applied sciences

MDPI

Article

Reducing LUT Count for FPGA-Based Mealy FSMs

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

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, 21, 600-richya str., 21021 Vinnytsia, Ukraine
3 Department of Infocommunication Engineering, Faculty of Infocommunications, Kharkiv National University of Radio Electronics, Nauky avenue 14, 61166 Kharkiv, Ukraine
4 Department of Technology, The Jacob of Paradies University, ul. Teatralna 25, 66-400 Gorzów Wielkopolski, Poland
* Correspondence: kkrzywicki@ajp.edu.pl

Received: 20 June 2020; Accepted: 23 July 2020; Published: 25 July 2020

Abstract: Very often, digital systems include sequential blocks which can be represented using a model of Mealy finite state machine (FSM). It is very important to improve such FSM characteristics as the number of used logic elements, operating frequency and power consumption. The paper proposes a novel design method optimizing LUT counts of LUT-based Mealy FSMs. The method is based on simultaneous use of such methods of structural decomposition as the replacement of FSM inputs and encoding of the collections of outputs. The proposed method results in three-level logic circuits of Mealy FSMs. These circuits have regular systems of interconnections. An example of FSM synthesis with the proposed method is given. The experiments with standard benchmarks were conducted. The results of experiments show that the proposed approach leads to reducing the LUT counts from 12% to 59% in average compared with known methods of synthesis of single-level FSMs. Furthermore, our approach provides better LUT counts as compared to methods of synthesis of two-level FSMs (from 9% to 20%). This gain is accompanied by a small loss of FSM performance.

Keywords: FPGA; LUT; Mealy FSM; structural decomposition; replacement of inputs; collections of outputs

1. Introduction

One of the features of our time is a wide application of digital systems in various spheres of human activity [1,2]. Modern digital systems include different combinational and sequential blocks [3,4]. The behaviour of a sequential block can be represented using the model of finite state machine (FSM) [5,6]. To improve characteristics of a digital system, it is necessary to improve such characteristics of FSMs as internal occupied resources, performance and power consumption. This necessity explains the continuous interest in developing methods aimed at optimizing these characteristics of FSM circuits. As a rule, the less internal occupied resources are used by an FSM circuit, the less power it consumes [7]. So, it is very important to reduce internal occupied resources consumed by an FSM circuit.

A sequential block can be represented as either Mealy or Moore FSM [5,6]. There are thousands of monographs and articles devoted to problems of FSM circuits design. The vast majority of these works are devoted to Mealy FSMs. Based on this analysis, we have chosen Mealy FSM as the object of research in our current article.

To diminish the required internal occupied resources, it is necessary to take into account specifics of logic elements implementing FSM circuits [8,9]. Nowadays, field programmable gate arrays (FPGAs) [10–12] are widely used in the implementation of digital systems [9,13–15]. Due to it,

we choose FPGA-based FSMs as a research object in the given article. In this article, we consider a case when look-up table (LUT) elements are used to implement logic circuits of Mealy FSMs.

A LUT is a block having S_L inputs and a single output [10,12]. A single LUT allows implementing an arbitrary Boolean function having up to S_L arguments [16,17]. However, the number of LUT inputs is rather small [10,12]. This feature leads to the need of functional decomposition of systems of Boolean functions (SBFs) representing FSM circuits [17,18]. In turn, this leads to multi-level FSM circuits with complex systems of interconnections [9].

One of the most crucial steps in the LUT-based design flow is the technology mapping [19–25]. During this step, an FSM circuit is converted into a network of interconnected LUTs. The outcome of technology mapping determines resulting characteristics of an FSM circuit. These characteristics are strongly interrelated.

The internal occupied resources consumed by a LUT-based FSM circuit include LUTs, flip-flops, interconnections, circuit of synchronization, input-output blocks. Obviously, to reduce the amount of required resources, it is very important to reduce the LUT count in a circuit. As follows from [26], the more LUTs are included into an FSM circuit, the more static power it consumes. Now, process technology has scaled considerably, with current design activity at 14 and 7 nm. Due to it, interconnection delay now dominates logic delay [26]. As noted in [16,27], the interconnections are responsible for the consuming up to 70% on power. So, it is very important to reduce the amount of interconnections to improve the characteristics of FSM circuits.

As shown in [26,28], the value $S_L = 6$ provides an optimal trade-off for the occupied chip area, performance, and power consumption of a LUT. However, the complexity of FPGA-based projects is constantly growing [9]. To overcome this contradiction, it is necessary to develop the methods of technology mapping that take into account rather small value of LUT's inputs.

The main contribution of this paper is a novel design method aimed at reducing the number of LUTs in circuits of FPGA-based Mealy FSMs. The proposed approach is based on joint usage of two known methods of structural decomposition (replacement of inputs and encoding of collections of outputs). The method leads to FSM circuits having three levels of logic and regular system of interconnections. The proposed method allows obtaining FSM circuits with fewer LUTs compared to circuits based on either single-level or two-level FSM models. Our current study is focused in Xilinx solutions [12].

The rest of the paper is organised as the follows. Section 2 presents the theoretical background of Mealy FSMs and peculiarities of FPGAs. Section 3 discusses the state of the art of FPGA-based technology mapping. Section 4 describes the main idea of the proposed method. The synthesis example is shown in Section 5. Section 6 gives results of experiments conducted on standard benchmarks [29]. A brief conclusion ends the paper.

2. Specifics of FPGAs and Mealy FSMs

The majority of modern FPGAs are organized using so called "island-style" architecture [16,28,30]. They include different configurable logic blocks (CLBs) and a matrix of programmable interconnections [10–12]. In this article, we consider CLBs consisting of LUTs and programmable flip-flops. To implement LUT-based circuits of sequential blocks, it is necessary to connect outputs of some LUTs with flip-flops [5].

An extremely small amount of LUT inputs leads to the necessity of functional decomposition [17] of functions representing combinational parts of FSMs. The functional decomposition produces multi-level circuits with irregular systems of interconnections. Such circuits resemble "spaghetti-type" programs [28]. Using terminology from programming, we can say that the functional decomposition produces LUT-based circuits with "spaghetti-type" interconnections.

A Mealy FSM is defined as a vector $< X, Y, A, \delta, \lambda, a_1 >$ [5], where $X = \{x_1, \ldots, x_L\}$ is a set of inputs, $Y = \{y_1, \ldots, y_N\}$ is a set of outputs, $A = \{a_1, \ldots, a_M\}$ is a set of internal states, δ is a function of transitions, λ is a function of output, and $a_1 \in A$ is an initial state. A Mealy FSM can be

represented using different tools, such as: state transition graphs [3,5], binary decision diagrams [31,32], and-inverter graphs [33], graph-schemes of algorithms [5]. In this article, we use state transition tables (STTs) for this purpose.

An STT includes the following columns [3,5]: a_m is a current state; a_s is a state of transition (a next state); X_h is a conjunction of inputs (or their compliments) determined a transition from a_m to a_s; Y_h is a collection of outputs generated during the transition from a_m to a_s. The column h includes the numbers of transitions ($h \in \{1, \ldots, H\}$). For example, the STT (Table 1) represents some Mealy FSM S_1.

Table 1. STT of Mealy FSM S_1.

a_m	a_s	X_h	Y_h	h
a_1	a_2	x_1	y_1y_7	1
	a_3	$\overline{x_1}$	y_2y_6	2
a_2	a_4	x_2	y_5	3
	a_4	$\overline{x_2}x_3$	y_2y_6	4
	a_5	$\overline{x_2}\,\overline{x_3}$	$y_1y_4y_6$	5
a_3	a_2	x_4	y_2y_3	6
	a_5	$\overline{x_4}x_5$	y_1y_7	7
	a_6	$\overline{x_4}\,\overline{x_5}$	y_4y_6	8
a_4	a_5	1	y_5	9
a_5	a_2	x_3x_6	y_2	10
	a_3	$x_3\overline{x_6}$	y_3y_4	11
	a_6	$\overline{x_3}x_7$	y_2y_6	12
	a_1	$\overline{x_3}\,\overline{x_7}$	y_5y_7	13
a_6	a_4	x_8	y_2	14
	a_1	$\overline{x_8}$	–	15

Using STT (Table 1), the following parameters of S_1 can be found: the number of inputs $L = 8$, the number of outputs $N = 7$, the number of states $M = 6$, and the number of transitions $H = 15$. Furthermore, Table 1 uniquely defines the functions of transitions and output of FSM S_1.

To find SBFs representing an FSM circuit, it is necessary [5]: (1) to encode states $a_m \in A$ by binary codes $K(a_m)$; (2) to construct sets of state variables $T = \{T_1, \ldots, T_R\}$ and input memory functions (IMFs) $\Phi = \{D_1, \ldots, D_R\}$ and (3) to transform an initial STT into a direct structure table (DST). States $a_m \in A$ are encoded during the step of state assignment [3].

In this article, we use the state codes having the minimum possible number of state variables R, where

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

This method is used, for example, in well-known academic system SIS [34]. There are other approaches for state encoding where the number of state variables differs from (1). For example, the academic system ABC [33] of Berkeley uses one-hot state assignment with $R = M$.

State codes are kept into a state register (RG). The RG consists of R flip-flops with mutual inputs of synchronization (*Clock*) and reset (*Start*). For LUT-based FSMs, D flip-flops are used to organize state registers [9]. The pulse *Clock* allows the functions $D_r \in \Phi$ to change the RG content.

To find functions representing an FSM circuit, it is necessary to create a direct structure table. A DST is an expansion of an STT by the columns with codes of current and next states ($K(a_m)$ and $K(a_s)$, respectively). Furthermore, a DST includes a column Φ_h with symbols $D_r \in \Phi$ corresponding to 1's in the code K(a_s) from the row h of a DST ($h \in \{1, \ldots, H\}$). The following SBFs are derived from a DST:

$$\Phi = \Phi(T, X); \quad (2)$$

$$Y = Y(T, X). \quad (3)$$

The systems (2) and (3) determine a structural diagram of Mealy FSM U_1 (Figure 1 from [35]). In Figure 1, the symbol *LUTer* denotes a logic block consisting of LUTs.

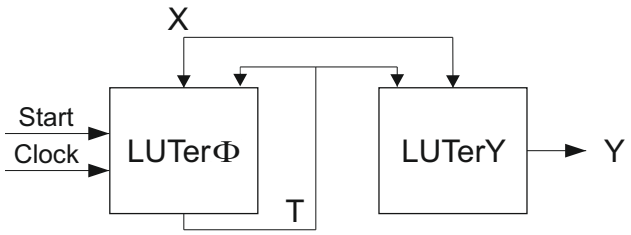

Figure 1. Structural diagram of LUT-based Mealy FSM U_1.

In the FSM U_1, the *LUTer*Φ implements the system (2), the LUTerY the system (3). If a function D_r is generated by a particular LUT, then the LUT's output is connected with a flip-flop [9]. The flip-flops form the state register distributed among the LUTs of *LUTer*Φ. It explains the existence of pulses *Start* and *Clock* as inputs of *LUTer*Φ.

The main specific of Mealy FSMs is the dependence of input memory functions and outputs on inputs and state variables. So, these functions have the same nature. This specific can be used to minimize hardware in LUT-based Mealy FSM circuits [35]. In Section 4, we will explain how to use this specific.

3. State of the Art

The process of technology mapping is associated with necessity of the solution of some optimization problems [9,35]. When designing FPGA-based FSMs, four optimization problems arise [35]: (1) the reduction of hardware amount, (2) the improvement of performance, (3) the reduction of power consumption, and (4) the improvement of testability. In this article, we propose a way for solution of the first problem.

Denote as $NL(f_i)$ the number of literals [3] in sum-of-products (SOPs) of functions (2) and (3). If the condition

$$NL(f_i) \leq S_L \quad (i \in \{1, \ldots, N+R\}) \tag{4}$$

takes place, then it is enough a single LUT to implement a circuit for any function $f_i \in \Phi \cup Y$. If the condition (4) is violated for some function $f_i \in \Phi \cup Y$, then the corresponding circuit is multi-level. In multi-level circuits, it is quite possible that the same inputs $x_l \in X$ appear on several logic levels. It results in FSM circuits with the spaghetti-type interconnections.

To improve the circuit characteristics, it is necessary to diminish the number of LUTs and make the system of interconnections more regular. It can be done using the following approaches:

1. The functional decomposition of functions representing Mealy FSM logic circuits [8,17,19,20,24,36].
2. The optimal state assignment [3,9,37–43].
3. The replacement of LUTs by embedded memory blocks (EMBs) [44–51].
4. The structural decomposition of FSM circuits [35,43,52].

The functional decomposition is a very powerful tool used in the process of technology mapping [19,53]. If the condition (4) is violated, then a function is broken down into smaller and smaller components. The process is terminated when any component has no more than S_L arguments.

If the condition (4) takes place, then a Mealy FSM logic circuit has exactly R + N LUTs. Otherwise, an FSM circuit is represented by $R + N + |\Psi|$ functions, where Ψ is a set of additional functions different from (2) and (3). New functions correspond to components of initial functions obtained in the process of decomposition.

A huge number of methods of functional decomposition are known. Some of them can be found, for example, in [19,20,23]. We do not discuss them in our article. All modern FPGA-based CAD

systems include program tools for functional decomposition. These tools can be found in academic systems [33,34,54,55], as well as in industrial packages [56–58]. The open system DEMAIN [54] includes powerful methods of functional decomposition. Due to this, we chose this system for comparison with our proposed approach.

The optimal state assignment [9] is a process of obtaining state codes optimizing systems of Boolean functions (2) and (3). One of the best academic optimal state assignment algorithms is JEDI distributed with the system SIS [34]. The JEDI is aimed at reducing the numbers of arguments in functions representing a Mealy FSM logic circuit. In this article, we compare JEDI-based FSMs with FSMs based on our proposed approach

Different state assignment strategies can be found in modern industrial CAD tools. For example, the design tool Vivado [57] uses the following approaches: the one-hot ($R = M$); compact; Gray codes; Johnson codes; speed encoding; automatic state assignment (auto). The same methods can be found in the package XST by Xilinx [56].

Because modern FPGAs include a lot of flip-flops, the one-hot state assignment is very popular in LUT-based design [41]. This approach allows producing less complicated combinational parts of FSM circuits than their counterparts based on the binary state encoding where $R = \lceil log_2 M \rceil$ [35]. As shown in [41], if $M \leq 8$, then FSMs based on binary state codes have better characteristics than their counterparts based on one-hot codes. The one-hot codes are more preferable if there is $M > 16$. However, the characteristics of LUT-based FSM circuits significantly depend on the number of inputs [35]. As it is shown in [42], if $L \leq 10$, then it is better to use one-hot state codes. Otherwise, binary state encoding allows producing better FSM circuits. So, both approaches should be compared with our proposed method. We chose the method Auto of Vivado as a method of binary state encoding. This method allows choosing codes producing FSM circuits with the best possible characteristics.

The main goal of both Gray and Johnson state encoding approaches is the reducing switching activity of an FSM circuit. It allows reducing the dynamic power consumption [35]. We do not discuss these methods in detail. Such an analysis can be found, for example, in [42].

So, a large number of state assignment methods are currently known. They are usually focused on optimizing one or more characteristics of FSM circuits. Some of them mostly focus on area reduction. It is very difficult to say which method is the best for a particular FSM. It depends on both the features of FSM and FPGA, as well as on the accepted criteria of FSM circuit optimality.

Each literal of SOP representing a function f_i corresponds to a wire in the FSM circuit. So, to diminish the number of interconnections, it is necessary to diminish the numbers of literals in Boolean functions (2) and (3). The fewer interconnections, the less power is consumed [28]. Therefore, the optimal state assignment must be performed regardless of whether the condition (4) is met or not.

Modern FPGAs include a lot of configurable embedded memory blocks [10–12]. Replacement of LUTs by EMB allows significantly improve the characteristics of resulting FSM circuits [41]. Because of it, there are a lot of design methods for EMB-based FSMs [43,44,46–49,59]. The survey of different methods of EMB-based design can be found in [60]. Unfortunately, these methods can be used only if there are "free" EMBs, which are not used to implement other parts of a digital system.

To optimize a LUT-based FSM circuit, it is necessary to eliminate a direct dependence of functions $y_n \in Y$ and $D_r \in \Phi$ on inputs $x_l \in X$. It is a main goal of different methods of a structural decomposition [35]. To eliminate this dependence, new functions $f_i \in \Psi$ are introduced to eliminate this dependence. These new functions depend on inputs and/or state variables. To optimize an FSM circuit, the following condition should take place:

$$|\Psi| \ll N + R. \tag{5}$$

Each system of new functions determines a separate block LUTer with its unique input and output variables. These blocks can be viewed as "hardware subroutines" by analogy with subroutines in programming [61,62]. If the relation (5) takes place, then the total number of LUTs implementing functions $f_i \in \Psi$ is significantly less than their total number in blocks $LUTer\Phi$ and $LUTerY$ of an

equivalent FSM U_1. Using hardware subroutines allows structuring an FSM circuit. The functions $f_i \in \Psi$ are used as arguments of functions (2) and (3). If the condition

$$|\Psi| \ll L + R \tag{6}$$

is true, then the total number of LUTs in an FSM circuit is significantly less than it is for an equivalent FSM U_1.

If (6) takes place, then the structural decomposition leads to reduced number of literals in SOPs of functions $f_i \in \Phi \cup Y$ as compared to initial functions (2) and (3). In turn, it reduces the total number of LUTs in blocks $LUTer\Phi$ and $LUTerY$ (as compared to U_1). If condition (4) is violated for some functions $f_i \in \Phi \cup Y \cup \Psi$, then the methods of functional and structural decomposition should be used together in the process of technology mapping. A survey of different methods of structural decomposition can be found in [35].

In this article, we discuss two methods of structural decomposition. They are the methods of replacement of inputs and encoding of outputs. They have been proposed to minimize the control memory size in microprogram control units (MCU) [63]. Next, they were applied in PLA-based FSMs [64]. Each of these methods was used separately in EMB-based FSM design [44,45,48,60]. However, they have never been used in LUT-based FSM design. In this article, we propose to combine these methods together to optimize characteristics of LUT-based Mealy FSMs.

The first method is a replacement of inputs $x_l \in X$ by additional variables $p_g \in P = \{p_1, \ldots, p_G\}$ where $G \ll L$ [64]. To do it, it is necessary to create an SBF

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

In MCUs, SBF (7) is implemented using a multiplexer. The additional variables are used as arguments of functions $f_i \in \Phi \cup Y$. These functions are represented as

$$\Phi = \Phi(T, P); \tag{8}$$

$$Y = Y(T, P). \tag{9}$$

The functions (8) and (9) have a regular nature [35]. So, they can be implemented as a memory block having $G + R$ address inputs and $N + R$ outputs.

Using this approach leads to FSM U_2 shown in Figure 2. It includes a multiplexer implementing SBF (7) and a memory block implementing systems (8) and (9).

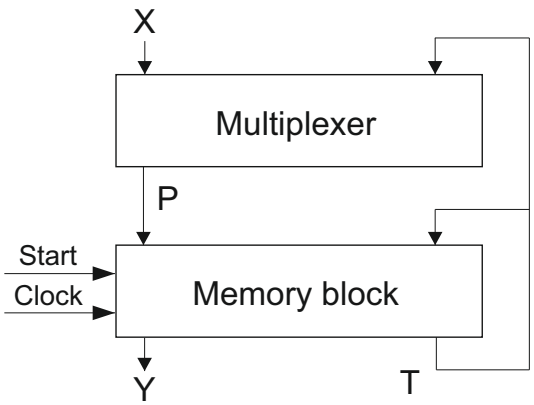

Figure 2. Structural diagram of FSM U_2.

In the classical MCU [63], only a single input is checked in each cycle of operation ($G = 1$). This results in rather slow control units. To decrease the number of cycles required for implementing

a control algorithm, it is necessary to increase the value of G. To optimize an MCU performance, the value of G is determined as [64]:

$$G = max(|X(a_1)|, \ldots, |X(a_M)|). \tag{10}$$

In (10), the symbol $X(a_m)$ stands for the set of inputs determining transitions from the state $a_m \in A$.

The model U_2 was used in the FPGA-based design. Different U_2-based approaches are discussed, for example, in [60]. In all discussed cases, EMBs implement systems (8) and (9). The system (7) is implemented with LUTs.

Collections of outputs (COs) $Y_q \subseteq Y (q \in \{1, \ldots, Q\})$ are generated during interstate transitions. The minimum number of bits in the code $K(Y_q)$ is determined as

$$R_Q = \lceil log_2 Q \rceil. \tag{11}$$

To encode COs by codes $K(Y_q)$, some additional variables $z_r \in Z = \{z_1, \ldots, z_{R_Q}\}$ are used.

If COs are encoded, then the system of outputs is represented as

$$Y = Y(Z). \tag{12}$$

In the case of MCU, the system (12) is implemented using two blocks, namely, a decoder and a coder [35].

To generate the additional variables $z_r \in Z$, it is necessary to find an SBF

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

In the case of MCU, both systems (13) and (2) are implemented by a memory block. In the case of FPGA-based design, a memory block is represented as a network of EMBs.

Using the encoding of COs in FPGA-based design leads to Mealy FSM U_3 shown in Figure 3.

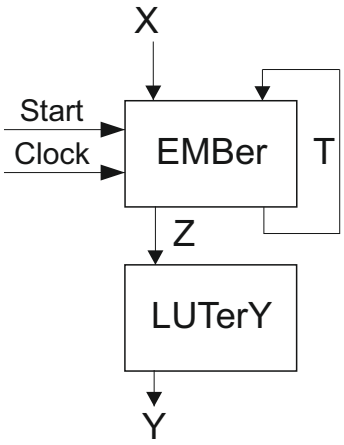

Figure 3. Structural diagram of FSM U_3.

In FSM U_3, the block EMBer implements systems (2) and (13). The block LUTer implements the system (12).

So far, these methods have been used separately to improve characteristics of circuits of FPGA-based FSMs. Moreover, some parts of FSM circuits have been implemented using EMBs [45,47]. In this article, we propose to use these methods together. Moreover, all functions are implemented by LUTs. This approach leads to Mealy FSM logic circuits having three levels of logic. The circuit for each level of logic can be viewed as a hardware subroutine. This approach allows structuring a resulting

FSM circuit and makes the system of interconnections more regular. We denote the proposed Mealy FSM by the symbol U_4.

4. Main Idea of the Proposed Method

Consider some Mealy FSM S_i represented by an STT. We assume that the following procedures have been executed: (1) the replacement of inputs; (2) the encoding of COs; (3) the encoding of states and (4) the transformation of initial STT into the DST of FSM U_1. To get SBFs representing U_4, we should transform the DST of FSM U_1 into a DST of Mealy FSM U_4.

To obtain the arguments of functions (8) and $Z(T, P)$, it is necessary to replace the column X_h of the DST of FSM U_1 by the column P_h. It is executed in the following way: if an additional variable $p_g \in P$ replaces an input $x_l \in X$ for a state $a_m \in A$, then the variable x_l (or its negation) from the column X_h is replaced by the variable p_g (or its negation) in the column P_h for all transitions from the state $a_m \in A$.

To obtain functions $Z(T, P)$, it is necessary to replace the column Y_h of the DST of FSM U_1 by the column Z_h. The filling of the column Z_h is executed in the following manner. If the h-th row of DST includes a CO $Y_q \subseteq Y$ such that the r-th bit of $K(Y_q)$ is equal to 1, then the symbol z_r should be written in the h-th row of the column Z_h of DST of FSM U_4.

Using a DST of FSM U_4, we can derive the systems (8) and

$$Z = Z(T, P). \tag{14}$$

Using the table of replacement of inputs, we can get the system (7). Next, using the content of collections of outputs, we can obtain the system (12).

Until now, the methods of replacement of inputs and encoding of the collections of outputs were used separately in EMB-based Mealy FSM design. In this article, we propose to use them together in LUT-based Mealy FSMs. There are three levels of logic blocks in the proposed Mealy FSM U_4. Its structural diagram is shown in Figure 4.

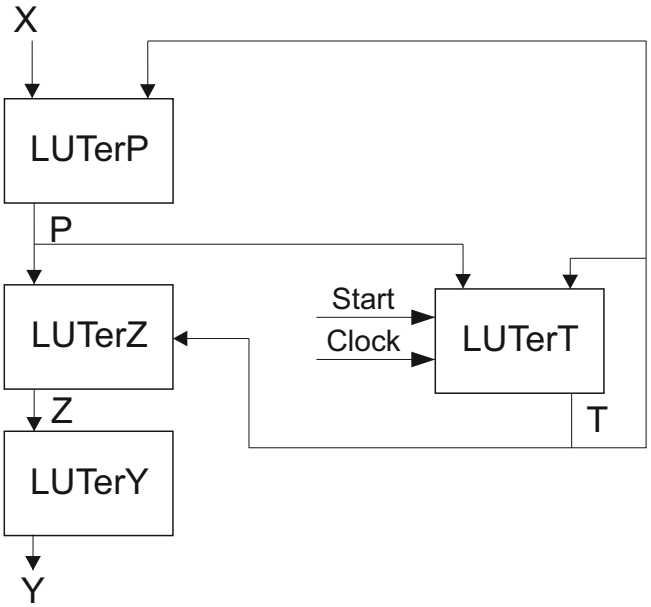

Figure 4. Structural diagram of Mealy FSM U_4.

In FSM U_4, the first level of logic is represented by a block *LUTerP*, the second level includes blocks *LUTerZ* and *LUTerT*, the third level includes a block *LUTerY*. The blocks implement the following SBFs: the *LUTerP* implements the system (7), the *LUTerZ* the system (14), the *LUTerT* the system (8), and the *LUTerY* the system (12).

If the condition

$$G + R \leq S_L \tag{15}$$

takes place, then the *LUTerZ* consists of R_Q LUTs and *LUTerT* of R LUTs. It is the best possible case. If the condition (15) is violated, then it is necessary to apply the methods of functional decomposition for some functions from SBFs (8) and (14).

If the condition

$$R_Q \leq S_L \tag{16}$$

takes place, then there are exactly N LUTs in the circuit of the *LUTerY*. If this condition is violated, then it is necessary to apply the methods of functional decomposition for some functions from SBF (12).

In this article, we propose a design method for Mealy FSM U_4. We assume that an FSM is represented by an STT. The method includes the following steps:

1. Executing the replacement of inputs by additional variables $p_g \in P$.
2. Executing the state assignment in a way optimizing the SBF $P = P(X, T)$.
3. Deriving the collections of outputs $Y_q \subseteq Y$ from the STT.
4. Executing the encoding of COs in a way optimizing the SBF $Y = Y(Z)$.
5. Creating the DST of FSM U_4 on the base of initial STT.
6. Deriving the SBFs (7), (8), (12) and (14) from the DST.
7. Implementing circuit of FSM using particular LUTs.

Some steps of the proposed method are connected with solution of optimization problems. We discuss these problems in the following Section.

5. Example of Synthesis

If a Mealy FSM S_j is synthesized using a model U_i, then we denote it by the symbol $U_i(S_j)$. Consider an example of synthesis for Mealy FSM $U_4(S_1)$. An FSM circuit will be implemented using LUTs with $S_L = 6$.

Executing the replacement of inputs. We start from constructing sets $X(a_m) \subseteq X$. A set $X(a_m)$ includes inputs $x_l \in X$ determining transitions from the state $a_m \in A$. Using Table 1 gives the following sets: $X(a_1) = \{x_1\}$, $X(a_2) = \{x_2, x_3\}$, $X(a_3) = \{x_4, x_5\}$, $X(a_4) = \emptyset$, $X(a_5) = \{x_3, x_6, x_7\}$ and $X(a_6) = \{x_8\}$.

Using (10) gives $G = max(1, 2, 2, 0, 3, 1) = 3$. So, there is the set $P = \{p_1, p_2, p_3\}$. Using (1) gives the number of state variables $R = 3$.

We should construct a table of replacement of inputs [35]. This table has M columns marked by states $a_m \in A$ and G rows marked by variables $p_g \in P$. If an input $x_l \in X$ is replaced by a variable $p_g \in P$ in the state $a_m \in A$, then there is the symbol x_l written at the intersection of the column a_m and the row p_g [64].

Inputs $x_l \in X$ written in a row p_g form a set $X(p_g) \subseteq X$. If $|X(p_g)| \leq S_L - R$, then the circuit generating $p_g \in P$ is implemented as a single LUT. In the discussed case, there is $S_L - R = 3$. To optimize the circuit of *LUTerP*, we should distribute inputs $x_l \in X$ in a way providing the relation $|X(p_g)| \leq 3$ ($g \in \{1, \ldots, G\}$). It could be done using the approach from [64]. One of the possible solutions is shown in Table 2.

Table 2. Replacement of inputs.

p_g \ a_m	a_1	a_2	a_3	a_4	a_5	a_6
p_1	x_1	x_2	–	–	x_6	–
p_2	–	x_3	x_4	–	x_3	–
p_3	–	–	x_5	–	x_7	x_8

Executing the state assignment. To optimize the circuit of $LUTerP$, it is necessary to diminish the number of literals in functions (7) [64]. It can be done due to a proper state assignment. These methods are based on results of the work [64].

Using (1) gives $R = 3$. So, there are the sets $T = \{T_1, T_2, T_3\}$ and $\Phi = \{D_1, D_2, D_3\}$. One of the possible outcomes of the state assignment is shown in Figure 5.

T_3 \ T_1T_2	00	01	11	10
0	a_1	a_2	a_3	a_6
1	a_4	a_5	$*$	$*$

Figure 5. State codes of Mealy FSM $U_4(S_1)$.

Deriving the collections of outputs. This step is executed in the trivial way. The collections $Y_q \subseteq Y$ are written in the column Y_h of an STT. Using Table 1, the following COs can be found: $Y_1 = \emptyset$, $Y_2 = \{y_1, y_7\}$, $Y_3 = \{y_2, y_6\}$, $Y_4 = \{y_5\}$, $Y_5 = \{y_1, y_4, y_6\}$, $Y_6 = \{y_2, y_3\}$, $Y_7 = \{y_4, y_6\}$, $Y_8 = \{y_5, y_7\}$, $Y_9 = \{y_2\}$ and $Y_{10} = \{y_3, y_4\}$.

To optimize the circuit of LUTerY, it is necessary to minimize the number of literals in functions (12). Furthermore, it allows minimizing the number of interconnections between the blocks $LUTerZ$ and $LUTerY$.

Executing the encoding of COs. We start this process from a system representing outputs $y_n \in Y$. as functions of collections $Y_q \subseteq Y$. It is the following system in the discussed case:

$$\begin{gathered} y_1 = Y_2 \vee Y_5; \quad y_2 = Y_3 \vee Y_6 \vee Y_9; \\ y_3 = Y_6 \vee Y_{10}; \quad y_4 = Y_5 \vee Y_7 \vee Y_{10}; \\ y_5 = Y_4 \vee Y_8; \quad y_6 = Y_3 \vee Y_5 \vee Y_7; \\ y_7 = Y_2 \vee Y_8. \end{gathered} \tag{17}$$

There are $Q = 10$ collections of outputs in the discussed case. Using (11) gives $R_Q = 4$ and $Z = \{z_1, \ldots, z_4\}$. Using the method [64] allows obtaining codes of COs shown in Figure 6.

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

Figure 6. Codes of COs of Mealy FSM $U_4(S_1)$.

Creating the DST of FSM $U_4(S_1)$. Having codes of states and COs, we can transform the initial STT (Table 1) into a DST of Mealy FSM $U_4(S_1)$ (Table 3).

Consider the row $h = 1$ of Table 3. There is $a_m = a_1$ and $a_s = a_2$. As follows from Figure 5, the code of a_2 is equal to 010. Due to it, there is the symbol D_2 in the column Φ_h. There is the symbol x_1 in the row 1 of STT. As follows from Table 2, the input x_1 is replaced by the variable p_1 for the state $a_1 \in A$. Due to it, there is the symbol p_1 in the first row of DST (Table 3). There is the collection of outputs $Y_2 = \{y_1, y_7\}$ in the first row of Table 1. As follows from Figure 6, there is $K(Y_2) = 0100$. Due to it, there is the symbol z_2 in the row 1 of DST (Table 3). All other rows of Table 3 are filled in the same way.

Deriving SBFs representing the circuit of $U_4(S_1)$. During this step, the functions (7), (8), (12) and (14) should be found. It can be done using Tables 2 and 3, as well as codes from Karnaugh maps shown in Figures 5 and 6.

We start from the SBF (7). We use the symbol A_m to denote a conjunction of state variables (or their complements) corresponding to the code $K(a_m)$.

Table 3. Direct structure table of Mealy FSM $U_4(S_1)$.

a_m	$K(a_m)$	a_s	$K(a_s)$	P_h	Z_h	Φ_h	h
a_1	000	a_2	010	p_1	z_2	D_2	1
		a_3	110	$\overline{p_1}$	z_1z_4	D_1D_2	2
a_2	010	a_4	001	p_1	z_2z_3	D_3	3
		a_4	001	$\overline{p_1}p_2$	z_1z_4	D_3	4
		a_5	011	$\overline{p_1}\,\overline{p_2}$	z_2z_4	D_2D_3	5
a_3	110	a_2	010	p_2	$z_1z_2z_3$	D_2	6
		a_5	011	$\overline{p_2}p_3$	z_2	D_2D_3	7
		a_6	100	$\overline{p_2}\,\overline{p_3}$	z_4	D_1	8
a_4	001	a_5	011	1	z_2z_3	D_2D_3	9
a_5	011	a_2	010	p_2p_1	z_1z_3	D_2	10
		a_3	110	$p_2\overline{p_1}$	z_3z_4	D_1D_2	11
		a_6	100	$\overline{p_2}p_3$	z_1z_4	D_1	12
		a_1	000	$\overline{p_2}\,\overline{p_3}$	z_3	–	13
a_6	100	a_4	001	p_3	z_1z_3	D_3	14
		a_1	000	$\overline{p_3}$	–	–	15

The following system can be derived from Table 2:

$$\begin{aligned} p_1 &= A_1x_1 \vee A_2x_2 \vee A_5x_6; \\ p_2 &= A_2x_3 \vee A_3x_4 \vee A_5x_3; \\ p_3 &= A_3x_5 \vee A_5x_7 \vee A_6x_8. \end{aligned} \tag{18}$$

Using codes from Figure 5, we can get the following minimized functions:

$$\begin{aligned} p_1 &= \overline{T_1}\,\overline{T_2}\,\overline{T_3}x_1 \vee \overline{T_1}\,T_2\,\overline{T_3}x_2 \vee T_2T_3x_6; \\ p_2 &= \overline{T_1}\,T_2x_3 \vee T_1T_2x_4; \\ p_3 &= T_1T_2x_5 \vee T_2T_3x_7 \vee T_1\overline{T_2}\,x_8. \end{aligned} \tag{19}$$

In the discussed case, the system (19) represents *LUTerP*. Each equation of (19) includes not more than six literals. Because $S_L = 6$, there are only $G = 3$ LUTs in the circuit of *LUTerP*.

Each row of DST of FSM U_4 corresponds to the product term

$$F_h = A_mB_h \quad (h \in \{1, \ldots, H\}). \tag{20}$$

In (20), the symbol B_h denotes a conjuction of variables p_g (or their compliments) written in the column P_h of DST.

The functions (8) and (14) depend on terms (20). For example, the following equations can be derived from Table 3:

$$\begin{aligned} D_1 &= F_2 \vee F_8 \vee F_{11} \vee F_{12} = \overline{T_1}\,\overline{T_2}\,\overline{T_3}\,\overline{p_1} \\ &\vee T_1T_2\overline{T_3}\,\overline{p_3}\,\overline{p_3} \vee \overline{T_1}T_2T_3p_2\,\overline{p_1} \vee \overline{T_1}T_2T_3\overline{p_2}p_3. \end{aligned} \tag{21}$$

$$z_1 = F_2 \vee F_4 \vee F_6 \vee F_{10} \vee F_{12} \vee F_{14} = \\ = T_1 T_2 \overline{T_3}\, \overline{p_2}\, \overline{p_3} \vee \ldots \vee T_1 \overline{T_2}\, \overline{T_3} p_3. \quad (22)$$

All other functions $D_r \in \Phi$ and $z_r \in Z$ are constructed in the same manner.

In the discussed case, there is $R + G = 6$. Because $S_L = 6$, the condition $NL(f_i) \leq S_L$ takes place for any function $f_i \in \Phi \cup Z$. It means that there are R=3 LUTs in the circuit of $LUTerT$ and $R_Q = 4$ LUTs in the circuit of $LUTerZ$.

Using system (17) and codes from Figure 6, we can get the following system:

$$y_1 = z_2\overline{z_3}; \quad y_2 = z_1; \quad y_3 = z_3 z_4; \\ y_4 = \overline{z_1} z_4; \quad y_5 = \overline{z_1} z_3 \overline{z_4}; \quad y_6 = \overline{z_3} z_4; \\ y_7 = z_2 \overline{z_3}\, \overline{z_4} \vee \overline{z_1}\, \overline{z_2} z_3 \overline{z_4}. \quad (23)$$

The analysis of (23) shows that there is no need in a LUT to implement the function y_2. Because $R_Q = 4$ is less than $S_L = 6$, the condition (16) takes place. So, it is necessary $N - 1 = 6$ LUTs to implement $LUTerY$.

In general case, each function from (12) has R_Q literals. For N functions, it gives $R_Q \cdot N$ literals. In the discussed case, there is $R_Q \cdot N = 4 \cdot 7 = 28$ literals. Each literal corresponds to the interconnection between blocks $LUTerZ$ and $LUTerY$. As follows from (23), there are 16 literals in this system. It gives a 42% savings in the number of interconnections compared to the general case. This economy is achieved due to chosen encoding of COs $Y_q \subseteq Y$. It should give economy in the power consumption.

So, there are $G = 3$ LUTs in $LUTerP$, $R = 3$ LUTs in $LUTerT$, $R_Q = 4$ LUTs in $LUTerZ$, and $N - 1 = 6$ LUTs in $LUTerY$. It gives 16 LUTs in the logic circuit of Mealy FSM $U_4(S_1)$.

The last step of design is connected with the placement and routing procedures [16]. It is executed using industrial CAD tools such as, for example, Vivado by Xilinx [57]. We do not discuss this step for a given example.

It is known that any sequential block can be represented using either model of Mealy FSM or Moore FSM [3]. There is the following specific of Moore FSM: its outputs depend only on states. It means that, for Moore FSMs, state codes can be viewed as the codes of collections of outputs. So, there is no sense in using additional variables encoding the collections of outputs. It means that the proposed approach can be used only in the case of LUT-based Mealy FSMs.

6. Experimental Results

To investigate the efficiency of proposed method, we use standard benchmarks from the library [29]. The library includes 48 benchmarks taken from the design practice. The benchmarks are rather simple, but they are very often used by different researches to compare new and known results [9,43]. The benchmarks are represented in KISS2 format. These benchmarks are Mealy FSMs, so we can directly use them in our research. The characteristics of these benchmark FSMs are shown in Table 4.

The process of obtaining synthesis research results in Vivado [57] has been divided into two stages. The first stage was a generation of the VHDL code based on benchmarks saved in the KISS2 format. Each benchmark was generated by the tool K2F [35,45] according to the given FSM model. It should be noted that generated code uses a specific Vivado code style [65] in order to provide the proper FSM extraction and fully synthesizable code. Then, in the next stage, the VHDL code was imported into Vivado (ver. 2019.1). The target device was the Xilinx Virtex 7 (XC7VX690TFFG1761) [66]. The chip includes LUTs with six inputs. The synthesis and optimization options were set to: *max_bram 0, opt_design -retarget -propconst -bram_power_opt* and the selected FSM state encoding method one of the following: *auto, one_hot, sequential, johnson* or *gray*. The final results presented in the tables are taken after the post-implementation.

Table 4. Characteristics of Mealy FSM benchmarks.

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

As we have found, our method can give economy in area if $R + L > S_L$. We have divided the benchmarks into categories using the values of $L + R$ and S_L. If $L + R \leq 6$, then benchmarks belong to category 0 (trivial FSMs); if $L + R \leq 12$, then to category 1 (simple FSMs); if $L + R \leq 18$, then to category 2 (average FSMs); if $L + R \leq 24$, then to category 3 (big FSMs); otherwise, they belong to category 4 (very big FSMs). Obviously, there is no sense to apply our approach to FSMs belonging to category 0. As our researches show, the higher the category, the more saving the proposed approach gives.

Four other methods were taken to compare with our approach. They are: (1) Auto of Vivado; (2) One-hot of Vivado; (3) JEDI-based FSMs and (4) DEMAIN-based FSMs. The results of experiments are shown in Table 5 (the number of LUTs) and Table 6 (the operating frequency).

Table 5. Experimental results (the number of LUTs).

Benchmark	Auto	One-Hot	JEDI	DEMAIN	Our Approach	Category
bbara	17	17	10	9	10	1
bbsse	33	37	24	26	26	1
bbtas	5	5	5	5	8	0
beecount	19	19	14	16	14	1
cse	40	66	36	38	33	1
dk14	10	27	10	12	12	1
dk15	5	16	5	6	6	1
dk16	15	34	12	14	11	1
dk17	5	12	5	6	8	0
dk27	3	5	4	4	7	0
dk512	10	10	9	10	12	0
donfile	31	31	22	26	21	1
ex1	70	74	53	57	40	2
ex2	9	9	8	9	8	1
ex3	9	9	9	9	11	0
ex4	15	13	12	13	11	1
ex5	9	9	9	9	10	0
ex6	24	36	22	23	21	1
ex7	4	5	4	4	6	1
keyb	43	61	40	42	37	1
kirkman	42	58	39	41	33	2
lion	2	5	2	2	6	0
lion9	6	11	5	5	8	0
mark1	23	23	20	21	19	1
mc	4	7	4	5	6	0
modulo12	7	7	7	7	9	0
opus	28	28	22	26	21	1
planet	131	131	88	94	78	2
planet1	131	131	88	94	78	2
pma	94	94	86	91	72	2
s1	65	99	61	64	54	2
s1488	124	131	108	112	89	2
s1494	126	132	110	117	90	2
s1a	49	81	43	54	38	2
s208	12	31	10	11	9	2
s27	6	18	6	6	6	1
s386	26	39	22	25	20	1
s420	10	31	9	10	8	4
s510	48	48	32	39	22	4
s8	9	9	9	9	9	1
s820	88	82	68	76	52	4
s832	80	79	62	70	50	4
sand	132	132	114	121	99	3
shiftreg	2	6	2	2	4	0
sse	33	37	30	32	26	1
styr	93	120	81	88	70	2
tma	45	39	39	41	30	2
Total	1792	2104	1480	1601	1318	
Percentage,%	135.96	159.63	112.29	121.47	100	

Table 6. Experimental results (the operating frequency, MHz).

Benchmark	Auto	One-Hot	JEDI	DEMAIN	Our Approach	Category
bbara	193.39	193.39	212.21	198.46	183.32	1
bbsse	157.06	169.12	182.34	178.91	159.24	1
bbtas	204.16	204.16	206.12	208.32	194.43	0
beecount	166.61	166.61	187.32	184.21	156.72	1
cse	146.43	163.64	178.12	174.19	153.24	1
dk14	191.64	172.65	193.85	187.32	162.78	1
dk15	192.53	185.36	194.87	188.54	175.42	1
dk16	169.72	174.79	197.13	189.83	164.16	1
dk17	199.28	167	199.39	172.19	147.22	0
dk27	206.02	201.9	204.18	205.10	181.73	0
dk512	196.27	196.27	199.75	197.49	175.63	0
donfile	184.03	184.00	203.65	194.83	174.28	1
ex1	150.94	139.76	176.87	186.14	164.32	2
ex2	198.57	198.57	200.14	199.75	188.95	1
ex3	194.86	194.86	195.76	193.43	174.44	0
ex4	180.96	177.71	192.83	178.14	168.39	1
ex5	180.25	180.25	181.16	181.76	162.56	0
ex6	169.57	163.80	176.59	174.12	156.42	1
ex7	200.04	200.84	200.6	200.32	191.43	1
keyb	156.45	143.47	168.43	157.16	136.49	1
kirkman	141.38	154	156.68	143.76	155.36	2
lion	202.43	204	202.35	201.32	185.74	0
lion9	205.3	185.22	206.38	205.86	167.28	0
mark1	162.39	162.39	176.18	169.65	153.48	1
mc	196.66	195.47	196.87	192.53	178.02	0
modulo12	207	207	207.13	207.37	189.7	0
opus	166.2	166.2	178.32	168.79	157.42	1
planet	132.71	132.71	187.14	185.73	174.68	2
planet1	132.71	132.71	187.14	185.73	173.29	2
pma	146.18	146.18	169.83	153.57	156.12	2
s1	146.41	135.85	157.16	149.17	145.32	2
s1488	138.5	131.94	157.18	153.12	141.27	2
s1494	149.39	145.75	164.34	159.42	155.63	2
s1a	153.37	176.4	169.17	158.12	166.36	2
s208	174.34	176.46	178.76	172.87	166.42	2
s27	198.73	191.5	199.13	198.43	185.15	1
s386	168.15	173.46	179.15	169.21	164.65	1
s420	173.88	176.46	177.25	172.87	186.35	4
s510	177.65	177.65	198.32	183.18	199.05	4
s8	180.02	178.95	181.23	180.39	168.32	1
s820	152	153.16	176.58	166.29	175.69	4
s832	145.71	153.23	173.78	160.03	174.39	4
sand	115.97	115.97	126.82	120.63	120.07	3
shiftreg	262.67	263.57	276.26	276.14	248.79	0
sse	157.06	169.12	174.63	169.69	158.14	1
styr	137.61	129.92	145.64	138.83	118.02	2
tma	163.88	147.8	164.14	168.19	137.48	2
Total	8127.08	8061.22	8718.87	8461.1	7917.1	
Percentage, %	102.65	101.82	110.13	106.87	100	

Tables 5 and 6 are organized in the same order. The rows are marked by the names of benchmarks, the columns by design methods. The rows "Total" include results of summation for corresponding values. The summarized characteristics of our approach (U_4—based FSMs) are taken as 100%. The rows "Percentage" show the percentage of summarized characteristics of FSM circuits implemented by other methods respectively to benchmarks based on our approach. Let us point out that the model

U_1 is used for designs with Auto, One-hot, JEDI and DEMAIN. Furthermore, for better visualization, summary data are presented in the Figures 7–9.

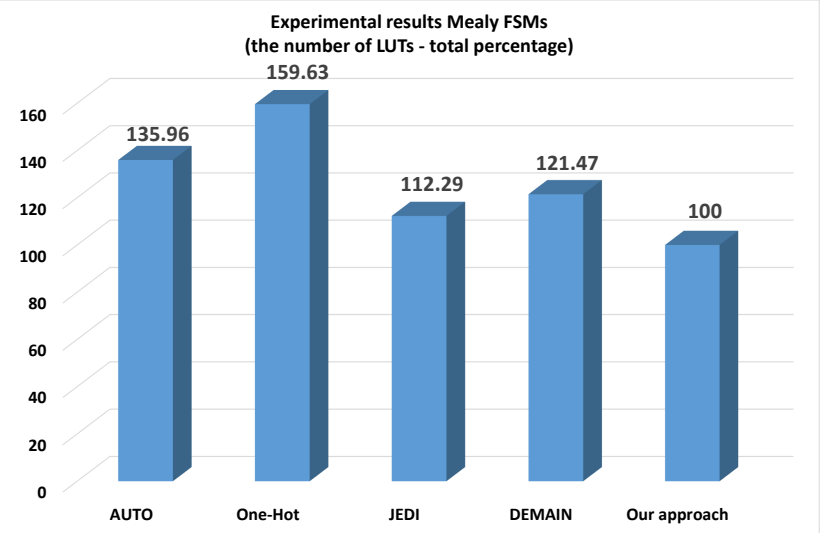

Figure 7. Experimental results (the number of LUTs—total percentage).

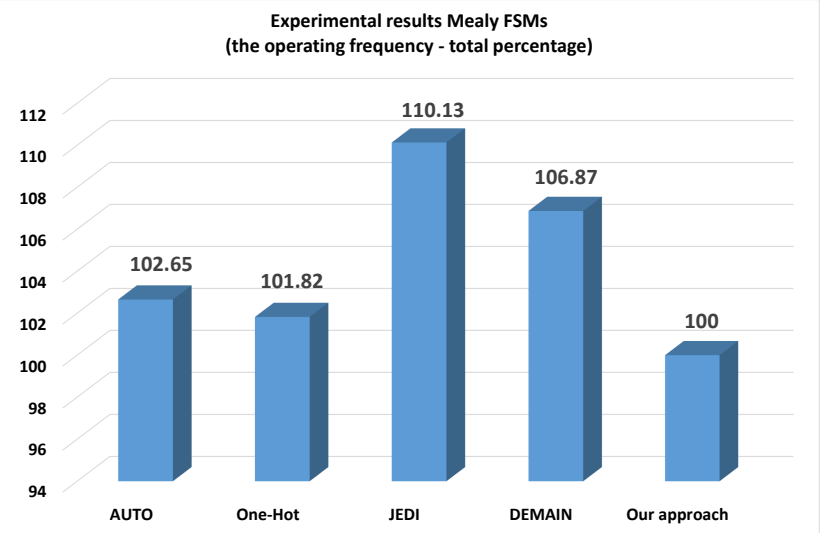

Figure 8. Experimental results (the operating frequency—total percentage).

As follows from Table 5 and Figure 7, the U_4—based FSMs required fewer LUTs than it is for other investigated methods. There is the following economy: (1) 35.65% regarding Auto; (2) 59.27% regarding One-hot; (3) 12.04% regarding JEDI-based FSMs and (4) 21.20% regarding DEMAIN-based FSMs. The higher is the category, the greater is the gain in LUTs. For trivial and simple FSMs, the better results are produced by either JEDI or DEMAIN. The gain are becoming more and more noticeable, starting from the average FSMs.

As follows from Table 6 and Figure 8, the U_4—based FSMs have a lower operating frequency than it is for other investigated methods. There is the following loss: (1) 2.65% regarding Auto; (2) 1.82% regarding One-hot; (3) 10.13% regarding JEDI-based FSMs and (4) 6.87% regarding DEMAIN-based FSMs. However, starting from big FSMs, the losses are getting smaller. It is connected with the fact that U_4—-based FSMs always have three levels of logic and more regular system of interconnections.

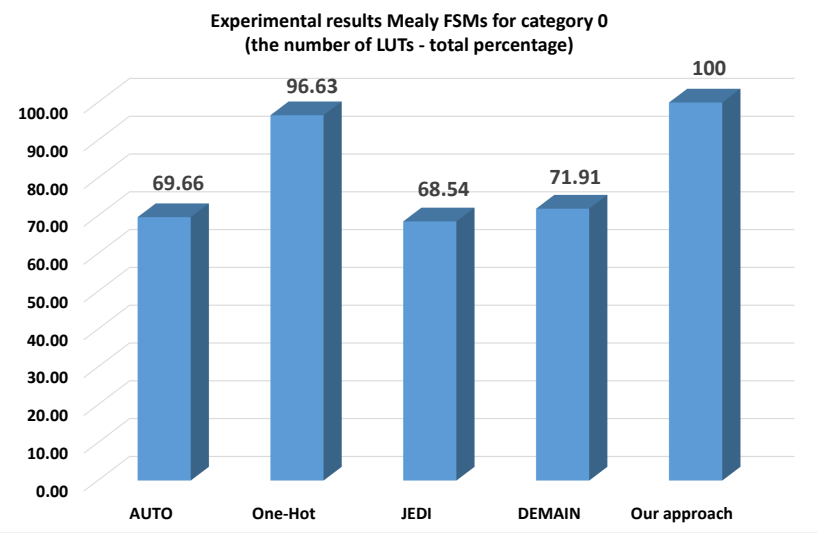

Figure 9. Experimental results for category 0 (the number of LUTs—total percentage).

The main goal of the proposed approach is to reduce the LUT count in circuits of FPGA-based Mealy FSMs. As follows from Table 5, the degree of reduction in the number of LUTs depends on the category of an FSM. To clarify this dependence, we have created Table 7 (experimental results for category 0), Table 8 (experimental results for category 1) and Table 9 (experimental results for categories 2–94). Furthermore, we present these results by graphs on Figures 9–11, respectively.

Table 7. Experimental results for category 0 (the number of LUTs).

Benchmark	Auto	One-Hot	JEDI	DEMAIN	Our Approach	Category
bbtas	5	5	5	5	8	0
dk17	5	12	5	6	8	0
dk27	3	5	4	4	7	0
dk512	10	10	9	10	12	0
ex3	9	9	9	9	11	0
ex5	9	9	9	9	10	0
lion	2	5	2	2	6	0
lion9	6	11	5	5	8	0
mc	4	7	4	5	6	0
modulo12	7	7	7	7	9	0
shiftreg	2	6	2	2	4	0
Total	62	86	61	64	89	
Percentage,%	69.66	96.63	68.54	71.91	100	

As follows from Table 7 and Figure 9, the proposed method produces FSM circuits having more LUTs than it is for other investigated methods. Our method has the following loss: (1) 30.34% regarding Auto; (2) 3.37% regarding One-hot; (3) 31.46% regarding JEDI-based FSMs and (4) 28.09% regarding DEMAIN-based FSMs. So, there is no sense in using our approach for designing trivial FSMs. However, it gives an economy in LUTs starting from simple FSMs (category 1).

As follows from Table 8 and Figure 10, the U_4—based FSMs of category 1 required fewer LUTs than it is for other methods. There is the following economy: (1) 23.03% regarding Auto; (2) 65.52% regarding One-hot; (3) 3.47% regarding JEDI-based FSMs and (4) 12.62% regarding DEMAIN-based FSMs. So, for the category 1, our approach produces FSM circuits slightly better than JEDI-based FSMs.

Table 8. Experimental results for category 1 (the number of LUTs).

Benchmark	Auto	One-Hot	JEDI	DEMAIN	Our Approach	Category
bbara	17	17	10	9	10	1
bbsse	33	37	24	26	26	1
beecount	19	19	14	16	14	1
cse	40	66	36	38	33	1
dk14	10	27	10	12	12	1
dk15	5	16	5	6	6	1
dk16	15	34	12	14	11	1
donfile	31	31	22	26	21	1
ex2	9	9	8	9	8	1
ex4	15	13	12	13	11	1
ex6	24	36	22	23	21	1
ex7	4	5	4	4	6	1
keyb	43	61	40	42	37	1
mark1	23	23	20	21	19	1
opus	28	28	22	26	21	1
s27	6	18	6	6	6	1
s386	26	39	22	25	20	1
s8	9	9	9	9	9	1
sse	33	37	30	32	26	1
Total	390	525	328	357	317	
Percentage,%	123.03	165.62	103.47	112.62	100	

Table 9. Experimental results for categories 2-4 (the number of LUTs).

Benchmark	Auto	One-Hot	JEDI	DEMAIN	Our Approach	Category
ex1	70	74	53	57	40	2
kirkman	42	58	39	41	33	2
planet	131	131	88	94	78	2
planet1	131	131	88	94	78	2
pma	94	94	86	91	72	2
s1	65	99	61	64	54	2
s1488	124	131	108	112	89	2
s1494	126	132	110	117	90	2
s1a	49	81	43	54	38	2
s208	12	31	10	11	9	2
styr	93	120	81	88	70	2
tma	45	39	39	41	30	2
sand	132	132	114	121	99	3
s420	10	31	9	10	8	4
s510	48	48	32	39	22	4
s820	88	82	68	76	52	4
s832	80	79	62	70	50	4
Total	1340	1493	1091	1180	912	
Percentage,%	146.93	163.71	119.63	129.39	100	

Our method produces best results for FSMs from categories 29-4 (Table 9 and Figure 11). It is very interesting that the gain respectively the one-hot approach is approximately the same as it is in the previous case. However, we provide a bigger gain for other investigated methods as compared to FSMs of the category 1. There is the following economy: (1) 46.93% regarding Auto; (2) 19.63% regarding JEDI-based FSMs and (3) 29.39% regarding DEMAIN-based FSMs. So, our approach produces FSM circuits with better amount of LUTs for Mealy FSMs having $L + R \geq 12$.

Till now, we compared our approach with U_1-based FSMs. However, we also compared the U_4-based FSMs with FSMs having two levels of logic. The structural diagrams of these FSMs are shown in Figure 12. The FSM U_5 is based on the replacement of inputs (Figure 12a). There is no

such an FSM in the known literature. We have got its structural diagram by transformation the FSM U_2 (Figure 2). We replaced the multiplexer by the block $LUTerP$; the memory block is replaced by $LUTerTY$. The $LUTerP$ implements the system (5), the LUTerTY the systems (6) and (7).

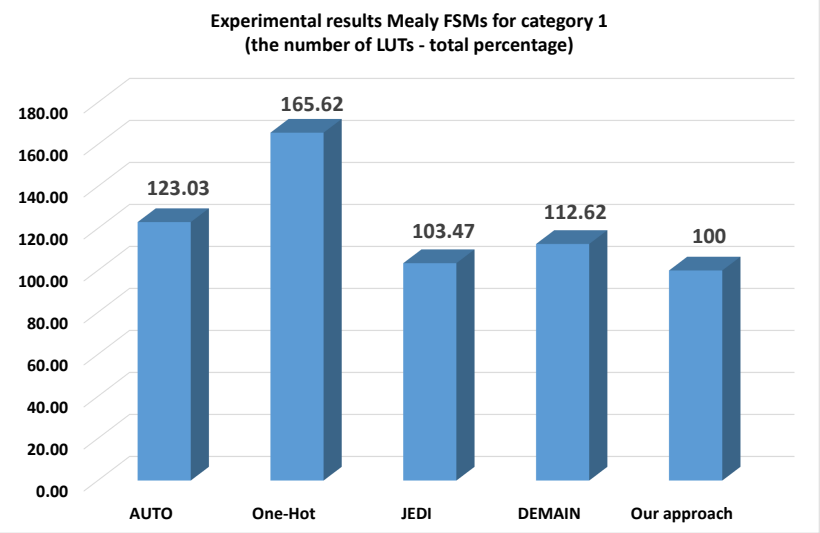

Figure 10. Experimental results for category 1 (the number of LUTs—total percentage).

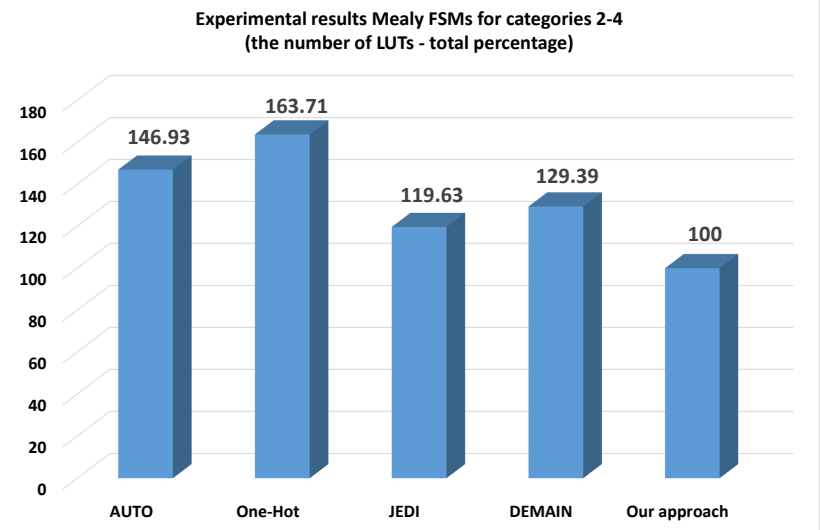

Figure 11. Experimental results for categories 2–4 (the number of LUTs—total percentage).

The FSM U_6 is based on the encoding of collections of outputs (Figure 12b). There is no such an FSM in the known literature. We have got its structural diagram by transformation the FSM U_3 (Figure 3). We replaced the block $EMBer$ by the block $LUTerTZ$ generating functions (2) and (11). As it is for FSM U_3, the $LUTerY$ implements the system (10).

The FSM U_7 is based on the transformation of state codes into outputs (Figure 12c) [35]. In this FSM, additional variables from the set V replace inputs for output functions. The FSM U_8 is based on the transformation of collections of outputs into state codes (Figure 12d) [35]. In this FSM, additional variables from the set V replace inputs for input memory functions. Both methods belong to the group of object transformation methods [35]. We do not discuss these approaches in this article. We just use them as examples of FSMs having two levels of logic.

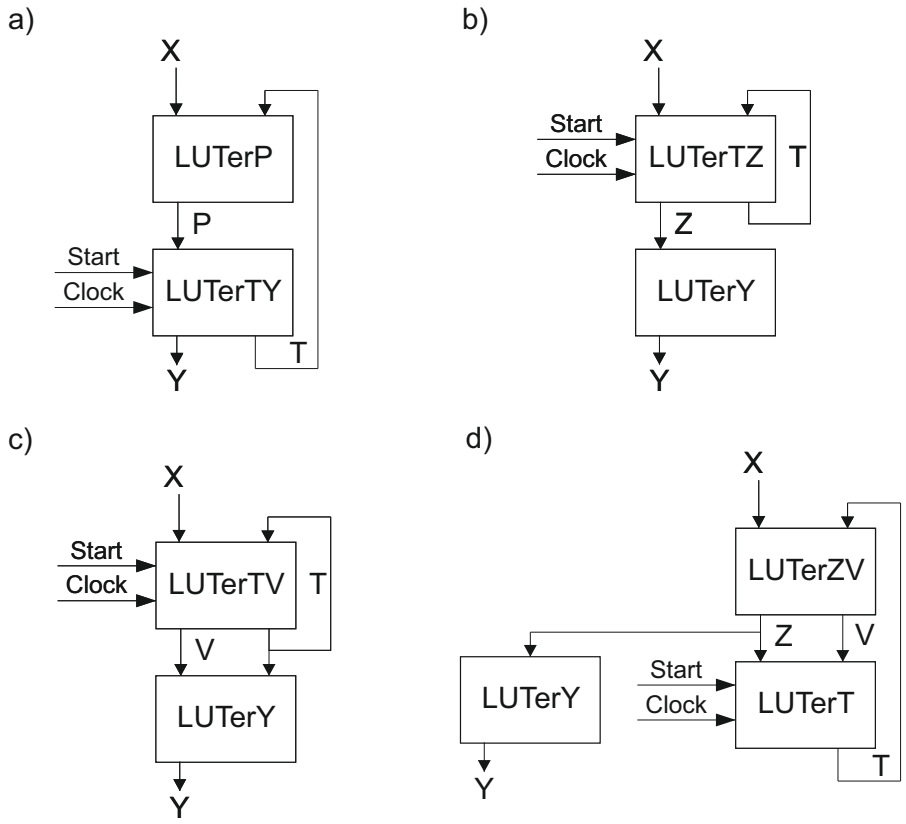

Figure 12. Structural diagrams of LUT-based Mealy FSMs with two-levels of logic: (**a**) replacement of inputs; (**b**) encoding of collections of outputs; (**c**) transformation of state codes into outputs; (**d**) transformation of collections of outputs into state codes.

We compared the FSMs (Figure 12) with our approach for the most complex benchmarks (categories 2–4). The results of experiments are shown in Table 10 and Figure 13. As follows from Table 10, our method produces better results for FSMs than it is for FSMs U_5-U_8. There is the following economy: (1) 17.11% regarding U_5; (2) 11.95% U_6 (3) 20.18% regarding U_7 and 4) 9.1% regarding U_8. So, our approach produces FSM circuits with better amount than two-level Mealy FSMs having $L + R \geq 12$. However, the gain is noticeably less than for U_1-based FSMs from these categories.

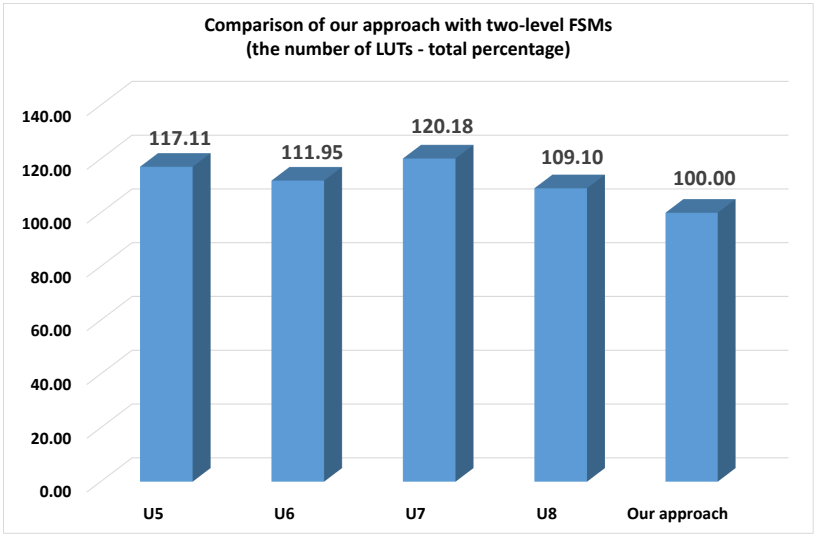

Figure 13. Comparison diagrams of our approach with two-level FSMs.

Table 10. Comparison of our approach with two-level FSMs.

Benchmark	U_5	U_6	U_7	U_8	Our Approach	Category
ex1	51	49	52	46	40	2
kirkman	38	37	40	37	33	2
planet	86	80	88	82	78	2
planet1	86	80	88	82	78	2
pma	84	88	90	76	72	2
s1	60	58	62	58	54	2
s1488	98	90	87	94	89	2
s1494	101	99	104	96	90	2
s1a	42	44	46	42	38	2
s208	11	12	11	11	9	2
styr	79	76	80	72	70	2
tma	40	37	42	38	30	2
sand	119	109	124	108	99	3
s420	10	11	11	10	8	4
s510	35	30	37	28	22	4
s820	67	64	70	61	52	4
s832	61	57	64	54	50	4
Total	1086	1021	1096	995	912	
Percentage,%	117.11	111.95	120.18	109.10	100	

So, the results of our experiments show that the proposed approach can reduce the LUT counts respectively to single- and two-level Mealy FSMs having $L + R \geq 12$. Of course, this conclusion is true only for benchmarks [27] and the device XC7VX690tffg1761-2 by Virtex-7, where LUTs have 6 inputs. It is almost impossible to make a similar conclusion for the general case. However, we hope that our approach rather good potential and can be used in CAD systems targeting FPGA-based Mealy FSMs.

7. Conclusions

Contemporary FPGA devices include a lot of look-up table elements. This allows implementing a very complex digital system using only a single chip. However, LUTs have rather small amount of inputs (for the vast majority of devices the value of S_L does not exceeds 6). This value is considered as optimal [26,28]. To design rather complex FSMs, the methods of functional decomposition are used. As a rule, this leads to multi-level FSM circuits with complex systems of spaghetti-type interconnections.

To optimize the LUT counts in FPGA-based FSM circuits, different methods of structural decomposition could be applied. As our researches [35] show, the structural decomposition can lead to FSM circuits having better characteristics than their counterparts based on the functional decomposition. They have regular system of interconnections and predicted number of logic levels.

The current article is devoted to a novel approach aimed at optimization of LUT-based Mealy FSMs. The proposed approach is based on simultaneous use of such methods of structural decomposition as the replacement of inputs and encoding of collections of outputs. Till now, the methods of replacement of inputs and encoding of the collections of outputs were used separately in EMB-based Mealy FSM design. In this article, we propose to use them together in LUT-based Mealy FSMs. Furthermore, we encode the collections in a way minimizing the number of interconnections between other blocks and the block generating FSM outputs. The proposed approach leads to three-level Mealy FSM circuits with regular systems of interconnections.

We compared the proposed approach with FSM circuits obtained using the Xilinx CAD tool Vivado 2019.1. These circuits were obtained using four different approaches: Auto by Vivado, One-hot by Vivado, JEDI and DEMAIN. The experiments clearly show that the proposed approach leads to reducing the number of LUTs in comparison with FSM circuits produced by other investigated methods. The results of experiments show that the proposed approach leads to reducing the LUT counts from 12% to 59% in average compared with known methods of synthesis of single-level FSMs.

Furthermore, our approach provides better LUT counts as compared to methods of synthesis of two-level FSMs (from 9% to 20%). However, our approach leads to slower FSM circuits as compared to other investigated methods. Thus, our approach reduces the overall performance of a digital system including U_4-based FSMs. So, the proposed method can be used if the LUT count is the dominant characteristic of a digital system.

There are two directions in our future research. The first is connected with development of design methods targeting FPGA chips of Intel (Altera). The second direction targets at EMB-based Mealy FSMs.

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

Funding: This research received no external funding.

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

Abbreviations

The following abbreviations are used in this manuscript:

CLB	configurable logic block
CO	collection of output
DST	direct structure table
EMB	embedded memory block
FSM	finite state machine
FPGA	field-programmable gate array
LUT	look-up table
MCU	microprogram control unit
SBF	systems of Boolean functions
SOP	sum-of-products
STT	state transition table

References

1. Marwedel, P. *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things*, 3rd ed.; Springer: Berlin, Germany, 2018.
2. Barkalov, A.; Titarenko, L.; Andrzejewski, G.; Krzywicki, K.; Kolopienczyk, M. Fault detection variants of the CloudBus protocol for IoT distributed embedded systems. *Adv. Electr. Comput. Eng.* **2017**, *17*, 3–10. [CrossRef]
3. Micheli, G.D. *Synthesis and Optimization of Digital Circuits*; McGraw–Hill: New York, NY, USA, 1994.
4. Bailliul J.; Samad T. *Encyclopaedia of Systems and Control*; Springer: London, UK, 2015.
5. Baranov, S. *Logic and System Design of Digital Systems*; TUTPress: Tallinn, Estonia, 2008.
6. Minns, P.; Elliot, I. *FSM-Based Digital Design Using Verilog HDL*; John Wiley and Sons: Hoboken, NJ, USA, 2008.
7. Jiminez, J.J.; Trojman, L.; Procel, L.-M. Power and area reduction of MD5 based on cryptoprocessors using novel approach of internal counters on the finite state machine. In Proceedings of the IEEE Fourth Ecuador Technical Chapter Meeting (ETCM), Guayaquil, Ecuador, 12–15 November 2019; pp. 1–4. [CrossRef]
8. Czerwinski, R.; Kania, D. *Finite State Machine Logic Synthesis for Complex Programmable Logic Devices*; Lecture Notes in Electrical Engineering; Springer: Berlin, Germany, 2013; Volume 231.
9. Sklyarov, V.; Skliarova, I.; Barkalov, A.; Titarenko, L. *Synthesis and Optimization of FPGA-Based Systems*; Lecture Notes in Electrical Engineering; Springer: Berlin, Germany, 2014; Volume 294.
10. Intel FPGAs and Programmable Devices. Available online: https://www.intel.pl/content/www/pl/pl/products/programmable.html (accessed on 24 July 2020).
11. Altera. Cyclone IV Device Handbook. 2020. Available online: http://www.altera.com/literature/hb/cyclone-iv/cyclone4-handbook.pdf (accessed on 24 July 2020).

12. Xilinx FPGAs. Available online: https://www.xilinx.com/products/silicon-devices/fpga.html (accessed on 24 July 2020).
13. Vázquez-Castillo, J.; Castillo-Atoche, A.; Carrasco-Alvarez, R.; Longoria-Gandara, O.; Ortegón-Aguilar, J. FPGA-Based Hardware Matrix Inversion Architecture Using Hybrid Piecewise Polynomial Approximation Systolic Cells. *Electronics* **2020**, *9*, *182*. [CrossRef]
14. Walters, E.G., III. Reduced-Area Constant-Coefficient and Multiple-Constant Multipliers for Xilinx FPGAs with 6-Input LUTs. *Electronics* **2017**, *6*, *101*. [CrossRef]
15. Agrawal, R.; Ahuja, K.; Hau Hoo, C.; Duy Anh Nguyen, T.; Kumar, A. ParaLarPD: Parallel FPGA Router Using Primal-Dual Sub-Gradient Method. *Electronics* **2019**, *8*, *1439*. [CrossRef]
16. Grout, I. *Digital Systems Design with FPGAs and CPLDs*; Elsevier: Amsterdam, The Netherlands, 2011.
17. Scholl, C. *Functional Decomposition with Application to FPGA Synthesis*; Kluwer Academic Publishers: Boston, MA, USA, 2001.
18. Rawski, M.; Łuba, T.; Jachna, Z.; Tomaszewicz, P. *Design of Embedded Control Systems*; The Influence of Functional Decomposition Onmodern Digital Design Process; Springer: Boston, MA, USA, 2005; pp. 193–203.
19. Kubica, M.; Kania, D. Technology Mapping Oriented to Adaptive Logic Modules. *Bull. Pol. Acad. Sci.* **2019**, *67*, 947–956.
20. Kubica, M.; Kania, D. Decomposition of multi-level functions oriented to configurability of logic blocks. *Bull. Pol. Acad. Sci.* **2017**, *65*, 317–331.
21. Mishchenko, A.; Chattarejee, S.; Brayton, R. Improvements to technology mapping for LUT-based FPGAs. *IEEE Trans. CAD* **2006**, *27*, 240–253.
22. Kubica, M.; Kania, D.; Kulisz, J. A technology mapping of fsms based on a graph of excitations and outputs. *IEEE Access* **2019**, *7*, 16123–16131. [CrossRef]
23. Kubica, M.; Kania, D. Area-oriented technologymapping for lut-based logic blocks. *Int. J. Appl. Math. Comput. Sci.* **2017**, *27*, 207–222. [CrossRef]
24. Machado, L.; Cortadella, J. Support-Reducing Decomposition for FPGA Mapping. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2020**, *39*, 213–224. [CrossRef]
25. Mishchenko, A.; Brayton, R.; Jiang, J.-H.R.; Jang, S. Scalable don't-care-based logic optimization and resynthesis. *ACM Trans. Reconfigurable Technol. Syst.* **2011**, *4*, 1–23. [CrossRef]
26. 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; ACM: New York, NY, USA, 2018; p. 6. [CrossRef]
27. Kilts, S. *Advanced FPGA Design: Architecture, Implementation, and Optimization*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 2007.
28. Kuon, I.; Tessier, R.; Rose, J. FPGA architecture: Survey and challenges—Found trends. *Electr. Des. Autom.* **2008**, *2*, 135–253.
29. McElvain, K. *LGSynth93 Benchmark*; Mentor Graphics: Wilsonville, OR, USA, 1993.
30. Krzywicki, K.; Barkalov, A.; Andrzejewski, G.; Titarenko, L.; Kolopienczyk, M. SoC research and development platform for distributed embedded systems. *Przegląd Elektrotechniczny* **2016**, *92*, 262–265. [CrossRef]
31. Opara, A.; Kubica, M.; Kania, D. Strategy of Logic Synthesis using MTBDD dedicated to FPGA. *Integr. Vlsi J.* **2018**, *62*, 142–158. [CrossRef]
32. Kubica, M.; Opara, A.; Kania, D. Logic synthesis for FPGAs based on cutting of BDD. *Microprocess. Microsyst.* **2017**, *52*, 173–187. [CrossRef]
33. 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.
34. Sentovich, E.M.; Singh, K.J.; Lavagno, L.; Moon, C.; Murgai, R.; Saldanha, A.; Savoj, H.; Stephan, P.R.; Brayton, R.K.; Sangiovanni-Vincentelli, A. *SIS: A System for Sequential Circuit Synthesis*; Tech. Rep.; University of California: Berkely, CA, USA, 1992.
35. Barkalov, A.; Titarenko, L.; Mielcarek, K.; Chmielewski, S. *Logic Synthesis for FPGA-Based Control Units—Structural Decomposition in Logic Design*; Lecture Notes in Electrical Engineering; Springer: Berlin, Germany, 2020; Volume 636.

36. Zając, W.; Andrzejewski, G.; Krzywicki, K.; Królikowski, T. Finite State Machine Based Modelling of Discrete Control Algorithm in LAD Diagram Language With Use of New Generation Engineering Software. *Procedia Comput. Sci.* **2019**, *159*, 2560–2569. [CrossRef]
37. Salauyou, V.; Ostapczuk, M. State Assignment of Finite-State Machines by Using the Values of Output Variables. In *Theory and Applications of Dependable Computer Systems, Proceedings of the DepCoS-RELCOMEX 2020, Brunow, Poland, 29 June–3 July 2020*; Zamojski, W., Mazurkiewicz, J., Sugier, J., Walkowiak, T., Kacprzyk, J., Eds.; Advances in Intelligent Systems and Computing; Springer: Cham, Switzerland, 2020; Volume 1173, pp. 543–553.
38. De Micheli, G.; Brayton, R.K.; Sangiovanni-Vincentelli, A. Optimal state assignment for finite statemachines. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2006**, *4*, 269–285. [CrossRef]
39. El-Maleh, A.H. A probabilistic pairwise swap search state assignment algorithm for sequential circuit optimization. *Integr. Vlsi J.* **2017**, *56*, 32–43. [CrossRef]
40. Klimowicz, A.S.; Solov'ev, V.V. Structural models of finite-state machines for their implementation on programmable logic devices and systems on chip. *J. Comput. Syst. Sci. Int.* **2015**, *54*, 230–242. [CrossRef]
41. Machalec, M.; Stastny, J. Synchronous FSM design methodology for Low-Power Smart Sensors and RFID Devices. *Electrorevue* **2010**, *1*, 1–9.
42. Sklyarov, V. Synthesis and implementation of RAM-based finite state machines in FPGAs. In *International Workshop on Field Programmable Logic and Applications*; Springer: Berlin/Heidelberg, Germany, 2000; pp. 718–727.
43. Barkalov, O.; Titarenko, L.; Mielcarek, K. Hardware reduction for lut-based mealy fsms. *Int. J. Appl. Math. Comput. Sci.* **2018**, *28*, 595–607. [CrossRef]
44. Rawski, M.; Selvaraj, H.; Łuba, T. An application of functional decomposition in ROM-based FSM implementation in FPGA devices. *J. Syst. Archit.* **2005**, *51*, 423–434. [CrossRef]
45. Kołopienczyk, M.; Titarenko, L.; Barkalov, A. Design of emb-based moore fsms. *J. Circuits Syst. Comput.* **2017**, *26*, 1–23. [CrossRef]
46. Rafla, N.I.; Gauba, I. A reconfigurable pattern matching hardware implementation using on-chip RAM-based FSM. In Proceedings of the 2010 53rd IEEE International Midwest Symposium on Circuits and Systems, Seattle, WA, USA, 1–4 August 2010; pp. 49–52.
47. Senhadji-Navarro, R.; Garcia-Vargas, I.; Jiménez-Moreno, G.; Civit-Balcells, A.; Guerra-Gutierrez, P. ROM based FSM implementation using input multiplexing in FPGA devices. *Electron. Lett.* **2004**, *40*, 1249–1251. [CrossRef]
48. Garcia-Vargas, I.; Senhadji-Navarro, R.; Jiménez-Moreno, G.; Civit-Balcells, A.; Guerra-Gutierrez, P. ROM-based finite state machine implementation in low cost FPGAs. In Proceedings of the IEEE International Symposium on Industrial Electronics ISIE 2007, Vigo, Spain, 4–7 June 2007; IEEE: Piscataway, NJ, USA, 2007; pp. 2342–2347.
49. Senhadji-Navaro, R.; Garcia-Vargas, I. High-Speed and Area-Efficient Reconfigurable Multiplexer Bank for RAM-Based Finite State Machine Implementations. *J. Circuits Syst. Comput.* **2015**, *24*, 1550101. [CrossRef]
50. Klimovich, A.S.; Solov'ev, V.V. Minimization of mealy finite-state machines by internal states gluing. *J. Comput. Syst. Sci. Int.* **2012**, *51*, 244–255. [CrossRef]
51. Barkalov, A.; Titarenko, L.; Mazurkiewicz, M.; Krzywicki, K. Encoding of terms in EMB-based Mealy FSMs. *Appl. Sci.* **2020**, *10*, 2762. [CrossRef]
52. Barkalov, A.; Titarenko, L.; Barkalov, A., Jr. Structural decomposition as a tool for the optimization of an FPGA-based implementation of a Mealy FSM. *Cybern. Syst. Anal.* **2012**, *48*, 313–322. [CrossRef]
53. Testa, E.; Amaru, L.; Soeken, M.; Mishchenko, A.; Vuillod, P.; Luo, J.; Casares, C.; Gaillardon, P.; Micheli, G.D. Scalable boolean methods in a modern synthesis flow. In Proceedings of the 2019 Design, Automation Test in Europe Conference Exhibition (DATE), Florence, Italy, 25–29 March 2019; pp. 1643–1648.
54. Selvaraj, H.; Nowicka, M.; Luba, T. Non-Disjoint Decomposition Strategy in Decomposition-Based Algorithms & Tools. In Proceedings of the International Conference on Computational Intelligence and Multimedia Application, Melbourne, Australia, 9–11 February 1998; pp. 34–42.
55. Michalski, T.; Kokosiński, Z. Functional decomposition of combinational logic circuits with PKmin. *Czas. Tech.* **2016**, *2016*, 191–202.
56. Xilinx. XST UserGuide.V. 11.3. Available online: https://www.xilinx.com/support/documentation/sw_manuals/xilinx11/xst.pdf (accessed on 24 July 2020).

57. Vivado. Available online: https://www.xilinx.com/products/design-tools/vivado.html (accessed on 24 July 2020).
58. Quartus Prime. 2020. Available online: https://www.intel.pl/content/www/pl/pl/software/programmable/quartus-prime/overview.html (accessed on 24 July 2020).
59. Senhadji-Navarro, R.; Garcia-Vargas, I. High-Performance Architecture for Binary-Tree-Based Finite State Machines. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2018**, *37*, 796–805. [CrossRef]
60. Garcia-Vargas, I.; Senhadji-Navarro, R. Finite state machines with input multiplexing: A performance study. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2015**, *34*, 867–871. [CrossRef]
61. Sass, R.; Schmidt, A. *Embedded System Design with platform FPGAs: Principles and Practices*; Morgan Kaufmann Publishers: Amsterdam, The Netherlands, 2010.
62. Dahl, O.; Dijkstra, E.; Hoare, C. *Structured Programming*; Academic Press: London, UK, 1972.
63. Wilkes, M.; Stringer, J. Microprogramming and the design of the control circuits in an electronic digital computer. In *Proceedings of Cambridge Philosophical Society*; Cambridge University Press: Cambridge, UK, 1953; Volume 49, pp. 230–238.
64. Baranov, S. *Logic Synthesis of Control Automata*; Kluwer Academic Publishers: Berlin, Germany, 1994.
65. *Vivado Design Suite User Guide: Synthesis*; UG901 (v2019.1); Xilinx, Inc.: San Jose, CA, USA, 2019.
66. *VC709 Evaluation Board for the Virtex-7 FPGA User Guide*; UG887 (v1.6); Xilinx, Inc.: San Jose, CA, USA, 2019.