electronics

MDPI

Article

Reducing Hardware in LUT-Based Mealy FSMs with Encoded Collections of Outputs

Alexander Barkalov [1,2,†], Larysa Titarenko [1,3,†] and Małgorzata Mazurkiewicz [4,*,†]

[1] Institute of Metrology, Electronics and Computer Science, University of Zielona Góra, ul. Licealna 9, 65-417 Zielona Góra, Poland
[2] Department of Computer Science and Information Technology, Vasyl Stus' Donetsk National University, 600-richya Str. 21, 21021 Vinnytsia, Ukraine
[3] Department of Infocommunication Engineering, Faculty of Infocommunications, Kharkiv National University of Radio Electronics, Nauky Avenue 14, 61166 Kharkiv, Ukraine
[4] Institute of Control & Computation Engineering, University of Zielona Góra, ul. Licealna 9, 65-417 Zielona Góra, Poland
* Correspondence: m.mazurkiewicz@issi.uz.zgora.pl
† These authors contributed equally to this work.

Abstract: A method is proposed that is focused on reducing the chip area occupied by logic elements creating the circuit of Mealy finite state machines (FSMs). The proposed method is aimed at FSM circuits implemented with internal resources of field-programmable gate arrays (FPGA). The required chip area is estimated by the number of look-up table (LUT) elements in a particular circuit. The method is based on mutual application of two methods of structural decomposition. The first of them is based on dividing the set of outputs and using unitary-maximum encoding of collections of FSM outputs. The second method is based on dividing the set of states by classes of compatible states. The optimization is achieved by replacing the maximum binary state codes by two-part codes proposed in this article. Each two-part state code consists of a code of a class including a particular state and a maximum binary code of this state inside a particular class. The proposed approach leads to three-level LUT-based Mealy FSM circuits. The first logic level generates three types of partial functions: unitary encoded outputs, variables encoding collections of outputs, and input memory functions. Each partial function is represented by a circuit including a single LUT. The LUTs from the second logic level generate final values of these functions. The LUTs from the third level implement outputs using collections of outputs. An example of synthesis applying the proposed method is discussed. The experiments were conducted using standard benchmark FSMs. Their results showed significant improving of the area occupied by an FSM circuit. The LUT count decreased on average by 9.49%. The positive side effect of the proposed method was increasing the value of the maximum operating frequency (on average, by 8.73%). The proposed method is advisable to use if a single-level LUT-based implementation of the FSM circuit is impossible.

Keywords: Mealy FSM; FPGA; LUT; synthesis; collections of outputs; two-part state codes; unitary-maximum codes

Citation: Barkalov, A.; Titarenko, L.; Mazurkiewicz, M. Reducing Hardware in LUT-Based Mealy FSMs with Encoded Collections of Outputs. *Electronics* **2022**, *11*, 3389. https://doi.org/10.3390/electronics11203389

Academic Editor: Andrea Boni

Received: 6 September 2022
Accepted: 17 October 2022
Published: 19 October 2022

1. Introduction

In the last few decades, there has been an increasingly profound influence of various digital systems on all aspects of human activity [1]. This applies primarily to personal computers, embedded systems, the Internet, and the Internet of Things [2]. All such systems are built using ultra-large integrated circuits such as ASICs and field-programmable gate arrays (FPGAs) [3]. Such chips are extremely complex: currently, they include 7–8 billion transistors [3]. This causes an urgent need to develop efficient computer-aided design (CAD) methods for implementing digital systems based on such chips. In the case of FPGA-based systems, technological mapping [4] is performed by industrial CAD systems

of firms manufacturing particular chips such as Intel (Altera) or AMD (Xilinx: San Jose, California, U.S.) [5,6]. These industrial CADs include a limited number of synthesis methods. However, it is possible to use them together with some external synthesis methods, where the original structural diagrams of FSM circuits are represented by HDL models [7]. Our current article is aimed at working out an FSM design method improving the characteristics of sequential blocks of FPGA-based digital systems. The step of synthesis is executed by our program tools, whereas the step of technology mapping is executed by the industrial CAD tool Vivado [8] by AMD (Xilinx).

One of the fundamental models used for representing sequential devices is the model of the Mealy finite state machine (FSM) [4]. In this paper, we discuss a case when FSM circuits are implemented using the internal resources of FPGAs [9]. The choice of these particular VLSI chips is determined by the fact that, now, a huge number of various projects are implemented using FPGAs [10]. Furthermore, it is clear that FPGAs will dominate logic design in the next few decades [7].

During the design process of FSM circuits, some optimization problems arise. As a rule, they are the following: minimizing the chip area occupied by an FSM circuit; reducing the value of the FSM cycle time (maximizing the operating frequency); minimizing the power consumption [11].

A digital system may include a number of sequential blocks [1]. The behavior of any sequential block, consisting of a combinational part and memory, can be represented using the FSM model. For each digital system, the ratio between the total complexity of processor units and memory units, on the one hand, and various FSMs, on the other hand, depends on the features of the system. The larger the proportion of the FSM-based part, the more important is the reduction in the chip area occupied by the circuits of each FSM. Solving this problem can lead to several positive effects [2]. Firstly, it can allow expanding the functionality of the whole system by expanding the functionality of the processor and other units of the system (since, after optimization of the FSM circuits, additional area will appear). Secondly, it may be possible to implement the system using less-powerful FPGA chips. Thirdly, a reduction in the area of the circuit leads to a decrease in power consumption, which is especially important for autonomous and mobile systems. In all these cases, the competitiveness of the designed digital system in relation to similar concurrent projects increases. All of the above shows the importance and necessity of optimizing the circuits of FSMs that are part of digital systems. Obviously, this area reduction is expedient as long as the performance of the optimized block remains within the specified limits [2].

In this paper, we discuss the case when FSM circuits are implemented using look-up table (LUT) elements [12]. The following internal resources are used for implementing FSM circuits: LUTs, flip-flops, synchronization tree, dedicated multiplexers, programmable interconnections, and input–outputs [12]. If an FSM circuit is implemented with these resources, then the occupied chip area is estimated as the number of LUTs (LUT count) [13]. As shown in [11], the reduction of the LUT count also reduces the power consumption. However, the area reduction can degrade the FSM's performance [14]. This degradation is an overhead of area reduction. In this paper, we propose a method that leads to simultaneously reducing the area with a slight increase in the FSM maximum operating frequency (compared to equivalent FSM circuits based on encoding of collections of outputs (COs) [11].

The main scientific novelty of our article is associated with the development of a new type of state code, called two-part codes. The first part contains the code for some class of the set of FSM states. The second part contains the code of state as an element of this class. Such an approach allows simultaneously reducing the number of LUTs in the FSM circuit and increasing its performance.

The main contribution of this paper is a novel method improving both the LUT counts and the performance of the FPGA-based circuits of Mealy FSMs. The improvement is associated with reducing the number of literals in Boolean functions representing an FSM circuit. A further decrease in the number of elements is associated with the proposed modification of the method of unitary-binary maximum encoding of output collections.

This modification allows using the resources of a single LUT to generate two outputs. The proposed method belongs to the methods of structural decomposition [11]. The results of the experiments showed that the proposed approach improves two characteristics of FSM circuits (LUT count and maximum operating frequency) as compared to the FSM circuits based on the maximum binary encoding of both states and collections of outputs [11].

The rest of the paper is organized as follows. Section 2 includes the basic information connected with the LUT-based Mealy FSM design. The state-of-the-art is analyzed in Section 3. Section 4 shows the main idea of the proposed approach. We show a synthesis example based on the proposed method in Section 5. The experimental results conducted with standard benchmarks are analyzed in Section 6. Finally, the article ends with a short conclusion.

In the further text, we use many notations. To facilitate understanding of the proposed method, we placed the main notation in Table 1.

Table 1. The main notation.

$SS = \{s_1, \ldots, s_G\}$	The set of FSM states.
$SI = \{i_1, \ldots, i_U\}$	The set of FSM inputs.
$SO = \{o_1, \ldots, o_W\}$	The set of FSM outputs.
G	The number of states.
U	The number of inputs.
W	The number of outputs.
R	The number of state variables.
$SSV = \{v_1, \ldots, v_R\}$	The set of state variables.
$SM = \{D_1, \ldots, D_R\}$	The set of input memory functions.
$SCO = \{CO_1, \ldots, CO_Q\}$	The set of collections of outputs.
$K(CO_q)$	The code of collection of outputs.
I_{LUT}	The number of LUT inputs.
$A(f_n)$	The number of arguments for a Boolean function f_n.
R_{CO}	The number of variables for encoding of collections of outputs.
$SOV = \{b_1, \ldots, b_{RCO}\}$	The set of variables encoding the collections of outputs.
$PS = \{CP^1, \ldots, CP^K\}$	The set of classes of compatible states.
$PC(s_g)$	The partial code of a state.
R_S	The number of variables encoding states inside the classes of compatible states.
K	The number of classes of compatible states.
R_C	The number of variables encoding classes of compatible states.
$TP(s_g)$	The two-part code of a state.
R_{TP}	The number of bits in the two-part state codes.

2. Basics of Mealy FSM Design with LUTs

In many cases, either a state transition graph (STG) or a state transition table (STT) is used to represent the behavior of the Mealy FSM [15]. Using these forms, the three following sets can be derived: a set of states $SS = \{s_1, \ldots, s_G\}$, a set of inputs $SI = \{i_1, \ldots, i_U\}$, and a set of outputs $SO = \{o_1, \ldots, o_W\}$. Therefore, an FSM has G states, U inputs, and W outputs. In the case of STG, its vertices correspond to states, and arcs correspond to interstate transitions. Above each arc, there are written FSM inputs causing a particular transition and outputs associated with this transition [15]. An STT represents an STG as a list of transitions, where each line corresponds to a particular transition. Therefore, if there are H

arcs in an STG, then there are H lines in a corresponding STT. For example, some FSM A_0 is represented by an STG shown in Figure 1.

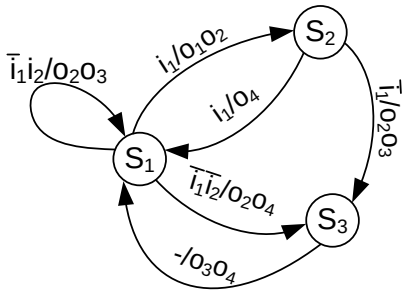

Figure 1. State transition graph of Mealy FSM Ex1.

FSM Ex1 is characterized by sets $SS = \{s_1, s_2, s_3\}$, $SI = \{i_1, i_2\}$, and $SO = \{o_1, \ldots, o_4\}$. Therefore, there are G = 3, U = 2, and W = 4. There are H = 6 arcs in this graph. Each arc is represented by a single line of Table 2.

Table 2. State transition table of Mealy FSM Ex1.

CS	*ST*	*In*	*Out*	*h*
s_1	s_2	i_1	$o_1 o_2$	1
	s_1	$\bar{i}_1 i_2$	$o_2 o_3$	2
	s_3	$\bar{i}_1 \bar{i}_2$	$o_2 o_4$	3
s_2	s_1	i_1	o_4	4
	s_3	$\bar{i}_1$	$o_2 o_3$	5
s_3	s_1	1	$o_3 o_4$	6

The transformation of the STG (Figure 1) into the STT (Table 2) is executed in a trivial way. Obviously, it is possible to transform any STT into the corresponding STG.

To design an FSM circuit, each state $s_g \in SS$ is represented by a binary code $SC(s_g)$. This is performed during the step of state assignment [15]. In the case of maximum binary (MB) state assignment, state codes consist of R_0 bits:

$$R_0 = \lceil \log_2 G \rceil. \tag{1}$$

Formula (1) determines a minimum possible number of state variables.

In a common case, the states are encoded using state variables from the set $SSV = \{v_1, \ldots, v_R\}$. Each bit of $SC(s_g)$ corresponds to a flip-flop. These flip-flops are combined into a code state register (RG). Very often, the RG consists of D flip-flops [4].

The maximum number of state code bits (R_{OH}) is used in the case of one-hot (OH) state assignment. The value of R_{OH} is determined as

$$R_{OH} = G. \tag{2}$$

The flip-flops are controlled by input memory functions (IMFs), a pulse of initialization *Start*, and a pulse of synchronization *Clock*. The IMFs form a set $SM = \{D_1, \ldots, D_R\}$. The value of R depends on the state assignment method used. It could be either R_0 or R_{OH} or some intermediate value. To load a code of the initial state $s_1 \in SS$ into the RG, the pulse *Start* is used. The IMFs determine a code of the next state. This code is loaded into the RG using the pulse *Clock*.

A Mealy FSM logic circuit consists of a combinational part and memory [4]. The memory is represented by the RG. The combinational part is represented by the following systems of Boolean functions (SBFs) [15]:

$$SM = F1(SSV, SI); \tag{3}$$

$$SO = F2(SSV, SI). \tag{4}$$

The systems (3) and (4) determine a structural diagram of Mealy FSM A_1 (Figure 2).

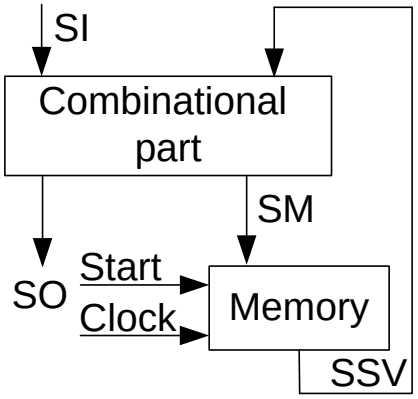

Figure 2. Structural diagram of Mealy FSM A_1.

The combinational part (Figure 2) implements SBFs (3) and (4). It is represented by a network of particular logic elements. The FSM memory is the register RG. The RG includes R master–slave flip-flops controlled by pulses *Start* and *Clock*.

The SBFs (3) and (4) are constructed using an FSM direct structure table (DST) [2]. To create a DST, it is convenient to transform the initial STG into the STT. An STT includes five columns [16]. A current state is shown in a column CS; a column ST includes a state of transition; an input signal I_h is shown in a column In; a column Out includes a collection of outputs (CO) O_h; h is a number of interstate transitions ($h \in \{1, \ldots, H\}$). The transformation of an STG into an equivalent STT is executed in a trivial way [16]. A DST includes all columns of a particular STT and three additional columns [15]. These additional columns are [16]: codes of the current state and the state of transition and the symbols of IMFs equal to 1 to load the next code into the RG.

The design method depends significantly on the properties of the logic elements used. In the case of LUT-based FSMs, an FSM circuit is implemented as a network of configurable logic blocks (CLBs). Each CLB includes LUTs, programmable flip-flops, and dedicated multiplexers. The resulting network is created using a programmable routing matrix [3]. Our paper targets FPGAs produced by Xilinx [12]. A peculiarity of their CLBs is the reconfigurability of the LUTs: the number of LUT inputs can be changed using dedicated multiplexers. There is some basic LUT having $I_{LUT} = 6$ inputs. Using multiplexers allows creating LUTs having either 7 or 8 inputs. These resulting LUTs have approximately the same performance as the basic LUTs [17].

The second peculiarity of AMD Xilinx FPGAs is the ability to share the resources of the basic LUT to implement two functions that depend on the same arguments. Therefore, a basic LUT is "split" by two LUTs having $I_{LUT} - 1$ inputs. We denote these two LUTs as a shared LUT (SLUT).

In the LUT-based FSMs, the RG is distributed among flip-flops of CLBs implementing SBF (3). Therefore, the RG is hidden inside the CLBs. This gives the structural diagram of LUT-based Mealy FSM A_1 shown in Figure 3.

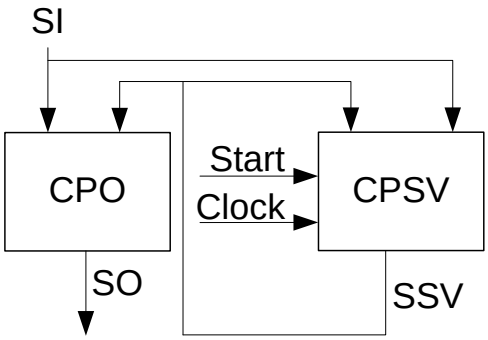

Figure 3. Structural diagram of LUT-based Mealy FSM A_1.

A block CPSV includes the hidden distributed register RG. The pulses *Start* and *Clock* control the operation of the RG. The LUTs of CPSV generate IMFs (3) entering the informational inputs of D flip-flops. As a result, the state variables $v_r \in SSV$ are generated. The block CPO generates outputs $o_w \in O$ represented by (4).

Each Boolean function $f_n \in SM \cup SO$ depends on $A(f_n)$ arguments ($n \in \{1, \ldots, R+W\}$). Our analysis shows that for some FSMs from the library [18], the value of $A(f_n)$ is around 20. However the number of LUT inputs (I_{LUT}) is very small [3]. Therefore, the following condition can take place:

$$A(f_n) > I_{LUT}. \tag{5}$$

If the condition (5) holds, then some serious problems can arise [4,11]. First of all, there is more than a single LUT in a circuit corresponding to a function that satisfies the condition (5). This increases both the number of logic levels and the number of interconnections inside this circuit. In turn, this increases both the propagation time and consumed power compared to an equivalent single-level circuit. Therefore, the fulfillment of the condition (5) has a significant negative effect on the quality of the LUT-based FSM circuit.

3. Related Works

If the condition (5) holds for some function, then the various methods of functional decomposition (FD) [4] are used for executing the step of technology mapping for a corresponding circuit. The decomposed function $f_n \in SM \cup SO$ is represented by a composition of partial functions. The logic circuit for each partial function includes only a single LUT. These LUTs are connected to create a final circuit.

The methods of functional decomposition are discussed in many books and papers, for example in [4,14,19–22]. The FD is a very powerful tool used in the process of technology mapping [4]. If the condition (5) holds, then such a function is broken down into smaller and smaller components. The decomposition is terminated when each component is represented by an SOP having no more than I_{LUT} arguments. The main drawback of FD-based FSM design is the multi-level nature of the produced circuits. In multi-level circuits, it is quite possible that the same inputs $i_u \in SI$ appear on several logic levels. This results in FSM circuits with "spaghetti-type" interconnections.

If the condition (5) is violated, then there are exactly $R+W$ LUTs in an FSM circuit. Otherwise, an FSM circuit is represented by $R+W+|\psi|$ functions, where ψ is a set of partial functions obtained in the process of decomposition. The partial functions are components of functions $f_n \in SM \cup SO$.

The negative effects of FD-based FSM circuits are well known [11]. They are connected with "spaghetti-type" interconnections typical for such circuits. For modern nanoelectronics, as mentioned in [23], "...wire delay has come to dominate logic delay". Therefore, FD-based FSM circuits are much slower than their single-level counterparts. Furthermore, the interconnections are responsible for up to 70% of total power consumption [23]. Due to this, the FD-based FSM circuits consume more power than their single-level counterparts.

To improve the characteristics of FSM circuits, the interconnection system should be improved. This can be performed, for example, by reducing the number of literals in

the sum-of-products (SOPs) representing functions $f_n \in SM \cup SO$. This reducing can be achieved by a proper state assignment [24].

One of the best state assignment algorithms is JEDI [25]. Its main approach for optimization is the following. If for states $s_i, s_j \in SS$, transitions depend on the same inputs $i_g \in SI$, then these states have adjacent codes. Therefore, the codes for such states are combined in generalized cubes having 2^r vertices ($r \in \{1, \ldots, R\}$). Due to this, JEDI reduces the number of literals in the SOPs of (3) and (4). This positive effect can lead to reducing the numbers of LUTs and their levels and interconnections in the corresponding FSM circuits. Therefore, applying JEDI can reduce the LUT count, cycle time, and power consumption of an FSM circuit.

The following conclusion can be made from the analysis of the works [4,26,27]: there is no state assignment approach that is the best for any FSM and for any logic elements used. Depending on the peculiarities of a particular FSM and the logic elements implementing its circuit, the same state assignment method can either improve or degrade some characteristics of the FSM circuit. For example, the maximum binary state assignment produces an FSM circuit with higher power consumption than for circuits based on either Gray or Johnson codes [28]. Next example, if an FSM has many unconditional transitions, then the sequential state assignment optimizes the area better than other state assignment methods [29].

The paper [26] showed the results of a comparison of FSM circuits based on the MB and OH state codes. As follows from [26], the OH state assignment allows improving the circuit characteristics for FSMs with $G > 16$. However, as shown in [4], the circuit characteristics strongly depend on the number of FSM inputs. Moreover, they depend on the number of inputs determining transitions from different states. As shown in [27], if there is $U > 10$, then the maximum binary state codes give better results compared to OH-based FSMs.

Therefore, the best state assignment method does not exist. This fact stimulates the development of new state encoding methods. The more choice a designer has, the higher the probability of finding a method that is most suitable for a particular FSM and available logic elements. One of the possible state assignment methods is suggested in this article. We discuss it in the following section.

The problems of optimizing the characteristics of FSM circuits are discussed in many works, such as, for example, [30–35]. An analysis of these and many other works allowed us to draw the following conclusion. As a rule, reducing the number of LUTs in the circuit leads to performance degradation. If an attempt is made to improve performance, then this is accompanied by LUT count growth. It is possible to reduce the value of the area–time product [13]. However, as before, the improvement of one characteristic leads to the deterioration of the other. Therefore, it would be desirable to propose a method that would simultaneously improve both the LUT count and the performance (the time of cycle) of the FPGA-based FSM circuit.

To optimize a LUT-based FSM circuit, it is necessary to eliminate the direct dependence of outputs and IMFs on FSM inputs $i_u \in SI$. This can be performed using the methods of structural decomposition [11]. The elimination of this dependence can be achieved by introducing some new functions $f_i \in \psi$. They depend on inputs and/or state variables. To optimize an FSM circuit, the following condition should take place:

$$|\psi| \ll W + R. \tag{6}$$

Each system of new functions has unique sets of input and output variables. Each such system determines a separate LUT-based block with its unique systems of input and output variables. If the condition (5) holds, then the total number of LUTs implementing functions $f_i \in \psi$ is significantly less than their total number in the combinational part of an equivalent FSM A_1 (Figure 3). The functions $f_i \in \psi$ are used as arguments of functions (2)

and (3). The total number of LUTs in an FSM circuit is significantly less than it is for the equivalent FSM A_1, if the following condition holds:

$$|\psi| \ll U + R. \tag{7}$$

A survey of methods of structural decomposition can be found in [11]. One of the known methods of structural decomposition is a method of encoding of collections of outputs [11]. The collections CO_q ($q \in \{1, \ldots, Q\}$) are generated during interstate transitions. They create a set $SCO = \{CO_1, \ldots, CO_Q\}$. The q-th CO is encoded by the maximum binary code $K(CO_q)$. The bit width of the code $K(CO_q)$ is determined as

$$R_{CO} = \lceil \log_2 Q \rceil. \tag{8}$$

The COs are encoded using additional variables $b_r \in SOV = \{b_1, \ldots, b_{RCO}\}$.

If the COs are encoded, then the system (4) is represented using two new systems:

$$SOV = SOV(SSV, SI); \tag{9}$$

$$SO = SO(SOV). \tag{10}$$

Using SBFs (3), (9), and (10) leads to Mealy FSM A_2. Its structural diagram is shown in Figure 4.

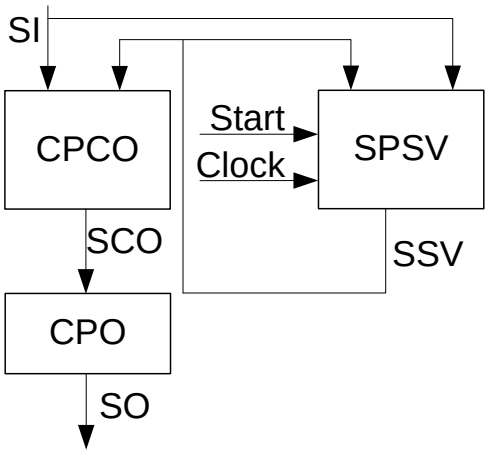

Figure 4. Structural diagram of Mealy FSM A_2.

In FSM A_2, the block of state variables SPSV implements SBF (3). Next, these functions enter the hidden RG. The block CPCO implements SBF (9). As a result, the codes of the COs are created. These codes enter the block CPO, which implements SBF (10).

The following conditions determine the best case for using the model A_2 [36]:

$$A(b_r) \le I_{LUT} \ \ (b_r \in \{b_1, \ldots, b_{RCO}\}); \tag{11}$$

$$R_{CO} \le I_{LUT}. \tag{12}$$

If (11) takes place, then there are exactly R_{CO} LUTs in the circuit of the CPCO. This circuit is single-level. If (12) holds, then there are no more than W LUTs in the circuit of the CPO. The number of LUTs in the CPO may be less than W if some of the functions (10) are represented by only one variable $b_r \in SOV$.

If the conditions (11) or (12) are violated, then the corresponding circuits have more than a single level of LUTs. In this case, it is necessary to apply FD-based methods for implementing these blocks (with all the ensuing negative consequences).

Summarizing this analysis, the following can be noted. All known state encoding methods do not exclude using various methods of functional decomposition to obtain

the final FSM circuit. As a result, they often lead to multilevel FSM circuits having many drawbacks (an increase in the propagation time and consumed power). Mostly, these drawbacks are connected with the spaghetti-type nature of interconnections inherent in FD-based circuits. Furthermore, the existence of the spaghetti-type interconnections significantly complicates the technology mapping process. The main challenge is to propose a state assignment method that allows the regularization of interconnections. The method proposed in our article can be considered as one of the possible answers to this challenge.

In this paper, we propose a method allowing the improvement of the circuit characteristics for LUT-based Mealy FSM A_2. We discuss the situation when: (1) the conditions (11) and (12) are violated and (2) the condition (5) holds for the function (3). To optimize the circuit of the CPO, we propose to use a modification of the known method of mixed encoding of COs [36]. To optimize the circuits of other blocks, we propose a method of two-part state assignment.

4. Main Idea of Proposed Method

To reduce the LUT count in combinational circuits implementing SBFs (3) and (9), we propose to design an FSM circuit using an approach similar to the one used in [24]. However, we propose a new type of state codes, which should replace the rather wide twofold codes used in [24]. We name these codes two-part state codes (TPCs).

To use TPCs, it is necessary to create a partition $PS = \{CP^1, \ldots, CP^K\}$ of the set SS by classes of compatible states $CP^k \in PS$. Each class $CP^k \in PS$ defines a set $SI^k \subseteq SI$ including inputs $i_u \in SI$ determining transitions from states $s_g \in CP^k$. There are U_k elements in the set $SI^k \subseteq SI$. There are G_k states in the k-th class of PS. These states can be encoded by partial codes $PC(s_g)$ having R_k bits:

$$R_k = \lceil \log_2 G_k \rceil. \tag{13}$$

It is enough $R_S = max(R_1, \ldots, R_K)$ variables to encode states inside any class $CP^k \in PS$. Therefore, the same variables are used to encode states as elements of different classes $CP^k \in PS$.

The partition should include the minimum possible number of classes, each of which satisfies the condition

$$U_k + R_k \leq I_{LUT}. \tag{14}$$

This problem can be solved using the approach from [36]. To distinguish the classes $CP^k \in PS$, they should be encoded by class codes $CC(CP^k)$. These codes have R_C bits:

$$R_C = \lceil \log_2 K \rceil. \tag{15}$$

To encode classes, we use class variables from the set $SV_2 = \{v_1, \ldots, v_{RC}\}$. To encode states $s_g \in CP^k$, state variables from the set $SV_1 = \{v_{RC+1}, \ldots, v_{RC+RS}\}$ are used. These variables create a set $SV = SV_1 \cup SV_2$. Therefore, the two-part code $TP(s_g)$ includes R_{TP} bits, where

$$R_{TP} = R_C + R_S. \tag{16}$$

The two-part code $TP(s_g)$ is represented as a concatenation of codes $CC(CP^k)$ and $PC(s_g)$ where $s_g \in CP^k$. If symbol "*" stands for the sign of concatenation, then the code $TP(s_g)$ is represented as

$$TP(s_g) = CC(CP^k) * PC(s_g). \tag{17}$$

To use SLUTs in the circuit of CPO, we propose to represent the set of outputs as two non-overlapping sets. The set SO_{oh} includes outputs represented as (4). Therefore, these outputs are represented by unitary codes. The set SO_{mb} consists of outputs represented as (10). Therefore, its elements are encoded by maximum binary codes. Now, the outputs

are encoded by unitary-maximum (UM) codes. Here, we used the idea from [36], but we modified this approach. We propose to form the set SO_{oh} in a way to fulfill the condition:

$$R_{CO} = I_{LUT} - 1. \tag{18}$$

Now, each class $CP^k \in PS$ determines the four following sets: (1) a partial set of inputs $SI^k \subseteq SI$; (2) a partial set of outputs $SO^k_{oh} \subseteq SO$; (3) a partial set of outputs $SO^k_{mb} \subseteq SO$; (4) a partial set of IMFs $SM^k \subseteq SM$. The partial sets of outputs include FSM outputs generated during transitions from the states $s_g \in CP^k$. The set $SM^k \subseteq SM$ includes the IMFs necessary to load the two-part state codes of the state of transitions.

The partial functions are represented by the following SBFs:

$$SO^k_{oh} = F3(SV_1, SI^k); \tag{19}$$

$$SO^k_{mb} = F4(SV_1, SI^k); \tag{20}$$

$$SM^k = F5(SV_1, SI^k). \tag{21}$$

We propose to use the method from [37] to represent some outputs as single literal functions. These outputs form a set SO^1_{mb}. The remaining outputs form a set $SO^2_{mb} = SO_{mb}/SO^1_{mb}$. Using partial functions (19)–(21) gives the following SBFs:

$$SO_{oh} = F6(SV_2, SO^1_{oh}, \ldots, SO^K_{oh}); \tag{22}$$

$$SOV = F7(SV_2, SO^1, \ldots, SO^K); \tag{23}$$

$$SV = F8(SV_2, SM^1, \ldots, SM^K). \tag{24}$$

To generate outputs $o_w \in SO^2_{mb}$, it is necessary to construct the functions:

$$SO^2_{mb} = F9(SOV). \tag{25}$$

If the condition (18) holds, then these outputs can be combined in pairs. Each pair is implemented by a SLUT.

The SBFs (19)–(25) are the base for designing the FSM A_3 proposed in this article. Its structural diagram is shown in Figure 5.

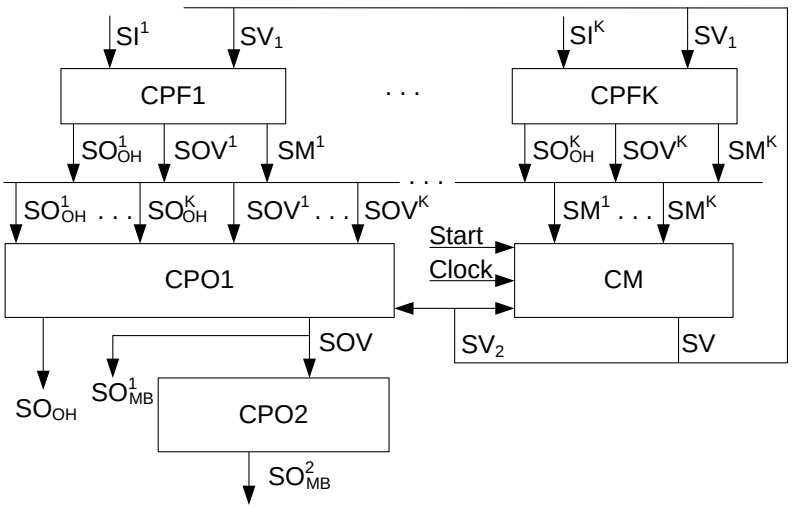

Figure 5. Structural diagram of Mealy FSM A_3.

The structural diagram of FSM A_3 includes three levels of logic blocks. Each of them is implemented using LUT-based CLBs. A block CPFk ($k \in \{1, \ldots, K\}$) implements SBFs (19)–(21). They represent the first logic level of the FSM circuit. This level is responsible

for generating partial functions of outputs and IMFs. A block CPO1 generates SBFs (22) and (23). This block can be implemented using multiplexers created from basic LUTs and dedicated multiplexers of CLBs [17]. In this case, the multiplexer control inputs are connected with the class variables $v_r \in SV_2$, whereas the data inputs are connected with wires corresponding to partial functions. In the best case, some outputs are represented as single literal functions $o_w \in SO^1_{mb}$. A block CM implements SBF (24). Its circuit is implemented in the same way as it is for CPO1. This block includes a hidden register controlled by pulses *Start* and *Clock*. The blocks CPO1 and CM represent the second logic level. Finally, the third logic level is represented by a block CPO2. This block generates outputs $o_w \in SO^2_{mb}$ represented by SBF (25).

As shown from our analysis of the benchmarks [18], the following relation holds for equivalent FSMs A_2 and A_3:

$$R_0 \leq R_{TP} \leq R_0 + 1. \tag{26}$$

Therefore, the replacement of model A_2 FSM by A_3 FSM does not lead to a significant difference in the number of state variables.

Only basic LUTs are used for implementing the second-level circuits if the following condition takes place:

$$R_C + K \leq I_{LUT}. \tag{27}$$

If either the relation $R_C + K = I_{LUT} + 1$ or $R_C + K = I_{LUT} + 2$ is true, then it is enough to have a single CLB to implement functions (19)–(21). In this case, there is only a single level of CLBs in the second-level circuits of FSM A_3. If the relation $R_C + K > I_{LUT} + 2$ takes place, then the circuits of CPO1 and the CM are multi-level.

In this paper, we propose a design method for LUT-based FSMs A_3. We assumed that the STG is an initial form of FSM representation. The proposed method includes the following steps:

1. Constructing a state transition table using the initial STG.
2. Dividing outputs by classes SO_{oh} and SO_{mb}.
3. Creating and encoding of collections of outputs $CO_q \subseteq SO_{mb}$.
4. Creating SBF (25) representing CPO2.
5. Constructing the partition PS with a minimum value of K.
6. Encoding of FSM states $s_g \in SS$ by two-part codes $TP(s_g)$.
7. Creating tables of blocks CPF1–CPFK.
8. Constructing SBFs (19)–(21) representing blocks CPF1–CPFK.
9. Creating tables for blocks CPO1 and CM.
10. Constructing SBFs (22)–(24) representing blocks CPO1 and CM.
11. Implementing the LUT-based circuit of P_{2F} FSM.

The outcome of Step 2 has a significant impact on the number of LUTs (and hence, the occupied chip area) in the blocks CPO1 and CPO2. The construction of the set SO_{oh} must be performed in such a way as to allow using one LUT to implement two FSM outputs. This leads to reducing the LUT count in the circuit of CPO2. Furthermore, it is very important to minimize the cardinality number of the set SO_{oh} to reduce the number of LUTs in the circuit of CPO1. Step 2 can be executed using the method from [24]. The outcome of Step 3 determines the area occupied by the circuit of CPO2. The encoding of the COs should be performed in a way minimizing the number of literals in SBF (25). This allows minimizing the number of interconnections (and hence, the occupied chip area) between the blocks CPO1 and CPO2. Step 3 is executed using the method from [37]. The outcome of Step 5 has a significant effect on the area occupied by the second-level blocks. The resulting partition should have the minimum possible number of classes. To solve this problem, the greedy algorithm [36] could be used. This approach minimizes the value of K, which in turn, makes it possible to satisfy the condition (27). The last step is executed using standard CAD tools such as Vivado [8] or Quartus [38].

5. Example of Synthesis

We discuss a case when CLBs having LUTs with $S_L = 5$ inputs are used for implementing the FSM circuit. We start the synthesis process using an STG (Figure 6) representing Mealy FSM Ex2.

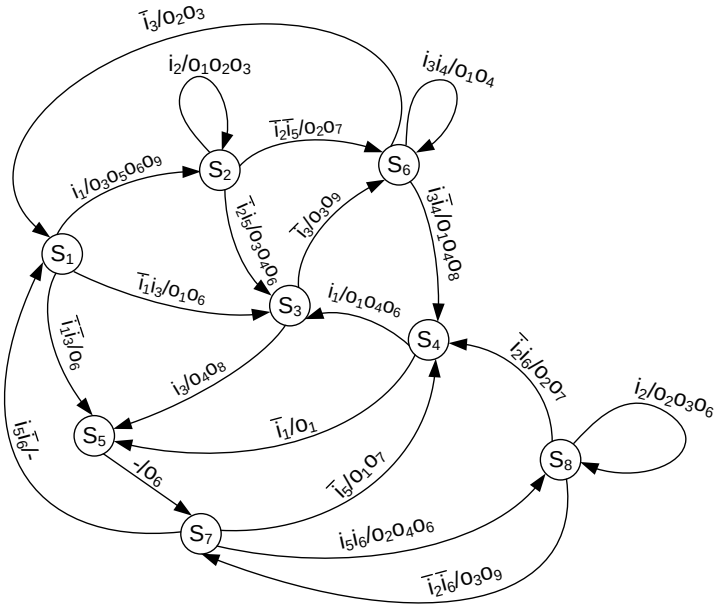

Figure 6. State transition graph of Mealy FSM Ex2.

Step 1. Transformation of an STG into an STT is performed in a trivial way. Starting from STG (Figure 6), we can obtain the STT of FSM Ex2 (Table 3). Table 3 has $H = 20$ rows. Each row corresponds to an arc of an STG (Figure 6).

Table 3. State transition table of FSM Ex2.

s	*ST*	*In*	*Out*	*h*
s_1	s_2	i_1	$o_3o_5o_6o_9$	1
	s_3	$\overline{i}_1 i_3$	o_1o_6	2
	s_5	$\overline{i}_1\,\overline{i}_3$	o_6	3
s_2	s_2	i_2	$o_1o_2o_3$	4
	s_3	$\overline{i}_2 i_5$	$o_3o_4o_6$	5
	s_6	$\overline{i}_2\,\overline{i}_5$	o_2o_7	6
s_3	s_5	i_3	o_4o_6	7
	s_6	$\overline{i}_3$	o_3o_9	8
s_4	s_3	i_1	$o_1o_4o_6$	9
	s_5	$\overline{i}_1$	o_1	10
s_5	s_7	1	o_6	11
s_6	s_6	i_3i_4	o_1o_4	12
	s_4	$i_3\overline{i}_4$	$o_1o_4o_8$	13
	s_1	$\overline{i}_3$	o_2o_3	14
s_7	s_8	i_5i_6	$o_2o_4o_6$	15
	s_1	$i_5\overline{i}_6$	–	16
	s_4	$\overline{i}_5$	o_1o_7	17
s_8	s_8	i_2	$o_2o_3o_6$	18
	s_4	$\overline{i}_2 i_6$	o_2o_7	19
	s_7	$\overline{i}_2\,\overline{i}_6$	o_3o_9	20

Using Table 3, we can derive the following sets: $SS = \{s_1, \ldots, s_8\}$, $SI = \{i_1, \ldots, i_6\}$, and $SO = \{o_1, \ldots, o_9\}$. This gives $G = 8$, $U = 6$, and $W = 9$.

Step 2. During transitions from states of Ex2, there are generated $Q = 18$ collections of outputs. They are represented in the column "Initial" of Table 4.

Table 4. Initial and transformed collections of outputs.

Initial		Transformed	
CO	**q**	**CO**	**q**
$o_3o_5o_6o_9$	2	$o_3o_5o_9$	1
o_1o_6	3	o_1	2
o_6	4	–	3
$o_1o_2o_3$	5	$o_1o_2o_3$	4
$o_3o_4o_6$	6	o_3o_4	5
o_2o_7	7	o_2o_7	6
o_4o_8	8	o_4o_8	7
o_3o_9	9	o_3o_9	8
$o_1o_4o_6$	10	o_1o_4	9
o_1	11	o_1	2
o_1o_4	12	o_1o_4	9
$o_1o_4o_8$	13	$o_1o_4o_8$	10
o_2o_3	14	o_2o_3	11
$o_2o_4o_6$	15	o_2o_4	12
o_1o_7	16	o_1o_7	13
$o_2o_3o_6$	17	o_2o_3	11

Using (8) gives $R_C = 5$. The condition (12) takes place, as well as the relation $R_C = I_{LUT}$. This means we cannot use SLUTs to implement the block CPO2. Therefore, it makes sense to use unitary-maximum encoding of the COs.

Step 3. Using the approach from [36] gives the sets $SO_{oh} = \{o_6\}$ and $SO_{mb} = \{o_1, \ldots, o_5, o_7, o_8, o_9\}$. There are $Q = 13$ transformed COs shown in the column "Transformed" of Table 4.

Now, using (8) gives $R_C = 4$. Applying the encoding method from [37], we can obtain the codes of the COs shown in Figure 7.

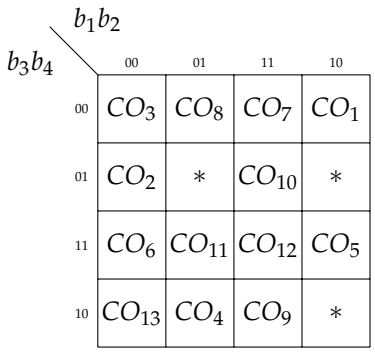

Figure 7. Codes of the COs for FSM Ex2.

Step 4. As follows from Figure 2, the COs are encoded using the elements of the set $SOV = \{b_1, \ldots, b_4\}$. Using the Karnaugh map (Figure 7) and the distribution of the outputs among the COs shown in Table 4, it is possible to create the following SBF:

$$\begin{aligned} &o_1 = b_4;\ o_2 = \overline{b_1} b_2 b_4 \vee \overline{b_2} b_3\ \overline{b_4};\ o_3 = b_2; \\ &o_4 = b_2 b_4 \vee b_1 b_3 \vee b_1 \overline{b_2};\ o_5 = b_1 b_2 \overline{b_3}; \\ &o_7 = \overline{b_1}\ \overline{b_2} b_3;\ o_8 = b_1 \overline{b_2}\ \overline{b_3}\ o_9 = b_2 \overline{b_3}. \end{aligned} \tag{28}$$

This SBF corresponds to SBF (10). It is a base for implementing the circuit of CPO2. As follows from (28), the outputs are represented by the following sets: $SO_{oh} = \{o_6\}$, $SO^1_{mb} = \{o_1, o_3\}$, and $SO^2_{mb} = \{o_2, o_4, o_5, o_7, o_8, o_9\}$. The outputs from the first two sets are generated by CPO1. The outputs $o_w \in SO^2_{mb}$ are generated by CPO2. The circuit of CPO2 is implemented using Boolean formulae from (28).

Step 5. Using the method proposed in [36], we can obtain the partition $PS = \{CP^1, CP^2\}$ where $CP^1 = \{s_1, s_3, s_4, s_6\}$ and $CP^2 = \{s_2, s_5, s_7, s_8\}$. The class $CP^1 \in PS$ determines the following partial sets: $SI^1 = \{i_1, i_3, i_4\}$, $SO^1_{oh} = \{o_6\}$, $SOV^1 = SOV$. The class $CP^2 \in PS$ determines the following partial sets: $SI^2 = \{i_2, i_5, i_6\}$, $SO^2_{oh} = \{o_6\}$, $SOV^2 = SOV$.

Step 6. In the discussed case, there is $K = 2$. Using (15) gives $R_C = 1$ and $SV_2 = \{v_1\}$. Each class of PS includes four states. Using (13) gives $R_1 = R_2 = 2$. Therefore, to encode states, it is enough $R_S = 2$ variables creating the set $SV_1 = \{v_2, v_3\}$. Let us use the following approach to encode the classes: the smaller the class index (k), the smaller the decimal value of this class code is. The same approach is used for encoding of the states. In the case of Ex2, this approach gives the two-part state codes shown in Figure 8.

v_1 \ v_2v_3	00	01	11	10	
0	s_1	s_3	s_4	s_6	$K(CP^1)$
1	s_2	s_5	s_7	s_8	$K(CP^2)$

Figure 8. Two-part state codes for FSM Ex2.

Each row of the Karnaugh map (Figure 8) contains a class of PSs. The variable v_1 create class codes $CC(PC^k)$, which are the following: $CC(PC^1) = 0$, $CC(PC^2) = 1$. Each column of the Karnaugh map (Figure 8) includes identical partial state codes $PC(s_g)$. These codes are created by variables v_2, v_3. The following codes can be found: $PC(s_1) = PC(s_2) = 00$, $PC(s_3) = PC(s_5) = 01$, $PC(s_4) = PC(s_7) = 10$, and $PC(s_6) = PC(s_8) = 11$. Therefore, for example, the two-part code of $s_3 \in IS$ is equal to 001, whereas the two-part code of $s_5 \in IS$ is equal to 101.

Step 7. Using information from the STT (Table 3), the state codes, the codes of the COs, make it possible to create tables of blocks CPF1–CPF2. The table of CPFk reflects interstate transitions from states $s_g \in CP^k$. The h-th row of this table shows a transition $\langle CS, ST \rangle$, where a current state is encoded in the partial code PC(CS) and a state of transition is represented by its two-part code FC(ST). The input signals are shown in the column Ink, the outputs represented by unitary codes shown in the column O^k_{oh}, the variables $b^k_r \in SOV$ shown in Outk, and the partial IMFs shown in the column $Mk(h \in \{1, \ldots, H_k\})$.

In the discussed case, Table 5 represents CPF1 and Table 6 represents the CPF2. The following relation takes place: $H_1 = H_2 = 10$.

Table 5. Table of block CPF1.

CS	PC(CS)	ST	FC(ST)	*In*1	O^1_{oh}	Out1	M1	h
s_1	00	s_2	100	i_1	o_6	b_1b_2	D_1	1
		s_3	001	$\overline{i_1}i_3$	o_6	b_4	D_3	2
		s_5	101	$\overline{i_1}\,\overline{i_3}$	o_6	–	D_1D_3	3
s_3	01	s_6	011	i_3	–	b_1	D_2D_3	4
		s_5	101	$\overline{i_3}$	–	b_2	D_1D_3	5
s_4	10	s_3	001	i_1	o_6	$b_1b_3b_4$	D_3	6
		s_5	101	$\overline{i_1}$	–	b_4	D_1D_3	7
s_6	11	s_6	011	i_3i_4	–	$b_1b_3b_4$	D_2D_3	8
		s_4	010	$i_3\overline{i_4}$	–	b_1b_4	D_2	9
		s_1	000	$\overline{i_3}$	–	b_2b_3	–	10

Table 6. Table of block CPF2.

CS	PC(CS)	ST	FC(ST)	*In*2	O^2_{oh}	Out2	M2	h
s_2	00	s_2	100	i_2	–	$b_2b_3b_4$	D_1	1
		s_3	001	$\overline{i_2}i_5$	o_6	$b_1b_2b_3$	D_3	2
		s_6	011	$\overline{i_2}\,\overline{i_5}$	–	b_3	D_2D_3	3
s_5	01	s_7	110	1	o_6	–	D_1D_2	4
s_7	10	s_8	111	i_5i_6	–	b_1b_3	$D_1D_2D_3$	5
		s_1	000	$i_5\overline{i_6}$	–	–	–	6
		s_4	010	$\overline{i_5}$	–	b_3b_4	D_2	7
s_8	11	s_8	111	i_2	o_6	b_2b_3	$D_1D_2D_3$	8
		s_4	010	$\overline{i_2}i_6$	–	b_3	D_2	9
		s_7	110	$\overline{i_2}\,\overline{i_6}$	–	b_2	D_1D_2	10

Step 8. The tables of CPF1–CPF2 were used to derive the systems (19)–(21). The sum-of-products of these functions consist of product terms created as conjunctions of state variables $v_2, v_3 \in SV_1$ and inputs $i_u \in SI^k (k \in \{1,2\})$.

$$\begin{aligned} D^1_1 &= \overline{v_2}\,\overline{v_3}i_1 \vee \overline{b_2}\,\overline{b_3} \vee v_2\overline{v_3}\,\overline{i_1} = f_1(v_2, v_3, i_1, i_3); \\ D^1_2 &= f_2(v_2, v_3, i_1, i_4); \\ D^1_3 &= f_3(v_2, v_3, i_1, i_3, i_4). \end{aligned} \tag{29}$$

$$\begin{aligned} b^1_1 &= f_4(v_2, v_3, i_1, i_3, i_4); \\ b^1_2 &= f_5(v_2, v_3, i_1, i_3); \\ b^1_3 &= f_6(v_2, v_3, i_1, i_3, i_4); \\ b^1_4 &= f_7(v_2, v_3, i_1, i_3, i_4); \\ o^1_6 &= f_8(v_2, v_3, i_1). \end{aligned} \tag{30}$$

The following SBFs are derived from Table 6:

$$\begin{aligned} D^2_1 &= f_9(v_2, v_3, i_2, i_5, i_6); \\ D^2_2 &= f_{10}(v_2, v_3, i_2, i_5, i_6); \\ D^2_3 &= f_{11}(v_2, v_3, i_2, i_5, i_6). \end{aligned} \tag{31}$$

$$\begin{aligned} b_1^2 &= f_{12}(v_2, v_3, i_2, i_5, i_6); \\ b_2^2 &= f_{13}(v_2, v_3, i_2, i_5, i_6); \\ b_3^2 &= f_{14}(v_2, v_3, i_2, i_5, i_6); \\ b_4^2 &= f_{15}(v_2, v_3, i_2, i_5); \\ o_6^2 &= f_{16}(v_2, v_3, i_2, i_5). \end{aligned} \tag{32}$$

Each function from (29)–(32) is represented as some f_n. In brackets, there are shown arguments used as literals in a particular SOP, for example, the partial function $D_1^1 = f_1$; it depends on four arguments (v_2, v_3, i_1, i_3). We need these data to understand which functions can be implemented by a SLUT.

Step 9. The tables of CPO1 and the CM are organized in the same order. They include columns "Function" and conjunctions of class variables for the classes $CP^1, \ldots, CP^K$. In the discussed case, there are two single-literal conjunctions $(\overline{v_1}, v_1)$. If a particular partial function is generated by CPFk, then there is 1 at the intersection of a row with this function and the column corresponding to this block. Otherwise, these intersections are marked by 0. In the discussed case, Table 7 represents the block CPO1 and Table 8 represents the block CM.

Table 7. Table of block CPO1.

Function	$\overline{v_1}$	v_1
o_6	1	1
b_1	1	1
b_2	1	1
b_3	1	1
b_4	1	1

Table 8. Table of block CM.

Function	$\overline{v_1}$	v_1
D_1	1	1
D_2	1	1
D_3	1	1

Step 10. The table of CPO1 is a base for deriving SBFs (22) and (23). The table of the CM is used to derive SBF (24). In the discussed case, SBF (33) represents the circuit of CPO1 and SBF (34) represents the block CM:

$$\begin{aligned} o_6 &= \overline{v_1} o_6^1 \vee v_1 o_6^2; \\ b_1 &= \overline{v_1} b_1^1 \vee v_1 b_1^2; \\ b_2 &= \overline{v_1} b_2^1 \vee v_1 b_2^2; \\ b_3 &= \overline{v_2} b_3^1 \vee v_1 b_3^2; \\ b_4 &= \overline{v_1} b_4^1 \vee v_1 b_4^2. \end{aligned} \tag{33}$$

$$\begin{aligned} D_1 &= \overline{v_1} D_1^1 \vee v_1 D_1^2; \\ D_2 &= \overline{v_1} D_2^1 \vee v_1 D_2^2; \\ D_3 &= \overline{v_1} D_3^1 \vee v_1 D_3^2. \end{aligned} \tag{34}$$

Step 11. To implement the circuit of FSM Ex2, it is necessary to map SBFs (28)–(34) into LUTs having $I_{LUT} = 5$. Each of these functions is represented by a single-LUT circuit. However, some functions can share the same basic LUT. There are 8 partial functions

generated by CPF1 and 8 partial functions generated by CPF2. CPO1 generates 5 functions, whereas 3 functions are implemented by the CM. The block CPO2 generates six functions. Therefore, the circuit of FSM Ex2 is represented by 30 functions. However, there are 25 LUTs in this circuit (Figure 9).

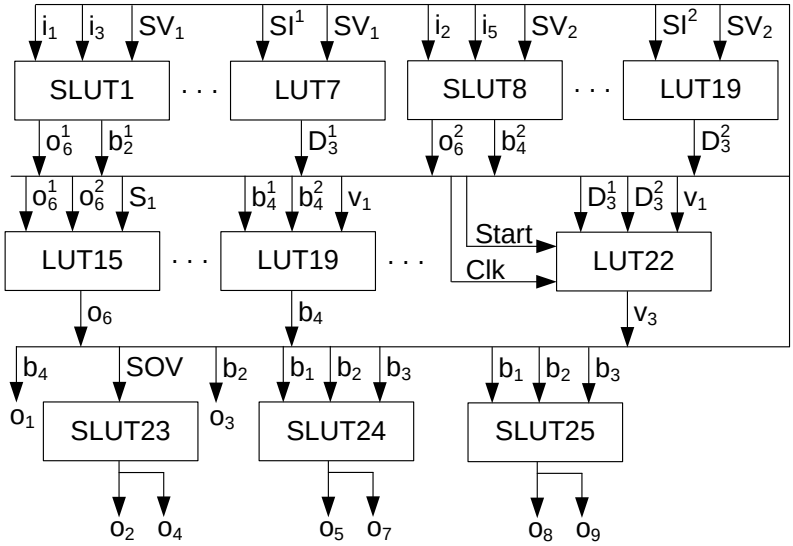

Figure 9. Logic circuit of FSM Ex2.

In this circuit, seven basic LUTs (LUT1–LUT7) implement partial functions (29) and (30). As follows from (30), functions o_6^1 and b_2^1 share the same four arguments. Therefore, they are implemented using SLUT1. Furthermore, functions o_6^2 and b_4^2 can be implemented by a shared LUT. This is SLUT8. Therefore, the circuit of CPF2 contains seven LUTs implementing SBFs (31) and (32). This means there are 14 LUTs on the first level of the logic circuit of FSM Ex2.

The elements LUT15–LUT19 implement SBF (33). The elements LUT20–LUT22 implement SBF (34). This means that there are 5 LUTs in the circuit of CPO1 and 3 LUTs in the circuit of the CM. In total, there are eight LUTs on the second level of the logic circuit of FSM Ex2.

The analysis of SBF (28) shows that three pairs of functions can be implemented using shared LUTs. These pairs are the following: $\langle o_2, o_4 \rangle$, $\langle o_5, o_7 \rangle$, and $\langle o_8, o_9 \rangle$. Therefore, there are three SLUTs on the third level of the logic circuit of FSM Ex2.

Therefore, in the discussed case, using unitary-maximum encoding of outputs allows reducing the number of LUTs on the third logic level. Furthermore, using shared LUTs leads to reducing the number of LUTs on the first logic level. Due to this, the number of basic LUTs is less than the number of generated functions.

To obtain the LUT-based circuit of FSM Ex2, each LUT should be represented by a truth table [15]. This can be performed in a trivial way, so we do not discuss this step for our example. Next, it is necessary to use some industrial CAD tools to execute a step of technology mapping [4].

The considered example is rather simple. It is intended to illustrate the main features of the proposed method. The next section shows the experimental results that allowed evaluating the effectiveness of the proposed method.

6. Experimental Results

To compare the LUT counts and maximum operating frequencies of FSM circuits based on various known state encoding methods and circuits of FSMs produced with the proposed method, we conducted some experiments. Their results are shown in this section. As a base for comparison, we used such methods as: (1) Auto of Vivado [8] (as a method of maximum binary state assignment); (2) One-hot of Vivado [8] (as an example of OH-state

assignment); (3) JEDI of SIS [25] (it is one of the best state assignment algorithms [4]). Furthermore, we used FSMs A_2 [36] as the object for comparison.

To conduct the experiments, we used standard benchmarks from the library LGSynth93 [18]. This library includes 48 benchmarks. The format KISS2 [15] was used for representing benchmark FSMs. These benchmarks have a wide range of such characteristics, such as numbers of states, inputs, and outputs. This library is used by different researchers to compare FSM circuits based on various design methods [14,23,39]. Table 9 has a list of the benchmarks and their main characteristics. The last column of this table includes the values of $U + R_0$ (the summation results for the number of FSM inputs and minimum number of state variables). We used the data from this column to choose benchmarks where our method can be applied.

Table 9. Characteristics of benchmarks from LGSynth93 [18].

Benchmark	U	W	H	G	$U + R_0$
bbara	4	2	60	10	8
bbsse	7	7	56	16	11
bbtas	2	2	24	6	5
beecount	3	4	28	7	6
cse	7	7	91	16	11
dk14	3	5	56	7	6
dk15	3	5	32	4	5
dk16	2	3	108	27	7
dk17	2	3	32	8	5
dk27	1	2	14	7	4
dk512	1	3	15	15	5
donfile	2	1	96	24	7
ex2	2	2	72	19	7
ex3	2	2	36	10	6
ex4	6	9	21	14	10
ex5	2	2	32	9	6
ex6	5	8	34	8	8
ex7	2	2	36	10	6
keyb	7	7	170	19	12
lion	2	1	11	4	4
lion9	2	1	25	9	6
mark1	5	16	22	15	9
mc	3	5	10	4	5
modulo12	1	1	24	12	5
opus	5	6	22	10	9
s27	4	1	34	6	7
s298	3	6	1096	218	11
s386	7	7	64	13	11
s8	4	1	20	5	7
shiftreg	1	1	16	8	4
sse	7	7	56	16	11

We conducted the experiments using the FPGA chip from the Virtex-7 family ($I_{LUT} = 6$). The chip is a part of the VC709 Evaluation Platform (xc7vx690tffg1761-2) [40]. The step of technology mapping was executed by the industrial CAD tool Vivado v2019.1 (64-bit) [41]. The Vivado reports were used to create the tables with the results of the experiments.

In the resulting tables, we show experimental results for 15 of 48 benchmarks [18]. We can explain this choice by the following. If the condition (5) is violated, then there are exactly $R_0 + W$ LUTs in the circuit of Mealy FSM A_1. This circuit includes only a single logic level. Therefore, if the condition (5) is violated, then the LUT-based circuit of Mealy FSM A_1 has the best characteristics of the LUT count (it has a minimum value), operating frequency (it has a maximum value), and power consumption (it has a minimum value). Of course, the further optimization of such a circuit makes no sense.

Our previous research [24,36] showed that the methods of SD can improve the characteristics of LUT-based FSM circuits if the following condition holds:

$$U + R_0 > 2I_{LUT}. \tag{35}$$

In the platform used, there was $I_{LUT} = 6$ (for basic LUTs). Therefore, it makes sense to check the efficiency of the proposed method using benchmarks for which the condition $U + R_0 > 12$ holds. The experimental results are shown in Table 10 (LUT count) and Table 11 (maximum operating frequency, MHz).

There are the following columns in Tables 10 and 11: BFSM (benchmark FSM); $A_1 - MB$ (results of experiments for FSMs with maximum binary state codes); $A_1 - OH$ (results of experiments for FSMs with one-hot state codes); $A_1 - JEDI$ (results of experiments for FSMs with JEDI-based codes); A_2 (results of experiments for FSMs with encoding of COs); A_3 (results of experiments for FSMs proposed in this paper); $U + R_0$. The results of the summation of values from the corresponding columns are shown in the row "Total". The row "Percentage" includes the percentage of summarized characteristics of investigated FSM circuits, respectively, to FSM A_3.

Table 10. Results of experiments (LUT count).

BFSM	$A_1 - MB$	$A_1 - OH$	$A_1 - JEDI$	A_2	A_3	$U + R_0$
ex1	70	74	53	46	42	16
kirkman	42	58	39	37	33	18
planet	131	131	88	85	80	14
planet1	131	131	88	85	80	14
pma	94	94	86	82	76	14
s1	65	99	61	59	57	14
s1488	124	131	108	99	89	15
s1494	126	132	110	96	87	15
s1a	49	81	43	46	44	15
s510	48	48	32	33	28	27
s820	88	82	68	62	52	25
s832	80	79	62	60	54	25
sand	132	132	114	110	98	18
styr	93	120	81	78	74	16
tma	45	39	39	37	33	13
Total	1318	1431	1072	1015	927	
Percentage, %	142.18	154.37	115.64	109.49	100.00	

As follows from Table 10, the application of the proposed approach led to FSM circuits with fewer LUTs than in the LUT-based circuits produced by other investigated methods. Our approach provides the following gain compared to equivalent other FSMs: (1) 42.18% compared with $A_1 - MB$ FSMs, (2) 54.37% compared with $A_1 - OH$ FSMs, (3) 15.64% compared with $A_1 - JEDI$ FSMs, and (4) 9.49% compared with A_2-based FSMs.

Let us remind that this gain was obtained for the benchmarks [18], the technology mapping algorithms of Vivado, and the internal resources of the Virtex-7 family. However, we think that our approach always leads to circuits with better characteristics if the conditions (5) and (12) are violated, whereas the conditions (18) and (27) take place. We can explain why this is the best case for replacing the model A_2 by the proposed model A_3:

1. If some of the functions $f_n \in SOV \cup SM$ satisfy the condition (5), then these functions should be broken down using various FD methods. This increases the number of partial functions. In turn, this increases the numbers of LUTs and their levels in the resulting FSM circuit. In this case, using our approach can help to avoid implementing multi-level circuits.
2. If (12) is violated, then the block CPO1 cannot be implemented by a single-level LUT-based circuit. This has the same consequences, as noted above. Therefore, it is necessary to represent some outputs in the unitary form.
3. If the condition (18) holds, then it is possible to use shared LUTs in the circuit of CPO2. Obviously, this reduces the number of LUTs in CPO2 compared with $|SO^2_{mb}|$.
4. If the condition (27) holds, then each function generated by CPO1 is represented by a single LUT. In this case, there are exactly $|SO_{oh}| + R_{CO}$ LUTs in the circuit of CPO1.

Table 11. Results of experiments (maximum operating frequency, MHz).

BFSM	$A_1 - MB$	$A_1 - OH$	$A_1 - JEDI$	A_2	A_3	$U + R_0$
ex1	150.94	139.76	176.87	192.12	216.42	16
kirkman	141.38	154.00	156.68	177.24	192.23	18
planet	132.71	132.71	187.14	211.42	226.83	14
planet1	132.71	132.71	187.14	211.42	226.83	14
pma	146.18	146.18	169.83	193.16	209.41	14
s1	146.41	135.85	157.16	182.46	198.24	14
s1488	138.50	131.94	157.18	187.44	208.26	15
s1494	149.39	145.75	164.34	193.57	216.32	15
s1a	153.37	176.40	169.17	198.31	220.12	15
s510	177.65	177.65	181.42	187.43	190.21	27
s820	152.00	153.16	176.58	181.21	192.36	25
s832	145.71	153.23	173.78	182.27	190.54	25
sand	115.97	115.97	126.82	142.48	166.42	18
styr	137.61	129.92	145.64	172.11	190.28	16
tma	163.88	147.80	164.14	182.29	221.08	13
Total	2184.41	2173.03	2493.89	2794.93	3065.55	
Percentage, %	71.26	70.89	81.35	91.17	100	

As follows from Table 11, our method makes it possible to obtain LUT-based FSM circuits with a bit higher value of maximum operating frequency than it is for all other investigated methods. If the condition (35) holds, then using the proposed model of Mealy FSM gives the following gain in the maximum operating frequency: (1) 28.74% compared

with $A_1 - MB$ FSMs, (2) 29.11% compared with $A_1 - OH$ FSMs, (3) 18.65% compared with $A_1 - JEDI$ FSMs, and (4) 8.83% compared with A_2-based FSMs.

Note that the gain decreases as the value $U + R_0$ increases. For example, for the benchmark tma $(U + R_0 = 13)$, moving from model A_2 to model A_3 gives a 21.4% increase in frequency. However, for the benchmark s510 $(U + R_0 = 27)$, moving from model A_2 to model A_3 gives only a 1.5% increase in frequency. We believe that this phenomenon is due to the fact that, as the value of $U + R_0$ increases, the difference in the number of logical levels between equivalent A_2- and A_3-based FSMs decreases. This is connected with the violation of the condition (27) for rather complex FSMs (such as s510).

In the experiments, we used the FPGA chip of the Virtex 7 family. All chips of this family included LUTs having $I_{LUT} = 6$ inputs [6]. An FSM designer cannot change this value. Obviously, the number of LUT inputs has a huge impact on the efficiency of the methods used for implementing digital circuits. Let us discuss the influence of this parameter on the characteristics of FSM circuits synthesized using the proposed method.

The more inputs a LUT has, the more likely it is that: (1) the circuits of each block will be single-level and (2) two output signals can be generated by a single LUT of CPO2 (this leads to a decrease in this block area). The more inputs a LUT has, the fewer elements must be removed from the set of FSM outputs to satisfy the condition (12). This leads to reducing the LUT number in the circuit of CPO1. In turn, a decrease in the number of LUT inputs leads to a violation of the conditions (18) and (27). As a result, the number of classes (K) for the partition of the set SS grows. This leads to such negative consequences as: (1) an increase in the number of blocks of the first level of the circuit (that is, its area grows) and (2) the growth in the number of logic levels in the circuits of CPO1 and CPO2. As a result, the number of interconnections increases, as well as the power consumption and the FSM cycle time. Thus, an increase in the number of inputs improves the characteristics of the circuit and a decrease in the number of inputs leads to the degradation of these characteristics. Unfortunately, the FSM designer cannot choose the number of LUT inputs, since this parameter is hard-coded by the LUT architecture used in each defined family [12].

The conducted experiments showed that reducing the FSM circuit area with a simultaneous increase in FSM performance is the main advantage of our method in relation to other investigated methods. This advantage begins to manifest itself, starting with the situation when the total number of FSM inputs (U) and state variables (R_0) is at least twice the number of LUT inputs. This means that the proposed method could be used if the condition (35) holds. Of course, the proposed method has some limitations. If the number of partition classes of the set of states exceeds the number of LUT inputs, then the circuits of both blocks CPO1 and CPO2 become multilevel. This leads to an increase in the number of interconnects, which in turn negatively affects the speed and area of the circuit. The second limitation is related to splitting the set of FSM outputs. If the number of outputs to be removed is significant, then this leads to a sharp increase in the area occupied by the circuit of CPO1. In this case, the reduction in the area of the block CPO2 may be insufficient, that is the total area of the blocks CPO1 and CPO2 will be greater than the area of the block CPO of the equivalent FSM A_2. As can be seen from the research results (Tables 10 and 11), such problems did not arise for the benchmarks used, although these benchmarks are quite complex. If such problems arise, then the proposed method may not be appropriate.

7. Conclusions

The reduction of the chip area occupied by an FSM circuit is one of the basic problems associated with FPGA-based design. In the case of LUT-based FSMs, the chip area is proportional to the number of LUTs in a particular circuit (LUT count of a circuit). This can be performed by decreasing the number of literals in the SOPs of Boolean functions representing an FSM circuit. In this paper, we propose to solve this problem by using two methods of structural decomposition (unitary-maximum binary representation of outputs and two-part state codes).

The proposed method is aimed at improving the area characteristics of LUT-based Mealy FSMs with encoding of the collections of outputs [36]. These FSMs are characterized by three-level logic circuits. If the number of arguments in Boolean functions representing an FSM circuit exceeds the number of basic LUT inputs, then the number of levels can be greatly increased. We propose to avoid such a phenomenon due to using methods of structural decomposition.

Simultaneous use of unitary-maximum codes of the outputs and two-part state codes allows improving the characteristics of LUT-based FSM circuits based on maximum encoding of both collections of outputs and FSM states. As a result, for rather complex FSMs, two of their basic characteristics are improved. Compared to FSMs A_2, the proposed approach improves the LUT count (on average, by 9.49%) and maximum operating frequency (on average, by 8.73%).

Therefore, for rather complex FSMs, the proposed approach allows improving the LUT count and maximum operating frequency. We think the proposed design method can be successfully used in implementing FPGA-based Mealy FSM circuits.

The proposed approach can be used to optimize LUT-based FSM circuits based on any structural decomposition method. Obviously, in each specific case, the proposed method should be modified taking into account the features of the initial models of FSM used. This determines the direction of our further research. In addition, we plan to use the two-part state assignment to optimize circuits of LUT-based Moore FSMs.

Author Contributions: Conceptualization, A.B., L.T. and M.M.; methodology, A.B., L.T. and M.M.; software, A.B., L.T. and M.M.; validation, A.B., L.T. and M.M.; formal analysis, A.B., L.T. and M.M.; investigation, A.B., L.T. and M.M.; writing—original draft preparation, A.B., L.T. and M.M.; 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
CO	collection of outputs
DST	direct structure table
FD	functional decomposition
FPGA	field-programmable gate array
FSM	finite state machine
IMF	input memory function
LUT	look-up table
MB	maximum binary
OH	one-hot
RG	state code register
SBF	systems of Boolean functions
SD	structural decomposition
STG	state transition graph
STT	state transition table

References

1. Baillieul, J.; Samad, T. (Eds.) *Encyclopedia of Systems and Control*; Springer: London, UK, 2015. [CrossRef]
2. Barkalov, A.; Titarenko, L.; Mazurkiewicz, M. *Foundations of Embedded Systems*; Studies in Systems, Decision and Control; Springer International Publishing: Cham, Switzerland, 2019; Volume 195.
3. Trimberger, S. *Field-Programmable Gate Array Technology*; Springer: New York, NY, USA, 2012.
4. Kubica, M.; Opara, A.; Kania, D. *Technology Mapping for LUT-Based FPGA*; Springer: Dordrecht, The Netherlands, 2021. [CrossRef]
5. Altera. Available online: http://www.altera.com (accessed on 31 August 2022).

6. Xilinx. Available online: http://www.xilinx.com (accessed on 31 August 2022).
7. Gazi, O.; Arli, A.C. *State Machines Using VHDL: FPGA Implementation of Serial Communication and Display Protocols*; Springer: Berlin, Germany, 2021. [CrossRef]
8. Vivado. Available online: https://www.xilinx.com/products/design-tools/vivado.html (accessed on 31 August 2022).
9. Trimberg, S. Three ages of FPGA: A Retrospective on the First Thirty Years of FPGA Technology. *IEEE Proc.* **2015**, *103*, 318–331. [CrossRef]
10. Ruiz-Rosero, J.; Ramirez-Gonzalez, G.; Khanna, R. Field Programmable Gate Array Applications—A Scientometric Review. *Computation* **2019**, *7*, 63. [CrossRef]
11. Barkalov, A.; Titarenko, L.; Krzywicki, K. Structural Decomposition in FSM Design: Roots, Evolution, Current State—A Review. *Electronics* **2021**, *10*, 1174. [CrossRef]
12. Kuon, I.; Tessier, R.; Rose, J. FPGA Architecture: Survey and Shallenges—Found Trends. *Electr. Des. Autom.* **2008**, *2*, 135–253.
13. Islam, M.M.; Hossain, M.; Shahjalal, M.; Hasan, M.K.; Jang, Y.M. Area-Time Efficient Hardware Implementation of Modular Multiplication for Elliptic Curve Cryptography. *IEEE Access* **2020**, *8*, 73898–73906 [CrossRef]
14. 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]
15. De Micheli, G. *Synthesis and Optimization of Digital Circuits*; McGraw–Hill: New York, NY, USA, 1994.
16. Baranov, S. *Logic and System Design of Digital Systems*; TUT Press: Tallinn, Estonia, 2008.
17. Chapman, K. *Multiplexer Design Techniques for Datapath Performance with Minimized Routing Resources*; Xilinx All Programmable: 2014; pp. 1–32. Available online: https://www.eeweb.com/wp-content/uploads/articles-app-notes-files-multiplexer-design-techniques-for-datapath-performance-1348763550.pdf (accessed on 31 August 2022).
18. LGSynth93. International Workshop on Logic Synthesis Benchmark Suite (LGSynth93). Available online: https://people.engr.ncsu.edu/brglez/CBL/benchmarks/LGSynth93/LGSynth93.tar (accessed on 31 August 2022).
19. Rawski, M.; Selvaraj, H.; Luba, T.; Szotkowski, P. Application of symbolic functional decomposition concept in FSM implementation targeting FPGA devices. In Proceedings of the Sixth International Conference on Computational Intelligence and Multimedia Applications (ICCIMA'05), Las Vegas, NV, USA, 16–18 August 2005; pp. 153–158. [CrossRef]
20. Jóźwiak, L.; Chojnacki, A. Effective and efficient FPGA synthesis through functional decomposition based on information relationship measures. In Proceedings of the Euromicro Symposium on Digital Systems Design, Warsaw, Poland , 4–6 September 2001; pp. 30–37. [CrossRef]
21. Kubica, M.; Kania, D. Area-oriented technology mapping for LUT-based logic blocks. *Int. J. Appl. Math. Comput. Sci.* **2017**, *27*, 207–222. [CrossRef]
22. Zgheib, G.; Ouaiss, I. Enhanced Technology Mapping for FPGAs with Exploration of Cell Configurations. *J. Circuits Syst. Comput.* **2015**, *24*, 1550039. [CrossRef]
23. Feng, W.; Greene, J.; Mishchenko, A. Improving FPGA Performance with a S44 LUT Structure. In *FPGA'18, Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 25–27 February 2018*; Association for Computing Machinery: New York, NY, USA, 2018; pp. 61–66. [CrossRef]
24. Barkalov, A.; Titarenko, L.; Mielcarek, K. Hardware reduction for LUT-based Mealy FSMs. *Int. J. Appl. Math. Comput. Sci.* **2018**, *28*, 595–607. [CrossRef]
25. Sentowich, E.; Singh, K.; Lavango., L.; Moon, C.; Murgai, R.; Saldanha, A.; Savoj, H.; Stephan, P,R.; Bryton, R.; Sangiovanni-Vincentelli, A. *SIS: A System for Sequential Circuit Synthesis*; Technical Report; University of California: Berkely, CA, USA, 1992.
26. Skliarova, I.; Sklyarov, V.; Sudnitson, A. *Design of FPGA-Based Circuits Using Hierarchical Finite State Machines*; TUT Press: Tallinn, Estonia, 2012.
27. Sklyarov, V. Synthesis and Implementation of RAM-based Finite State Machines in FPGAs. In *Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing*; Hartenstein, R.W., Grünbacher, H., Eds.; Springer: Berlin/Heidelberg, Germany, 2000; pp. 718–727. [CrossRef]
28. Nahiyan, A.; Farahmandi, F.; Mishra, P.; Forte, D.; Tehranipoor, M. Security-Aware FSM Design Flow for Identifying and Mitigating Vulnerabilities to Fault Attacks. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2019**, *38*, 1003–1016. [CrossRef]
29. Jimenez, J.; Trojman, L.; Procel, L.M. Power and Area Reduction of MD5 based on Cryptoprocessor Using novel approach of Internal Counters on the Finite State Machine. In Proceedings of the 2019 IEEE Fourth Ecuador Technical Chapters Meeting (ETCM), Guayaquil, Ecuador, 11–15 November 2019; pp. 1–4. [CrossRef]
30. Barkalov, A.; Titarenko, L.; Krzywicki, K.; Saburova, S. Improving Characteristics of LUT-Based Mealy FSMs with Twofold State Assignment. *Electronics* **2021**, *10*, 901. [CrossRef]
31. Solov'ev, V. Implementation of finite-state machines based on programmable logic ICs with the help of the merged model of Mealy and Moore machines. *J. Commun. Technol. Electron.* **2013**, *58*, 172–177. [CrossRef]
32. Park, J.; Yoo, H. Area-Efficient Differential Fault Tolerance Encoding for Finite State Machines. *Electronics* **2020**, *9*, 1110. [CrossRef]
33. Kubica, M.; Kania, D. Technology Mapping of FSM Oriented to LUT-Based FPGA. *Appl. Sci.* **2020**, *10*, 3926. [CrossRef]
34. Klimowicz, A.; Salauyou, V. State Merging and Splitting Strategies for Finite State Machines Implemented in FPGA. *Appl. Sci.* **2022**, *12*, 8134. [CrossRef]
35. Solov'ev, V.V. Synthesis of Fast Finite State Machines on Programmable Logic Integrated Circuits by Splitting Internal States. *Int. J. Comput. Syst. Sci.* **2022**, *61*, 360–371. [CrossRef]

36. 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. [CrossRef]
37. Achasova, S. *Synthesis Algorithms for Automata with PLAs*; M: Soviet Radio: Moscow, Russia, 1987.
38. Quartus II. Available online: https://www.intel.com/content/www/us/en/programmable/downloads/software/quartus-ii-we/121.html (accessed on 31 August 2022).
39. Kubica, M.; Opara, A.; Kania, D. Logic Synthesis for FPGAs Based on Cutting of BDD. *Microprocess. Microsyst.* **2017**, *52*, 173–187. [CrossRef]
40. Xilinx. VC709 Evaluation Board for the Virtex-7 FPGA. Available online: https://www.xilinx.com/support/documentation/boards_and_kits/vc709/ug887-vc709-eval-board-v7-fpga.pdf (accessed on 31 August 2022).
41. Xilinx. 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 31 August 2022).