \cup O$.

To optimize FSM circuits, it is necessary to reduce the value of $N(F)$. The importance of this problem has led to the development of a significant number of methods of FD [22]. Various algorithms of FD are included into CAD tools aimed in implementation of FPGA-based digital systems.

The values of $NL(f_i)$ could be reduced due to proper state assignment [4]. For example, it is possible to represent a state using only a single state variable. This is achieved by the one-hot state assignment, when $RS = M$ [49]. The one-hot state assignment requires using the SRG with $M$ bits. However, this is not a problem because modern FPGAs include a lot of programmable flip-flops. Due to this, this approach is very popular in FPGA-based design. For example, this method is used in the academic CAD system ABC by Berkeley [50,51]. It is also used in industrial CAD packages such as, for example, Vivado of Xilinx [52] and Quartus of Intel (Altera) [53].

The Equation (1) determines so called maximum binary state codes [4]. The negative effect of one-hot state assignment is an increase in the number of IMFs compared with their minimum possible number (1). However, these IMFs are much simpler than in the case of maximum binary state assignment [1]. These approaches have been compared, for example, in [54]. The research [54] shows that using one-hot codes improves FSM characteristics if there is $M > 16$. However, the number of state variables is not the only factor influencing the circuit characteristics. The limited number of LUT inputs increases the effect of the number of FSM inputs on the characteristics of LUT-based FSM circuits [1]. It is shown in [35] that the one-hot state assignment produces worse FSM circuits if the number of FSM inputs exceeds 10.

So, in one case the best results are produced by the method of maximum binary state assignment, and in the other case it is better to use the one-hot state codes. Thus, it is necessary to check which method will give the best results for a particular FSM. Due to this, we have compared the FSM circuits produced by our proposed approach with FSM circuits
produced by three other methods of state assignment. As a base for comparison, we chose the algorithm JEDI [9,55], the methods of binary state assignment Auto and the One-hot state assignment of Vivado [52] by Xilinx [15]. We chose Vivado because it aims in Xilinx FPGA chips. We chose JEDI because it is one of the best maximum binary state assignment methods [5].

It is possible to encode states in a way reducing the power consumption [56]. The majority of such methods are based on reducing the switch activity of an FSM circuit [48,57]. To do it, it is necessary to minimize the Hamming distance for codes of states with the maximum amount of transitions [57]. However, our research [12,38,39] shows that the power consumption can be reduced due to the reducing the number of interconnections inside an FSM circuit. To reduce the number of interconnections, it is necessary to minimize the numbers of arguments in SBFs (2) and (3) [4]. This can be done using various methods of state assignment.

The structural decomposition is an efficient way of reducing LUT counts in Mealy FSMs logic circuits [11]. The main idea of these methods is the elimination of direct connection between FSM inputs \( i_l \in I \) and state variables \( T_r \in T \), on the one hand, and outputs \( o_n \in O \) and IMFs \( D_r \in D \), on the other hand. This is connected with introducing additional functions forming a set \( F \) having \( N(F) \) elements. The functions \( f_j \in F \) depend on FSM inputs and state variables. In turn, FSM outputs and IMFs use these additional functions as arguments creating literals of corresponding SOPs. To optimize LUT count due to applying the methods of structural decomposition (SD), the following conditions should take places [11]:

\[ N(F) \ll N + R_S; \]  
\[ N(F) \ll L + R_S. \]  

All known methods of SD are based on conditions (6) and (7). These methods are analysed, for example, in [11]. If condition (5) is violated for some functions \( f_j \in F \), then joint application of FD- and SD-based decomposition methods is necessary [12].

One of the SD-based methods is a method of twofold state assignment [12,32]. Let us analyse this method. The method is based on construction a partition \( \pi_S = \{S^1, \ldots, S^K\} \) of the set of Mealy FSM states. Each class \( S^k \in \pi_S \) includes \( M_k \) states. The maximum binary codes \( C(s_m) \) are used for encoding states as elements of some class \( S^k \in \pi_S \). There are \( R(S^k) \) bits in codes \( C(s_m) \) of states \( s_m \in S^k \), where

\[ R(S^k) = \lceil \log_2(M_k + 1) \rceil. \]  

To encode states \( s_m \in S^k \), the variables \( \tau_r \in \tau^k \) are used. The sets \( \tau^1, \ldots, \tau^K \) form a set \( \tau \) having \( R_0 \) elements:

\[ R_0 = R_1 + \ldots + R_K. \]  

Each class \( S^k \in \pi_S \) determines three sets. The set \( I^k \subseteq I \) includes inputs causing transitions from states \( s_m \in S^k \). There are \( L_k \) elements in the set \( I^k \subseteq I \). The set \( O^k \subseteq O \) includes outputs generating during the transitions from states \( s_m \in S^k \). The set \( D^k \subseteq D \) includes IMFs equal to 1 during the transitions from states \( s_m \in S^k \).

This method can be used if the following condition takes place:

\[ R(S^k) + L_k \leq NI_{LUT} \quad (k \in \{1, \ldots, K\}). \]  

In this case, it is possible to use the model of \( P_T \) Mealy FSM (Figure 3).
A class $S^k \in \tau_S$ corresponds to a BlockSk implementing SBFs

$$D^k = D^k(t^k, I^k),$$

$$O^k = O^k(t^k, I^k).$$

The circuit of each BlockSk is implemented with LUTs having $N_{LUT}$ inputs. The functions (11) and (12) represent the partial SOPs of FSM input memory functions and outputs. The BlockTO creates final values of functions from the set $D \cup O$. To do it, each LUT of this block implements disjunctions having up to $K$ inputs:

$$D_r = \bigvee_{k=1}^{K} D^k_r \ (r \in \{1, \ldots, R_S\});$$

$$o_n = \bigvee_{k=1}^{K} o^k_n \ (n \in \{1, \ldots, N\}).$$

The outputs of LUTs producing functions (13) are connected with inputs of D flip-flops. This explains the existence of pulses Reset and Clk as inputs of BlockTO. So, this block produces outputs $o_n \in O$ and state variables $T_r \in T$.

To create SBFs (11) and (12), it is necessary to have state variables $\tau_r \in \tau$. These variables are generated by the Block$\tau$. This block transforms state variables $T_r \in T$ and generates the following SBF:

$$\tau = \tau(T).$$

As follows from [12], the circuits of $P_T$ FSMs require fewer LUTs than the circuits of equivalent $P$ Mealy FSMs. If condition (10) takes place, circuits of $P_T$ FSMs also have fewer levels of logic. Due to this, they are faster than the circuits of equivalent $P$ Mealy FSMs.

As follows from Figure 3, both inputs $i_l \in I$ and state variables $\tau_r \in \tau$ enter only LUTs of the first level of FSM circuit. The partial functions (11) and (12) enter only LUTs of the second level of FSM circuit. At last, the state variables $T_r \in T$ enter only LUTs of the Block$\tau$ creating the third level of logic. Due to this, the circuits of $P_T$ FSMs have regular systems of interconnections. This distinguishes them from the circuits of FD-based $P$ FSMs having complex systems of “spaghetti-type” interconnections. Our research [11,32] shows that the circuits of $P_T$ FSMs consume less power than the circuits of equivalent $P$ Mealy FSMs.

The analysis of Figure 3 shows that $P_T$ FSMs have a drawback. Namely, they include Block$\tau$ in the path leading from inputs $i_l \in I$ to state variables $\tau_r \in \tau$. The conversion (15) takes some time, which is added to the FSM cycle time. In this article, we propose a way of eliminating the Block$\tau$ from FSM circuit. It allows reducing the cycle time and, therefore, increasing the value of maximum operating frequency of a resulting FSM circuit.
4. Main Idea of the Proposed Method

Assume that we have constructed a partition $\pi_S$ for some Mealy FSM. To eliminate the converter of state codes $K(s_m)$ into state codes $C(s_m)$, we propose to use extended state codes $EC(s_m)$. For a state $s_m \in \{1, \ldots, M\}$, the extended state code determines the state as an element of both sets $S$ and $S^k \in \pi_S$. The number of bits in $EC(s_m)$ is equal to $R_0$ determined by (9). To encode states by ESCs, we use state variables $\tau_r \in \tau$, where $|\tau| = R_0$. If $s_m \in S^k$, then only variables $\tau_r \in \tau^k$ can differ from zero.

Using only ESCs leads to $PE$ Mealy FSMs. There is a structural diagram of $PE$ Mealy FSM shown in Figure 4.

![Figure 4. Structural diagram of PE Mealy FSM.](image)

In $PE$ Mealy FSMs, the blocks of the first level implement partial functions (11) and (12). These functions enter inputs of LUTs from $Block\tau O$. As in the case of $PT$ FSMs, the functions (12) are transformed into FSM outputs. However, the functions (11) are transformed directly into state variables $\tau_r \in \tau$. Due to this, there is no need in the $Block\tau$ used in the case of $PT$ Mealy FSM (Figure 3).

By removing the $Block\tau$, the three-level FSM circuit turns into a two-level one. So, we can expect that $PE$ Mealy FSMs have better performance than equivalent $PT$ Mealy FSMs. Results of our experimental studies, shown in Section 6, have confirmed this assumption.

In this article, we propose a synthesis method for $PE$ Mealy FSMs. The method aims in LUT-based FSMs where an LUT has $NI_{LUT}$ inputs. We assume that an FSM is represented by its state transition table. Furthermore, we assume that there exists a partition $\pi_S$ satisfying the condition (10). There are the following steps in the proposed synthesis method:

1. Constructing the partition $\pi_S$ with the minimum cardinality number $K$.
2. Encoding of FSM states by extended state codes $EC(s_m)$.
3. Creating DST of $PE$ Mealy FSM.
4. Creating tables representing $BlockS1$–$BlockSK$.
5. Creating table representing $Block\tau O$.
6. Constructing SBFs representing blocks of FSM circuit.
7. Implementing $PE$ Mealy FSM circuit with particular LUTs and other resources of FPGA chip.

To create the partition $\pi_S$, we can use the methods from [11,32]. These methods try to minimize the number of LUTs in the resulting FSM circuit. Firstly, the number of shared outputs in the sets $S^k \in \pi_S$ should be minimum one. This reduces the number of LUTs implementing partial outputs in the circuit of $BlockSK$. Furthermore, this allows to reduce the number of interconnections among $BlockSK$ and LUTs of $Block\tau O$. The circuit of block $Block\tau O$ is guaranteed to have a single level of logic if the following condition is true:

$$K \leq NI_{LUT}. \quad (16)$$

If condition (16) is violated, then the circuit of $Block\tau O$ is still a single-level one, if each partial output is generated by no more than $NI_{LUT}$ blocks of the first level of logic. Due to
this, it makes sense to minimize the appearance of shared FSM outputs in different sets \( S^k \in \pi_S \).

Each class \( S^k \in \pi_S \) is characterised by a set \( \delta(S^k) \) including states of transitions from the states \( s_m \in S^k \). The methods [31,32] minimize the appearance of shared states of transition in different classes of the partition \( \pi_S \). This allows minimizing the number of partial input memory functions generated by a particular block of logic. In turn, this minimizes the numbers of LUTs in circuits of all blocks of \( P_T \) FSM circuit. In the case of \( P_E \) FSM, we still use this method.

The difference in the organization of \( P_T \) and \( P_E \) FSMs leads to a change in the method of state assignment. In [32], state are encoded in a way minimizing the number of LUTs in the circuit of LUTs. There is no LUT in \( P_E \) FSMs. We propose to encode states in a way minimizing the number of LUTs generating functions \( (11) \). To do it, we encode the states \( s_m \in \delta(S^k) \) in a way maximizing the number of zeros in the same bits of codes for different states of transition within each set \( \delta(S^k) \).

Consider the following example. There is a set \( \delta(S^3) = \{s_3,s_5\} \). If these states have the codes \( EC(s_3) = 01101 \), \( EC(s_5) = 10011 \), then all five partial input memory functions are generated by BlockS1. To generate them, it is necessary to use five LUTs. If there are codes \( EC(s_3) = 00001 \) and \( EC(s_5) = 00011 \), then only two partial input memory functions should be generated. To do it, only 2 LUTs are necessary. We use this approach to encode states of \( P_E \) Mealy FSMs.

5. Example of Synthesis

If a model of \( P_E \) FSM is used to implement the circuit of some FSM \( A_y \), then we denote this as \( P_E(A_y) \). In this section, we discuss an example of synthesis of \( P_E \) Mealy FSM starting from STT (Table 1). To implement the circuit of Mealy FSM \( P_E(A_0) \), we use LUTs having \( NI_{LUT} = 5 \).

Step 1. Using the approach in [32] gives the partition \( \pi_S = \{S^1,S^2\} \) of the set of states. It includes the classes \( S^1 = \{s_1,s_3,s_4\} \) and \( S^2 = \{s_2,s_5,s_6\} \). There is \( M_1 = M_2 = 3 \) in the discussed case. Using \( (8) \) gives \( R(S^1) = R(S^2) = 2 \). As follows from the analysis of Table 1, the following sets exist for each class \( S^k \in \pi_S \): \( I^1 = \{i_1,i_2,i_3\} \), \( O^1 = \{o_1,o_2,o_4,o_6,o_7\} \), \( \delta(S^1) = \{s_1,s_2,s_3,s_5,s_6\} \) for \( S^1 \); \( I^2 = \{i_4,i_5\} \), \( O^2 = \{o_1,o_3,o_5\} \), \( \delta(S^2) = \{s_1,s_3,s_4,s_6\} \) for \( S^2 \). So, the following relation takes place: \( L_1 = L_2 = 2 \). Because of \( NI_{LUT} = 5 \), the condition \( (10) \) is true for each class of the partition \( \pi_S = \{S^1,S^2\} \).

As we can see, there are no identical elements in the sets \( I^1 \) and \( I^2 \) \( (I^1 \cap I^2 = \emptyset) \). The fewer identical elements in different sets \( I^k \subseteq I \), the fewer connections between the sources of FSM inputs and LUTs of the first level of logic. In our particular case, the absolute minimum of zero is reached.

There is only a single common output \( o_1 \) in sets \( O^1 \) and \( O^2 \). It means that only a single LUT of BlockTO is necessary to generate FSM outputs. All other outputs are generated by LUTs from corresponding blocks of the first level of logic circuit. In general, the fewer identical elements in different sets \( O^k \subseteq O \), the smaller the number of LUTs in the second level of logic.

Step 2. There is \( R(S^1) = R(S^2) = 2 \). From \( (9) \), we can get \( R_0 = 4 \). This gives the set \( \tau = \{\tau_1,\ldots,\tau_4\} \). We can create the following sets for the state encoding: \( \tau^1 = \{\tau_1,\tau_2\} \) and \( \tau^2 = \{\tau_3,\tau_4\} \). If \( s_m \in S^1 \), then \( \tau_3 = \tau_4 = 0 \) in the code \( EC(s_m) \). There is \( \tau_1 = \tau_2 = 0 \) in the code \( EC(s_m) \) for the states \( s_m \in S^2 \). Furthermore, we should maximize the number of zeros in the same bits of codes for different states of transition within each set \( \delta(S^k) \).

In our example, it is possible to encode only states \( s_m \in \delta(S^2) \) in such a way. The outcome of state assignment is shown in Figure 5.
As follows from the Karnaugh map (Figure 5), there is $\tau_3 = 0$ in extended codes of states $s_m \in S.1$. It means that only three LUTs are used for generating input memory functions in BlockS2. Furthermore, we can see that variables $\tau_3$ and $\tau_4$ are insignificant for conjunctions $S_m$ for the codes of states $s_m \in S.2$. This allows diminishing the number of literals in terms (4) up to $R(S^k)$. For example, there is $S_3 = \tau_1 \land \tau_2$. These minimized conjunctions are used in SOPs of functions (11) and (12).

Step 3. The transition from an STT to a DST is executed in the trivial way. The columns $s_m, s_T, I_h, O_h, h$ contain the same symbols in both tables. The state codes are taken from a corresponding Karnaugh map. In the discussed example, the codes are taken from Figure 5. There is a symbol $D_r$ in the row $h$ of the column $D_h$, if there is 1 in the code of state of transition $s_T \in S$ in this row. In the discussed case, the DST of FSM $P_E(A_0)$ is represented by Table 2. This table includes $h = 15$ rows.

### Table 2. DST of $P_E$ FSM $A_0$

| $s_m$ | $EC(s_m)$ | $s_T$ | $EC(s_T)$ | $I_h$ | $O_h$ | $D_h$ | $h$ |
|-------|------------|-------|------------|-------|-------|-------|-----|
| $s_1$ | 0100       | $s_2$ | 0010       | $i_1$ | $o_102$ | $D_3$ | 1   |
|       |            | $s_2$ | 0010       | $i_1^21_2$ | $o_6$ | $D_3$ | 2   |
|       |            | $s_3$ | 1000       | $i_11_2$ | $o_204$ | $D_1$ | 3   |
| $s_2$ | 0010       | $s_3$ | 1000       | $i_4$ | $o_3$ | $D_1D_2$ | 4   |
|       |            | $s_4$ | 1100       | $i_4$ | $o_5$ | $D_1D_2$ | 5   |
| $s_3$ | 1000       | $s_1$ | 0100       | $i_2^21_3$ | $o_406$ | $D_2$ | 6   |
|       |            | $s_5$ | 0011       | $i_2^21_3$ | $o_102$ | $D_3D_4$ | 7   |
|       |            | $s_5$ | 0011       | $i_2^21_3$ | $o_7$ | $D_3D_4$ | 8   |
| $s_4$ | 1100       | $s_2$ | 0010       | $i_2^21_3$ | $o_206$ | $D_3$ | 9   |
|       |            | $s_4$ | 1100       | $i_2^21_3$ | $o_407$ | $D_1D_2$ | 10  |
|       |            | $s_6$ | 0001       | $i_2^21_3$ | $o_1$ | $D_4$ | 11  |
| $s_5$ | 0011       | $s_6$ | 0001       | 1     | $o_3$ | $D_4$ | 12  |
| $s_6$ | 0001       | $s_1$ | 0100       | $i_4$ | – | $D_2$ | 13  |
|       |            | $s_4$ | 1100       | $i_41_5$ | $o_103$ | $D_1D_2$ | 14  |
|       |            | $s_6$ | 0001       | $i_41_5$ | $o_5$ | $D_4$ | 15  |

### Step 4. To construct a table of BlockSk, it is necessary to select the rows of a DST with transitions from the states $s_m \in S^k$. Obviously, this is executed in a trivial way. Applying this approach to Table 2 gives two tables. Table 3 represents the BlockS1; Table 4 represents the BlockS2. There are 9 rows in Table 3 and 6 rows in Table 4. So, together, these tables include all 15 rows of the original DST (Table 2).
Table 3. Table of BlockS1.

| $s_m$ | $EC(s_m)$ | $s_T$ | $EC(s_T)$ | $I_h$ | $O_h$ | $D_h$ | $h$ |
|------|-----------|-------|-----------|------|------|------|----|
| $s_1$ | 0100 | $s_2$ | 0010 | $i_1$ | $o_1$ | $o_2$ | $D_3$ | 1 |
|      |      | $s_2$ | 0010 | $i_2$ | $o_6$ |     | $D_3$ | 2 |
|      |      | $s_3$ | 1000 | $i_1$ | $o_2$ | $o_4$ | $D_1$ | 3 |
| $s_3$ | 0100 | $s_1$ | 0011 | $i_2$ | $o_6$ |     | $D_2$ | 4 |
|      |      | $s_4$ | 0011 | $i_2$ | $o_2$ |     | $D_3D_4$ | 5 |
|      |      | $s_5$ | 0011 | $i_2$ | $o_7$ |     | $D_3D_4$ | 6 |
| $s_4$ | 1100 | $s_2$ | 0010 | $i_2$ | $o_6$ |     | $D_3$ | 7 |
|      |      | $s_3$ | 1100 | $i_2$ | $o_7$ |     | $D_3D_2$ | 8 |
|      |      | $s_6$ | 0001 | $i_2$ | $o_1$ |     | $D_4$ | 9 |

Table 4. Table of BlockS2.

| $s_m$ | $EC(s_m)$ | $s_T$ | $EC(s_T)$ | $I_h$ | $O_h$ | $D_h$ | $h$ |
|------|-----------|-------|-----------|------|------|------|----|
| $s_2$ | 0010 | $s_3$ | 1000 | $i_1$ | $o_3$ |     | $D_1$ | 1 |
|      |      | $s_4$ | 1100 | $i_1$ | $o_5$ |     | $D_3D_2$ | 2 |
|      |      | $s_5$ | 0011 |     |     |     | $D_4$ | 3 |
| $s_6$ | 0001 | $s_1$ | 0100 | $i_1$ |     |     | $D_2$ | 4 |
|      |      | $s_2$ | 1100 | $i_2$ | $o_1$ |     | $D_3D_2$ | 5 |
|      |      | $s_3$ | 0001 | $i_2$ | $o_5$ |     | $D_4$ | 6 |

Step 5. The table of BlockτO includes two columns. The column Function includes all functions from sets of input memory functions and FSM outputs. The column Block is divided by K sub-columns ($S_1, \ldots, SK$). The sub-column $Sk$ corresponds to BlockSk. If a function $f_i \in D \cup O$ is generated by BlockSk, then there is 1 at the intersection of the row $f_i$ and the column $Sk$.

In the discussed case, there are $R_0 + N = 11$ rows in the table of BlockτO. This block is represented by Table 5.

Table 5. Table of BlockτO.

| Function | Block | Function |
|----------|-------|----------|
| D1       | S1    | S2       |
| D2       | S1    | S2       |
| D3       | S1    | S2       |
| D4       | S1    | S2       |
| o1       | S1    | S2       |
| o2       | S1    | S2       |

There is a transparent connection between Tables 3 and 4, on the one hand, and Table 5, on the other hand. For example, there is $D_3 = 0$ in Table 4. So, there is 0 at the intersection of the row $D_3$ and the column $S_2$ of Table 5. Next, there is $o_4 = 1$ in Table 3. So, there is 1 at the intersection of the row $o_4$ and the column $S_1$ of Table 5. All other rows of Table 5 are filled on the base of a similar analysis.
Step 6. Systems (11) and (12) are extracted from tables of BlockS1–BlockSK. The following SBFs can be derived from Table 3 (after minimization):

\[
\begin{align*}
D_1^1 &= F_3 \lor F_8 = \tau_1 \tau_2 \bar{i}_1 \lor \tau_1 \tau_2 \bar{i}_9; \\
D_2^1 &= F_4 \lor F_8 = \tau_1 \tau_2 \bar{i}_2 \lor \tau_1 \tau_2 \bar{i}_3; \\
D_3^1 &= F_1 \lor F_2 \lor F_3 \lor F_6 \lor F_7; \\
D_4^1 &= F_5 \lor [F_6 \lor F_9] = \tau_1 \tau_2 \bar{i}_3 \lor \tau_1 \bar{i}_2. \\
\end{align*}
\]  
(17)

\[
\begin{align*}
o_1^1 &= F_1 \lor F_5 \lor F_9 = \tau_1 \tau_2 \bar{i}_3 \lor \tau_1 \tau_2 \bar{i}_2; \\
o_2^1 &= F_1 \lor F_3 \lor F_7; \\
o_4^1 &= F_3 \lor F_4 \lor F_8; \\
o_6^1 &= F_2 \lor [F_4 \lor F_7] = \tau_1 \tau_2 \bar{i}_2 \lor \tau_1 \bar{i}_3; \\
o_7^1 &= F_6 = \tau_1 \tau_2 \bar{i}_2. \\
\end{align*}
\]  
(18)

The following systems are derived from Table 4:

\[
\begin{align*}
D_1^2 &= [F_1 \lor F_2] \lor F_5 = \tau_3 \tau_4 \lor \tau_3 \tau_4 \bar{i}_5; \\
D_2^2 &= F_2 \lor F_4 \lor F_5; \\
D_4^2 &= D_3^2 = F_3 \lor F_6 = \tau_3 \tau_4 \lor \tau_3 \tau_4 \bar{i}_5. \\
\end{align*}
\]  
(19)

\[
\begin{align*}
o_1^2 &= F_3 = \tau_3 \tau_4 \bar{i}_5; \\
o_2^2 &= F_1 \lor F_3 \lor F_5; \\
o_2^2 &= F_2 \lor F_6 = \tau_3 \tau_4 \lor \tau_3 \tau_4 \bar{i}_5. \\
\end{align*}
\]  
(20)

The following SBF is derived from Table 5:

\[
\begin{align*}
D_1 &= D_1^1 \lor D_2^2; \\
D_2 &= D_1^1 \lor D_2^2; \\
D_3 &= D_3^1; \\
D_4 &= D_1^1 \lor D_2^2; \\
o_1 &= o_2^1 \lor o_2^2; \\
o_2 &= o_2^1; \\
o_3 &= o_5^2; \\
o_4 &= o_4^1; \\
o_5 &= o_5^2; \\
o_6 &= o_6^1; \\
o_7 &= o_7^2. \\
\end{align*}
\]  
(21)

Systems (17)–(21) represent logic circuit of FSM \( P_E(A_0) \). Let us analyse the LUT counts for each level of logic. To do it, we should analyse systems (17)–(21).

As follows from (17) and (18), there are 9 LUTs in the circuit of BlockS1. The LUT count is determined by the number of equations in (17) and (18). From SBFs (19) and (20) follows that there are 6 LUTs in the circuit of BlockS2.

If some equation of SBF (21) has more than a single product term, then the corresponding LUT is included into BlockClk. Otherwise, a corresponding function is generated by some of block of the first level of logic. So, there are 4 LUTs in the circuit of BlockClk.

So, there are \( 9 + 6 + 4 = 19 \) LUTs in the circuit of FSM \( P_E(A_0) \). The pulses Clk and Reset enter CLBs generating functions \( D_i \in D \). The circuit is shown in Figure 6.

**Figure 6.** Logic circuit of FSM \( P_E(A_0) \).
Step 7. To implement the circuit of FSM $P_E(A_0)$, it is necessary to use very complicated methods of technology mapping [26]. This can be done using, for example, the CAD tool Vivado by Xilinx [52]. This package solves the problems of mapping, placement, routing, testing, finding characteristics of an FSM circuit (such as the LUT count, number of CLBs, number of flip-flops, maximum operating frequency, power consumption). We do not show the results of implementation for this particular example. In the next section, we use Vivado to investigate the efficiency of our method compared with some other design methods.

6. Experimental Results

The results of experiments are shown in this section. To conduct experiments, we use benchmark FSMs from the library [58]. The library includes 48 benchmarks represented in the format KISS2. These benchmarks have a wide range of basic characteristics (numbers of states, inputs, and outputs). The characteristics of these benchmarks can be found in many articles and books, for example, in [11,27,37]. These benchmarks are used very often by different researchers to compare area and time characteristics of FSMs obtained using different design methods. The characteristics of the benchmarks are shown in Table 6.

Table 6. Characteristics of Mealy FSM benchmarks.

| Benchmark | L  | N  | $R_S + L$ | $M/R_S$ | H  | Class |
|-----------|----|----|----------|---------|----|-------|
| bbara     | 4  | 2  | 8        | 12/4    | 60 | 1     |
| bbse      | 7  | 7  | 12       | 26/5    | 56 | 1     |
| bbtas     | 2  | 2  | 6        | 9/4     | 24 | 0     |
| beecount  | 3  | 4  | 7        | 10/4    | 28 | 1     |
| cse       | 7  | 7  | 12       | 32/5    | 91 | 1     |
| dk14      | 3  | 5  | 8        | 26/5    | 56 | 1     |
| dk15      | 3  | 5  | 8        | 17/5    | 32 | 1     |
| dk16      | 2  | 3  | 9        | 75/7    | 108| 1     |
| dk17      | 2  | 3  | 6        | 16/4    | 32 | 0     |
| dk27      | 1  | 2  | 5        | 10/4    | 14 | 0     |
| dk512     | 1  | 3  | 6        | 24/5    | 15 | 0     |
| donfile   | 2  | 1  | 7        | 24/5    | 96 | 1     |
| ex1       | 9  | 19 | 16       | 80/7    | 138| 2     |
| ex2       | 2  | 2  | 7        | 25/5    | 72 | 1     |
| ex3       | 2  | 2  | 6        | 14/4    | 36 | 0     |
| ex4       | 6  | 9  | 11       | 18/5    | 21 | 1     |
| ex5       | 2  | 2  | 6        | 16/4    | 32 | 0     |
| ex6       | 5  | 8  | 9        | 14/4    | 34 | 1     |
| ex7       | 2  | 2  | 12       | 17/5    | 36 | 1     |
| keyb      | 7  | 7  | 12       | 22/5    | 170| 1     |
| kirkman   | 12 | 6  | 18       | 48/6    | 370| 2     |
| lion      | 2  | 1  | 5        | 5/3     | 11 | 0     |
| lion9     | 2  | 1  | 6        | 11/4    | 25 | 0     |
| mark1     | 5  | 16 | 10       | 22/5    | 22 | 1     |
| mc        | 3  | 5  | 6        | 8/3     | 10 | 0     |
| modulo12  | 1  | 1  | 5        | 12/4    | 24 | 0     |
| Benchmark | L | N | R₅ + L | M/R₅ | H | Class |
|-----------|---|---|--------|------|---|-------|
| 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| 4     |
| 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     |

To conduct the experiments, we used a personal computer with the following characteristics: CPU: Intel Core i7 6700 K 4.2@4.4 GHz, Memory: 16 GB RAM 2400 MHz CL15. As a platform for implementing FSM circuits we used the Virtex-7 VC709 Evaluation Platform (xc7vx690tffg1761-2) [59]. The FPGA chip of this platform includes LUTs with 6 inputs. We used CAD tool Vivado v2019.1 (64-bit) [52] to execute the technology mapping. The results of experiments are taken from reports produced by Vivado. As the source information for the CAD tool, we used VHDL-based FSM models obtained by the transformation of files in KISS2 format into VHDL codes. The transformation is executed by the CAD tool K2F [11].

We compared area (the LUT count) and time (the maximum operating frequency) characteristics of FSMs based on five different approaches. Three of them are P Mealy FSMs based on: (1) Auto of Vivado (it uses binary state codes); (2) One-hot of Vivado; (3) JEDI. The fourth objects for comparison are P₇-based FSMs [12,32]. We compared these four FSMs with our approach.

It is known [11] that area and time characteristics of LUT-based FSM circuits depend strongly on the relation between numbers of inputs (L) and state variables (R₅), on the one hand, and the number of LUT inputs N_ILUT, on the other hand. Due to this, we have divided the benchmarks into five following classes. The benchmarks belong to class of trivial FSMs (class 0) if R₅ + L ≤ 6. The benchmarks belong to class of simple FSMs (class 1) if R₅ + L ≤ 12. The benchmarks belong to class of average FSMs (class 2) if R₅ + L ≤ 18. The benchmarks belong to class of big FSMs (class 3) if R₅ + L ≤ 24. The benchmarks belong to class of very big FSMs (class 4) if R₅ + L > 24. As research [39] shows, the larger the class number, the bigger the gain from using methods of structural decomposition.
The class 0 includes the benchmarks bbtas, dk17, dk512, ex3, ex5, lion, lion9, mc, modulo12, and shiftreg. The class 1 contains the benchmarks bbara, bbsse, beecount, cse, dk14, dk15, dk16, donfile, ex2, ex4, ex6, ex7, keyb, mark1, opus, s27, s386, s8, and sse. The class 2 consists of the benchmarks ex1, kirkman, planet, planet1, pma, s1, s1488, s1494, s1a, s208, sty, and tma. The class 3 includes the benchmark sand. At last, the benchmarks s420, s510, s820, and s832 create the class 4.

The results of experiments are shown in Tables 7–15. There is the same organization of Tables 7, 9, 11 and 13–15. We marked the table columns by the names of investigated methods. The table rows contain the names of benchmarks. Within each table, the results for the same class are shown in adjacent rows. There are results of summation of values from columns in the row “Total”. The row “Percentage” includes the percentage of summarized characteristics of FSM circuits produced by other methods, respectively, to PE-based FSMs. To point that the model of P FSM is used for methods Auto, One-hot, and JEDI, we name these methods as P-Auto, P-One-hot, and P-JEDI. Tables 8, 10 and 12 show summarized experimental results for different classes of benchmark FSMs. Let us analyse the experimental results taken from reports produced by Vivado.

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

| Benchmark | P-Auto | P-One-Hot | P-JEDI | $P_T$ FSM | $P_E$ FSM |
|-----------|--------|-----------|--------|-----------|-----------|
| **Class 0** | | | | | |
| bbtas | 5 | 5 | 5 | 5 | 5 |
| dk17 | 5 | 12 | 5 | 5 | 5 |
| dk27 | 3 | 5 | 4 | 6 | 4 |
| dk512 | 10 | 10 | 9 | 8 | 9 |
| ex3 | 9 | 9 | 9 | 8 | 9 |
| ex5 | 9 | 9 | 9 | 8 | 10 |
| lion | 2 | 5 | 2 | 2 | 4 |
| lion9 | 6 | 11 | 5 | 5 | 6 |
| mc | 4 | 7 | 4 | 4 | 6 |
| modulo12 | 7 | 7 | 7 | 7 | 7 |
| shiftreg | 2 | 6 | 2 | 4 | 6 |
| **Class 1** | | | | | |
| bbara | 17 | 17 | 10 | 11 | 13 |
| bbsse | 33 | 37 | 24 | 22 | 26 |
| beecount | 19 | 19 | 14 | 12 | 14 |
| cse | 40 | 66 | 36 | 32 | 34 |
| dk14 | 16 | 27 | 10 | 12 | 12 |
| dk15 | 15 | 16 | 12 | 6 | 9 |
| dk16 | 15 | 34 | 12 | 10 | 11 |
| donfile | 31 | 31 | 24 | 19 | 21 |
| ex2 | 9 | 9 | 8 | 8 | 9 |
| ex4 | 15 | 13 | 12 | 10 | 11 |
| ex6 | 24 | 36 | 22 | 20 | 22 |
| ex7 | 4 | 5 | 4 | 4 | 6 |
| keyb | 43 | 61 | 40 | 37 | 38 |
| mark1 | 23 | 23 | 20 | 18 | 20 |
As follows from Table 7, the $P_T$-based FSMs consume the minimum amount of LUTs compared with other investigated approaches. The $P_T$-based FSMs require 7.7% more LUTs than equivalent $P_T$-based FSMs. However, all other FSMs require more LUTs than it is for our approach. Our approach consumes fewer LUTs than it is for P-Auto (25.47% of gain), P-One-hot (46.1% of gain) and P-JEDI-based FSMs (6.67% of gain).

To show a dependence of the gain in LUTs on the class of benchmarks, we have created Table 8. It shows the gain for classes 0, 1 and 2–4.

Table 8. Summarized results for FSM classes (the number of LUTs).

| Class | P-Auto | P-One-Hot | P-JEDI | $P_T$ FSM | $P_E$ FSM | Total Percentage |
|-------|--------|-----------|--------|-----------|-----------|------------------|
| 0     | 62     | 86        | 61     | 62        | 71        | 100.00           |
|       | 87.32  | 121.13    | 85.92  | 87.32     | 100.00    |                  |
| 1     | 406    | 525       | 337    | 304       | 342       | 100.00           |
|       | 118.71 | 153.51    | 98.54  | 88.89     | 100.00    |                  |
| 2–4   | 1340   | 1493      | 1091   | 964       | 1028      | 100.00           |
|       | 130.35 | 145.23    | 106.13 | 93.77     | 100.00    |                  |
Table 9. Experimental results (maximum operating frequency, MHz).

| Benchmark | P-Auto | P-One-Hot | P-JEDI   | P_T FSM | P_E FSM |
|-----------|--------|-----------|----------|---------|---------|
|           |        |           |          |         |         |
| **Class 0** |        |           |          |         |         |
| 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   | 208.17  | 208.17  |
| ex3       | 194.86 | 194.86    | 195.76   | 201.12  | 201.12  |
| ex5       | 180.25 | 180.25    | 181.16   | 182.01  | 182.01  |
| lion      | 202.43 | 204       | 202.35   | 200.18  | 200.18  |
| lion9     | 205.3  | 185.22    | 206.38   | 207.13  | 207.13  |
| mc        | 196.66 | 195.47    | 196.87   | 196.12  | 196.12  |
| modulo12  | 207    | 207       | 207.13   | 208.12  | 208.12  |
| shiftreg  | 262.67 | 263.57    | 276.26   | 256.69  | 256.69  |
| **Class 1** |        |           |          |         |         |
| bbara     | 193.39 | 193.39    | 212.21   | 210.37  | 252.44  |
| bbsse     | 157.06 | 169.12    | 182.34   | 198.65  | 238.38  |
| beecount  | 166.61 | 166.61    | 187.32   | 201.43  | 241.72  |
| cse       | 146.43 | 163.64    | 178.12   | 206.55  | 247.86  |
| dk14      | 191.64 | 172.65    | 193.85   | 186.53  | 223.84  |
| dk15      | 192.53 | 185.36    | 194.87   | 189.14  | 226.97  |
| dk16      | 169.72 | 174.79    | 197.13   | 211.52  | 253.82  |
| donfile   | 184.03 | 184       | 203.65   | 231.63  | 248.19  |
| ex2       | 198.57 | 198.57    | 200.14   | 201.34  | 241.61  |
| ex4       | 180.96 | 177.71    | 192.83   | 197.76  | 237.31  |
| ex6       | 169.57 | 163.8     | 176.59   | 198.65  | 238.35  |
| ex7       | 200.04 | 200.84    | 200.6    | 200.69  | 240.83  |
| keyb      | 156.45 | 143.47    | 168.43   | 187.48  | 224.98  |
| mark1     | 162.39 | 162.39    | 176.18   | 189.58  | 227.47  |
| opus      | 166.2  | 166.2     | 178.32   | 177.84  | 213.4   |
| s27       | 198.73 | 191.5     | 199.13   | 198.76  | 238.53  |
| s386      | 168.15 | 173.46    | 179.15   | 182.63  | 218.87  |
| s8        | 180.02 | 178.95    | 181.23   | 178.32  | 213.65  |
| sse       | 157.06 | 169.12    | 174.63   | 189.64  | 205.41  |
| **Classes 2–4** |        |           |          |         |         |
| ex1       | 150.94 | 139.76    | 176.87   | 212.94  | 276.82  |
| kirkman   | 141.38 | 154       | 156.68   | 174.73  | 227.15  |
| planet    | 132.71 | 132.71    | 187.14   | 193.49  | 251.54  |
| planet1   | 132.71 | 132.71    | 187.14   | 193.49  | 251.54  |
| pma       | 146.18 | 146.18    | 169.83   | 184.45  | 239.83  |
| s1        | 146.41 | 135.85    | 157.16   | 170.19  | 221.47  |
As follows from Table 8, the $P_T$–based FSMs have worse results than equivalent P-Auto FSMs (12.68% of loss on the number of LUTs), P-JEDI FSMs (14.08% of loss on the number of LUTs) and $P_T$-based FSMs (12.68% of loss on the number of LUTs). So, for the class 0, the P-JEDI based FSMs have the minimum amount of LUTs. For the class 1, our approach loses out relative to the other two approaches (11.01% to $P_E$–based FSMs and 1.46% to P-JEDI-based FSMs). However, our approach produces more economical circuits than it is for P-Auto (18.71% of gain) and P-One-hot (53.51% of gain). The $P_T$–based FSMs only lose to the $P_T$-based FSMs (6.23%). So, for the classes 2–4, our approach defeats P-Auto FSMs (30.35% of gain), P-One-hot FSMs (45.23% of gain), and P-JEDI FSMs (6.13% of gain). It means that our approach allows obtaining FSM circuits with a number of LUTs comparable to this number for equivalent P-JEDI- and $P_T$-based FSMs. At the same time, the loss of our approach decreases as the complexity of FSMs increases: the larger the class number, the smaller our loss relative to the best solutions (P-JEDI- and $P_T$-based FSMs).

We thought that our approach would allow us to obtain FSM circuits with higher performance than it is for circuits based on models of either $P$ or $P_T$ FSMs. To test this assumption, we have created Table 9. It includes values of maximum operating frequency measured in megahertz.

As follows from Table 9, the $P_E$–based FSMs have the higher values of maximum operating frequency compared with other investigated FSMs. Our approach provides the following gain: (1) 25.49% compared with P-Auto-based FSMs; (2) 26.09% compared with P-One-hot-based FSMs; (3) 10.06% compared with P-JEDI-based FSMs; (4) 15.9% compared with $P_T$-based FSMs. Our research has shown that the frequency gain depends on the class to which an FSM belongs. This conclusion is supported by data from Table 10.

### Table 9. Cont.

| Benchmark | P-Auto | P-One-Hot | P-JEDI | $P_T$ FSM | $P_E$ FSM |
|-----------|--------|-----------|--------|-----------|-----------|
| s1488     | 138.5  | 131.94    | 157.18 | 187.95    | 244.31    |
| s1494     | 149.39 | 145.75    | 164.34 | 186.22    | 242.05    |
| s1a       | 153.37 | 176.4     | 169.17 | 178.84    | 214.53    |
| s208      | 174.34 | 176.46    | 178.76 | 196.37    | 255.28    |
| styr      | 137.61 | 129.92    | 145.64 | 178.65    | 232.24    |
| tma       | 163.88 | 147.8     | 164.14 | 181.22    | 235.59    |
| sand      | 115.97 | 115.97    | 126.82 | 163.18    | 221.14    |
| s420      | 173.88 | 176.46    | 177.25 | 181.62    | 263.32    |
| s510      | 177.65 | 177.65    | 198.32 | 209.36    | 297.76    |
| s820      | 152    | 153.16    | 176.58 | 192.14    | 268.1     |
| s832      | 145.71 | 153.23    | 173.78 | 192.87    | 274.22    |
| Total     | 8127.08| 8061.22   | 8718.87| 9172.66   | 10,906.96 |
| Percentage, % | 74.51 | 73.91 | 79.94 | 84.10 | 100.00 |

As follows from Table 8, the $P_T$–based FSMs have worse results than equivalent P-Auto FSMs (12.68% of loss on the number of LUTs), P-JEDI FSMs (14.08% of loss on the number of LUTs) and $P_T$-based FSMs (12.68% of loss on the number of LUTs). So, for the class 0, the P-JEDI based FSMs have the minimum amount of LUTs. For the class 1, our approach loses out relative to the other two approaches (11.01% to $P_E$–based FSMs and 1.46% to P-JEDI-based FSMs). However, our approach produces more economical circuits than it is for P-Auto (18.71% of gain) and P-One-hot (53.51% of gain). The $P_T$–based FSMs only lose to the $P_T$-based FSMs (6.23%). So, for the classes 2–4, our approach defeats P-Auto FSMs (30.35% of gain), P-One-hot FSMs (45.23% of gain), and P-JEDI FSMs (6.13% of gain). It means that our approach allows obtaining FSM circuits with a number of LUTs comparable to this number for equivalent P-JEDI- and $P_T$-based FSMs. At the same time, the loss of our approach decreases as the complexity of FSMs increases: the larger the class number, the smaller our loss relative to the best solutions (P-JEDI- and $P_T$-based FSMs).

We thought that our approach would allow us to obtain FSM circuits with higher performance than it is for circuits based on models of either $P$ or $P_T$ FSMs. To test this assumption, we have created Table 9. It includes values of maximum operating frequency measured in megahertz.

As follows from Table 9, the $P_E$–based FSMs have the higher values of maximum operating frequency compared with other investigated FSMs. Our approach provides the following gain: (1) 25.49% compared with P-Auto-based FSMs; (2) 26.09% compared with P-One-hot-based FSMs; (3) 10.06% compared with P-JEDI-based FSMs; (4) 15.9% compared with $P_T$-based FSMs. Our research has shown that the frequency gain depends on the class to which an FSM belongs. This conclusion is supported by data from Table 10.

### Table 10. Summarized results for FSM classes (maximum operating frequency).

| Class | P-Auto | P-One-Hot | P-JEDI | $P_T$ FSM | $P_E$ FSM | Total Percentage |
|-------|--------|-----------|--------|-----------|-----------|------------------|
| 0     | 2254.90| 2199.70   | 2275.35| 2256.44   | 2256.44   | 100.00           |
| 1     | 3339.55| 3335.57   | 3576.72| 3738.51   | 4433.63   | 100.00           |
| 2–4   | 2532.63| 2525.95   | 2866.80| 3177.71   | 4216.89   | 100.00           |
As follows from Table 10, \( P_E \)-based FSMs have the same operating frequency as equivalent \( P_T \) FSMs from the class 0. We can explain this phenomenon by the fact that there is no code transformer al \( P_T \) FSMs. Furthermore, \( P \)-JEDI-based FSMs have slightly higher frequency (0.84%). However, our approach gives a slight advantage over \( P \)-Auto-based FSMs (0.07%) and \( P \)-One-hot-based FSMs (2.51%). It follows from Table 10 that for trivial automata, the method of organizing the FSM circuit is practically irrelevant. The difference in frequency depends mainly on the state encoding method.

Starting from simple FSMs (the class 1), our approach allows producing the fastest circuits. There is the following gain in maximum operating frequency: (1) 24.68% compared with \( P \)-Auto-based FSMs; (2) 24.77% compared with \( P \)-One-hot-based FSMs; (3) 19.33% compared with \( P \)-JEDI-based FSMs; (4) 15.68% compared with \( P_T \)-based FSMs. The gain is even greater for FSMs of classes 2–4. This is the following: (1) 39.94% compared with \( P \)-Auto-based FSMs; (2) 40.1% compared with \( P \)-One-hot-based FSMs; (3) 32.02% compared with \( P \)-JEDI-based FSMs; (4) 24.64% compared with \( P_T \)-based FSMs. As we can see, starting from simple FSMs, the difference in frequency depends mainly on the architecture of FSM.

When comparing different variants of the FSM circuit implementation, an integral estimate is often used, which is equal to the product of the chip area occupied by a circuit by the performance [45]. In the case of LUT-based FSMs, the circuit quality is estimated by the product of the LUT count by the minimum cycle time [45]. Obviously, the time of cycle is inversely to the operating frequency. The lower the value of this product, the better is the quality of the corresponding FSM circuit.

### Table 11. Experimental results (products of LUT counts by propagation times).

| Benchmark | P-Auto | P-One-Hot | P-JEDI | \( P_T \) FSM | \( P_E \) FSM |
|-----------|--------|----------|--------|-------------|-------------|
| **Class 0** |        |          |        |              |             |
| bbtas     | 24.49  | 24.49    | 24.26  | 24.95       | 24.95       |
| dk17      | 25.09  | 71.86    | 25.08  | 25.02       | 25.02       |
| dk27      | 14.56  | 24.76    | 19.59  | 30.51       | 20.34       |
| dk512     | 50.95  | 50.95    | 45.06  | 38.43       | 43.23       |
| ex3       | 46.19  | 46.19    | 45.97  | 39.78       | 44.75       |
| ex5       | 49.93  | 49.93    | 49.68  | 43.95       | 54.94       |
| lion      | 9.88   | 24.51    | 9.88   | 9.99        | 19.98       |
| lion9     | 29.23  | 59.39    | 24.23  | 24.14       | 28.97       |
| mc        | 20.34  | 35.81    | 20.32  | 20.40       | 30.59       |
| modulo12  | 33.82  | 33.82    | 33.80  | 33.63       | 33.63       |
| shiftnreg | 7.61   | 22.76    | 7.24   | 15.58       | 23.37       |
| **Class 1** |        |          |        |              |             |
| bbarar    | 87.91  | 87.91    | 47.12  | 52.29       | 51.50       |
| bsse      | 210.11 | 218.78   | 131.62 | 110.75      | 109.07      |
| beecount  | 114.04 | 114.04   | 74.74  | 59.57       | 57.92       |
| cse       | 273.17 | 403.32   | 202.11 | 154.93      | 137.17      |
| dk14      | 83.49  | 156.39   | 51.59  | 64.33       | 53.61       |
| dk15      | 77.91  | 86.32    | 61.58  | 31.72       | 39.65       |
| dk16      | 88.38  | 194.52   | 60.87  | 47.28       | 43.34       |
| donfile   | 168.45 | 168.48   | 117.85 | 82.03       | 84.61       |
| ex2       | 45.32  | 45.32    | 39.97  | 39.73       | 37.25       |
The results of the comparison relative to this estimate are shown in Table 11. The time of cycle is represented in nanoseconds. So, the data inside Table 11 are represented as “number of LUTs × nsecs”.

As follows from Table 11, our approach leads to FSM circuits with better integral estimates than it is for other investigated methods. There is the following gain in the integral estimates: (1) 101.63% compared with P-Auto-based FSMs; (2) 133.62% compared with P-One-hot-based FSMs; (3) 45.54% compared with P-JEDI-based FSMs; (4) 16.89% compared with $P_T$-based FSMs. This is connected with the fact that our approach produces two-level FSM circuits. At the same time, a circuit for each function for any level of logic requires only a single LUT.

| Benchmark | P-Auto | P-One-Hot | P-JEDI | $P_T$ FSM | $P_T$ FSM |
|-----------|--------|-----------|--------|-----------|-----------|
| ex4       | 82.89  | 73.15     | 62.23  | 50.57     | 46.35     |
| ex6       | 141.53 | 219.78    | 124.58 | 100.68    | 92.30     |
| ex7       | 20.00  | 24.90     | 19.94  | 19.93     | 24.91     |
| keyb      | 274.85 | 425.18    | 237.49 | 197.35    | 168.90    |
| mark1     | 141.63 | 141.63    | 113.52 | 94.95     | 87.92     |
| opus      | 168.47 | 168.47    | 123.37 | 129.33    | 117.15    |
| s27       | 30.19  | 93.99     | 30.13  | 30.19     | 33.54     |
| s386      | 154.62 | 224.84    | 122.80 | 98.56     | 100.52    |
| s8        | 49.99  | 50.29     | 49.66  | 56.08     | 56.17     |
| sse       | 210.11 | 218.78    | 171.79 | 137.10    | 141.18    |

**Classes 2–4**

| Benchmark | P-Auto | P-One-Hot | P-JEDI | $P_T$ FSM | $P_T$ FSM |
|-----------|--------|-----------|--------|-----------|-----------|
| ex1       | 463.76 | 529.48    | 299.66 | 197.24    | 158.95    |
| kirkman   | 297.07 | 376.62    | 248.91 | 200.31    | 162.89    |
| planet    | 987.11 | 987.11    | 470.24 | 413.46    | 345.87    |
| planet1   | 987.11 | 987.11    | 470.24 | 413.46    | 345.87    |
| pma       | 643.04 | 643.04    | 506.39 | 422.88    | 333.57    |
| s1        | 443.96 | 728.74    | 388.14 | 334.92    | 275.43    |
| s1488     | 895.31 | 992.88    | 687.11 | 489.49    | 392.94    |
| s1494     | 843.43 | 905.66    | 669.34 | 504.78    | 388.35    |
| s1a       | 319.49 | 459.18    | 254.18 | 229.26    | 219.08    |
| s208      | 68.83  | 175.68    | 55.94  | 45.83     | 43.09     |
| styr      | 675.82 | 923.65    | 556.17 | 408.62    | 340.17    |
| tma       | 274.59 | 263.87    | 237.60 | 182.10    | 152.81    |
| sand      | 1138.23| 1138.23   | 898.91 | 618.95    | 488.38    |
| s420      | 57.51  | 175.68    | 50.78  | 44.05     | 37.98     |
| s510      | 270.19 | 270.19    | 161.36 | 138.52    | 104.11    |
| s820      | 578.95 | 535.39    | 385.09 | 301.86    | 220.07    |
| s832      | 549.04 | 515.56    | 356.77 | 279.98    | 222.45    |
| Total     | 12,228.61| 14,168.64| 8844.90| 7089.45| 6064.86 |
| Percentage, % | 201.63 | 233.62 | 145.84 | 116.89 | 100.00 |
To check the dependence of the integral estimate on the value of $R_S + L$, we split Table 11 and built Table 12, which shows integral estimates for FSMs from classes 0, 1 and 2–4.

### Table 12. Summarized results for FSM classes (products of LUT counts by propagation times).

| Class | P-Auto | P-One-Hot | P-JEDI | $P_T$ FSM | Total Percentage |
|-------|--------|-----------|--------|-----------|-----------------|
| 0     | 312.09 | 444.47    | 305.10 | 306.38    | 349.79          |
|       | 89.22  | 127.07    | 87.22  | 87.59     | 100.00          |
| 1     | 2423.08| 3116.09   | 1842.98| 1557.37   | 1483.07         |
|       | 163.38 | 210.11    | 124.27 | 105.01    | 100.00          |
| 2–4   | 9493.45| 10,608.08 | 6696.83| 5225.70   | 4232.00         |
|       | 224.33 | 250.66    | 158.24 | 123.48    | 100.00          |

In the case of class 0 (Table 12), the circuits of $P_E$-based FSMs have the worst values of integral estimates compared with the circuits produced by P-Auto, P-JEDI, and $P_T$ FSMs. There is the following loss: (1) 10.78% compared with P-Auto; (2) 12.78% compared with P-JEDI and (3) 12.41% compared with $P_T$ FSMs. So, the loss is approximately the same relative to these three models. However, $P_E$–based FSMs have a gain of 27.07% compared with LUT-based P-One-hot FSMs.

### Table 13. Comparison with our previous works (the number of LUTs).

| Benchmark    | [12] | [32] | [38] | [39] | $P_E$ FSM |
|--------------|------|------|------|------|-----------|
| Class 0      |      |      |      |      |           |
| bbtas        | 5    | 8    | 8    | 9    | 5         |
| dk17         | 5    | 9    | 8    | 10   | 5         |
| dk27         | 6    | 8    | 7    | 9    | 4         |
| dk512        | 8    | 12   | 12   | 14   | 9         |
| ex3          | 8    | 12   | 11   | 14   | 9         |
| ex5          | 8    | 13   | 10   | 12   | 10        |
| lion         | 2    | 6    | 6    | 8    | 4         |
| lion9        | 5    | 7    | 8    | 10   | 6         |
| mc           | 4    | 8    | 6    | 8    | 6         |
| modulo12     | 7    | 10   | 9    | 11   | 7         |
| shifreg      | 4    | 6    | 4    | 6    | 6         |
| Class 1      |      |      |      |      |           |
| bbara        | 11   | 11   | 10   | 14   | 13        |
| bbssse       | 22   | 23   | 26   | 29   | 26        |
| beccount     | 12   | 13   | 14   | 16   | 14        |
| cse          | 32   | 35   | 33   | 35   | 34        |
| dk14         | 12   | 12   | 12   | 14   | 12        |
| dk15         | 6    | 11   | 6    | 11   | 9         |
| dk16         | 10   | 12   | 11   | 13   | 11        |
| donfile      | 19   | 19   | 21   | 24   | 21        |
| ex2          | 8    | 10   | 8    | 10   | 9         |
Starting from simple FSMs (the class 1), our approach allows producing the circuits with significantly better integral estimates as their counterparts. There is the following gain in values of area-time products: (1) 63.38% compared with P-Auto-based FSMs; (2) 110.11% compared with P-One-hot-based FSMs; (3) 24.27% compared with P-JEDI-based FSMs; (4) 5.01% compared with $P_T$-based FSMs. The gain is even greater for FSMs of classes 2–4. This is the following: (1) 124.33% compared with P-Auto-based FSMs; (2) 150.66% compared with P-One-hot-based FSMs; (3) 58.24% compared with P-JEDI-based FSMs; (4) 23.48% compared with $P_T$-based FSMs. As we can see, starting from simple FSMs, the difference in the values of area-time products depends mainly on the architecture of FSM. Due to this, the circuits based on $P_T$ FSMs also benefit in comparison to P-Auto, P-One-hot, and P-JEDI FSMs.
As follows from results of experiments, using the model of $P_E$ FSMs allows obtaining FSM circuits with higher operating frequency and less values of integral estimates than they are for other investigated models. Winning starts already from simple FSMs for whom the following relation takes place: $(L + R_S) - N_{LUT} > 0$. The gain from using our method increases as the difference between the number of FSM inputs and state variables, on one side, and the number of inputs of LUT, on the other side, increases.

In our previous papers [12, 32, 38, 39], we have proposed various methods for improving characteristics of LUT-based Mealy FSMs. All these methods lead to three-level FSM circuits. In [12], there is proposed a synthesis method based on the twofold state assignment and one-hot encoding of outputs. In [32], there is proposed a synthesis method based on the twofold state assignment and encoding of collections of outputs. These methods are also shown in our book [11]. In [38], there is proposed a synthesis method based on the replacement of FSM inputs and encoding of collections of outputs. In [39], there is proposed a synthesis method based on the transformation of codes of collections of outputs into state codes.

To compare our new results with results [12, 32, 38, 39], we have created three additional tables. Table 13 includes the experimental results for the number of LUTs. Table 14 shows the results for the maximum operating frequency. Table 15 contains the products of LUT counts by propagation times.

**Table 14. Comparison with our previous works (maximum operating frequency, MHz).**

| Benchmark | [12] | [32] | [38] | [39] | $P_E$ FSM |
|-----------|------|------|------|------|----------|
| **Class 0** | | | | | |
| bbtas | 200.38 | 198.18 | 194.43 | 201.47 | 200.38 |
| dk17 | 199.87 | 147.21 | 147.22 | 172.99 | 199.87 |
| dk27 | 196.65 | 184.61 | 181.73 | 190.32 | 196.65 |
| dk512 | 208.17 | 175.02 | 175.63 | 187.45 | 208.17 |
| ex3 | 201.12 | 176.95 | 174.44 | 187.26 | 201.12 |
| ex5 | 182.01 | 169.39 | 162.56 | 162.56 | 182.01 |
| lion | 200.18 | 188.13 | 185.74 | 195.73 | 200.18 |
| lion9 | 207.13 | 172.57 | 167.28 | 183.45 | 207.13 |
| mc | 196.12 | 177.62 | 178.02 | 182.95 | 196.12 |
| modulo12 | 208.12 | 190.99 | 189.70 | 201.74 | 208.12 |
| shiftreg | 256.69 | 251.75 | 248.79 | 253.72 | 256.69 |
| **Class 1** | | | | | |
| bbara | 210.37 | 184.15 | 183.32 | 210.21 | 252.44 |
| bbsse | 198.65 | 162.62 | 159.24 | 193.43 | 238.38 |
| beecount | 201.43 | 156.44 | 156.72 | 194.47 | 241.72 |
| cse | 206.56 | 157.87 | 153.24 | 182.62 | 247.86 |
| dk14 | 186.53 | 161.11 | 162.78 | 201.39 | 223.84 |
| dk15 | 189.14 | 177.38 | 175.42 | 206.74 | 226.97 |
| dk16 | 211.52 | 165.78 | 164.16 | 199.14 | 253.82 |
| donfile | 231.63 | 179.63 | 174.28 | 206.83 | 248.19 |
| ex2 | 201.34 | 192.45 | 188.95 | 196.58 | 241.61 |
| ex4 | 197.76 | 169.77 | 168.39 | 196.18 | 237.31 |
As follows from Table 13, the circuits of [38]-based FSMs use a minimal number of LUTs compared to other investigated methods. The $P_E$-based FSMs require 7.7% more LUTs than equivalent [12]-based FSMs, 1.32% more LUTs than [32]-based FSMs, and 8.54% more LUTs than [38]-based FSMs. Our approach consumes fewer LUTs than it is [39]-based FSMs (0.49% of gain).

As follows from Table 14, our approach allows obtaining FSM circuits with the highest operating frequency. There is the following gain in operating frequency: (1) 15.9% compared with [12]-based FSMs; (2) 26.43% compared with [32]-based FSMs; (3) 27.81% compared with [38]-based FSMs; (4) 17.44% compared with [39]-based FSMs.

### Table 14. Cont.

| Benchmark | [12]    | [32]    | [38]    | [39]    | $P_E$ FSM |
|-----------|---------|---------|---------|---------|-----------|
| ex6       | 198.65  | 170.55  | 156.42  | 187.53  | 238.35    |
| ex7       | 200.69  | 199.19  | 191.43  | 204.16  | 240.83    |
| keyb      | 187.48  | 140.41  | 136.49  | 178.59  | 224.98    |
| mark1     | 189.58  | 157.61  | 153.48  | 182.37  | 227.47    |
| opus      | 177.84  | 158.49  | 157.42  | 186.34  | 213.40    |
| s27       | 198.76  | 187.47  | 185.15  | 201.26  | 238.53    |
| s386      | 182.63  | 167.92  | 164.65  | 192.34  | 218.87    |
| s8        | 178.32  | 171.46  | 168.32  | 191.32  | 213.65    |
| sse       | 189.64  | 165.31  | 158.14  | 171.18  | 205.41    |

| Classes 2–4 |
|--------------|
| ex1          | 212.93  | 167.20  | 164.32  | 180.72  | 276.82    |
| kirkman     | 174.73  | 157.37  | 155.36  | 184.62  | 227.15    |
| planet      | 193.49  | 176.91  | 174.68  | 212.45  | 251.54    |
| planet1     | 193.49  | 175.83  | 173.29  | 212.45  | 251.54    |
| pma         | 184.45  | 162.11  | 156.12  | 192.43  | 239.83    |
| sl          | 170.19  | 157.64  | 145.32  | 145.32  | 221.47    |
| s1488       | 187.95  | 142.77  | 141.27  | 182.14  | 244.31    |
| s1494       | 186.22  | 156.43  | 155.63  | 186.49  | 242.05    |
| s1a         | 178.84  | 168.76  | 166.36  | 188.92  | 214.53    |
| s208        | 196.37  | 168.33  | 166.42  | 192.15  | 255.28    |
| styr        | 178.65  | 120.14  | 118.02  | 164.52  | 232.24    |
| tma         | 181.22  | 139.34  | 137.48  | 182.72  | 235.59    |
| sand        | 163.18  | 127.72  | 120.07  | 143.14  | 221.14    |
| s420        | 181.62  | 187.21  | 186.35  | 218.62  | 263.32    |
| s510        | 209.36  | 201.54  | 199.05  | 221.19  | 297.76    |
| s820        | 192.14  | 176.99  | 175.69  | 195.73  | 268.10    |
| s832        | 192.87  | 179.83  | 174.39  | 199.18  | 274.22    |

| Total       | 9172.65 | 8024.15 | 7917.10 | 9005.11 | 10,906.96 |

| Percentage, % | 84.10 | 73.57 | 72.19 | 82.56 | 100.00 |
Table 15. Comparison with our previous works (products of LUT counts by propagation times).

| Benchmark | [12] | [32] | [38] | [39] | PC FSM |
|-----------|------|------|------|------|--------|
| **Class 0** |      |      |      |      |        |
| bbtas     | 24.95| 40.37| 41.15| 44.67| 24.95  |
| dk17      | 25.02| 61.14| 54.34| 57.81| 25.02  |
| dk27      | 30.51| 43.33| 38.52| 47.29| 20.34  |
| dk512     | 38.43| 68.56| 68.33| 74.69| 43.23  |
| ex3       | 39.78| 67.82| 63.06| 74.76| 44.75  |
| ex5       | 43.95| 76.75| 61.52| 73.82| 54.94  |
| lion      | 9.99 | 31.89| 32.30| 40.87| 19.98  |
| lion9     | 24.14| 40.56| 47.82| 54.51| 28.97  |
| mc        | 20.40| 45.04| 33.70| 43.73| 30.59  |
| modulo12  | 33.63| 52.36| 47.44| 54.53| 33.63  |
| shiftreg  | 15.58| 23.83| 16.08| 23.65| 23.37  |
| **Class 1** |      |      |      |      |        |
| bbbara    | 52.29| 59.73| 54.55| 66.60| 51.50  |
| bbsse     | 110.75| 141.43| 163.28| 149.93| 109.07|
| beecount  | 59.57| 83.10| 89.33| 82.27| 57.92  |
| cse       | 154.92| 221.70| 215.35| 191.65| 137.17|
| dk14      | 64.33| 74.48| 73.72| 69.52| 53.61  |
| dk15      | 31.72| 62.01| 34.20| 53.21| 39.65  |
| dk16      | 47.28| 72.39| 67.01| 65.28| 43.34  |
| donfile   | 82.03| 105.77| 120.50| 116.04| 84.61  |
| ex2       | 39.73| 51.96| 42.34| 50.87| 37.25  |
| ex4       | 50.57| 76.57| 65.32| 66.27| 46.35  |
| ex6       | 100.68| 140.72| 134.25| 122.65| 92.30  |
| ex7       | 19.93| 40.16| 31.34| 39.18| 24.91  |
| keyb      | 197.35| 299.12| 271.08| 223.98| 168.90|
| mark1     | 94.95| 145.93| 123.79| 115.15| 87.92  |
| opus      | 129.33| 138.81| 133.40| 123.43| 117.15|
| s27       | 30.19| 32.01| 32.41| 39.75| 33.54  |
| s386      | 98.56| 142.93| 121.47| 114.38| 100.52|
| s8        | 56.08| 46.66| 53.47| 57.50| 56.17  |
| sse       | 137.10| 157.28| 164.41| 169.41| 141.18|
| **Classes 2–4** |      |      |      |      |        |
| ex1       | 197.25| 251.20| 243.43| 243.47| 158.95|
| kirkman   | 200.31| 260.53| 212.41| 189.58| 162.89|
| planet    | 413.46| 452.21| 446.53| 385.97| 345.87|
| planet1   | 413.46| 454.98| 450.11| 385.97| 345.87|
| pma       | 422.88| 456.48| 461.18| 394.95| 333.57|
| s1        | 334.92| 342.55| 371.59| 399.12| 275.43|
As follows from Table 15, the circuits of $P_E$-based FSMs have the best values of area-time products. There is the following gain: (1) 16.89% compared with $[12]$-based FSMs; (2) 47.89% compared with $[32]$-based FSMs; (3) 40.87% compared with $[38]$-based FSMs; (4) 29.88% compared with $[39]$-based FSMs.

We have proposed the method based on extended state codes to improve the time characteristics of $P_T$ FSMs. Note that the gain in frequency is accompanied by a slight increase in the number of LUTs compared with equivalent $P_T$ FSMs. We think that our approach can be used instead of $P_T$ FSMs if the performance is the main criterion for the optimality of LUT-based FSM circuits.

### 7. Conclusions

Modern FPGAs have up to 7 billion transistors [13]. It means that a very complex digital system may be implemented using a single FPGA chip. The complexity of the implemented systems is constantly increasing, but the number of LUT inputs remains very small. As research [30,31] states, there is no sense in having LUTs with more than 6 inputs. If an FSM circuit is represented by functions for which the condition (5) is violated, then the technology mapping is based on applying various methods of functional decomposition. In turn, this leads to multi-level LUT-based FSM circuits having complicated systems of interconnections.

The characteristics of LUT-based FSM circuits may be improved using various methods of structural decomposition [11]. Very often, FSM circuits based on the structural decomposition have much better characteristics compared with their counterparts based on the functional decomposition [11,12,38,39]. Our research [12] shows that LUT-based Mealy FSM circuits with the twofold state assignment have better characteristics (fewer LUTs and lower power consumption) than their counterparts based on functional decomposition. However, to apply this approach, it is necessary to create the extended state codes. It leads to using a block of state code transformer adding some delay in the cycle time.

In our current article, we propose to use only extended state codes for the state assignment. As a result, we propose a structural diagram and the design method of $P_E$ Mealy FSMs. The elimination of code transformer allows increasing the maximum operating frequency in comparison with $P_T$-based FSMs. In $P_E$ Mealy FSMs, outputs $o_n \in O$ are produced simultaneously with functions $D_r \in D$. As a result, we achieved an increase in operating frequency (up to 23.48%) accompanied by a small increase (up to 12.68%) in the FPGA resources used.

| Benchmark | $[12]$ | $[32]$ | $[38]$ | $[39]$ | $P_E$ FSM Total |
|-----------|--------|--------|--------|--------|----------------|
| s1488     | 489.49 | 644.39 | 630.00 | 510.60 | 392.94         |
| s1494     | 504.78 | 594.52 | 578.29 | 504.05 | 388.35         |
| s1a       | 229.26 | 248.87 | 228.42 | 222.32 | 219.08         |
| s208      | 45.83  | 65.35  | 54.08  | 57.25  | 43.09          |
| styr      | 408.62 | 674.21 | 593.12 | 474.11 | 340.17         |
| tma       | 182.10 | 222.48 | 218.21 | 186.08 | 152.81         |
| sand      | 618.95 | 782.96 | 824.52 | 719.58 | 488.38         |
| s420      | 44.05  | 53.42  | 42.93  | 45.74  | 37.98          |
| s510      | 138.52 | 153.82 | 110.52 | 103.98 | 104.11         |
| s820      | 301.86 | 333.35 | 295.98 | 286.11 | 220.07         |
| s832      | 279.98 | 333.65 | 286.71 | 261.07 | 222.45         |
| Total     | 7089.45| 8969.40| 8543.53| 7877.31| 6064.86        |
| Percentage, % | 116.89 | 147.89 | 140.87 | 129.88 | 100.00         |
The results of experiments show that the performance gain increases as the complexity of an FSM (the number of FSM inputs and state variables) increases. At the same time, the increase in the FSM complexity leads to a decrease in the loss in the number of LUTs. Furthermore, our approach provides better area-time products starting from FSMs for which the total number of inputs and state variables exceeds twice the number of inputs $N_{I_{LUT}}$.

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

**Funding:** This research received no external funding.

**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:

- CLB: configurable logic block
- DST: direct structure table
- ESC: extended state code
- FD: functional decomposition
- FSM: finite state machine
- FPGA: field-programmable gate array
- IMP: input memory function
- LUT: look-up table
- SBF: systems of Boolean functions
- SD: structural decomposition
- SOP: sum-of-products
- SRG: state register
- STG: state transition graph
- STT: state transition table
- $N_{I_{LUT}}$: number of LUT inputs
- $I = \{i_1, \ldots, i_L\}$: set of FSM inputs
- $O = \{o_1, \ldots, o_N\}$: set of FSM outputs
- $S = \{s_1, \ldots, s_M\}$: set of FSM states
- $L$: number of inputs
- $N$: number of outputs
- $M$: number of states
- $H$: number of interstate transitions
- $K(s_m)$: binary code of state $s_m \in S$
- $R_s$: number of state variables in $K(s_m)$
- $T = \{T_1, \ldots, T_{R_s}\}$: set of state variables
- $D = \{D_1, \ldots, D_{R_s}\}$: set of input memory functions
- $F_h$: product term corresponding to $h$-th row of DST
- $s_m$: conjunction of state variables corresponding to code of state $s_m \in S$
- $N L(f_j)$: number of literals in SOP of function $f_j$
- $N(F)$: number of additional functions different from $f_j \in D \cup O$
- $F$: set of additional functions
- $\pi_S = \{S^1, \ldots, S^K\}$: partition of the set of FSM states
- $S^k \in \pi_S$: class number $k$ of partition $\pi_S = \{S^1, \ldots, S^K\}$
- $C(s_m)$: code of state as an element of a class $S^k$
- $R(S^k)$: number of bits necessary to encode states from a class $S^k$
- $L_k$: number of inputs determining transitions from states of class $S^k$
- $\tau_r \in \pi^k$: state variables encoding states $s_m \in S^k$
References

1. Sklyarov, V.; Skliarova, I.; Barkalov, A.; Titarenko, L. Synthesis and Optimization of FPGA-Based Systems; Springer: Berlin, Germany, 2014.

2. Branco, S.; Ferreira, A.G.; Cabral, J. Machine Learning in Resource-Scarce Embedded Systems, FPGAs, and End-Devices: A Survey. Electronics 2019, 8, 1289. [CrossRef]

3. Zajać, W.; Andrzejewski, G.; Krzywicki, K.; Królakowski, T. Finite State Machine Based Modelling of Discrete Control Algorithm in LAD Diagram Language with Use of New Generation Engineering Software. Proc. Comput. Sci. 2019, 159, 2560–2569. [CrossRef]

4. Micheli, G.D. Synthesis and Optimization of Digital Circuits; McGraw–Hill: Cambridge, MA, USA, 1994.

5. Krzywicki, K.; Barkalov, A.; Andrzejewski, G.; Titarenko, L.; Kołopięczyk, M. SoC research and development platform for distributed embedded systems. Przegląd Elektrotechniczny 2016, 92, 262–265. [CrossRef]

6. Czerwinski, R.; Kania, D. Finite State Machine Logic Synthesis for Complex Programmable Logic Devices; Springer: Berlin/Heidelberg, Germany, 2013.

7. Andrzejewski, G.; Zajać, W.; Krzywicki, K.; Królakowski, T. On some aspects of Concurrent Control Processes Modelling and Implementation in LAD Diagram Language With Use of New Generation Engineering Software. Proc. Comput. Sci. 2020, 176, 2173–2183. [CrossRef]

8. El-Maleh, A.H. A Probabilistic Tabu Search State Assignment Algorithm for Area and Power Optimization of Sequential Circuits. Arab. J. Sci. Eng. 2020, 45, 6273–6285. [CrossRef]

9. Skorupski, M. Analysis of Influence of the State Assignment on Area of Microprogram Control Units. Master’s Thesis, University of Zielona Gora, Zielona Gora, Poland, 2020.

10. Gąski, G.D.; Abdi, S.; Gerstlauer, A.; Schirner, G. Embedded System Design: Modeling, Synthesis and Verification; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2009.

11. Barkalov, A.; Titarenko, L.; Mielcarek, K.; Chmielowski, S. Logic Synthesis for FPGA-Based Control Units—Structural Decomposition in Logic Design; Springer: Berlin/Heidelberg, Germany, 2020.

12. Barkalov, A.; Titarenko, L.; Mielcarek, K. Improving characteristics of LUT-based Mealy FSMs. Int. J. Appl. Math. Comput. Sci. 2020, 30, 745–759.

13. Trimberger, S.M. Field-Programmable Gate Array Technology; Springer Science & Business Media: Berlin, Germany, 2012.

14. Altera. Cyclone IV Device Handbook. Available online: http://www.altera.com/literature/hb/cyclone-iv/cyclone4-handbook.pdf (accessed on 15 February 2021).

15. Xilinx FPGAs. Available online: https://www.xilinx.com/products/silicon-devices/fpga.html (accessed on 15 February 2021).

16. Wang, Z.; Tang, Q.; Guo, B.; Wei, J.-B.; Wang, L. Resource Partitioning and Application Scheduling with Module Merging on Dynamically and Partially Reconfigurable FPGAs. Electronics 2020, 9, 1461. [CrossRef]

17. Zhang, F.; Guo, C.; Zhang, S.; Chen, L.; Li, X.; Sun, H.; Meng, Y.; Chen, Q. Research on Hex Programmable Interconnect Points Test in Island-Style FPGA. Electronics 2020, 9, 2177. [CrossRef]

18. Ruiz-Rosero, J.; Ramirez-Gonzalez, G.; Khanna, R. Field Programmable Gate Array Applications—A Scientometric Review. Computation 2019, 7, 63. [CrossRef]

19. Minns, P.; Elliot, I. FSM-Based Digital Design Using Verilog HDL; JohnWiley and Sons: Hoboken, NJ, USA, 2008.

20. Grout, I. Digital Systems Design with FPGAs and CPLDs; Elsevier Science: Amsterdam, The Netherlands, 2011.

21. Intel FPGAs and Programmable Devices. Available online: https://www.intel.pl/content/www/pl/pl/products/programmable.html (accessed on 15 February 2021).

22. Kuo, I.; Tessier, R.; Rose, J. FPGA architecture: Survey and challenges—Found trends. Electr. Des. Autom. 2008, 2, 135–253.

23. Scholl, C. Functional Decomposition with Application to FPGA Synthesis; Kluwer Academic Publishers: Boston, MA, USA, 2001.

24. Kubica, M.; Kania, D. Technology mapping oriented to adaptive logic modules. Bull. Pol. Acad. Sci. 2019, 67, 947–956.

25. Machado, L.; Cortadella, J. Support-reducing decomposition for FPGA mapping. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 2018, 39, 213–224. [CrossRef]

26. Kubica, M.; Kania, D. Decomposition of multi-level functions oriented to configurability of logic blocks. Bull. Pol. Acad. Sci. 2017, 67, 317–331.

27. 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 (FPGA’18), Monterey, CA, USA, 25–27 February 2018; p. 6. [CrossRef]

28. Rawski, M.; Luba, T.; Jachna, Z.; Tomaszewicz, P. The Influence of Functional Decomposition Onmodern Digital Design Process. In Design of Embedded Control Systems; Springer: Boston, MA, USA, 2005; pp. 193–203.
29. Mishchenko, A.; Brayton, R.; Jiang, J.H.R.; Jang, S. Scalable do not-care-based logic optimization and resynthesis. ACM Trans. Reconfig. Technol. Syst. TRETs 2011, 4, 1–23.

30. Salauyouv, V.; Ostapczuk, M. State Assignment of Finite-State Machines by Using the Values of Output Variables. In Theory and Applications of Dependable Computer Systems. DepCoS-RELCOMEX 2020; Advances in Intelligent Systems and Computing; Zamojski, W., Mazurkiewicz, J., Sugier, J., Walkowiak, T., Kacprzyk, J., Eds.; Springer: Cham, Switzerland, 2020; Volume 1173, pp. 543–553.

31. Kilts, S. Advanced FPGA Design: Architecture, Implementation, and Optimization; Wiley-IEEE Press: Hoboken, NJ, USA, 2007.

32. Barkalov, O.; Titarenko, L.; Mielcarek, K. Hardware reduction for LUT-based Mealy FSMs. Int. J. Appl. Math. Comput. Sci. 2018, 28, 595–607. [CrossRef]

33. Baranov, S.I. Logic and System Design of Digital Systems; TUT Press: Tallinn, Estonia, 2008.

34. Sklarova, D.; Sklarov, V.A.; Sudnitsyn, A. Design of FPGA-Based Circuits Using Hierarchical Finite State Machines; TUT Press: Tallinn, Estonia, 2012.

35. Mishchenko, A.; Chattarejee, S.; Brayton, R. Improvements to technology mapping for LUT-based FPGAs. IEEE Trans. CAD 2006, 27, 240–253.

36. Kubicza, 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]

37. Baranov, A.; Titarenko, L.; Krzywicki, K. Reducing LUT Count for FPGA-Based Mealy FSMs. Appl. Sci. 2020, 10, 5115. [CrossRef]

38. Baranov, A.; Titarenko, L.; Krzywicki, K.; Saburova, S. Improving the Characteristics of Multi-Level LUT-Based Mealy FSMs. Electronics 2020, 9, 1859. [CrossRef]

39. Baranov, S. Logic Synthesis of Control Automata; Kluwer Academic Publishers: Dordrecht, The Netherlands, 1994.

40. Opara, A.; Kubica, M.; Kania, D. Strategy of Logic Synthesis using MTBDD dedicated to FPGA. Integr. VLSI J. 2018, 62, 142–158. [CrossRef]

41. Klimovich, A.S.; Solovev, V.V. Minimization of mealy finite-state machines by internal states gluing.

42. Baranov, S. Synthesis of Control Automaton. In Logic Synthesis for Control Automata; Springer: Berlin/Heidelberg, Germany, 2000; pp. 718–727.

43. De Micheli, G.; Brayton, R.K.; Sangiovanni-Vincentelli, A. Optimal state assignment for finite state machines. IEEE Trans. CAD 1985, 4, 269–285. [CrossRef]

44. El-Maleh, A.H. A probabilistic pairwise swap search state assignment algorithm for sequential circuit optimization. Integr. VLSI J. 2017, 56, 32–43. [CrossRef]

45. Islam, M.M.; Hessain, M.S.; Shahjalal, M.D.; 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]

46. Benini, L.; De Micheli, G. State assignment for low power dissipation. IEEE J. Solid State Circuits 1995, 30, 258–268. [CrossRef]

47. Villa, T.; Kam, T.; Brayton, R.K.; Sangiovanni-Vincentelli, A. Synthesis of Finite State Machines: Logic Optimization; Springer Science & Business Media: Berlin, Germany, 2012.

48. De Micheli, G.; Brayton, R.K.; Sangiovanni-Vincentelli, A. Optimal state assignment for finite state machines. IEEE Trans. Comp. Aided Des. Integr. Circuits Syst. 1985, 4, 269–285. [CrossRef]

49. Rawski, M.; Selvaraj, H.; Luba, T. An application of functional decomposition in ROM-based FSM implementation in FPGA devices. J. Syst. Archit. 2005, 51, 423–434. [CrossRef]

50. ABC System. Available online: https://people.eecs.berkeley.edu/~alanmi/abc/ (accessed on 15 February 2021).

51. 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.

52. Vivado Design Suite User Guide: Synthesis. UG901 (v2019.1). Available online: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug901-vivado-synthesis.pdf (accessed on 15 February 2021).

53. Quartus Prime. Available online: https://www.intel.pl/content/www/pl/pl/software/programmable/quartus-prime/overview.html (accessed on 15 February 2021).

54. Kacprzyk, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 24–40.

55. Khatri, S.P.; Gulati, K. Advanced Techniques in Logic Synthesis, Optimizations and Applications; Springer: New York, NY, USA, 2011.

56. ABC System. Available online: https://people.eecs.berkeley.edu/~alanmi/abc/ (accessed on 15 February 2021).

57. Sutter, G.; Todorovich, E.; López-Buedo, S.; Boemo, E. Low-power FSMs in FPGA: Encoding alternatives. In Integrated Circuit Design, Power and Timing Modeling, Optimization and Simulation; Springer: Berlin/Heidelberg, Germany, 2002; pp. 363–370.

58. McElvain, K. LGSynth93 Benchmark; Mentor Graphics: Wilsonville, OR, USA, 1993.

59. Xilinx, Inc. VC709 Evaluation Board for the Virtex-7 FPGA User Guide; UG887 (v1.6); Xilinx, Inc.: San Jose, CA, USA, 2019. Available online: https://www.xilinx.com/support/documentation/boards_and_kits/vc709/ug887-vc709-eval-board-v7-fpga.pdf (accessed on 11 March 2019).