electronics

MDPI

Review

Structural Decomposition in FSM Design: Roots, Evolution, Current State—A Review

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

Abstract: The review is devoted to methods of structural decomposition that are used for optimizing characteristics of circuits of finite state machines (FSMs). These methods are connected with the increasing the number of logic levels in resulting FSM circuits. They can be viewed as an alternative to methods of functional decompositions. The roots of these methods are analysed. It is shown that the first methods of structural decomposition have appeared in 1950s together with microprogram control units. The basic methods of structural decomposition are analysed. They are such methods as the replacement of FSM inputs, encoding collections of FSM outputs, and encoding of terms. It is shown that these methods can be used for any element basis. Additionally, the joint application of different methods is shown. The analysis of change in these methods related to the evolution of the logic elements is performed. The application of these methods for optimizing FPGA- based FSMs is shown. Such new methods as twofold state assignment and mixed encoding of outputs are analysed. Some methods are illustrated with examples of FSM synthesis. Additionally, some experimental results are represented. These results prove that the methods of structural decomposition really improve the characteristics of FSM circuits.

Keywords: finite state machine; synthesis; microprogram control unit; logic elements; structural decomposition; PROM; PLA; PAL; CPLD; FPGA

Citation: Barkalov, A.; Titarenko, L.; Krzywicki, K. Structural Decomposition in FSM Design: Roots, Evolution, Current State—A Review. *Electronics* **2021**, *10*, 1174. https://doi.org/10.3390/electronics10101174

Academic Editors: Paolo Colantonio and Alessandro Cidronali

Received: 14 April 2021
Accepted: 12 May 2021
Published: 14 May 2021

1. Introduction

The development of information technologies has led to the widespread use of various digital systems in different areas of mankind's activity [1–9]. It is known that digital systems consist of various combinational and sequential blocks [10,11]. As a rule, the circuits of combinational blocks are regular [12]. A designer can use standard library elements of computer-aided design (CAD) systems to implement such circuits [11]. For example, a multi-bit adder can be represented as a composition of standard single-bit adders. The sequential blocks could be very complex (for example, control units of computers) or rather simple (such as binary counters). It is known that the circuits of complex sequential blocks are irregular [10,12]. As a rule, there are no standard library solutions for such blocks. It means that each sequential block is synthesised anew. To synthesise the logic circuit of a sequential block, some tools are used to present the law of its behaviour.

Very often, the behaviour of sequential blocks is represented using the model of a finite state machine (FSM) [10,13,14]. Three characteristics of an FSM circuit significantly influence the characteristics of a digital system. These characteristics are the hardware amount, the operating frequency (the performance), and the power consumption. Because

of it, there is continuous interest in developing the various approaches that aimed at optimizing the basic characteristics of FSM circuits. As a rule, the less hardware is consumed by a sequential block's circuit, the less power it requires [15–20]. Accordingly, it is very important to reduce the amount of hardware that is consumed by an FSM circuit.

The development of various methods of optimizing the characteristics of the FSM circuits has been started since 1951. A characteristic feature of such methods is the consideration of the characteristics of the logic elements that are used for the design of FSM circuits. At various times, various logic elements were used for implementing FSM circuits. Among these elements, there are logic gates, decoders, multiplexors, read-only memories (ROMs), programmable ROMs (PROMs), programmable logic arrays (PLAs), programmable array logic (PAL), complex programmable logic devices (CPLDs), and field-programmable gate arrays (FPGAs). Some of these logic elements have been used together.

The structural decomposition is one of the approaches used for reducing the hardware amount [21–24]. The roots of this approach go back to 1951, when M. Wilkes put forward the idea of a microprogram control unit (MCU) [25,26]. Over the following times, Wilkes' ideas were modified with a change in the elemental basis used for implementing FSM circuits.

The main idea of the structural decomposition is the following. An FSM circuit is represented by some big logic blocks. Each such a block has its own unique input variables and output functions. The outputs of some blocks are used as the inputs of other blocks. This allows for eliminating the direct connection between FSM inputs and outputs. In the best case, the logic circuit of each block has exactly a single level of logic elements [21,27,28]. In this article, we present a rather brief survey of the known methods of structural decomposition. At the same time, almost half of the article is devoted to the methods of structural decomposition used for optimizing the circuits of FPGA-based FSMs.

The main contribution of this paper is a survey of methods of structural decomposition of FSM circuits. The analysis of these methods shows that the structural decomposition is a powerful tool that allows for significantly improving the characteristics of FSM circuits as compared to their counterparts based on other known approaches.

The rest of the paper is organized as the following. Section 2 presents the theoretical background of finite state machines. Section 3 discusses the methods of implementing microprogram control units. Section 4 presents the methods of structural decomposition used in application-specific integratedcircuits. Section 5 considers the methods targeting simple programmable logic devices. Section 6 considers the structural decomposition of FPGA-based FSMs. A brief conclusion ends the paper.

2. Implementing Circuits of Finite State Machines

An FSM can be defined as a tuple $A = \langle S, I, O, \delta, \lambda, s_1 \rangle$ [10,13], where $S = \{s_1, \ldots, s_M\}$ is a set of internal states, $I = \{i_1, \ldots, i_L\}$ is a set of inputs, $O = \{o_1, \ldots, o_N\}$ is a set of outputs, δ is a transition function, λ is a function of output, and $s_1 \in S$ is an initial state. An FSM can be represented using such tools as: state transition graphs [10], binary decision diagrams [29], and-inverter graphs [30], graph-schemes of algorithms [13].

The most obvious way to represent an FSM is the state transition graph. For example, the STG that is shown in Figure 1 represents a Mealy FSM A_1.

The FSM states are represented by the nodes $s_1, \ldots, s_4$. The arcs define the interstate transitions that are determined by the input signals that are the conjunctions of inputs (or their complements). These conjunctions are written above the arcs together with the outputs generated during these transitions. Using STG (Figure 1), we can find the following parameters of Mealy FSM A_1: the number of inputs $L = 3$, the number of outputs $N = 5$, the number of states $M = 4$, and the number of transitions $H = 8$. Additionally, this STG uniquely defines the functions of transitions and output of FSM A_1.

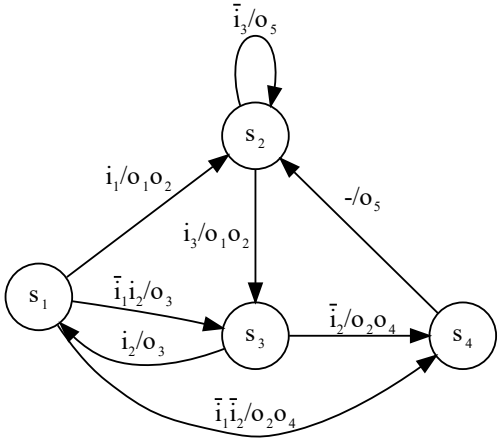

Figure 1. The state transition graph of Mealy finite state machine (FSM) A_1.

To design an FSM circuit, an STG should be transformed into the corresponding STT. An STT includes the following columns [10,13]: s_C is a current state; s_T is a state of transition; I_h an input signal determining a transition from s_C to s_T; and, O_h is a subset of the set of outputs generated during the transition from the current state s_C to the state of transition s_T. We name this subset a collection of outputs. The numbers of the transitions ($h \in \{1, \ldots, H\}$) are shown in the last column of the STT.

In this article, we mostly use STTs for initial representation of FSMs. For example, the FSM A_1 is represented by the STT (Table 1). We hope that there is the transparent connection between the STG (Figure 1) and STT (Table 1).

Table 1. State transition table (STT) of Mealy finite state machine (FSM) A_1.

s_C	s_T	I_h	O_h	h
s_1	s_2	i_1	o_1o_2	1
	s_3	$\bar{i}_1 i_2$	o_3	2
	s_4	$\bar{i}_1\,\bar{i}_2$	o_2o_4	3
s_2	s_3	i_3	o_1o_2	4
	s_2	$\bar{i}_3$	o_5	5
s_3	s_1	i_2	o_3	6
	s_4	$\bar{i}_2$	o_2o_4	7
s_4	s_2	1	o_5	8

There are two main types of FSM, namely, Mealy [31] and Moore [32] FSMs. The first of them was proposed in 1955 by G. Mealy; the second was proposed in 1956 by E. Moore. In both cases, the function δ determines the states of transition as functions depending on the current states and inputs. So, it is the following function:

$$\delta(A, X) = A. \tag{1}$$

For Mealy FSMs, the function λ determines the outputs as functions depending on the current states and inputs. It gives the following function:

$$\lambda(A, X) = O. \tag{2}$$

For Moore FSMs, the function λ determines the outputs as functions depending only on the current states. So, it is the following function:

$$\lambda(A) = O. \tag{3}$$

The difference among (2) and (3) leads to a difference in the synthesis methods of Mealy and Moore FSMs. We now explain the stages of Mealy FSM's synthesis starting from Table 1.

In 1965, Viktor Glushkov proved a theorem of the structural completeness [33]. According to this theorem, an FSM circuit is represented as a composition of the combinational part and the memory. The memory is necessary for keeping the history of the FSM's operation. The history is represented by FSM internal states. This fundamental approach is still widely used for the synthesis of FSM circuits [34–38].

An FSM logic circuit is represented by some systems of Boolean functions (SBFs) [10,13]. To find these SBFs for Mealy FSMs, it is necessary to [13]: (1) encode states $s_m \in S$ by binary codes $K(s_m)$; (2) construct sets of state variables $T = \{T_1, \ldots, T_R\}$ and input memory functions (IMFs) $D = \{D_1, \ldots, D_R\}$; and, (3) transform an initial STT into a direct structure table (DST). The states $s_m \in S$ are encoded during the step of state assignment [10].

The minimum possible number of state variables R_S is determined as

$$R_S = \lceil log_2 M \rceil. \tag{4}$$

The approach based on (4) defines so-called maximum binary codes [10]. This method is used, for example, in the well-known academic system SIS [39]. However, the number of state variables can be different from (4). For example, the one-hot state codes with $R = M$ are used in the academic system ABC [30,40] of Berkeley. The maximum binary codes and one-hot codes define the extreme points of the encoding space. There are other approaches for state assignment where the following relation holds: $\lceil log_2 M \rceil \leq R_S \leq M$.

A state register (RG) keeps the state codes. The register includes R memory elements (flip-flops) having shared inputs of synchronization (*Clock*) and reset (*Start*). Very often, master–slave D flip-flops are used to organize state registers [41,42]. The pulse *Clock* allows the functions $D_r \in D$ to change the RG content.

After the execution of the state assignment, we should create a direct structure table. A DST includes all of the columns of an STT and three additional columns. These columns include the current state codes $K(s_C)$ and the codes $K(s_T)$ of the states of transitions. Finally, a column Φ_h includes the symbols $D_r \in D$ corresponding to 1's in the code $K(s_T)$ from the row h of a DST ($h \in \{1, \ldots, H\}$). A DST is a base to construct the following SBFs:

$$D = D(T, I); \tag{5}$$

$$Y = Y(T, I). \tag{6}$$

The systems (5) and (6) determine a structural diagram of P Mealy FSM (Figure 2) [42].

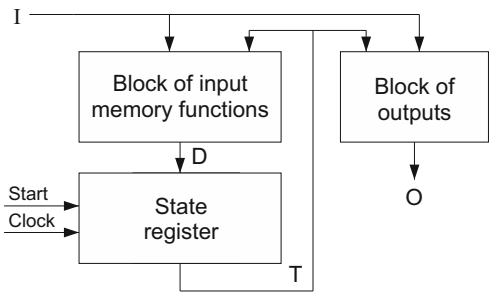

Figure 2. Structural diagram of *P* Mealy FSM.

The block of input memory functions generates the functions (5). The block of outputs generates the system (6). The pulse *Start* loads the code of the initial state to RG. The pulse of synchronization *Clock* allows information to be written to the register.

A DST of Moore FSM is a base for deriving the systems (5) and

$$O = O(T). \tag{7}$$

A P Moore FSM is represented by a structural diagram that is similar to the one shown in Figure 2. However, as follows from SBF (7), there is no connection between the inputs $i_l \in I$ and block of outputs.

We now discuss how to obtain systems (5) and (6) for P Mealy FSM A_1. There is $M = 4$. Using (4) gives the value of $R_S = 2$. This determines the sets $T = \{T_1, T_2\}$ and $D = \{D_1, D_2\}$. Let us encode states in the trivial way: $K(s_1) = 00, \ldots, K(s_4) = 11$. Having state codes allows transforming Table 1 (the initial STT) to Table 2. Table 2 is the DST of P FSM A_1.

Table 2. Direct structure table (DST) of Mealy FSM A_1.

s_C	$K(s_C)$	s_T	$K(s_T)$	I_h	O_h	D_h	h
s_1	00	s_2	01	i_1	o_1o_2	D_2	1
		s_3	10	$\overline{i_1}i_2$	o_3	D_1	2
		s_4	11	$\overline{i_1}\,\overline{i_2}$	o_2o_4	D_1D_2	3
s_2	01	s_3	10	i_3	o_1o_2	D_1	4
		s_2	01	$\overline{i_3}$	o_5	D_2	5
s_3	10	s_1	00	i_2	o_3	-	6
		s_4	11	$\overline{i_2}$	o_2o_4	D_1D_2	7
s_4	11	s_2	01	1	o_5	D_2	8

To fill the column D_h, we should take into account that the value of $D_r \in D$ is equal to the value of the r-th bit of code $K(s_T)$ [13]. Systems (5) and (6) are represented as a sum-of-products (SOPs) [10,43]. These SOPs include product terms $F_h \in F$ corresponding to rows of a DST. The elements of the set of terms F are determined as

$$F_h = S_C \wedge I_h \quad (h \in 1, \ldots, H). \tag{8}$$

In (8), the first member S_C is a conjunction of state variables corresponding to a code of the current state $K(s_C)$ from the h-th row of DST. There are the following conjunctions S_C in the discussed case: $S_1 = \overline{T_1}\,\overline{T_2}, \ldots, S_4 = T_1T_2$.

Using Table 2, we can obtain the following SBFs:

$$\begin{aligned} &o_1 = F_1 \vee F_4 = \overline{T_1}\,\overline{T_2}\,i_1 \vee \overline{T_1}\,T_2\,i_3; \\ &o_2 = F_1 \vee F_3 \vee F_4 \vee F_7; \\ &o_3 = F_2 \vee F_6; \quad o_4 = F_3 \vee F_7; \\ &o_5 = F_5 \vee F_8 = \overline{T_1}\,T_2\,\overline{i_3} \vee T_1\,T_2. \end{aligned} \tag{9}$$

$$\begin{aligned} &D_1 = [F_2 \vee F_3] \vee F_4 \vee F_7 = \overline{T_1}\,\overline{T_2}\,\overline{i_1} \vee\ \overline{T_1}\,T_2\,i_3 \vee T_1\,\overline{T_2}\,\overline{i_1}; \\ &D_2 = F_1 \vee F_3 \vee F_5 \vee F_7 \vee F_8. \end{aligned} \tag{10}$$

The SBF (9) determines the circuit of block of outputs and the SBF (10) determines the circuit of block of input memory functions.

The hardware amount in an FSM circuit depends on the combination of SBF characteristics (the numbers of literals, functions, and product terms of SOPs) and specifics of the used logic elements (the number of inputs, outputs and product terms). Denote, by $NA(f_i, F_h)$, the number of literals in a term F_h of the SOP of a function f_i, and, by $NT(f_i)$,

the number of terms in a SOP of this function. Obviously, the following conditions are true for a SOP of any function $f_i \in D \cup O$:

$$NA(f_i, F_h) \leq L + R_S; \tag{11}$$

$$NT(f_i) \leq H. \tag{12}$$

Consider the SOP of function D_1 from SBF (10). Each term of this SOP includes $NA(D_1, F_h) = 3$ literals. There are $NT(D_1) = 3$ terms in this SOP. If NAND gates having $NI_{NAND} = 3$ inputs are used for implementing a logic circuit corresponding to D_1, then there are four gates and two levels of gates in the circuit. This is the best solution, because the circuit includes the minimum possible number of gates (the minimum hardware amount), their levels (the maximum operating frequency), and interconnections.

However, if there is $NI_{NAND} = 2$, then the SOP should be transformed. After the transformation, the SOP is represented by the following formula:

$$D_1 = \overline{\overline{T_1}\ \overline{T_2}\ \overline{i_1} \cdot \overline{T_1}\ T_2\ i_3 \cdot T_1\ \overline{T_2}\ \overline{i_1}} \tag{13}$$

Twelve gates are necessary for implementing the function (13). The resulting circuit has six levels of gates. Thus, an imbalance between the characteristics of the function and logic elements leads to an increase in the number of gates and levels of logic in the resulting logic circuit.

This situation can occur for any logical elements (logic gates, ROMs, PROMs, PLAs, PALs, CPLDs, FPGAs, and so on). In this case, it is necessary to optimize the characteristics of a resulting logic circuit. The structural decomposition is one of the ways for such an optimization [21].

3. Roots of Structural Decomposition

The control units' circuits of the first computers were characterized by an irregular structure [44–48] with all the ensuing consequences. In 1951, Professor of Cambridge M. Wilkes proposed a principle of microprogram control [25,26]. According to this principle, each computer instruction is represented as a microprogram kept into a special control memory (CM). A microprogram consists of microinstructions. Each microinstruction has an operational part with control outputs (microoperations) and an address part having data used for generating an address of transition (the address of the next microinstruction to be executed). A special register is used to keep the microinstruction address. This approach allows for obtaining a microprogram control unit (MCU) with a regular circuit, which is quite simple to implement and test. A trivial structural diagram of the MCU is shown in Figure 3.

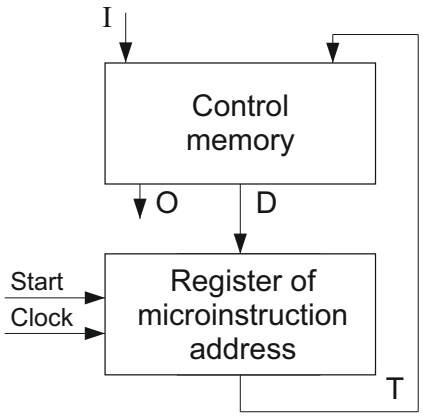

Figure 3. Trivial implementation of microprogram control unit.

The MCU (Figure 3) uses the microinstruction address from the register and logical conditions (inputs) $i_l \in I$ to generate outputs $o_n \in O$ and the next address represented by variables $T_r \in T$. A comparison of Figures 2 and 3 shows that the MCU is a finite state machine in which blocks of input memory functions and outputs are replaced by the control memory. At the same time, microinstructions correspond to FSM states; microinstruction addresses correspond to state codes. This connection between FSMs and MCUs was first noted in [49].

The circuit of control memory was implemented using ROM [50–52]. For MCU (Figure 3), the required volume of such a ROM, V_{ROM}, is determined as

$$V_{ROM} = (N + R_S)2^{L+R_S}. \quad (14)$$

For average FSMs [13], there is $N = 50, R_S = 8, L = 30$. If such a control unit is implemented as MCU (Figure 3), then it is necessary for 10^{13} bits of control memory. In the 1950s, the use of such a big control memory would lead to a significant increase in the cost of a computer. Because of it, Figure 3 rather shows an idea of MCU, not the practical way of its implementation.

In order to diminish the required value of V_{ROM}, two approaches have been proposed by M. Wilkes. The first of them is the selection of an input that should be used for generation of the transition address. As a rule, only a single logic condition is selected in each cycle of MCU operation. This allows for reducing the length of the address part of microinstruction. This approach leads to a two-level MCU shown in Figure 4.

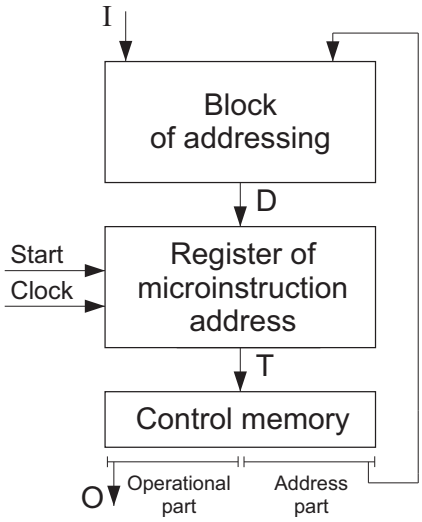

Figure 4. Two-level microprogram control unit.

The second approach is an encoding of collections of microoperations $Y_q \subseteq O$ by maximum binary codes $K(Y_q)$ having R_Q bits. In practical cases [53], there is $R_Q \leq 8$. This allows reducing the length of the operational part of microinstruction up to

$$R_Q = \lceil log_2 Q \rceil. \quad (15)$$

In (15), we use Q to denote the number of different COs for a particular STT.

If an MCU is implemented starting from STT (Table 1), then the following collections of outputs (COs) can be found: $Y_1 = \{o_1, o_2\}$, $Y_2 = \{o_3\}$, $Y_3 = \{o_2, o_4\}$, $Y_4 = \{o_5\}$. Accordingly, there is $Q = 4$. Using (15) gives $R_Q = 2$. Let us use elements of the set $Z = \{z_1, \ldots, z_{RQ}\}$ for encoding of the COs. It gives the set $Z = \{z_1, z_2\}$. Figure 5 shows one of the possible outcomes of encoding.

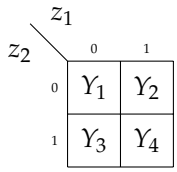

Figure 5. Maximum codes of collections of outputs.

As follows from Figure 5, there is $K(Y_1) = 00, ..., K(Y_4) = 11$. The system of outputs is represented by the following SOP:

$$\begin{aligned} o_1 &= Y_1 = \overline{z_1}\,\overline{z_2}; \quad o_2 = Y_1 \vee Y_3 = \overline{z_1}; \\ o_3 &= Y_2 = z_1\,\overline{z_2}; \quad o_4 = Y_3 = \overline{z_1}\,z_2; \\ o_5 &= Y_4 = z_1\,z_2. \end{aligned} \tag{16}$$

To implement the system (16), we should include in the MCU a block of outputs. This block consists of a decoder (DC) and a coder. Hence, this block has two levels of logic. The decoder transforms codes of COs into one-hot codes corresponding to COs. The coder transforms these one-hot codes into outputs. In the general case, the outputs are represented by the system

$$O = O(Z). \tag{17}$$

If both of the approaches are used simultaneously, then there are three levels of logic blocks in the MCU. Figure 6 shows the structural diagram of three-level MCU with the compulsory addressing of microinstructions [50,54].

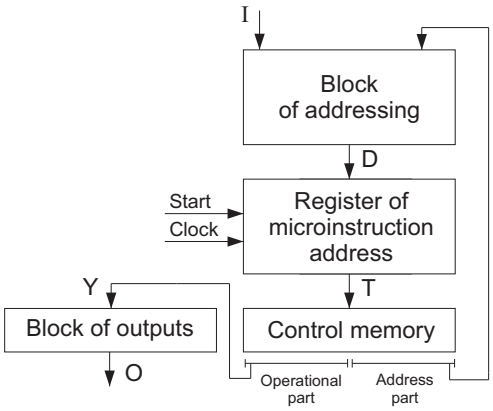

Figure 6. Three-level microprogram control unit.

In the case of the compulsory addressing of microinstructions, the microinstruction format includes the operational part having R_Q bits and the address part having $R_L = \lceil log_2 L \rceil$ bits with a code of logical condition to be checked and two address fields. The first address field includes an address of transition, if a logical condition to be checked is equal to 0 (or an address of unconditional transition). The second address field includes an address of transition if a logical condition to be checked is equal to 1. If a microprogram includes M microinstructions, then the number of address bits is determined by (4). Accordingy, each microinstruction has $R_Q + R_L + 2 \times R_S$ bits. If $R_Q = R_S = 8$ and $R_L = 6$, then the value of V_{ROM} is equal to $30 \times 256 = 7680$ bits.

The block of addressing generates address variables $D_r \in D$. These variables depend on the inputs and microinstruction address part. This block is implemented on multiplexers (MXs) [52]. The block of outputs generates outputs as functions of the MCU operational part. Hence, an MCU is a Moore FSM.

The MCU with the block of addressing became the prototype of the FSMs with replacement of inputs. In literature [41] such FSMs are called *MP* FSMs, where "*M*" means "multiplexer". The MCU with the block of outputs became the prototype of *PY* FSMs with

encoding of collections of outputs. The three-level MCU (Figure 6) corresponds to *MPY* FSM. This means that various methods of structural decomposition can be used together.

The method of encoding of fields of compatible outputs (FCOs) was proposed to eliminate the coder from the block of outputs [55]. The outputs are compatible if they are not written in the same rows of STT. The set O is divided by I classes of compatible outputs:

$$O = O^1 \cup O^2 \cup \ldots \cup O^I. \tag{18}$$

Outputs $o_n \in O^i$ are encoded by maximum binary codes $K_i(o_n)$. There are R_i bits in the code $K_i(o_n)$:

$$R_i = \lceil log_2(N_i + 1) \rceil. \tag{19}$$

In (19), we use the symbol N_i to denote the number of outputs in the class O^i. The one is added to N_i to take the relation $o_n \notin O^i$ into account.

The outputs $o_n \in O^i$ are encoded using variables $z_r \in Z^i$. The total number of operational part bits, R_{FCO}, is determined by summation of the values of (19). The structural diagram of the MCU based on this principle is the same as the one shown in Figure 6. However, the block of outputs consists of I decoders DC^i. A decoder DC^i generates outputs from the field FCO_i.

This approach was used in optimizing control units of IBM/360 [56]. Additionally, they became the prototypes of *PD* FSMs [41].

There are three possible organizations of the block of outputs that are shown in Figure 7.

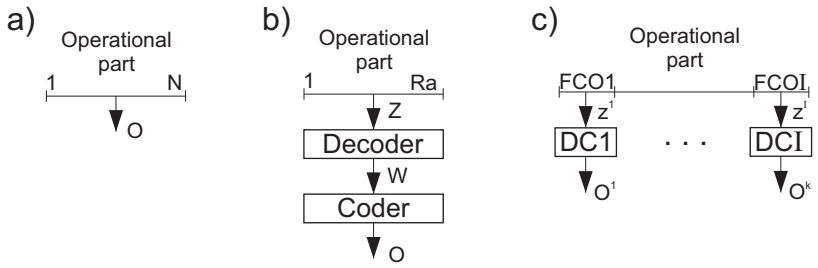

Figure 7. Organization of block of outputs with one-hot (**a**), maximum encoding (**b**) and encoding of FCO (**c**).

As follows from Figure 7, the one-hot organization (Figure 7a) leads to the fastest MCUs having the longest operational part. The block of outputs is absent. The maximum encoding of collections of outputs (Figure 7b) results in the two-level block of outputs. This is the slowest solution, but it provides the shortest operational part. As follows from Figure 7c, the encoding of FCOs results in a single-level block of outputs. This approach provides a compromise solution with the average delay and hardware amount.

The value of V_{ROM} can be reduced due to using the nanomemory [44,54]. We now explain the idea of this approach (Figure 8).

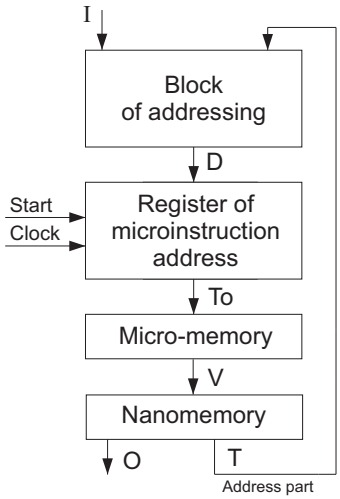

Figure 8. Organization of microprogram control unit (MCU) with nanomemory.

If there are M_o unique microinstructions in a microprogram, then they are encoded using R_o variables $v_r \in V$, where R_o is determined as it is for R_S. These codes are kept into a micro-memory (the first level of control memory). There are M codes in the micro-memory. The second level of memory (nanomemory) keeps the operational and addresses parts of these microinstructions. For example, there is $M = 1024$, $M_0 = 256$, and a microinstruction contains 64 bits. In the case of MCU (Figure 6), there are $V_0 = 1024 \times 64 = 64k = 65{,}536$ bits. We have $R_0 = 10$. Accordingly, there are 1024×10 bits of the micro-memory and $256 \times 64 = 16k$ bits of the nanomemory. Hence, there are 26,624 bits of the control memory for MCU (Figure 8). It means that that approach allows reducing the volume of control memory by 2.46 time when compared to the MCU (Figure 6). This approach is a prototype of *PH* FSMs with encoding of product terms [41].

One fundamental law follows from the analysis of different methods of minimizing the value of V_{ROM}. This is the following: the reducing hardware amount leads to an increase in the delay time of the resulting circuit (due to an increase in the number of logic levels). This law holds for all methods of structural decomposition.

4. Structural Decomposition in Matrix-Based Fsms

If an FSM is a part of an application-specific integrated circuit (ASIC) [57], then its circuit can be implemented using custom matrices [13,58]. These matrices are used as either AND-planes or OR-planes [59]. Each plane is a system of wires connected by CMOS transistors. Two wires (direct and compliment values of corresponding arguments) represent each literal of a SOP. Each term of a SOP corresponds to a wire.

To implement a matrix circuit of Mealy FSM, it is enough to use a single AND-matrix M_1 and a single OR-matrix M_2. This is a trivial matrix implementation of P Mealy FSM (Figure 9).

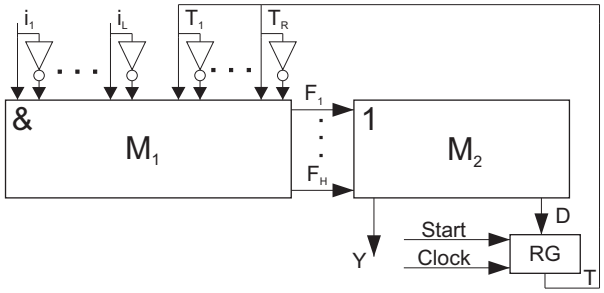

Figure 9. Trivial matrix implementation of P Mealy FSM.

The trivial matrix circuit (Figure 9) represents a P Mealy FSM [41]. This is the fastest matrix solution. However, such a solution is very redundant.

The hardware amount of matrix circuits is defined in conventional units of area (CUA) of matrices [13]. These areas are determined as the following:

$$S(M_1) = 2(L + R_S) \cdot H; \tag{20}$$

$$S(M_2) = H(N + R_S). \tag{21}$$

In (20) and (21), the symbols $S(M_1)$, $S(M_2)$ stand for area of matrices M_1 and M_2, respectively.

If there is $L = 30$, $N = 50$, $R_S = 8$, and $H = 2000$ (an average FSM [13]), then $S(M_1) = 152{,}000$ CUA and $S(M_2) = 116{,}000$ CUA. It gives the total area equal to 268,000 CUA. If each product term of SBFs (5) and (6) includes $3 + R = 11$ literals, then there are $11 \times 2000 = 22{,}000$ useful interconnections in M_1. If each term enters SOPs of five functions, then there are $5 \times 2000 = 10{,}000$ useful interconnections in M_2. Hence, only 32,000 interconnections are used for implementing an FSM circuit. Because there are 268,000 interconnections, only 12% of the area is really used.

Two methods of structural decomposition were used to reduce the chip area that is occupied by an FSM circuit, namely [58]:

1. The replacement of inputs (MP FSM).
2. The encoding of collections of outputs (PY FSM).

To design an MP FSM, it is necessary to replace the set I by some set $P = \{p_1, \ldots, p_G\}$. This makes sense if

$$G \ll L. \tag{22}$$

The value of G is determined by the maximum number of inputs causing transitions from states $s_m \in S$ [58]. Consider the DST of Mealy FSM A_2 (Table 3).

Table 3. Direct structure table (DST) of Mealy FSM A_2.

s_C	$K(s_C)$	s_T	$K(s_T)$	I_h	O_h	D_h	h
s_1	000	s_2	001	i_1	o_1o_2	D_3	1
		s_3	010	$\overline{i_1}$	o_3	D_2	2
s_2	001	s_2	001	i_2	o_2o_4	D_3	3
		s_3	010	$\overline{i_2}i_3$	o_1o_2	D_2	4
		s_4	100	$\overline{i_2}\,\overline{i_3}$	o_5	D_1	5
s_3	010	s_4	100	i_4	o_3	D_1	6
		s_5	011	$\overline{i_4}i_5$	o_3o_5	D_2D_3	7
		s_6	101	$\overline{i_4}\,\overline{i_5}$	o_1o_2	D_1D_3	8
s_4	100	s_3	010	i_6	o_2o_4	D_2	9
		s_1	000	$\overline{i_6}$	-	-	10
s_5	011	s_2	001	i_7	o_3	D_3	11
		s_2	001	$\overline{i_7}i_8$	o_1o_2	D_3	12
		s_4	100	$\overline{i_7}\,\overline{i_8}$	o_3o_5	D_1	13
s_6	101	s_1	000	1	-	-	14

In the case of A_2, we have $G = 2$. Accordingly, there is a set $P = \{p_1, p_2\}$.

To replace inputs, it is necessary to create the following SBF:

$$P = P(T, I). \tag{23}$$

This SBF is constructed using a table of replacement. In the discussed case, Table 4 presents the table of replacement.

Table 4. Table of replacement of P Mealy FSM A_2.

s_m	s_1	s_2	s_3	s_4	s_5	s_6
p_1	i_1	i_2	i_4	-	i_7	-
p_2	-	i_3	i_5	i_6	i_8	-
$K(s_m)$	000	001	010	100	011	101

Using Table 4 gives the following SBF:

$$p_1 = \overline{T_1}\,\overline{T_2}\,\overline{T_3}\,i_1 \vee \overline{T_1}\,\overline{T_2}\,T_3\,i_2 \vee \overline{T_1}T_2\overline{T_3}\,i_4 \vee \overline{T_1}T_2T_3\,i_7; \quad (24)$$
$$p_2 = \overline{T_1}\,\overline{T_2}\,T_3\,i_3 \vee \overline{T_1}T_2\overline{T_3}\,i_5 \vee T_1\,\overline{T_2}\,\overline{T_3}\,i_6 \vee \overline{T_1}T_2T_3\,i_8.$$

The SOP for p_1 includes terms v_1–v_4; the SOP for p_2 includes terms v_5–v_8. Hence, there is $NT(P) = 8$.

We should construct a table of MP FSM to design the circuit of MP FSM. It can be done by a transformation of the DST of P FSM. The transformation is reduced to the replacement of the column I_h by the column P_h [58]. In the discussed case, this leads to Table 5.

Table 5. DST of MP Mealy FSM A_2.

s_C	$K(s_C)$	s_T	$K(s_T)$	P_h	O_h	D_h	h
s_1	000	s_2	001	p_1	o_1o_2	D_3	1
		s_3	010	$\overline{p_1}$	o_3	D_2	2
s_2	001	s_2	001	p_1	o_2o_4	D_3	3
		s_3	010	$\overline{p_1}p_2$	o_1o_2	D_2	4
		s_4	100	$\overline{p_1}\,\overline{p_2}$	o_5	D_1	5
s_3	010	s_4	100	p_1	o_3	D_1	6
		s_5	011	$\overline{p_1}p_2$	o_3o_5	D_2D_3	7
		s_6	101	$\overline{p_1}\,\overline{p_2}$	o_1o_2	D_1D_3	8
s_4	100	s_3	010	p_2	o_2o_4	D_2	9
		s_1	000	$\overline{p_2}$	-	-	10
s_5	011	s_2	001	p_1	o_3	D_3	11
		s_2	001	$\overline{p_1}p_2$	o_1o_2	D_3	12
		s_4	100	$\overline{p_1}\,\overline{p_2}$	o_3o_5	D_1	13
s_6	101	s_1	000	1	-	-	14

From Table 5, we can find that the IMFs and outputs of MP FSM are represented by the following SBFs:

$$D = D(T, P); \quad (25)$$
$$O = O(T, P). \quad (26)$$

Systems (23)–(25) determine a matrix circuit of MP FSM that is shown in Figure 10.

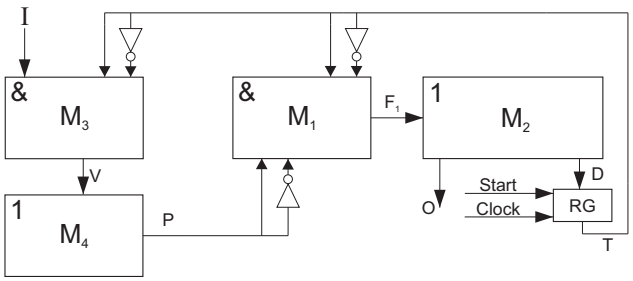

Figure 10. Structural diagram of MP Mealy FSM.

In the *MP* Mealy FSM (Figure 10), the matrix M_3 implements terms of SBF (23). The matrix M_4 transforms terms $v_r \in V$ into functions (23). The matrix M_1 implements terms $F_h \in F$. These terms correspond to the rows of DST. The matrix M_2 generates functions (25) and (26). These matrices have the following areas:

$$\begin{aligned} S(M_1) &= 2(G + R_S) \cdot H; \\ S(M_2) &= H(N + R_S); \\ S(M_3) &= (L + 2R_S) \cdot NT(P); \\ S(M_4) &= NT(P) \cdot G. \end{aligned} \tag{27}$$

To optimize the matrix M_2, the method of encoding of COs can be used [58]. As it is for MCU, Q COs are encoded by binary codes $K(Y_q)$. These codes have R_Q bits, where the expression (15) determines the value of R_Q.

For FSM A_2, the following COs can be found: $Y_1 = \emptyset, Y_2 = \{o_1, o_2\}, Y_3 = \{o_3\}, Y_4 = \{o_2, o_4\}, Y_5 = \{o_5\}, Y_6 = \{o_3, o_5\}$. Accordingly, there is $Q = 6$, $R_Q = 3$, $Z = \{z_1, z_2, z_3\}$. To minimize the number of literals in (17), it is necessary to encode COs $Y_q \subseteq O$ using the approach [60]. In the discussed case, Figure 11 shows the outcome of encoding.

z_3 \ z_1z_2	00	01	11	10
0	Y_1	Y_4	Y_3	Y_6
1	*	Y_2	*	Y_5

Figure 11. Optimal codes of collections of outputs.

Using codes (Figure 11), we can get the following SBF:

$$\begin{aligned} &o_1 = Y_2 = z_2z_3; \quad o_2 = Y_2 \vee Y_4 = \overline{z_1}z_2; \\ &o_3 = Y_3 \vee Y_6 = z_1\overline{z_3}; \quad o_4 = Y_4 = \overline{z_1}\, z_2\, \overline{z_3}; \\ &o_5 = Y_6 = z_1\, \overline{z_2}\, \overline{z_3}. \end{aligned} \tag{28}$$

There are $N = 5$ terms in (28). In the general case, there are $NT(O)$ terms in (17). They form a set W.

To implement a *PY* FSM circuit, it is necessary to create a DST of *PY* FSM. For the FSM A_2, it is Table 6.

Table 6. DST of *PY* Mealy FSM A_2.

s_C	$K(s_C)$	s_T	$K(s_T)$	I_h	Z_h	Φ_h	h
s_1	000	s_2	001	i_1	z_2z_3	D_3	1
		s_3	010	$\overline{i_1}$	z_1z_2	D_2	2
s_2	001	s_2	001	i_2	z_2	D_3	3
		s_3	010	$\overline{i_2}i_3$	z_2z_3	D_2	4
		s_4	100	$\overline{i_2}\,\overline{i_3}$	z_1z_3	D_1	5
s_3	010	s_4	100	i_4	z_1z_2	D_1	6
		s_5	011	$\overline{i_4}i_5$	z_1	D_2D_3	7
		s_6	101	$\overline{i_4}\,\overline{i_5}$	z_2z_3	D_1D_3	8
s_4	100	s_3	010	i_6	z_2	D_2	9
		s_1	000	$\overline{i_6}$	-	-	10
s_5	011	s_2	001	i_7	z_1z_2	D_3	11
		s_2	001	$\overline{i_7}i_8$	z_2z_3	D_3	12
		s_4	100	$\overline{i_7}\,\overline{i_8}$	z_1	D_1	13
s_6	101	s_1	000	1	-	-	14

The DST is a base for deriving SBFs (5) and

$$Z = Z(T, I). \tag{29}$$

The SBFs (5), (17), and (29) determine a *PY* Mealy FSM whose structural diagram is shown in Figure 12.

In *PY* FSM, the matrix M_2 implements functions $D_r \in D$ and variables $z_r \in Z$. The matrix M_5 transforms $z_r \in Z$ into terms of SBF (17). The matrix M_6 generates outputs $o_n \in D$. These matrices have the following areas:

$$\begin{aligned} S(M_1) &= 2(L + R_S) \cdot H; \\ S(M_2) &= H(R_S + R_Q); \\ S(M_5) &= 2R_Q \cdot NT(O); \\ S(M_6) &= NT(O) \cdot N. \end{aligned} \tag{30}$$

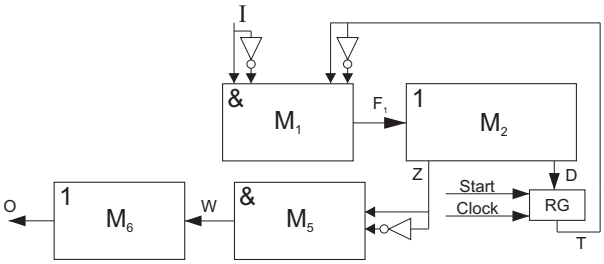

Figure 12. Structural diagram of *PY* Mealy FSM.

These approaches can be used simultaneously [58]. This leads to *MPY* Mealy FSM (Figure 13).

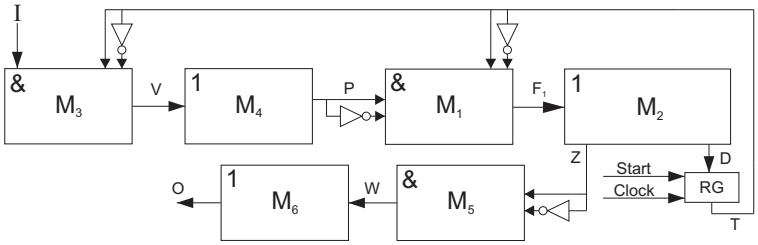

Figure 13. Structural diagram of *MPY* Mealy FSM.

In the matrix circuit (Figure 13), the matrices M_3 and M_4 implement the SBF (23), the matrices M_5 and M_6 implement the SBF (17). The matrices M_1 and M_2 implement SBFs (25) and

$$Z = Z(T, P). \tag{31}$$

There are two levels of logic in the matrix circuit of *P* Mealy FSM (Figure 9). This circuit has six levels of logic. Obviously, the *P* FSM is three times faster than an equivalent *MPY* FSM (Figure 13). Let us compare areas of equivalent FSMs.

As shown in [58], the average FSMs have the following characteristics: $L = 30$, $N = 50$, $R_S = 8$, $H = 2000$, $G = 4$, $NT(P) = 50$, $R_Q = 6$, $NT(O) = 80$. This gives the following: $S(M_1) = 2(G + R_S) \cdot H = 48{,}000$, $S(M_2) = H(R_S + R_Q) = 28{,}000$, $S(M_3) = (L + 2R_S) \cdot NT(P) = 2300$, $S(M_4) = NT(P) \cdot G = 200$, $S(M_5) = 2R_Q \cdot NT(O) = 960$, $S(M_6) = NT(O) \cdot N = 4000$. Now, we have the following total area of *MPY* FSM circuit: 83,460 CUA. There are 268,000 CUA of the area of *P* Mealy FSM (Figure 9). This gives around 69% of economy. Accoridngly, an increase in the number of levels of a matrix circuit leads to an average reduction in area by 3.23 times. Of course, the FSM performance practically decreases to the same extent.

Accordingly, the methods of structural decomposition can be used for optimizing matrix circuits of Mealy FSMs. The same is true for Moore FSMs [61]. For further reducing the area, it is necessary to apply various methods of joint minimization of SBFs [43].

5. Structural Decomposition in Spld-Based Fsms

In the 1970s, a wide range of so-called simple programmable logic devices (SPLDs) appeared. This class includes programmable logic arrays (PLAs), programmable read-only memories (PROMs), and programmable array logic (PAL) [62–65]. A SPLD is a general purpose chip whose hardware can be configured by an end user to implement a particular product [66–69].

There is one common feature of SPLDs. Namely, they can be viewed as a composition of AND and OR arrays [62–65,70]. A typical SPLD structure is exactly the same as the one shown in Figure 9. Accordingly, SPLDs can implement SOPs representing the systems of Boolean functions.

In the case of PROM, the AND-array is fixed. It creates an address decoder. The OR-array is programmable. A PROM is the best tool for implementing SBFs that are represented by truth tables [10]. The number of address inputs of a PROM was rather small. Acccordingly, PROMs were used for implementing only parts of FSM circuits [71].

The joint using PROMs and multiplexers (MXs) leads to *MP* FSMs. The MXs implement the replacement of inputs that are represented by (23). The PROMs implement systems (25) and (26). To keep state codes, the register RG is used (Figure 14a). The joint using PROMs, decoders (DCs), and MXs leads to *MPD* FSMs (Figure 14b). To implement *MPY* FSMs, it is necessary to use MXs and PROMs (Figure 14c).

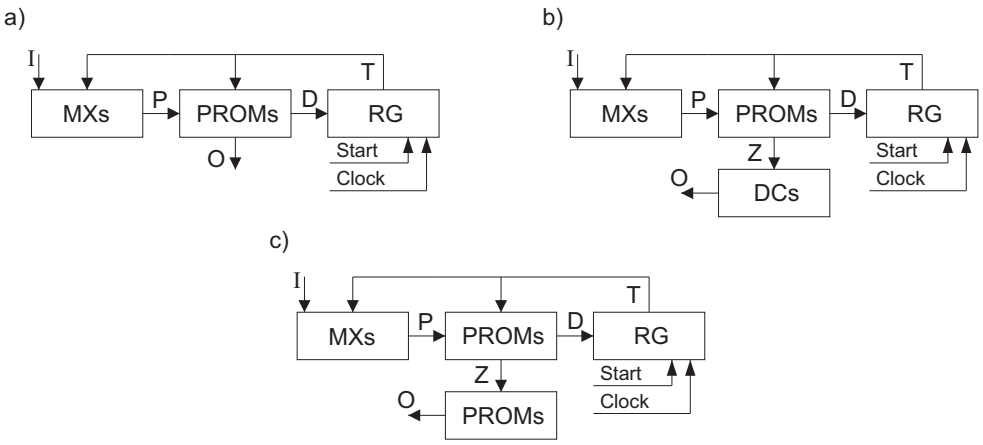

Figure 14. Organization of *MP* (**a**), *MPD* (**b**), and *MPY* (**c**) programmable read-only memory (PROM)-based FSMs.

As follows from Figure 14, different logic elements implement different parts of FSM circuits. This approach is a heterogeneous implementation of FSM circuit [71]. Of course, it is enough to use only memory blocks for implementing an FSM circuit [72].

The PLAs have the following specifics: both of the arrays are programmable [62,63]. Because of it, PLAs are used for implementing reduced SOPs [43] of SBFs [13,41,70]. Typical PLAs have $S_{PLA} = 16$ inputs, $t_{PLA} = 8$ outputs, and $q_{PLA} = 48$ terms [73,74].

As a rule, FSM circuits were represented by networks of PLAs [13,75]. To optimize the number of chips in a circuit, the methods of structural decomposition were used. Additionally, the principle of heterogeneous implementation was used. For example, *MPY* FSMs could be implemented using MXs, PLAs, and PROMs (Figure 15).

Different approaches were used for optimizing characteristics of PLA-based FSMs [76–82]. One of the new approaches was an encoding of FSM terms [78], leading to *PH* FSMs.

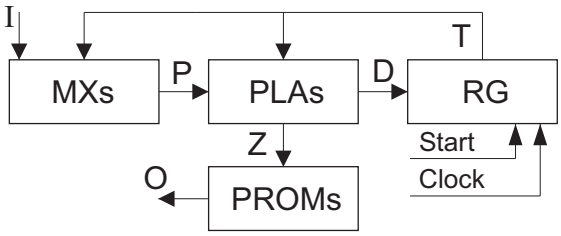

Figure 15. Heterogeneous circuit of *MPY* FSM.

In this case, terms $F_h \in F$, corresponding to rows of STT, were encoded by binary codes $K(F_h)$ having R_H bits:

$$R_H = \lceil log_2 H \rceil. \tag{32}$$

To encode terms, variables $z_r \in Z$ were used, where $|Z| = R_H$. The following SBFs represent *PH* FSMs:

$$\begin{aligned} Z &= Z(I, T); \\ D &= D(Z); \\ O &= O(Z). \end{aligned} \tag{33}$$

These SBFs were implemented using PLAs (for Z) and PROMs (for D, O). Such a composition of PLAs and PROMs leads to *PH* FSM (Figure 16).

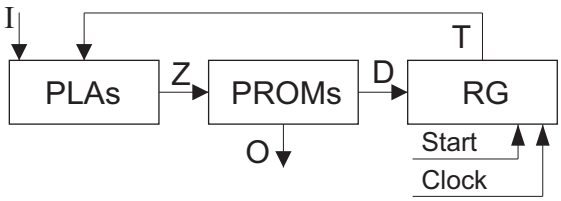

Figure 16. The structural diagram of *PH* FSM.

To implement a *PH* FSM, it is necessary to: (1) encode terms $F_h \in F$; (2) create a DST of *PH* FSM; (3) create SBFs (33); and, (4) program PLAs and PROMs. For example, there is $H = 14$ for Mealy FSM A_2 (Table 3). Using (32) gives $R_H = 4$ and $Z = \{z_1, \ldots, z_4\}$. Let us encode terms in the trivial way: $K(F_1) = 0000, \ldots, K(F_{14}) = 1101$. Table 7 is a DST of *PH* Mealy FSM A_2. Table 8 shows the PROMs' contents.

Obviously, *PH* FSM (Figure 16) can be transformed into *MPH*, *MPHY*, *MPHD*, *PHY*, and *PHD* FSMs. To optimize circuits with decoders, the method [50] can be used.

To optimize hardware of PLA-based FSMs, it is possible to use the methods that are based on transformation of objects [27,83,84]. The following objects are characteristic for the Mealy FSMs [71]: states, outputs, and collections of outputs. The main idea of this approach is a representation of some objects as functions of other objects and additional variables.

Table 7. DST of *PH* Mealy FSM A_2.

s_C	$K(s_C)$	I_h	F_h	$K(F_h)$	Z_h	h
s_1	000	i_1	F_1	0000	-	1
		$\overline{i_1}$	F_2	0001	z_4	2
s_2	001	i_2	F_3	0010	z_3	3
		$\overline{i_2}i_3$	F_4	0011	z_3z_4	4
		$\overline{i_2}\,\overline{i_3}$	F_5	0100	z_2	5
s_3	010	i_4	F_6	0101	z_2z_4	6
		$\overline{i_4}i_5$	F_7	0110	z_2z_3	7
		$\overline{i_4}\,\overline{i_5}$	F_8	0111	$z_2z_3z_4$	8
s_4	100	i_6	F_9	1000	z_2	9
		$\overline{i_6}$	F_{10}	1001	z_2z_4	10
s_5	011	i_7	F_{11}	1010	z_2z_3	11
		$\overline{i_7}i_8$	F_{12}	1011	$z_1z_3z_4$	12
		$\overline{i_7}\,\overline{i_8}$	F_{13}	1100	z_1z_2	13
s_6	101	1	F_{14}	1101	$z_1z_2z_4$	14

Table 8. Contents of programmable read-only memories (PROMs) of *PH* Mealy FSM A_2.

F_h	$K(F_h)$	o_1	o_2	o_3	o_4	o_5	D_1	D_2	D_3	h
F_1	0000	1	1	0	0	0	0	0	1	1
F_2	0001	0	0	1	0	0	0	1	0	2
F_3	0010	0	1	0	1	0	0	0	1	3
F_4	0011	1	1	0	0	0	0	1	0	4
F_5	0100	0	0	0	0	1	1	0	0	5
F_6	0101	0	0	1	0	0	1	0	0	6
F_7	0110	0	0	1	0	1	0	1	1	7
F_8	0111	1	1	0	0	0	1	0	1	8
F_9	1000	0	1	0	1	0	0	1	0	9
F_{10}	1001	0	0	0	0	0	0	0	0	10
F_{11}	1010	0	0	1	0	0	0	0	1	11
F_{12}	1011	1	1	0	0	0	0	0	1	12
F_{13}	1100	0	0	1	0	1	1	0	0	13
F_{14}	1101	0	0	0	0	0	0	0	0	14

The transformation of states into outputs leads to P_S FSMs (Figure 17a). The transformation of states into COs leads to P_SY FSMs (Figure 17b). The transformation of COs into states leads to P_OY FSMs (Figure 17c).

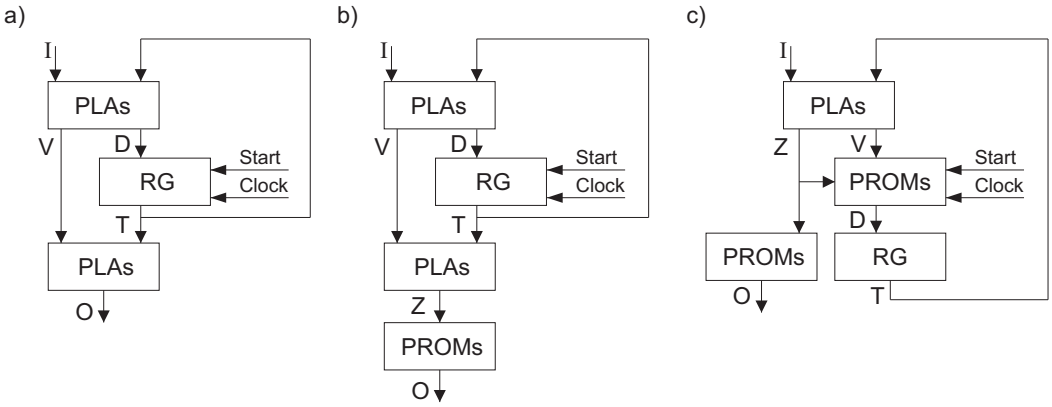

Figure 17. Structural diagrams of Mealy FSMs with transformation of states into outputs **(a)**, states into collections of outputs (**b**), and collections of outputs into states (**c**).

As follows from Figure 17a, additional variables $v_r \in V$ replace inputs $i_e \in I$ in the SBF of outputs:

$$O = O(T, V). \tag{34}$$

If $|V| \ll L$, then the SOPs of (34) are much simpler than SOPs of (6). In $P_S Y$ FSMs (Figure 17b), the following SBFs are generated:

$$V = V(T, I); \tag{35}$$

$$Z = Z(T, V). \tag{36}$$

In the case of $P_O Y$ FSMs (Figure 17c), the following new SBF is implemented:

$$D = D(V, Z). \tag{37}$$

As follows from [83], the transformation of objects improves performance as compared with *MPY* FSMs. Because of it, they are used in FPGA-based design [21].

The PAL chips have the following specific [64,85]: the AND array is programmable and OR-array is fixed. the terms of PAL are assigned to macrocells [23,74]. The evolution of this conception led to complex programmable logic devices (CPLDs) [15,69,86]. There are a huge number of publications related to PAL- and CPLD-based synthesis [64,73,85,87–91]. We do not discuss these methods in this survey. However, we note that the structural decomposition is used in CPLD-based FSMs [23].

6. Structural Decomposition in Fpga-Based Fsms

6.1. Basic Methods of Structural Decomposition in Design with Luts and Embs

Field-programmable gate arrays are widely used for implementing circuits of various digital systems [12,15,69,92]. To implement an FSM circuit, the following internal resources of FPGA chip can be used: look-up table (LUT) elements, embedded memory blocks (EMBs), programmable flip-flops, programmable interconnections, input-output blocks, and block of synchronization. LUTs and flip-flops form configurable logic blocks (CLBs). The "island-style" architecture is used in the majority of FPGAs [17,93,94].

A LUT is a block having S_L inputs and a single output [95–98]. If a Boolean function depends on up to S_L arguments [67], then the corresponding circuit only includes a single LUT. However, the number of LUT inputs is very limited [95–97]. Due to it, the methods of functional decomposition are used to implement the FPGA-based FSM circuits [99–103]. As a result, the FSM circuits have a lot of logic levels and a complex systems of interconnections [29]. Such circuits resemble programs that are based on intensive use of "go-to" operators [104]. Using terminology from programming, we can say that the functional decomposition produces the "spaghetti-type" LUT-based FSM circuits.

Modern FPGAs include a lot of configurable embedded memory blocks [95,96]. These CLBs allow for implementing systems of regular functions [28]. If at least a part of the FSM circuit is implemented using EMBs, then the characteristics of this circuit can be significantly improved [16]. Because of it, there are a lot of design methods targeting EMB-based FSMs [16,105–115]. In [28], there is the survey of various methods of EMB-based FSM design. However, very often, practically all available EMBs are used for implementing the operational blocks of digital systems. Accordingly, the EMB-based FSM design methods can only be applied if a designer has some "free" EMBs.

An EMB can be characterized by a pair $\langle S_A, t_F \rangle$, where S_A is a number of address inputs and t_F is a number of memory cell outputs. A single EMB can keep a truth table of an SBF including up to t_F Boolean functions depended on up to S_A arguments [116]. A pair $\langle S_A, t_F \rangle$ defines a configuration of an EMB with the constant total number of bits (size of EMB):

$$V_0 = 2^{S_A} \times t_F. \tag{38}$$

The parameters S_A and t_F could be defined by a designer [66]. It means that EMBs are configurable memory blocks [67]. The following configurations exist for modern EMBs [95,96]: $\langle 15, 1 \rangle, \langle 14, 2 \rangle, \ldots, \langle 9, 64 \rangle$. Accordingly, modern EMBs are very flexible and can be tuned to meet characteristics of a particular FSM. This explains the existence of a wide spectrum of EMB-based design methods [16,105–115].

If the condition

$$2^{(R_S+L)}(R_S + N) \leq V_0 \tag{39}$$

holds, then a single EMB implements an FSM circuit [28]. If (39) is violated, then an FSM circuit could be implemented as: (1) a homogenous network of EMBs or (2) a heterogeneous network where LUTs and EMBs are used together [16,114].

There are three approaches for implementing combinational parts of CLB-based FSMs. They are the following: (1) using only LUTs; (2) using only EMBs; and, (3) using the heterogeneous approach, when both LUTs and EMBs are applied [28].

One of the most crucial steps in the CLB-based design flow is the technology mapping [29,117,118]. The outcome of the technology mapping is a network of interconnected CLBs representing an FSM circuit. This step largely determines the resulting characteristics of an FSM circuit. These characteristics are strongly interrelated.

A chip area occupied by a CLB-based FSM circuit is mostly determined by the number of CLBs and the system of their interconnections. Obviously, to reduce the area, it is necessary to reduce the CLB count in an FSM circuit. As follows from [119], the more LUTs are included into an FSM circuit, the more 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" [18]. As noted in [120], the interconnections are responsible for the consume up to 70% of power. Accordingly, it is very important to reduce the amount of interconnections to improve the characteristics of FSM circuits. All of this can be done using methods of structural decomposition.

As follows from (39), an FSM circuit can be implemented by a single EMB if the following conditions hold for a configuration $\langle S_A, t_F \rangle$:

$$S_A \geq R_S + L; \tag{40}$$

$$t_F \geq R_S + N. \tag{41}$$

As a rule, the modern EMBs are synchronous blocks. Hence, there is no need in an additional register to keep FSM state codes [28]. Figure 18 shows a trivial EMB-based circuit of Mealy FSM.

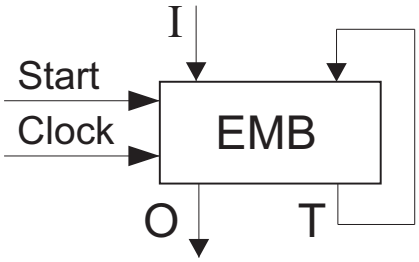

Figure 18. A trivial embedded memory block (EMB)-based circuit of Mealy FSM.

To design such a circuit, it is necessary to [28]: (1) execute the state assignment; (2) construct a DST on the base of an STT; and, (3) create the truth table corresponding to the DST. This truth table has $L + R_S$ columns containing an address of a particular cell. Each cell has $R_S + N$ bits. Transitions from any state $s_m \in S$ are represented by $H(s_m)$ rows of the truth table [28]:

$$H(s_m) = 2^L. \tag{42}$$

The following parameters can be found for A_1 (Table 2): the number of inputs $L = 3$, and the number of state variables $R_S = 2$. Accordingly, using (42) gives $H(s_m) = 8$. If an input $i_e \in I$ is insignificant for transitions from a state $s_m \in S$, then there are the same values of IMFs and outputs for cells with addresses having either $i_e = 0$ or $i_e = 1$. This rule is illustrated by Table 9 with the transitions from state s_2 from Table 2.

Table 9. Part of truth table for FSM A_1.

Address					Contents of Cells							q	h
T_1	T_2	i_1	i_2	i_3	o_1	o_2	o_3	o_4	o_5	D_1	D_2		
0	1	0	0	0	0	0	0	0	1	0	1	9	5
0	1	0	0	1	1	1	0	0	0	1	0	10	4
0	1	0	1	0	0	0	0	0	1	0	1	11	5
0	1	0	1	1	1	1	0	0	0	1	0	12	4
0	1	1	0	0	0	0	0	0	1	0	1	13	5
0	1	1	0	1	1	1	0	0	0	1	0	14	4
0	1	1	1	0	0	0	0	0	1	0	1	15	5
0	1	1	1	1	1	1	0	0	0	1	0	16	4

In Table 9, the number of a cell is shown in the column q. The column h is added to compare Tables 2 and 9. The even rows of Table 9 correspond to $i_3 = 1$, and the odd rows correspond to $i_3 = 0$.

The transition from LUTs to EMBs is similar to the transition from gates to large scale integration circuits. This transition improves all the characteristics of an FSM circuit, namely, the chip area that is occupied by FSM circuit, the FSM performance and power consumption. If conditions (40) and (41) are violated, then methods of structural decomposition can be used [21]. In this case, an FSM circuit is represented as a network of EMBs and LUTs.

The analysis of numerous literature has shown that the following methods of structural decomposition are used in EMB-based FSM design:

1. The replacement of inputs $i_e \in I$ by additional variables $p_g \in P$ leading to MP FSMs [16,107–113].
2. The maximum encoding of collections of outputs leading to PY FSMs [28].
3. Mixed encoding of outputs leading to PY_M FSMs [121].
4. The encoding of product terms leading to PH FSMs [122].

Following the notation of [21], we denote, as LUTer, a block consisting of LUTs and. as EMBer, a block consisting of EMBs. The structural diagram of *MP* Mealy FSM is shown in Figure 19.

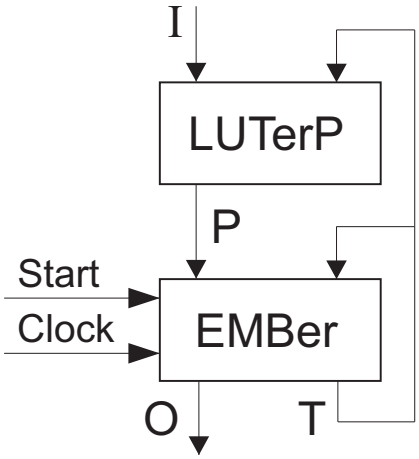

Figure 19. Structural diagram of field-programmable gate array (FPGA)-based *MP* Mealy FSM.

In *MP* FSM, the LUTerP implements SBF (23), the EMBer contains a truth table of SBFs (25) and (26). As follows from Figures 18 and 19, the outputs $o_n \in O$ are synchronized. This is necessary to stabilize FSM outputs [42]. The *MP* Mealy FSM can be used if the following condition holds:

$$2^{G+R_S}(N + R_S) \leq V_o. \tag{43}$$

Clearly, the *MP* FSM (Figure 19) uses an idea of the two-level MCU (Figure 4) in an FPGA environment. The state variables create the address part of microinstructions. The number of EMBs in EMBer is determined as

$$n_{EMB} = \left\lceil \frac{R_S + N}{t_F} \right\rceil. \tag{44}$$

To diminish the value of n_{EMB}, the maximum encoding of COs $Y_q \subseteq O$ can be used [21]. The replacement of inputs can be used together with this approach. This results in the *MPY* Mealy FSM (Figure 20).

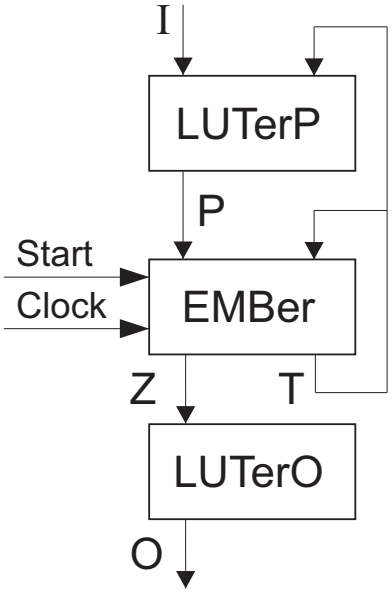

Figure 20. The structural diagram of FPGA-based *MPY* Mealy FSM.

In *MPY* FSM, the EMBer implements SBFs (23) and (31). The LUTerO transforms codes $K(O_q)$ into outputs $o_n \in O$. To do it, SBF (17) is implemented by LUTerO. Now, the number of EMBs in EMBer is determined as

$$n_{EMB} = \left\lceil \frac{R_S + R_Q}{t_F} \right\rceil. \tag{45}$$

The value of R_Q is determined by (15).

If the condition

$$R_Q \leq S_L \tag{46}$$

holds, then a single-level circuit of LUTerO includes up to N LUTs. If (46) is violated, then a mixed encoding of outputs [121] can be used. The idea of this approach is the following.

Let it be $Q = 17$, $R_Q = 5$, and $S_L = 4$. The analysis of these values shows that the condition (46) is violated. Let the set of COs include COs $Y_5 = \{o_1, o_3, o_4\}$ and $Y_8 = \{o_1, o_4\}$. If we eliminate $o_3 \in O$ from Y_5, then $Y_5 \equiv Y_8$. Now, there is $R_Q = 4 = S_L$. The eliminated outputs form a set O_E. The set of outputs is represented as $O = O_E \cup O_L$, where $O_L \cap O_E = \emptyset$. This leads to MPY_M Mealy FSM (Figure 21).

In MPY_M FSM, the outputs $o_n \in O_E$ are represented by SBF (26). The outputs $o_n \in O_L$ are represented by (17). The outputs $o_n \in O_E$ are represented by one-hot codes, the outputs $o_n \in O_L$ by maximum binary codes. Because of that, this is a mixed encoding of outputs.

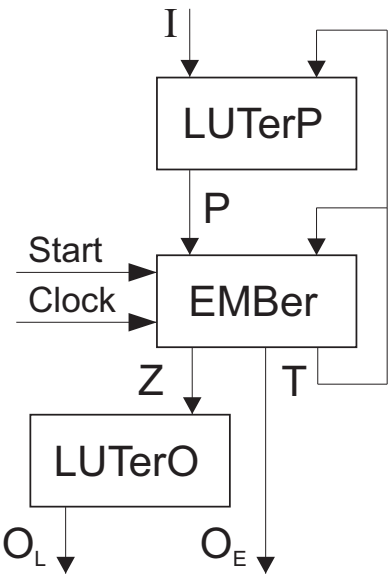

Figure 21. Structural diagram of MPY_M Mealy FSM.

In [121], there is proposed a method allowing to create such a partition of the set O. It allows for eliminating the minimum possible number elements of O to create the set O_E.

This approach can be used to diminish the number of CLBs in the circuit of LUTerO. For example, there is $S_L = 6$ for LUTs of Virtex 7 [96]. If $R_Q = 6$, then the number of LUTs in the circuit of LUTerO is equal to N. However, the CLB can be organized as two LUTs having five shared inputs. If the mixed encoding of outputs gives the set O_L with $R_Q = 5$, then the number of LUTs in LUTerO is determined as $\lceil |O_L|/2 \rceil$. The closer the values of N and $|O_L|$ are, the greater the saving in the number of CLBs.

Two approaches are possible for implementing EMB-based Mealy FSMs [122]. In both cases, the binary codes $K(F_h)$ encode the terms $F_h \in F$. These codes have R_H bits. The variables $z_r \in Z$ are used for encoding of terms, where $|Z| = R_H$. The value of R_H is determined by (32). The system $Z = Z(T, I)$ represents the block of terms [122]. This system can be implemented as either the network of LUTs (Figure 22a) or the network of EMBs (Figure 22b).

Both methods should be used. Finally, the method leading to the minimum hardware should be selected [122].

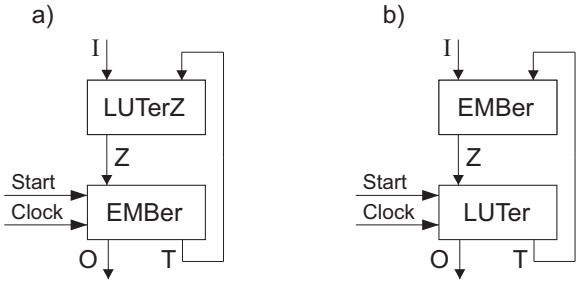

Figure 22. Structural diagrams of *PH* Mealy FSM with $\langle LUTer, EMBer \rangle$ (**a**) and $\langle EMBer, LUTer \rangle$ (**b**) organization.

6.2. Structural Decomposition in Lut-Based Design

As mentioned in [12], EMBs are widely used for implementing various blocks of digital systems. Accordingly, it is quite possible that only LUTs can be used for implementing FSM circuits. The methods of structural decomposition may be used in LUT-based FSMs [21]. They are used to improve LUT counts (and other characteristics) of LUT-based *P* Mealy FSMs (Figure 23).

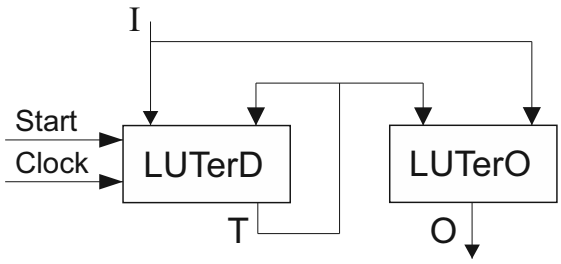

Figure 23. Structural diagram of look-up table (LUT)-based *P* Mealy FSM.

In *P* FSMs, the LUTerD implements SBF (5) and the LUTerO implements SBF (6). Each function $f_i \in D \cup O$ is represented by a SOP having $NA(f_i)$ literals. In the best case, there are R_S LUTs in the circuit of LUTerD and N LUTs in the circuit of LUTerO. The following relation determines this case:

$$NA(f_i) \leq S_L \quad (i \in \{1, \ldots, R_S + N\}). \tag{47}$$

If (47) is violated, then a *P* FSM is represented by a multi-level circuit. To improve LUT count of such circuits, the model of *MPY* FSM can be used.

This approach is proposed in [123]. It leads to a three-level circuit that is shown in Figure 24.

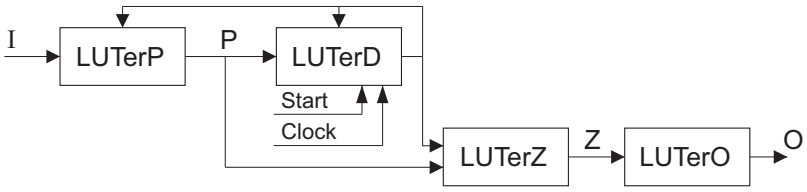

Figure 24. Structural diagram of LUT-based *MPY* Mealy FSM.

In *MPY* FSM, the LUTerP implements system (23). It generates additional variables $p_g \in P$ replacing inputs $i_e \in I$. The LUTerD generates input memory functions that are represented by (25). The LUTerZ generates variables $z_r \in Z$ used for encoding of collections

of outputs. This block implements SBF (31). The LUTerO implements outputs $o_n \in O$ that are represented by SBF (17).

The method of synthesis of LUT-based *MPY* FSM includes the following steps [123]:

1. Executing the replacement of inputs.
2. Executing the state assignment optimizing (23).
3. Deriving collections of outputs from the STT.
4. Executing the encoding of COs.
5. Creating the DST of *MPY* FSM.
6. Deriving SBFs (25) and (31) from the DST.
7. Implementing FSM circuit using particular LUTs.

In [123], the results of experiments conducted to compare the characteristics of various models of LUT-based FSMs are shown. The standard benchmarks [124] were used for investigation. These benchmarks are Mealy FSMs; they are represented in KISS2 format. Table 10 contains the characteristics of these benchmark FSMs.

To conduct experiments [123], the CAD tool Vivado (ver. 2019.1) [125] was used with the target chip XC7VX690T2FFG1761 (Xilinx Virtex 7) [126]. There is $S_L = 6$ for LUTs of Virtex 7 family.

Four other methods were compared with *MPY* FSMs. They were Auto of Vivado, one-hot of Vivado, JEDI [39,127], and DEMAIN [128]. The benchmarks were divided by five categories. To do it, the values of $R_S + L$ and $S_L = 6$ were used. If $R_S + L \leq 6$, then benchmarks belong to category 0; if $6 < R_S + L \leq 12$, it is the category 1; if $12 < R_S + L \leq 18$, then it defines the category 2; if $18 < R_S + L \leq 24$, then benchmarks belong to category 3; finally, the relation $R_S + L > 24$, determines category 4.

Table 11 (the LUT counts) and Table 12 (the maximum operating frequency) represent the results of investigations [123]. As follows from Table 11, MPY-based FSMs have minimum number of LUTs. As follows from Table 12, MPY-based FSMs are the slowest. However, this disadvantage is reduced with the increase in the number of category.

Table 10. Characteristics of Mealy FSM benchmarks.

Benchmark	L	N	$R+L$	M/R	H
			Category 0		
bbtas	2	2	6	9/4	24
dk17	2	3	6	16/4	32
dk27	1	2	5	10/4	14
dk512	1	3	6	24/5	15
ex3	2	2	6	14/4	36
ex5	2	2	6	16/4	32
lion	2	1	5	5/3	11
lion9	2	1	6	11/4	25
mc	3	5	6	8/3	10
modulo12	1	1	5	12/4	24
shiftreg	1	1	5	16/4	16
			Category 1		
bbara	4	2	8	12/4	60
bbsse	7	7	12	26/5	56
beecount	3	4	7	10/4	28
cse	7	7	12	32/5	91
dk14	3	5	8	26/5	56
dk15	3	5	8	17/5	32
dk16	2	3	9	75/7	108
donfile	2	1	7	24/5	96
ex2	2	2	7	25/5	72
ex4	6	9	11	18/5	21
ex6	5	8	9	14/4	34
ex7	2	2	12	17/5	36
keyb	7	7	12	22/5	170
mark1	5	16	10	22/5	22
opus	5	6	10	18/5	22
s27	4	1	8	11/4	34
s386	7	7	12	23/5	64
s8	4	1	8	15/4	20
sse	7	7	12	26/5	56

Table 10. *Cont.*

Benchmark	L	N	$R+L$	M/R	H
			Categories 2–4		
ex1	9	19	16	80/7	138
kirkman	12	6	18	48/6	370
planet	7	19	14	86/7	115
planet1	7	19	14	86/7	115
pma	8	8	14	49/6	73
s1	8	7	14	54/6	106
s1488	8	19	15	112/7	251
s1494	8	19	15	118/7	250
s1a	8	6	15	86/7	107
s208	11	2	17	37/6	153
styr	9	10	16	67/7	166
tma	7	9	13	63/6	44
sand	11	9	18	88/7	184
s420	19	2	27	137/8	137
s510	19	7	27	172/8	77
s820	18	19	25	78/7	232
s832	18	19	25	76/7	245

Table 11. Experimental results for *MPY* Mealy FSMs [123] (LUT counts).

Benchmark	Auto	One-Hot	JEDI	DEMAIN	*MPY*
			Category 0		
bbtas	**5**	**5**	**5**	**5**	8
dk17	**5**	12	**5**	6	8
dk27	**3**	5	4	4	7
dk512	10	10	**9**	10	12
ex3	**9**	**9**	**9**	**9**	11
ex5	**9**	**9**	**9**	**9**	10
lion	**2**	5	**2**	**2**	6
lion9	6	11	**5**	**5**	8
mc	**4**	7	**4**	5	6
modulo12	**7**	**7**	**7**	**7**	9
shiftreg	**2**	6	**2**	**2**	4

Table 11. *Cont.*

Benchmark	Auto	One-Hot	JEDI	DEMAIN	*MPY*
			Category 1		
bbara	17	17	10	**9**	10
bbsse	33	37	**24**	26	26
beecount	19	19	**14**	16	**14**
cse	40	66	36	38	**33**
dk14	**10**	27	**10**	12	12
dk15	**5**	16	**5**	6	6
dk16	15	34	12	14	**11**
donfile	31	31	22	26	**21**
ex2	9	9	**8**	9	**8**
ex4	15	13	12	13	**11**
ex6	24	36	22	23	**21**
ex7	**4**	5	**4**	**4**	6
keyb	43	61	40	42	**37**
mark1	23	23	20	21	**19**
opus	28	28	22	26	**21**
s27	**6**	18	**6**	**6**	**6**
s386	26	39	22	25	**20**
s8	**9**	**9**	**9**	**9**	**9**
sse	33	37	30	32	**26**
			Categories 2–4		
ex1	70	74	53	57	**40**
kirkman	42	58	39	41	**33**
planet	131	131	88	94	**78**
planet1	131	131	88	94	**78**
pma	94	94	86	91	**72**
s1	65	99	61	64	**54**
s1488	124	131	108	112	**89**
s1494	126	132	110	117	**90**
s1a	49	81	43	54	**38**
s208	12	31	10	11	**9**
styr	93	120	81	88	**70**
tma	45	39	39	41	**30**
sand	132	132	114	121	**99**
s420	10	31	9	10	**8**
s510	48	48	32	39	**22**
s820	88	82	68	76	**52**
s832	80	79	62	70	**50**
Total	1792	2104	1480	1601	**1321**
Percentage	135.65%	159.27%	112.04%	121.20%	**100%**

Table 12. Experimental results for *MPY* Mealy FSMs [123] (the operating frequency, MHz).

Benchmark	Auto	One-Hot	JEDI	DEMAIN	*MPY*
Category 0					
bbtas	204.16	204.16	206.12	**208.32**	194.43
dk17	199.28	167.00	**199.39**	172.19	147.22
dk27	**206.02**	201.90	204.18	205.10	181.73
dk512	196.27	196.27	**199.75**	197.49	175.63
ex3	194.86	194.86	**195.76**	193.43	174.44
ex5	180.25	180.25	181.16	**181.76**	162.56
lion	202.43	**204.00**	202.35	201.32	185.74
lion9	205.30	185.22	**206.38**	205.86	167.28
mc	196.66	195.47	**196.87**	192.53	178.02
modulo12	207.00	207.00	207.13	**207.37**	189.7
shiftreg	262.67	263.57	**276.26**	276.14	248.79
Category 1					
bbara	193.39	193.39	**212.21**	198.46	183.32
bbsse	157.06	169.12	**182.34**	178.91	159.24
beecount	166.61	166.61	**187.32**	184.21	156.72
cse	146.43	163.64	**178.12**	174.19	153.24
dk14	191.64	172.65	**193.85**	187.32	162.78
dk15	192.53	185.36	**194.87**	188.54	175.42
dk16	169.72	174.79	**197.13**	189.83	164.16
donfile	184.03	184.00	**203.65**	194.83	174.28
ex2	198.57	198.57	**200.14**	199.75	188.95
ex4	180.96	177.71	**192.83**	178.14	168.39
ex6	169.57	163.80	**176.59**	174.12	156.42
ex7	200.04	**200.84**	200.60	200.32	191.43
keyb	156.45	143.47	**168.43**	157.16	136.49
mark1	162.39	162.39	**176.18**	169.65	153.48
opus	166.20	166.20	**178.32**	168.79	157.42
s27	198.73	191.50	**199.13**	198.43	185.15
s386	168.15	173.46	**179.15**	169.21	164.65
s8	180.02	178.95	181.23	**180.39**	168.32
sse	157.06	169.12	**174.63**	169.69	158.14

Table 12. *Cont.*

Benchmark	Auto	One-Hot	JEDI	DEMAIN	*MPY*
			Categories 2–4		
ex1	150.94	139.76	176.87	**186.14**	164.32
kirkman	141.38	154.00	**156.68**	143.76	155.36
planet	132.71	132.71	**187.14**	185.73	174.68
planet1	132.71	132.71	**187.14**	185.73	173.29
pma	146.18	146.18	**169.83**	153.57	156.12
s1	146.41	135.85	**157.16**	149.17	145.32
s1488	138.50	131.94	**157.18**	153.12	141.27
s1494	149.39	145.75	**164.34**	159.42	155.63
s1a	153.37	**176.40**	169.17	158.12	166.36
s208	174.34	176.46	**178.76**	172.87	166.42
styr	137.61	129.92	**145.64**	138.83	118.02
tma	163.88	147.80	164.14	**168.19**	137.48
sand	115.97	115.97	**126.82**	120.63	120.07
s420	173.88	176.46	177.25	172.87	**186.35**
s510	177.65	177.65	198.32	183.18	**199.05**
s820	152.00	153.16	**176.58**	166.29	175.69
s832	145.71	153.23	173.78	160.03	**174.39**
Total	8127.08	8061.22	**8718.87**	8461.10	7917.10
Percentage	102.65%	101.82%	**110.13%**	106.87%	100%

6.3. New Methods of Structural Decomposition

In all thw discussed methods, only maximum state codes are used when the value of R_S is determined by (4). In [129–131], there is a method of twofold state assignment proposed. In this case, any state $s_m \in S$ has two codes. The code $K(s_m)$ determines the state as an element of the set S. The code $C(s_m)$ defines the state as an element of some partition class.

To use the method [129,130], it is necessary to construct a partition $\Pi_S = \{S^1, \ldots, S^K\}$ of the set of states S. For each class $S^k \in \Pi_S$, the following condition holds:

$$R_k + L_k \leq S_L \quad (k = \overline{1, K}). \tag{48}$$

In (48), the symbol R_k denotes the length (the number of bits) of a code $C(s_m)$ for states $s_m \in S^k$; the symbol L_k defines the number of inputs $i_e \in I$ determining the transitions from states $s_m \in S^k$.

Each class $S^k \in \Pi_S$ determines a DST_k with transitions from states $s_m \in S^k$. This table includes inputs from the set $I^k \subseteq I$, outputs from the set $O^k \subseteq O$, and IMFs that are equal to 1 for transitions from states $s_m \in S^k$. These IMFs form a set $D^k \subseteq D$. A DST_k determines the SBFs

$$D^k = D^k(\tau^k, I^k); \tag{49}$$

$$O^k = O^k(\tau^k, I^k). \tag{50}$$

The variables $\tau_r \in \tau^k$ encode states as elements of the set $S^k \subseteq S$.

This approach determines P_T Mealy FSMs. The logic circuits of P_T FSMs include three levels of logic blocks. Figure 25 showsn the structural diagram of P_T FSM.

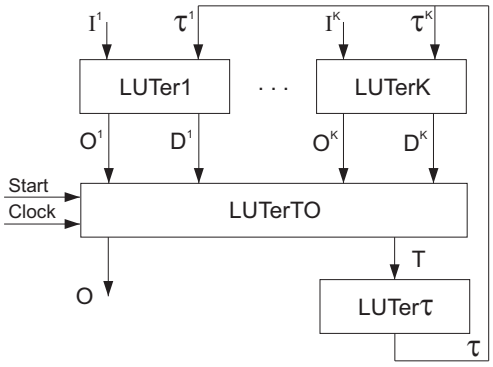

Figure 25. Structural diagram of P_T Mealy FSM.

In P_T Mealy FSM, the LUTerk ($k \in \{1, \ldots, K\}$) implements SBF (49) and (50). The LUTerTO implements the following SBFs:

$$D = D(D^1, \ldots, D^K); \tag{51}$$

$$O = O(O^1, \ldots, O^K). \tag{52}$$

The LUTerτ transform state codes $K(s_m)$ into state codes $C(s_m)$. To do it, the following SBF is implemented:

$$\tau = \tau(T). \tag{53}$$

The structural diagram (Figure 25) determines a case of the one-hot encoding of outputs [130]. In [129], there was a method proposed combining the twofold state assignment with the maximum encoding of COs. This leads to P_TY Mealy FSM, as shown in Figure 26.

In P_TY FSM, the SBFs (50) and (52) are replaced by SBFs:

$$Z^k = Z^k(\tau^k, I^k); \tag{54}$$

$$Z = Z(Z^1, \ldots, Z^K). \tag{55}$$

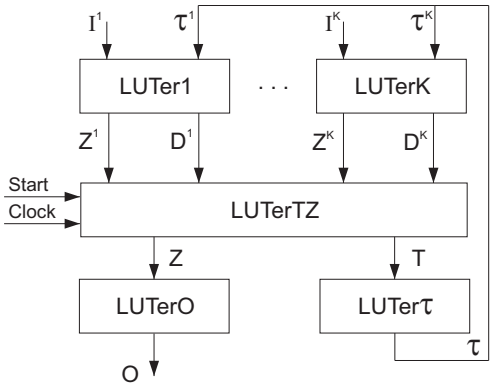

Figure 26. Structural diagram of P_TY Mealy FSM.

Because of (48), each function (49), (50), and (54) are implemented as a single-level circuit; moreover, each function is implemented by a circuit having exactly one LUT. If there is

$$K \leq S_L, \tag{56}$$

then it is enough a single LUT to implement a circuit for each determined by (52) and (54). If there is

$$R_S \leq S_L, \tag{57}$$

then the circuit of the LUTerτ is a single-level one. If the condition (46) holds, then there are up to N LUTs in the circuit of LUTerO.

In the best case, the conditions (46), (48), (56), and (57) are true. This best case determines the three-level LUT-based circuits of both P_T and P_TY Mealy FSMs. Logic circuits of P_TY FSMs consume fewer LUTs than equivalent PY FSMs, as shown in [129]. The experimental results [130] show that the logic circuits of P_T FSMs consume fewer LUTs than this is for the equivalent P Mealy FSMs.

Using the twofold state assignment improves the characteristics of EMB-based FSMs, as shown in [122]. In [122], this method is used to improve LUT count in PH Mealy FSMs (Figure 22b). The method is based on finding a partition $\Pi_F = \{F^1, \ldots, F^k\}$ of the set of terms F. For each class of this partition, the following condition holds:

$$R_k \leq S_L \quad (k \in \{1, \ldots, K\}). \tag{58}$$

The value of R_k can be found as $\lceil log_2 H_k \rceil$, where H_k is a number of elements in the set F^k.

The binary codes $K(F_h)$ encode the classes $F^k \in \Pi_F$. These codes have R_c bits, where

$$R_c = \lceil log_2 K \rceil. \tag{59}$$

The code of a term $F_h \in F$ is represented as

$$K(F_h) = C(F^k) * C(F_h). \tag{60}$$

In (60), $C(F_h)$ is a code of a term as an element of the set $F^k \subseteq F$, $*$ is a sign of concatenation. To encode terms, the variables $z_r \in Z$ are used. To use free outputs of EMB, the set D is represented as $D_E \cup D_L$ and the set O is represented as $O_E \cup O_L$. The classes of Π_F are encoded using variables $v_r \in V$. Now, the PH FSM is represented, as shown in Figure 27.

In [122], the results of experiments are shown. The following models were compared: P FSMs (Figure 23), MP FSMs (Figure 19), PH FSMs (Figure 22b), and the proposed approach (Figure 27). Table 13 (LUT counts), Table 14 (the maximum operating frequency), and Table 15 (the consumed power) show the results of experiments for some benchmarks [124].

The experiments have been conducted for the benchmarks [124], the evolution board with chip XC7VX690TFFG1761-2 [126] and CAD tool Vivado [125]. It is enough a single EMB of Virtex 7 to implement the logic circuits for any from 33 benchmarks [124], as shown in [122]. A network of LUTs and EMBs is used to implement circuits for other benchmarks.

It is possible to improve the characteristics of LUT-based FSM circuits using the transformation of objects [21]. For example, there is a structural diagram of P_oY Mealy FSM shown in Figure 28 [132].

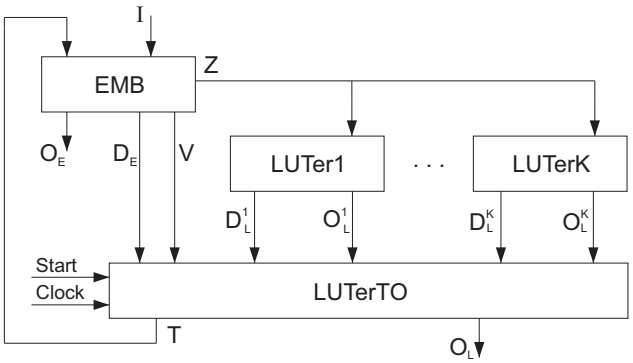

Figure 27. The structural diagram of PH Mealy FSM.

Table 13. Experimental results [122] (LUT counts).

Benchmark	***P***	***MP***	***PH***	**[122]**
ex1	22	**19**	48	36
kirkman	30	26	27	**11**
planet	21	**16**	51	38
planet1	21	**16**	51	38
pma	28	23	27	**14**
s1	26	23	24	**12**
s1488	24	**21**	52	37
s1494	28	**24**	50	39
s208	29	23	8	**7**
s420	38	36	8	**7**
s510	39	36	22	**15**
s820	40	**34**	47	36
s832	41	**34**	47	35
sand	27	23	29	**16**
styr	26	20	31	**18**
Total	440	374	522	**359**
Percentage	123%	104%	145%	**100%**

Table 14. Experimental results [122] (the operating frequency, MHz).

Benchmark	***P***	***MP***	***PH***	**[122]**
ex1	141.43	105.78	158.28	**212.93**
kirkman	125.78	107.81	155.11	**174.73**
planet	122.01	105.41	124.31	**187.95**
planet1	122.01	105.41	124.31	**187.95**
pma	115.41	114.49	127.65	**186.22**
s1	124.49	117.80	132.85	**178.84**
s1488	127.80	112.79	131.77	**186.37**
s1494	122.79	124.92	135.73	**181.62**
s208	144.92	128.04	144.05	**209.36**
s420	148.04	112.66	152.65	**192.14**
s510	122.66	111.42	138.75	**192.87**
s820	121.42	88.65	133.36	**163.18**
s832	98.65	115.57	100.53	**184.69**
sand	135.57	104.68	146.78	**178.65**
styr	114.68	116.47	115.69	**181.22**
Total	1887.66	1671.90	2021.82	**2798.72**
Percentage	67.4%	59.7%	72.2%	**100%**

Table 15. Experimental results [122] (the consumed power, Watts).

Benchmark	*P*	*MP*	*PH*	[122]
ex1	3.560	3.290	3.014	**2.918**
kirkman	4.922	3.562	2.811	**2.476**
planet	3.222	3.756	1.727	**1.527**
planet1	3.222	3.756	1.727	**1.527**
pma	4.778	4.915	4.257	**3.683**
s1	3.694	3.813	3.578	**3.058**
s1488	1.586	2.412	**1.449**	1.785
s1494	1.730	2.398	1.453	**1.302**
s208	3.005	3.544	2.574	**2.248**
s420	1.604	3.384	1.543	**1.292**
s510	1.883	1.996	1.878	**1.682**
s820	2.465	2.161	**1.756**	1.843
s832	2.515	2.504	2.193	**1.732**
sand	2.579	2.578	2.385	**2.017**
styr	1.467	1.556	1.307	**1.112**
Total	42.232	45.625	33.652	**30.202**
Percentage	139.8%	151%	111.4%	**100%**

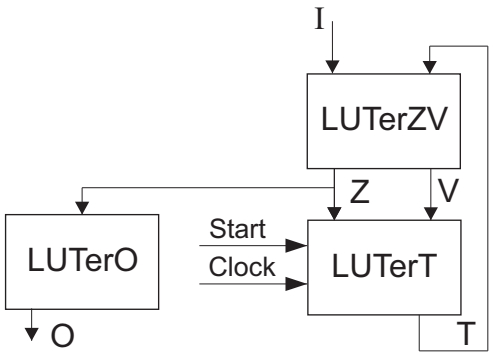

Figure 28. Structural diagram of P_oY Mealy FSM.

In P_oY FSM, the LUTerZV implements SBFs (35) and

$$Z = Z(T, I). \tag{61}$$

The LUTerT generates the functions from the SBF (37) and the LUTerO implements SBF (17). This approach is used to: (1) improve the operating frequency of multi-level MPY FSMs and (2) reduce the LUT count as compared with P FSMs if the condition (47) is violated.

If condition (47) is violated for functions $f_i \in V \cup Z$, then the LUTerZV is represented by a multi-level circuit. To improve the characteristics of P_oY FSMs, the following approach is proposed in [132].

The set S is divided by classes $S^k \in \Pi_S$, such that the condition (48) holds for each class of Π_S. Next, states $s_m \in S^k$ are encoded by codes $C(s_m)$ having the minimum possible number of bits. The following SBFs should be implemented [132]: (54), (55), (17), (37), and

$$V^k = V^k(\tau^k, I^k); \tag{62}$$

$$V = V(V^1, \ldots, V^K). \tag{63}$$

This approach leads to $P_{oT}Y$ FSMs. The circuit of $P_{oT}Y$ FSM includes three levels of LUTs (Figure 29).

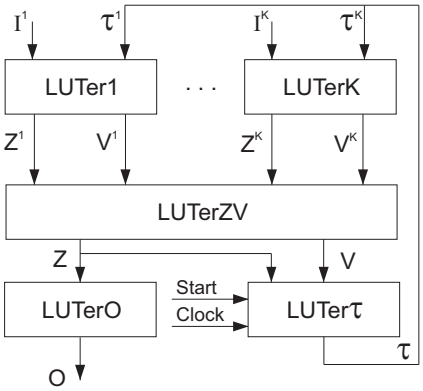

Figure 29. The structural diagram of LUT-based $P_{oT}Y$ Mealy FSM.

In $P_{oT}Y$ FSM, the LUTerk ($k \in \{1, \ldots, K\}$) implements SBFs (54) and (62). The LUTerZV generates functions $z_r \in Z$ and $v_r \in V$. They are represented by SBFs (55) and (63). The LUTerO implements SBF (17), the LUTerτ generates functions (37).

There are experimental results in [132] that are obtained using the CAD tool Vivado [125] and the evolution board with Virtex 7 FPGA chip [126]. The following characteristics have been compared: the LUT counts (Table 16), maximum operating frequency (Table 17), and area-time products (Table 18).

As follows from Table 16, the P_oY FSMs require fewer LUTs than other investigated methods. The $P_{oT}Y$ FSMs consume more LUTs (8.84%) when compared to P_oY FSMs. However, other FSMs are based on functional decomposition. Their circuits require more LUTs than for $P_{oT}Y$ FSMs. The gain increases along with the growth of the category number.

As follows from Table 17, the $P_{oT}Y$-based FSMs have the highest operating frequency as compared to other investigated methods. The following can be found from Table 18: the $P_{oT}Y$-based FSMs produce circuits with better area-time products then it is for other investigated methods. Starting from average FSMs, $P_{oT}Y$-based circuits have better area-time products.

Table 16. Experimental results [132] (LUT counts).

Benchmark	Auto	One-Hot	JEDI	P_oY	$P_{oT}Y$
			Category 0		
bbtas	**5**	**5**	**5**	8	9
dk17	**5**	12	**5**	8	10
dk27	**3**	5	4	7	9
dk512	10	10	**9**	12	14
ex3	**9**	**9**	**9**	11	14
ex5	**9**	**9**	**9**	10	12
lion	**2**	5	**2**	6	8
lion9	6	11	**5**	8	10
mc	**4**	7	**4**	6	8
modulo12	**7**	**7**	**7**	9	11
shiftreg	**2**	6	**2**	4	6

Table 16. *Cont.*

Benchmark	Auto	One-Hot	JEDI	P_oY	$P_{oT}Y$
			Category 1		
bbara	17	17	**10**	**10**	14
bbsse	33	37	**24**	26	29
beecount	19	19	**14**	**14**	16
cse	40	66	36	**33**	35
dk14	16	27	**10**	12	14
dk15	15	16	12	**8**	11
dk16	15	34	12	**11**	13
donfile	31	31	24	**21**	24
ex2	9	9	**8**	**8**	10
ex4	15	13	12	**11**	13
ex6	24	36	22	**21**	23
ex7	**4**	5	**4**	6	8
keyb	43	61	40	**37**	40
mark1	23	23	20	**19**	21
opus	28	28	22	**21**	23
s27	**6**	18	**6**	**6**	8
s386	26	39	22	**20**	22
s8	**9**	**9**	**9**	**9**	11
sse	33	37	30	**26**	29
			Categories 2–4		
ex1	70	74	53	**40**	44
kirkman	42	58	39	**33**	35
planet	131	131	88	**78**	82
planet1	131	131	88	**78**	82
pma	94	94	86	**72**	76
s1	65	99	61	**54**	58
s1488	124	131	108	**89**	93
s1494	126	132	110	**90**	94
s1a	49	81	43	**38**	42
s208	12	31	10	**9**	11
styr	93	120	81	**70**	78
tma	45	39	39	**30**	34
sand	132	132	114	**99**	103
s420	10	31	9	**8**	10
s510	48	48	32	**22**	23
s820	88	82	68	**52**	56
s832	80	79	62	**50**	52
Total	1808	2104	1489	**1320**	1448
Percentage	124.86%	145.30%	102.83%	**91.16%**	100%

Hence, using the methods of structural decomposition allows for improving characteristics of FPGA-based FSMs. Three-level circuits improve the LUT count and two-level circuits improve the performance. These methods can be applied together with other optimization methods used in FSM design [21].

Table 17. Experimental results [132] (the maximum operating frequency, MHz).

Benchmark	Auto	One-Hot	JEDI	P_oY	$P_{oT}Y$
Category 0					
bbtas	204.16	204.16	**206.12**	194.43	201.47
dk17	199.28	167.00	**199.39**	147.22	172.99
dk27	**206.02**	201.90	204.18	181.73	190.32
dk512	196.27	196.27	**199.75**	175.63	187.45
ex3	194.86	194.86	**195.76**	174.44	187.26
ex5	180.25	180.25	**181.16**	162.56	162.56
lion	202.43	**204.00**	202.35	185.74	195.73
lion9	205.30	185.22	**206.38**	167.28	183.45
mc	196.66	195.47	**196.87**	178.02	182.95
modulo12	207.00	207.00	**207.13**	189.70	201.74
shiftreg	262.67	263.57	**276.26**	248.79	253.72
Category 1					
bbara	193.39	193.39	**212.21**	183.32	210.21
bbsse	157.06	169.12	182.34	159.24	**193.43**
beecount	166.61	166.61	187.32	156.72	**194.47**
cse	146.43	163.64	178.12	153.24	**182.62**
dk14	191.64	172.65	193.85	162.78	**201.39**
dk15	192.53	185.36	194.87	175.42	**206.74**
dk16	169.72	174.79	197.13	164.16	**199.14**
donfile	184.03	184.00	203.65	174.28	**206.83**
ex2	198.57	198.57	**200.14**	188.95	196.58
ex4	180.96	177.71	192.83	168.39	**196.18**
ex6	169.57	163.80	176.59	156.42	**187.53**
ex7	200.04	200.84	200.60	191.43	**204.16**
keyb	156.45	143.47	168.43	136.49	**178.59**
mark1	162.39	162.39	176.18	153.48	**182.37**
opus	166.20	166.20	178.32	157.42	**186.34**
s27	198.73	191.50	199.13	185.15	**201.26**
s386	168.15	173.46	179.15	164.65	**192.34**
s8	180.02	178.95	181.23	168.32	**191.32**
sse	157.06	169.12	**174.63**	158.14	171.18

Table 17. *Cont.*

Benchmark	Auto	One-Hot	JEDI	P_oY	$P_{oT}Y$
			Categories 2–4		
ex1	150.94	139.76	176.87	164.32	**180.72**
kirkman	141.38	154.00	156.68	155.36	**184.62**
planet	132.71	132.71	187.14	174.68	**212.45**
planet1	132.71	132.71	187.14	173.29	**212.45**
pma	146.18	146.18	169.83	156.12	**192.43**
s1	146.41	135.85	**157.16**	145.32	145.32
s1488	138.50	131.94	157.18	141.27	**182.14**
s1494	149.39	145.75	164.34	155.63	**186.49**
s1a	153.37	176.40	169.17	166.36	**188.92**
s208	174.34	176.46	178.76	166.42	**192.15**
styr	137.61	129.92	145.64	118.02	**164.52**
tma	163.88	147.80	164.14	137.48	**182.72**
sand	115.97	115.97	126.82	120.07	**143.14**
s420	173.88	176.46	177.25	186.35	**218.62**
s510	177.65	177.65	198.32	199.05	**221.19**
s820	152.00	153.16	176.58	175.69	**195.73**
s832	145.71	153.23	173.78	174.39	**199.18**
Total	8127.08	8061.22	8718.87	7873.36	**9005.11**
Percentage	90.25%	89.52%	96.82%	87.43%	**100%**

Table 18. Experimental results [132] (area-time products, LUTs $\times ns$).

Benchmark	Auto	One-Hot	JEDI	P_oY	$P_{oT}Y$
			Category 0		
bbtas	24.49	24.49	**24.26**	41.15	44.67
dk17	25.09	71.86	**25.08**	54.34	57.81
dk27	**14.56**	24.76	19.59	38.52	47.29
dk512	50.95	50.95	**45.06**	68.33	74.69
ex3	46.19	46.19	**45.97**	63.06	74.76
ex5	49.93	49.93	**49.68**	61.52	73.82
lion	**9.88**	24.51	**9.88**	32.30	40.87
lion9	29.23	59.39	**24.23**	47.82	54.51
mc	20.34	35.81	**20.32**	33.70	43.73
modulo12	33.82	33.82	**33.80**	47.44	54.53
shiftreg	7.61	22.76	**7.24**	16.08	23.65

Table 18. *Cont.*

Benchmark	Auto	One-Hot	JEDI	P_oY	$P_{oT}Y$
			Category 1		
bbara	87.91	87.91	**47.12**	54.55	66.60
bbsse	210.11	218.78	**131.62**	163.28	149.93
beecount	114.04	114.04	**74.74**	89.33	82.27
cse	273.17	403.32	202.11	215.35	**191.65**
dk14	83.49	156.39	**51.59**	73.72	69.52
dk15	77.91	86.32	61.58	**45.60**	53.21
dk16	88.38	194.52	**60.87**	67.01	65.28
donfile	168.45	168.48	117.85	120.50	**116.04**
ex2	45.32	45.32	**39.97**	42.34	50.87
ex4	82.89	73.15	**62.23**	65.32	66.27
ex6	141.53	219.78	124.58	134.25	**122.65**
ex7	20.00	24.90	**19.94**	31.34	39.18
keyb	274.85	425.18	237.49	271.08	**223.98**
mark1	141.63	141.63	**113.52**	123.79	115.15
opus	168.47	168.47	**123.37**	133.40	123.43
s27	30.19	93.99	**30.13**	32.41	39.75
s386	154.62	224.84	122.80	121.47	**114.38**
s8	49.99	50.29	**49.66**	53.47	57.50
sse	210.11	218.78	171.79	**164.41**	169.41
			Categories 2–4		
ex1	463.76	529.48	299.66	**243.43**	243.47
kirkman	297.07	376.62	248.91	212.41	**189.58**
planet	987.11	987.11	470.24	446.53	**385.97**
planet1	987.11	987.11	470.24	450.11	**385.97**
pma	643.04	643.04	506.39	461.18	**394.95**
s1	443.96	728.74	388.14	**371.59**	399.12
s1488	895.31	992.88	687.11	630.00	**510.60**
s1494	843.43	905.66	669.34	578.29	**504.05**
s1a	319.49	459.18	254.18	228.42	**222.32**
s208	68.83	175.68	55.94	**54.08**	57.25
styr	675.82	923.65	556.17	593.12	**474.11**
tma	274.59	263.87	237.60	218.21	**186.08**
sand	1138.23	1138.23	898.91	824.52	**719.58**
s420	57.51	175.68	50.78	**42.93**	45.74
s510	270.19	270.19	161.36	110.52	**103.98**
s820	578.95	535.39	385.09	295.98	**286.11**
s832	549.04	515.56	356.77	286.71	**261.07**
Total	12228.61	14168.64	8844.90	8554.93	**7877.31**
Percentage	155.24%	179.87%	112.28%	108.60%	**100%**

7. Conclusions

Since the 1950s, digital systems have increasingly influenced different areas of our lives. The control units and other sequential blocks are very important parts of digital systems. Very often, the behaviour of sequential blocks is represented using a model of finite state machine. During these 70 years, several generations of logic elements that are used to implement FSM circuits have changed. However, one thing remained unchanged: regardless of the generation of logic elements, there is always the problem of reducing their number in the FSM circuit. This problem arises if a single-level FSM circuit with minimum possible amount of elements cannot be implemented. One of the ways for reducing the required hardware is the applying various methods of structural decomposition.

These approaches have roots in various methods that are used for optimizing the size of the control memory of microprogram control units. The following basic methods of structural decomposition are known: the replacement of FSM inputs, encoding of the collections of outputs, encoding of product terms corresponding to interstate transitions, and transformation of objects. Using these methods requires taking the peculiarities of logic elements into account. Recently, two new methods of structural decomposition have appeared. These new methods are: (1) the twofold state assignment and (2) the mixed encoding of FSM outputs. These methods are focused on FPGA-based FSMs.

This orientation is related to the fact that FPGA devices are very often used for implementing digital systems. These chips include a lot of LUT elements and embedded memory blocks. It allows implementing very complex digital systems. Embedded memory blocks are effective tools for implementing FSM circuits. However, it is quite possible that all available EMBs are used for implementing various blocks of a digital system. In this case, an FSM circuit is implemented as a network of LUTs. The main specific of LUTs is a very small number of inputs (for the vast majority of FPGAs the value of S_L is less than 7). This feature makes it necessary to use the methods of functional decomposition in the FPGA-based design. As a rule, this leads to multi-level FSM circuits that are characterized by the very complex systems of "spaghetti-type" interconnections.

The optimization of the chip area that is occupied by a LUT-based FSM circuit can be achieved due to applying various methods of structural decomposition. Numerous studies show that the structural decomposition produces the FSM circuits having better characteristics than their counterparts based on the functional decomposition. The FSM circuits that are based on the structural decomposition are characterized by the regular system of interconnections and predicted number of logic levels. The same is true for the heterogeneous implementation of FSM circuits when LUTs and EMBs are used simultaneously.

In this review, we have shown the roots of structural decomposition methods and their development starting from the 1950s. Our research shows that these methods can be used for optimizing FSM circuits that were implemented with any logic elements (PROMs, PLAs, PALs, CPLDs, FPGAs, and custom matrices of ASIC). Now, the majority of digital systems are implemented using FPGAs and ASICs. It is difficult to imagine what elements will replace them in the future. However, one thing remains clear: these elements will also have limits on the number of inputs, outputs, and terms. The results of the research presented in this article allow us to conclude that the methods of structural decomposition will be used in the future generations of the logic elements implementing FSM circuits.

Author Contributions: Conceptualization, A.B., L.T. and K.K.; methodology, A.B., L.T. and K.K.; formal analysis, 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.

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
COF	collection of output functions
CO	collection of output
CPLD	complex programmable logic device
DST	direct structure table
EMB	embedded memory block
ESC	extended state code
FCO	field of compatible outputs
FD	functional decomposition
FSM	finite state machine
FPGA	field-programmable gate array
IMF	input memory function
LUT	look-up table
PAL	programmable array Logic
PLA	programmable logic array
PROM	programmable read-only memory
ROM	read-only memory
SBF	systems of Boolean functions
SD	structural decomposition
SOP	sum-of-products
SRG	state register
STT	state transition table

References

1. Alur, R. *Principles of Cyber-Physical Systems*; MIT Press: Cambridge, MA, USA, 2015.
2. Suh, S.C.; Tanik, U.J.; Carbone, J.N.; Eroglu, A. *Applied Cyber-Physical Systems*; Springer: New York, NY, USA, 2014.
3. Krzywicki, K.; Barkalov, A.; Andrzejewski, G.; Titarenko, L.; Kolopienczyk, M. SoC research and development platform for distributed embedded systems. *Prz. Elektrotech.* **2016**, *92*, 262–265. [CrossRef]
4. Nowosielski, A.; Małecki, K.; Forczmański, P.; Smoliński, A.; Krzywicki, K. Embedded Night-Vision System for Pedestrian Detection. *IEEE Sens. J.* **2020**, *20*, 9293–9304. [CrossRef]
5. Barkalov, A.; Titarenko, L.; Mazurkiewicz, M. *Foundations of Embedded Systems*; Springer International Publishing: New York, NY, USA, 2019.
6. Lee, E.A.; Seshia, S.A. *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*; MIT Press: Cambridge, MA, USA, 2017.
7. 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]
8. Zajac, 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]
9. Marwedel, P. *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems, and the Internet of Things*, 3rd ed.; Springer International Publishing: New York, NY, USA, 2018.
10. De Micheli, G. *Synthesis and Optimization of Digital Circuits*; McGraw-Hill: Cambridge, MA, USA, 1994.
11. Gajski, D.D.; Abdi, S.; Gerstlauer, A.; Schirner, G. *Embedded System Design: Modeling, Synthesis and Verification*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2009.
12. 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.
13. Baranov, S. *Logic Synthesis of Control Automata*; Kluwer Academic Publishers: Dordrecht, The Netherlands, 1994.
14. El-Maleh, A.H. Finite state machine-based fault tolerance technique with enhanced area and power of synthesised sequential circuits. *IET Comput. Digit. Tech.* **2017**, *11*, 159–164. [CrossRef]
15. Jenkins, J.H. *Designing with FPGAs and CPLDs*; Prentice Hall: Hoboken, NJ, USA, 1994 .
16. Tiwari, A.; Tomko, K.A. Saving power by mapping finite-state machines into embedded memory blocks in FPGAs. In Proceedings of the Proceedings Design, Automation and Test in Europe Conference and Exhibition, Paris, France, 16–20 February 2004; Volume 2, pp. 916–921.
17. Trimberger, S.M. *Field-Programmable Gate Array Technology*; Springer Science & Business Media: Berlin, Germany, 2012.
18. Feng, W.; Greene, J.; Mishchenko, A. Improving FPGA Performance with a S44 LUT structure. In Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA"18), Monterey, CA, USA, 25–27 February 2018; p. 6. [CrossRef]

19. Benini, L.; De Micheli, G. State assignment for low power dissipation. *IEEE J. Solid-State Circuits* **1995**, *30*, 258–268. [CrossRef]
20. Agrawal, R.; Borowczak, M.; Vemuri, R. A state encoding methodology for Side-Channel security vs. power Trade-Off exploration. In Proceedings of the 2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID), Delhi, India, 5–9 January 2019; pp. 70–75.
21. 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/Heidelberg, Germany, 2020; Volume 636.
22. Barkalov, A.; Titarenko, L. *Logic Synthesis for FSM-Based Control Units*; Springer: Berlin, Germany, 2009; Volume 53.
23. Czerwinski, R.; Kania, D. *Finite State Machine Logic Synthesis for Complex Programmable Logic Devices*; Vol. 231 of Lecture Notes in Electrical Engineering; Springer: Berlin/Heidelberg, Germany, 2013.
24. Barkalov, A.; Titarenko, L.; Mazurkiewicz, M.; Krzywicki, K. Improving LUT count of FPGA-based sequential blocks. *Bull. Pol. Acad. Sci. Tech. Sci.* **2021**. [CrossRef]
25. Wilkes, M.V. The best way to design an automatic calculating machine. In Proceedings of the Manchester University Computer Inaugural Conference, London, UK, 9–12 July 1951; pp. 16–18.
26. Wilkes, M.V.; Stringer, J.B. Micro-programming and the design of the control circuits in an electronic digital computer. In *Mathematical Proceedings of the Cambridge Philosophical Society*; Cambridge University Press: Cambridge, UK, 1953; Volume 49, pp. 230–238.
27. 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]
28. Barkalov, A.; Titarenko, L.; Kolopienczyk, M.; Mielcarek, K.; Bazydlo, G. *Logic Synthesis for FPGA-Based Finite State Machines*; Springer: Chem, Switzerland, 2015; pp. 2–31.
29. Kubica, M.; Opara, A.; Kania, D. *Technology Mapping for LUT-Based FPGA*; Springer: Cham, Switzerland, 2021.
30. 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.
31. Mealy, G.H. A method for synthesizing sequential circuits. *Bell Syst. Tech. J.* **1955**, *34*, 1045–1079. [CrossRef]
32. Moore, E.F. Gedanken-experiments on sequential machines. *Autom. Stud.* **1956**, *34*, 129–153.
33. Glushkov, V.M. *Synthesis of Digital Automata*; Foreign Technology Div Wright-Patterson Afb Ohio: Dayton, OH, USA, 1965.
34. Issa, H.H.; Ahmed, S.M.E. FPGA implementation of floating point based cuckoo search algorithm. *IEEE Access* **2019**, *7*, 134434–134447. [CrossRef]
35. Senhadji-Navarro, R.; Garcia-Vargas, I. Methodology for Distributed-ROM-based Implementation of Finite State Machines. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2020**. [CrossRef]
36. Klimowicz, A. Combined State Splitting and Merging for Implementation of Fast Finite State Machines in FPGA. In *International Conference on Computer Information Systems and Industrial Management*; Springer: Cham, Switzerland, 2020; pp. 65–76.
37. Gazi, O.; Arlı, A.Ç. VHDL Implementation of Finite State Machines and Practical Applications. In *State Machines Using VHDL*; Springer: Cham, Switzerland, 2021; pp. 55–113.
38. Yan, Z.; Jiang, H.; Li, B.; Yang, M. A Flowchart Based Finite State Machine Design and Implementation Method for FPGA. In *International Conference on Internet of Things as a Service*; Springer: Cham, Switzerland, 2020; pp. 295–310.
39. Sentowich, E.; 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*; University of California: Berkely, CA, USA, 1992.
40. ABC System. Available online: https://people.eecs.berkeley.edu/~alanmi/abc/ (accessed on 6 April 2021).
41. Baranov, S.; Skliarov, V. *Digital Devices with Programmable LSIs with Matrix Structure*; Radio and Communications; Radio Sviaz: Moscow, Russia, 1986.
42. Skliarov, V. *Synthesis of Automata with Matrix LSIs*; Nauka i Technika: Minsk, Belarus, 1984.
43. McCluskey, E.J. *Logic Design Principles with Emphasis on Testable Semicustom Circuits*; Prentice-Hall, Inc.: Hoboken, NJ, USA, 1986.
44. Agerwala, T. Microprogram optimization: A survey. *IEEE Comput. Archit. Lett.* **1976**, *25*, 962–973. [CrossRef]
45. Agrawala, A.K.; Rauscher, T.G. *Foundations of Microprogramming*; Academic Press: New York, NY, USA, 1976.
46. Chu, Y. *Computer Organization and Microprogramming*; Prentice Hall: Hoboken, NJ, USA, 1972. [CrossRef]
47. Flynn, M.J.; Rosin, R.F. Microprogramming: An introduction and a viewpoint. *IEEE Trans. Comput.* **1971**, *100*, 727–731.
48. Habib, S. *Microprogramming and Firmware Engineering Methods*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 1988.
49. Palagin, A.; Rakitskij, A. Three structures of microprogram control units. *Control Mach. Syst.* **1984**, *3*, 40–43.
50. Kravcov, L.; Chernicki, G. *Design of Microprogram Control Units*; Energy: Leningrad, Russia, 1976. [CrossRef]
51. Dasgupta, S. The organization of microprogram stores. *ACM Comput. Surv. (CSUR)* **1979**, *11*, 39–65.
52. Husson, S.S.; Mm, S. *Microprogramming: Principles and Practices*; Prentice-Hall Inc.: Englewood Cliffs, NJ, USA, 1970.
53. Salisbury, A.B. *Microprogrammable Computer Architectures*; Elsevier Science Inc.: Amsterdam, The Netherlands, 1976.
54. Baranov, S.; Barkalov, A. Microprogramming: principles, methods, applications. *Foreign Radioelectron.* **1984**, *5*, 3–29.
55. Schwartz, S.J. An algorithm for minimizing read only memories for machine control. In Proceedings of the 9th Annual Symposium on Switching and Automata Theory, Schenedtady, NY, USA, 15–18 October 1968; pp. 28–33. [CrossRef]
56. Tucker, S.G. Microprogram control for System/360. *IBM Syst. J.* **1967**, *6*, 222–241.
57. Solovjev, V.; Chyzy, M. Refined CPLD macrocell architecture for the effective FSM implementation. In Proceedings of the 25th EUROMICRO Conference, Informatics: Theory and Practice for the New Millennium, Milan, Italy, 8–10 September 1999; Volume 1, pp. 102–109.

58. Baranov, S.I. *Synthesis of Microprogram Machines*; Energiya: Leningrad, Russia, 1979.
59. Navabi, Z. *Embedded Core Design with FPGAs*; McGraw-Hill Professional: New York, NY, USA, 2006.
60. Achasova, S. Synthesis algorithms for automata with PLAs. *Sov. Radio* **1987**, *3*, 22–33.
61. Barkalov, A.; Węgrzyn, M. *Design of Control Units with Programmable Logic*; University of Zielona Góra Press: Zielona Góra, Poland, 2006.
62. Baranov, S.; Barkalov, A. Application of programmable logic arrays in digital systems. *Foregin Radioelectron.* **1982**, *6*, 67–79.
63. Baranov, S.; Sinjov, V. Programmable logic arrays in digital systems. *Foregin Radioelectron.* **1976**, *1*, 78–84.
64. Palagin, A.; Barkalov, A.; Usifov, S.; Szwets, A. Synthesis of microprogram automata with PLIs. *Kiev IC NAN* **1992**, *92*, 18–26.
65. Gorman, K. The programmable logic array: A new approach to microprogramming. *Electron. Des. News* **1973**, *18*, 68–75.
66. Maxfield, C. *The Design Warrior's Guide to FPGAs: Devices, Tools and Flows*; Elsevier: Amsterdam, The Netherlands, 2004.
67. Maxfield, C. *FPGAs: Instant Access*; Elsevier: Amsterdam, The Netherlands, 2011.
68. Hemel, A. The PLA: A different kind of ROM. *Electron. Des.* **1976**, *24*, 28–47. [CrossRef]
69. Brown, S.; Rose, J. Architecture of FPGAs and CPLDs: A tutorial. *IEEE Des. Test Comput.* **1996**, *13*, 42–57.
70. Bibilo, P. *Synthesis of Combinational PLA Structures for VLSI*; Nauka i Tehnika: Minsk, Belarus, 1992.
71. Below, P.L.A.L. *Digital Systems Design with Programmable Logic*; Addison-Wesley: Boston, MA, USA, 1990.
72. Sasao, T. *Memory-Based Logic Synthesis*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2011.
73. Solovjov, V. Design of functional blocks of digital systems with programmable logic devices. *Bestprint* **1996**, *7*, 40–52.
74. Solovjov, V. *Design of Digital Systems Basing on Programmable Logic Integrated Circuits*; Hotline–Telecom: Moscow, Russia, 2001.
75. Baranov, S.I. *Logic and System Design of Digital Systems*; TUT Press: Tallinn, Estonia, 2008. [CrossRef]
76. Baer, J.L.; Koyama, B. On the minimization of the width of the control memory of microprogrammed processors. *IEEE Comput. Archit. Lett.* **1979**, *28*, 310–316.
77. Novikov, S. Synthesis Of Logic-Circuits With Programmable Logic-Arrays. *Avtomatika I Vychislitelnaya Tekhnika* **1977**, *5*, 1–4.
78. Skilarov, V. *Synthesis of Microprogram Automata with Standard PLAs*; Automatic Control and Computer Sciences; Allerton Press Inc.: New York, NY, USA, 1983; pp. 28–35.
79. Skilarov, V. Using decoders in microprogram automata with matrix structure. *Izwiestia Wuzow Priborostrojenie* **1982**, *12*, 27–31.
80. Sorokin, B. A Method of Synthesis of Microprogram Automata on Standard ROMs and PLAs. *Avtomatika I Vychislitelnaya Tekhnika* **1984**, *2*, 69–77.
81. Barkalov, A. Multilevel PLA schemes for microprogram automata. *Cybern. Syst. Anal.* **1995**, *31*, 489–495.
82. Barkalov, A. Optimization of multilevel circuit of mealy FSM with PLAs. *Control Syst. Mach.* **1994**, *93*, 13–16.
83. Barkalov, A.A.; Barkalov, A.A.J. Design of Mealy finite-state machines with the transformation of object codes. *Int. J. Appl. Math. Comput. Sci.* **2005**, *15*, 151–158. [CrossRef]
84. Barkalov, A.; Titarenko, L.; Mielcarek, K.; Wegrzyn, M. Design of EMB-based mealy FSMs with transformation of output functions. *IFAC-PapersOnLine* **2015**, *48*, 197–201.
85. Palagin, A.; Barkalov, A.; Usifov, S.; Starodubov, K.; Svetc, A. Realization of microprogrammed automata on CPLD. *Control Syst. Mach.* **1991**, *8*, 18–22.
86. Zeidman, B. *Designing with FPGAS and CPLDS*; CRC Press: Boca Raton, FL, USA, 2002. [CrossRef]
87. Kania, D. Two-level logic synthesis on PALs. *Electron. Lett.* **1999**, *35*, 879–880.
88. Kania, D. Two-level logic synthesis on PAL-based CPLD and FPGA using decomposition. In Proceedings of the 25th EUROMICRO Conference, Informatics: Theory and Practice for the New Millennium, Milan, Italy, 8–10 September 1999; Volume 1, pp. 278–281. [CrossRef]
89. Kania, D. Coding capacity of PAL-based logic blocks included in CPLDs and FPGAs. *IFAC Proc. Vol.* **2000**, *33*, 167–172.
90. Kania, D. An Efficient Algorithm for Output Coding in PAL Based CPLDs. *Int. J. Eng.* **2002**, *15*, 325–328. [CrossRef]
91. Kania, D.; Milik, A. Logic Synthesis based on decomposition for CPLDs. *Microprocess. Microsyst.* **2010**, *34*, 25–38. [CrossRef]
92. Bomar, B.W. Implementation of microprogrammed control in FPGAs. *IEEE Trans. Ind. Electron.* **2002**, *49*, 415–422.
93. Kuon, I.; Tessier, R.; Rose, J. *FPGA Architecture: Survey and Challenges*; Now Publishers Inc.: Delft, The Netherlands, 2008. [CrossRef]
94. Trimberger, S.M.S. Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology: This Paper Reflects on How Moore's Law Has Driven the Design of FPGAs Through Three Epochs: the Age of Invention, the Age of Expansion, and the Age of Accumulation. *IEEE Solid-State Circuits Mag.* **2018**, *10*, 16–29.
95. Altera. Cyclone IV Device Handbook. Available online: http://www.altera.com/literature/hb/cyclone-iv/cyclone4-handbook.pdf (accessed on 6 April 2021).
96. Xilinx FPGAs. Available online: https://www.xilinx.com/products/silicon-devices/fpga.html (accessed on 6 April 2021).
97. Intel FPGAs and Programmable Devices. Available online: https://www.intel.pl/content/www/pl/pl/products/programmable.html (accessed on 6 April 2021).
98. Kilts, S. *Advanced FPGA Design: Architecture, Implementation, and Optimization*; Wiley-IEEE Press: Hoboken, NJ, USA, 2007.
99. Łuba, T.; Rawski, M.; Jachna, Z. Functional decomposition as a universal method of logic synthesis for digital circuits. In Proceedings of the 9th International Conference Mixed Design of Integrated Circuits and Systems MixDes, Wroclaw, Poland, 20–22 June 2002; Volume 2, pp. 285–290.

100. Scholl, C. *Functional Decomposition with Applications to FPGA Synthesis*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2013.
101. Nowicka, M.; Luba, T.; Rawski, M. FPGA-based decomposition of boolean functions. Algorithms and implementation. In *ACS'98: Advanced Computer Systems*; Instytut Informatyki Politechniki Szczecinskiej: Szczecin, Poland, 1998; pp. 502–509. [CrossRef]
102. Łuba, T. Multi-level logic synthesis based on decomposition. *Microprocess. Microsyst.* **1994**, *18*, 429–437. [CrossRef]
103. Machado, L.; Cortadella, J. Support-reducing decomposition for FPGA mapping. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2018**, *39*, 213–224.
104. Dahl, O.J.; Dijkstra, E.W.; Hoare, C.A.R. *Structured Programming*; Academic Press Ltd.: Cambridge, MA, USA, 1972.
105. Kolopienczyk, M.; Barkalov, A.; Titarenko, L. Hardware reduction for RAM-based Moore FSMs. In Proceedings of the 7th International Conference on Human System Interactions (HSI), Costa da Caparica, Portugal, 16–18 June 2014; pp. 255–260. [CrossRef]
106. Kołopieńczyk, M.; Titarenko, L.; Barkalov, A. Design of EMB-based Moore FSMs. *J. Circuits Syst. Comput.* **2017**, *26*, 1750125. [CrossRef]
107. Das, N.; Priya, P.A. FPGA implementation of reconfigurable finite state machine with input multiplexing architecture using hungarian method. *Int. J. Reconfigurable Comput.* **2018**. [CrossRef]
108. 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.
109. 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 2007 IEEE International Symposium on Industrial Electronics, Vigo, Spain, 4–7 June 2007; pp. 2342–2347. [CrossRef]
110. 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]
111. Senhadji-Navarro, R.; Garcia-Vargas, I. High-performance architecture for Binary-Tree-Based finite state machines. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2017**, *37*, 796–805. [CrossRef]
112. Senhadji-Navarro, R.; Garcia-Vargas, I.; Jimenez-Moreno, G.; Civit-Ballcels, A. ROM-based FSM implementation using input multiplexing in FPGA devices. *Electron. Lett.* **2004**, *40*, 1249–1251.
113. 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. [CrossRef]
114. Rawski, M.; Selvaraj, H.; Łuba, T. An application of functional decomposition in ROM-based FSM implementation in FPGA devices. *J. Syst. Archit.* **2005**, *51*, 424–434.
115. Rawski, M.; Tomaszewicz, P.; Borowik, G.; Łuba, T. 5 logic synthesis method of digital circuits designed for implementation with embedded memory blocks of FPGAs. In *Design of Digital Systems and Devices*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 121–144.
116. 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.
117. Mishchenko, A.; Chattarejee, S.; Brayton, R. Improvements to technology mapping for LUT-based FPGAs. *IEEE Trans. CAD* **2006**, *27*, 240–253. [CrossRef]
118. 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.
119. Cong, J.; Yan, K. Synthesis for FPGAs with embedded memory blocks. In Proceedings of the 2000 ACM/SIGDA Eighth International Symposium on Field Programmable Gate Arrays, Monterey, CA, USA, 10–11 February 2000; pp. 75–82.
120. Barkalov, A.; Bukowiec, A. Synthesis of mealy finite states machines for interpretation of verticalized flow-charts. *Theor. Appl. Inform.* **2005**, *5*, 39–51. [CrossRef]
121. Barkalov, A.; Titarenko, L.; Chmielewski, S. Mixed encoding of collections of output variables for LUT-based mealy FSMs. *J. Circuits Syst. Comput.* **2019**, *28*, 1950131. [CrossRef]
122. Barkalov, A.; Titarenko, L.; Mazurkiewicz, M.; Krzywicki, K. Encoding of terms in EMB-based Mealy FSMs. *Appl. Sci.* **2020**, *10*, 2762. [CrossRef]
123. Barkalov, A.; Titarenko, L.; Krzywicki, K. Reducing LUT Count for FPGA-Based Mealy FSMs. *Appl. Sci.* **2020**, *10*, 5115.
124. McElvain, K. *LGSynth93 Benchmark*; Mentor Graphics: Wilsonville, OR, USA, 1993.
125. 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 6 April 2021).
126. *VC709 Evaluation Board for the Virtex-7 FPGA User Guide*; UG887 (v1.6); Xilinx, Inc.: San Jose, CA, USA, 2019.
127. Lin, B. Synthesis of multiple-level logic from symbolic high-level description languages. In Proceedings of the IFIP International Conference on Very Large Scale Integration, Munich, Germany, 16–18 August 1989.
128. Rawski, M.; Łuba, T.; Jachna, Z.; Tomaszewicz, P. The influence of functional decomposition on modern digital design process. In *Design of Embedded Control Systems*; Springer: Boston, MA, USA, 2005; pp. 193–204. [CrossRef]
129. Barkalov, O.; Titarenko, L.; Mielcarek, K. Hardware reduction for LUT-based Mealy FSMs. *Int. J. Appl. Math. Comput. Sci.* **2018**, *28*, 595–607.

130. Barkalov, A.; Titarenko, L.; Mielcarek, K. Improving characteristics of LUT-based Mealy FSMs. *Int. J. Appl. Math. Comput. Sci.* **2020**, *30*, 745–759. [CrossRef]
131. Barkalov, A.; Titarenko, L.; Krzywicki, K.; Saburova, S. Improving Characteristics of LUT-Based Mealy FSMs with Twofold State Assignment. *Electronics* **2021**, *10*, 901. [CrossRef]
132. Barkalov, A.; Titarenko, L.; Krzywicki, K.; Saburova, S. Improving the Characteristics of Multi-Level LUT-Based Mealy FSMs. *Electronics* **2020**, *9*, 1859.