BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 69(2), 2021, Article number: e136728
DOI: 10.24425/bpasts.2021.136728

Improving LUT count of FPGA-based sequential blocks

Alexander BARKALOV[1,2], Larysa TITARENKO[1,3], Małgorzata MAZURKIEWICZ[1], and Kazimierz KRZYWICKI[4]

[1] University of Zielona Góra, ul. Licealna 9, 65-417 Zielona Góra, Poland
[2] Vasyl' Stus Dohetsk National University, 21, 600-richya str., Vinytsia, 21021, Ukraine
[3] Kharkiv National University of Radio Electronics, Nauky avenye, 14, 6166, Kharkiv, Ukraine
[4] The Jacob of Paradies University, ul. Teatralna 25, 66-400 Gorzów Wielkopolski, Poland

Abstract. Very often, a digital system includes sequential blocks which can be represented using a model of the finite state machine (FSM). It is very important to improve such FSM characteristics as the number of used logic elements, operating frequency and consumed energy. The paper proposes a novel technology-dependant design method targeting LUT-based Mealy FSMs. It belongs to the group of structural decomposition methods. The method is based on encoding the product terms of Boolean functions representing the FSM circuit. To diminish the number of LUTs, a partition of the set of internal states is constructed. It leads to three-level logic circuits of Mealy FSMs. Each function from the first level requires only a single LUT to be implemented. The method of constructing the partition with the minimum amount of classes is proposed. There is given an example of FSM synthesis with the proposed method. The experiments with standard benchmarks were conducted. They show that the proposed method can improve such FSM characteristics as the number of used LUTs. This improvement is accompanied by a decrease in performance. A positive side effect of the proposed method is a reduction in power consumption compared with FSMs obtained with other design methods.

Key words: FPGA; LUT; Mealy FSM; synthesis; structural decomposition; product terms; partition.

1. Introduction

Different sequential blocks are essential parts of modern digital systems [1–3]. Very often, the model of Mealy finite state machine (FSM) [4, 5] is used to specify sequential blocks. For example, FSM specify: 1) the complex hyper-tangent and exponential functions and other complex functions [6]; 2) the hardware-software interfaces of embedded systems [7]; 3) the blocks of integral stochastic computing [8]; 4) the activation functions for deep neural networks [9, 10]; 5) different stages of cascaded digital processing systems [11, 12]; 6) the control units of computers and other complex digital systems [5, 13–19]. Based on the popularity of FSMs, we also use this model in our current research.

Nowadays, field-programmable gate arrays (FPGA) [20–22] are very popular in digital systems design [14]. FPGA chips are used either as standalone chips [23–25] or as parts of systems-on-chips (SoC) [26, 27]. Modern SoCs are so powerful that they can implement complex digital systems. In this case, FPGAs are often used to implement hardware-software interfaces [28] and different accelerators [29]. Based on this analysis, we selected FPGAs as the basis for implementing circuits of sequential blocks.

To implement an FSM circuit, it is enough only two components of FPGA fabric [30]. These components are logic elements and a matrix of programmable interconnections [20–22]. A logic element includes a look-up table (LUT) element having S_L inputs, a programmable flip-flop and multiplexers. A LUT can implement an arbitrary Boolean function having not more than S_L arguments. The presence of flip-flops allows implementing sequential circuits.

An FSM circuit has three important characteristics [13, 14]: 1) a chip area occupied by an FSM circuit; 2) the performance and 3) the consumed power. For FPGA-based FSMs, these characteristics depend strongly on the number of LUTs in the circuit. In addition, the performance depends on the numbers of logic levels in the circuit. In turn, the consumed power is largely dependent on the total length of interconnections [24, 25].

The most crucial step in the FPGA-based design flow is a step of technology mapping [31, 32]. During this step, an FSM circuit is converted into a network of logic elements. To improve the FSM circuits characteristics, different methods of structural decomposition [33] can be used.

The main goal of this paper is to present a method of technology mapping those targets LUT-based Mealy FSMs. The method is based on the encoding of terms of Boolean functions representing an FSM logic circuit. The proposed method targets FPGA chips by Xilinx [22].

Section 2 presents the theoretical background of Mealy FSMs and peculiarities of FPGAs. Section 3 discusses the state-of-the-art in FPGA-based technology mapping. Section 4 describes the main idea of proposed method. The synthesis example is shown in Section 5. Section 6 is devoted to constructing a required partition for the set of terms. Section 7 gives results of experiments conducted on standard benchmarks [34]. A brief conclusion ends the paper.

*e-mail: m.mazurkiewicz@issi.uz.zgora.pl

Manuscript submitted 2020-06-30, revised 2020-12-01, initially accepted for publication 2020-12-11, published in April 2021

2. Background of Mealy FSM and FPGAs

A Mealy FSM can be represented using a lot of approaches. It could be state transition graphs [35], and – inversion graphs [36, 37], binary decision diagrams [16, 32, 38], just to name a few of them. To explain our approach, we choose state transition tables (STT) [35]. An STT has H rows corresponding to transitions between internal states $a_m \in A$, where $A = \{a_1, \ldots, a_M\}$ is a set of states. The transitions are determined by inputs $x_l \in X$, where $X = \{x_1, \ldots, x_L\}$ is a set of inputs. During transitions $\langle a_m, a_s \rangle$ some outputs $y_n \in Y$ are generated where $Y = \{y_1, \ldots y_N\}$ is a set of outputs. So, an STT determines three sets (X, Y, A) and two functions: the transition function and output function [35]. We discuss only FSMs having the explicit initial state $a_1 \in A$. It means that FSM operation begins from the state $a_1 \in A$.

Consider an STT (Table 1) representing some Mealy FSM S_1. The following sets can be derived from Table 1: $A = \{a_1, \ldots, a_6\}$, $X = \{x_1, \ldots, x_5\}$, $Y = \{y_1, \ldots, y_8\}$. So, there is $M = 6$, $L = 5$, and $N = 8$.

Table 1
STT of Mealy FSM S_1

a_m	a_s	X_h	Y_h	h
a_1	a_2	x_1	$y_1y_2y_4$	1
	a_3	$\bar{x}_1$	$y_3y_4y_8$	2
a_2	a_3	x_3	y_1	3
	a_5	$\bar{x}_3x_4$	y_5	4
	a_6	$\bar{x}_3\bar{x}_4$	y_6	5
a_3	a_4	x_1x_2	y_2	6
	a_5	$x_1\bar{x}_2$	y_8	7
	a_5	$\bar{x}_1$	y_3	8
a_4	a_2	x_1x_2	y_6	9
	a_3	$x_1\bar{x}_2$	y_5y_8	10
	a_5	$\bar{x}_1x_3$	y_2	11
	a_6	$\bar{x}_1\bar{x}_3$	y_1y_4	12
a_5	a_6	x_4	y_6	13
	a_4	$\bar{x}_4x_5$	y_7	14
	a_1	$\bar{x}_4\bar{x}_5$	–	15
a_6	a_2	x_3	$y_3y_5y_8$	16
	a_1	$\bar{x}_3$	y_4	17

To design an FSM circuit, it is necessary to encode the states $a_m \in A$ by binary codes $K(a_m)$. There are R bits in these codes. We discuss a case when

$$R = \lceil \log_2 M \rceil. \tag{1}$$

This step is a state assignment [35]. Its outcome significantly affects all characteristics of FSM circuits [5, 13, 14].

The states are encoded using state variables from the set $T = \{T_1, \ldots, T_R\}$. State codes are kept in a state register (RG). As a rule, D flip-flops are used in FPGA-based design [14]. To change the content of RG, input memory functions (IMF) $D_r \in \Phi$ are used, where $\Phi = \{D_1, \ldots, D_R\}$.

The FSM logic circuit is represented by the following systems of Boolean functions (SBF):

$$\Phi = \Phi(T, X); \tag{2}$$
$$Y = Y(T, X). \tag{3}$$

There is $M = 6$ for FSM S_1. Using (1) gives $R = 3$. In turn, it gives sets $T = \{T_1, T_2, T_3\}$ and $\Phi = \{D_1, D_2, D_3\}$.

To find SBFs (2)–(3), it is necessary to transform the initial STT into a direct structure table (DST). It has the following columns [12, 39]: a_m is a current state; $K(a_m)$ is a code of the state $a_m \in A$; a_s is a state of transition; $K(a_s)$ is a code of the state $a_s \in A$; X_h is an input signal causing the transition $\langle a_m, a_s \rangle$ and equal to a conjunction of some inputs $x_l \in X$ (or their complements); Y_h is a collection of outputs $y_n \in Y$ generated during the transition $\langle a_m, a_s \rangle$; Φ_h is a set of IMF equal to 1 to replace the code $K(a_m)$ by the code $K(a_s)$ into RG; h is a number of transition $(h \in \{1, \ldots, H\})$.

Each function from (2)–(3) is represented as a sum-of-products (SOP) [35]. Each SOP includes up to H product terms $F_h \in F = \{F_1, \ldots, F_H\}$. A product term F_h corresponds to the $h-th$ row of a DST [5]:

$$F_h = A_m X_h. \tag{4}$$

In (4), A_m is a conjunction of state variables (or their complements) corresponding to the code $K(a_m)$ of the state from the column a_m of DST for the row h $(h \in \{1, \ldots, H\})$.

This background information facilitates making the following conclusion. The main feature of Mealy FSMs is a direct dependence of functions $y_n \in Y$ and $D_r \in \Phi$ on variables $x_l \in X$ and $T_r \in T$. At the same time, the SOP of any function (2)–(3) can include up to $L + R$ arguments.

Modern FPGA chips include a lot of LUTs [20–22]. For example, the device 7VH870T of Virtex-7 by Xilinx includes 547000 LUTs with $S_L = 6$ [40]. Each LUT could be connected with the input of a programmable flip-flop. This connection produces a logic element (Fig. 1).

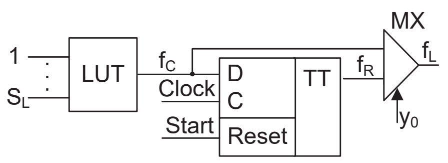

Fig. 1. Structural diagram of LUT-based logic element

A LUT can implement an arbitrary Boolean function having no more than S_L arguments. The output of LUT (f_c) is connected with input D of the flip-flop. Using a multiplexer MX allows by-passing the flip-flop. So, the output f_L of a logic element could be either combinational $(y_o = 0)$ or registered $(y_o = 1)$. Flip-flops are used to implement a distributed register. Its operation is controlled by pulses *Start* and *Clock*. If *Start* $= 1$, then $f_R = 0$. So, the pulse *Start* loads the code with all zeros into RG [5]. The pulse *Clock* allows changing the RG content.

The trivial structural diagram of Mealy FSM U_1 is shown in Fig. 2. Here the symbol LUTer determines a circuit implemented with logic elements shown in Fig. 1.

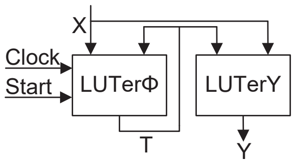

Fig. 2. Structural diagram of Mealy FSM U_1

In FSM U_1, the $LUTer\Phi$ implements the system (2), the $LUTerY$ implements the system (3). The register RG is distributed among the logic elements of $LUTer\Phi$. It explains the presence of pulses $Start$ and $Clock$ as inputs of $LUTer\Phi$.

If any function $f_i \in \Phi \cup Y$ is implemented by a single LUT, then there is only a single level of logic in both $LUTer\Phi$ and $LUTerY$. Moreover, there are only $R+N$ LUTs in the circuit of U_1. It is the best possible situation [39]. Obviously, such a circuit is characterised by a maximum operating frequency and minimum power consumption. But it is possible only if the following condition takes place:

$$NA(f_i) \leq S_L. \tag{5}$$

In (5), the symbol $NA(f_i)$ stands for the number of arguments in a function $f_i (i \in \{1, \ldots, R+N\})$.

If condition (5) is violated, then different approaches are used to improve the characteristics of FSM circuit. Some of them are analysed in Section 3.

3. State-of-the-art

The main stage of VLSI-based logic synthesis is technology mapping [31, 32, 41]. This stage's outcome affects significantly the characteristics of FSM circuits [14]. During this stage, the specifics of used logic elements should be taken into account [13, 14]. For LUT-based circuits, the main specific is a rather small number of LUT inputs ($S_L \leq 6$) [14, 24]. The functional decomposition (FD) is a key issue of LUT-based logic synthesis [42–44]. Different academic tools supporting FD are discussed, for example, in [32].

If condition (5) is violated for some function $f_i \in \Phi \cup Y$, then this function is broken down into smaller and smaller components. This is equivalent to adding new functions forming set Ψ. The process is terminated when each function representing a part of FSM circuit meets the condition (5). As a result, the FSM circuit has many levels of logic. So, the circuits of both $LUTer\Phi$ and $LUTerY$ are multi-level [43]. Obviously, multilevel circuits have lower performance and a more complex system of interconnections than their single-level counterparts.

To improve the characteristics of FSM circuits, it is necessary to reduce the number of arguments in SOPs (2)–(3). It could be done due to the proper state assignment [5, 35]. During this step, the specifics of logic elements should be taken into account [13, 14, 45–47].

One of the most popular state assignment methods is JEDI which is distributed with the system SIS [48]. We think that a similar method is used in ABC system by Berkeley [37]. Modern industrial packages use a lot of different state assignment approaches. For example, the following methods are used in design tools XST and Vivado by Xilinx [49, 50]: one-hot; compact; automatic; Gray codes; Johnson codes; speed encoding.

So, there are a lot of state assignment methods. It is really difficult to say which is the best for a particular FSM. To choose the best method, a designer should take into account the peculiarities of the following issues: 1) logic elements used; 2) an FSM model; 3) an FSM behaviour [33].

Methods of structural decomposition (SD) are based on the elimination of direct dependency of functions $f_i \in \Phi \cup Y$ on inputs $x_l \in X$ [33]. These new functions depend on $x_l \in X$ and $T_r \in T$. Each system of new functions determines a separate block LUTer having its own unique inputs and outputs. The functions $f_i \in \Psi$ are arguments of functions $y_n \in Y$ and $D_r \in \Phi$. As a rule, the following relation takes place: $|\Psi| \ll N+R$. Due to this, the total number of LUTs in LUTers implementing functions $f_i \in \Psi$ is significantly less than in blocks $LUTerY$ and $LUTer\Phi$ of FSM U_1.

The SD leads to reducing the number of arguments in SOPs of functions $y_n \in Y$ and $D_r \in \Phi$ as compared to functions (2)–(3). In turn, it reduces the number of LUTs in $LUTerY$ and $LUTre\Phi$ (as compared to U_1). If condition (5) is violated for some functions $f_i \in \Phi \cup Y \cup \Psi$, the methods of FD should be used for executing the technology mapping.

The methods of SD belong to the group of methods leading to multilevel FSM circuits [33]. The thorough analysis of SD-based multilevel FSM circuits can be found in [33]. There are the following methods of SD: the replacement of FSM inputs; the encoding of collections of outputs; the transformation of objects (states or collections of outputs); the two-fold state assignment. The number of arguments in SBFs representing SD-based circuits can be reduced using various known methods of states assignment. Due to the increase in the number of logic levels, SD-based circuits might be slower than equivalent FSM circuits based on functional decomposition. But, as a rule, SD-based circuits include fewer LUTs and consume less power compared with equivalent FD-based circuits [33, 39].

In this article, we discuss one of the methods of SD. It is the method of product terms encoding (PTE) [33]. Further, when we are talking about product terms, we mean conjunctions (4) included into SBFs (2)–(3). The original approach [33] is the following.

Let terms (4) form a set $F = \{F_1, \ldots, F_H\}$. Let us encode each term $F_h \in F$ by a binary code $K(F_h)$ having R_H bits:

$$R_H = \lceil \log_2 H \rceil. \tag{6}$$

Let us use the elements of set $Z = \{z_1, \ldots, z_{RH}\}$ for the PTE. Let us find the following systems:

$$Z = Z(T, X); \tag{7}$$

$$\Phi = \Phi(Z); \tag{8}$$

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

It leads to Mealy FSM U_2 (Fig. 3). In FSM U_2, the $LUTerZ$ implements the SBF (7), the $LUTer\Phi Y$ the systems (8)–(9). The pulses $Start$ and $Clock$ are connected with flip-flops of LEs implementing functions $D_r \in \Phi$.

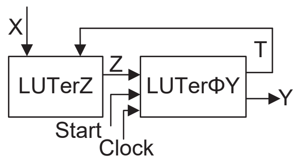

Fig. 3. Structural diagram of Mealy FSM U_2

In [51], it is proposed to implement SBF (7) using embedded memory blocks. In this case, the $LUTerZ$ is replaced by a block $EMBerZ$ including one or more EMBs. To optimize the circuit of $LUTer\Phi Y$, the set of terms is divided by classes. It allows improving area and power consumption as compared with LUT-based FSM U_2 [51]. But this approach can be used only if not all available EMBs are used for implementing various blocks of a digital system including a particular FSM. Also, there is no sense in using EMBs if the value of $L+R$ exceeds the maximum possible number of EMB address inputs. So, LUT-based design methods are more versatile than EMB-based methods. Due to this, in this article, we propose a LUT-based method of FSM design.

If $R_H \ll L+R$, then the number of LUTs in $LUTer\Phi Y$ is significantly less than their total number in the equivalent FSM U_1. The same is true for the number of logic levels. But functions (7) could be rather complex. It leads to a multi-level circuit of $LUTerZ$. In this article, we propose an approach for reducing hardware in $LUTerZ$.

The proposed method uses the idea of two-fold state assignment [33, 39]. But the methods [33, 39] have one serious drawback. Namely, each state $a_m \in A$ should have two codes. One of them determines each state as an element of the set A. The second code is necessary to determine a state as an element of some class of a partition of the set. This determines the need to use a special code converter consuming some LUTs and interconnections. In our article, we propose a method that allows the elimination of this code converter.

4. The main idea of the proposed method

Let a Mealy FSM be represented by an STT having H rows. Let us use LUTs with S_L inputs to implement FSM circuit. Let us encode the terms $F_h \in F$ by binary codes $K(F_h)$ having R_H bits. Let us form systems (7)–(9). Let condition (5) be violated for functions (7). In this case, we propose the following approach.

Let us find a partition $\Pi_A = \{A^1, \dots, A^K\}$ of the set A such that the following condition takes place:

$$R_k + L_k \le S_L \quad (k \in \{1, \dots, K\}). \tag{10}$$

In (10), R_k is the number of state variables encoding the states $a_m \in A^k$, L_k is the number of inputs $x_l \in X^k$ determining transitions from $a_m \in A^k$.

Let us use R_k state variables $T_r \in T$ to encode the states $a_m \in A^k$:

$$R_k = \lceil \log_2(|A^k| + 1) \rceil \quad (k \in \{1, \dots, K\}). \tag{11}$$

Now, sets T and Φ have R_T elements, where

$$R_T = R_1 + R_2 + \dots + R_K\,. \tag{12}$$

Each state $a_m \in A$ uniquely defines the terms $F_h \in F$ corresponding to transitions $\langle a_m, a_s \rangle$. So, the partition Π_A uniquely determines a partition $\Pi_F = \{\, F^1, \dots, F^K \}$ of the set F. Each class $F^k \in \Pi_F$ includes terms determined by the class $A^k \in \Pi_A$.

Let us encode the terms $F_h \in F$ in a way minimizing the numbers of arguments in SBFs (8)–(9). It can be done using, for example, the methods from [52].

Each class $A^k \in \Pi_A$ determines sets $X^k \subseteq X$ and $Z^k \subseteq Z$. A set $X^k \subseteq X$ includes inputs determining transitions from $a_m \in A^k$. A set $Z^k \subseteq Z$ includes additional variables $z_r \in Z$ equal to 1 for codes $K(F_h)$, where $F_h \in F$.

Based on this preliminary information, we propose the structural diagram of Mealy FSM U_3 (Fig. 4).

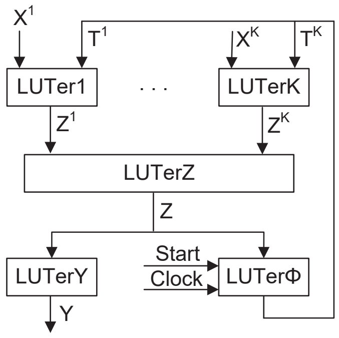

Fig. 4. Structural diagram of Mealy FSM U_3

In FSM U_3, the block $LUTerk$ implements functions

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

In (13), the symbol T^k stands for a set $T^k \subseteq T$ such that variables $T_r \in T^k$ are used to encode states $a_m \in A^k$.

The block $LUTerZ$ generates functions $z_r \in Z$. They are just disjunctions of functions $z_r^k \in Z^k$:

$$z_r = \bigvee_{k=1}^{K} z_r^k \quad (r \in \{1, \dots, R_H\}). \tag{14}$$

The $LUTer\Phi$ implements SBF (8), the $LUTerY$ SBF (9).

In each cycle of operation, only a single $LUTerk$ is “active”. It means that there are $z_r^k = 1$ only for this block. There are only zeros on outputs of other blocks $LUTerk$. These blocks are “idle”. We use the following relation showing that a block in idle:

$$T_r \in T^k \to T_r = 0. \tag{15}$$

The relation (15) explains the presence of 1 in (11). The analysis of FSM U_3 shows the following specifics. Firstly, the circuit has exactly three levels of logic blocks. Secondly, each level of U_3 has unique input and output variables. For example, the inputs $x_l \in X$ and state variables $T_r \in T$ enter only the blocks from the first level. It leads to an FSM circuit with a regular system of interconnections.

Let the symbol $U_i(S_j)$ mean that an FSM U_i is synthesized starting from STT for Mealy FSM S_j. In this article, we propose a design method for Mealy FSM $U_3(S_j)$. It includes the following steps:

1. Finding the partition Π_A that satisfies to (10).
2. Executing the state assignment for each class $A^k \in \Pi_A$.
3. Executing the encoding of terms $F_h \in F$.
4. Constructing tables for blocks $LUTer1$ - $LUTerK$.
5. Constructing SBFs (13)–(14).
6. Finding SBFs (8)–(9).
7. Implementing FSM circuit with LUTs having S_L inputs.

Let us discuss an example of synthesis for Mealy FSM $U_3(S_1)$. We use LUTs with $S_L = 5$ in the example.

5. Example of synthesis

Step 1 is very important. It largely determines the number of LUTs in a resulting circuit. We discuss this step in Section 6. In Section 5, let us just use a partition $\Pi_A = \{A_1, A_2\}$ with the classes $A^1 = \{a_1, a_3, a_4\}$ and $A^2 = \{a_2, a_5, a_6\}$.

Using STT (Table 1) and the partition Π_A gives the following sets: $X^1 = \{x_1, x_2, x_3\}$ and $X^2 = \{x_3, x_4, x_5\}$. Using (11) and (12) gives $R_1 = R_2 = 2$, $R_T = 4$, $T = \{T_1, \dots, T_4\}$. Let $T^1 = \{T_1, T_2\}$ and $T^2 = \{T_3, T_4\}$. Because $R_1 = R_2 = 2$ and $L_1 = L_2 = 3$, the condition (10) takes place if $S_L = 5$.

Let us encode the states $a_m \in A$ in the flowing way: $K(a_1) = K(a_2) = 01$, $K(a_3) = K(a_5) = 10$ and $K(a_4) = K(a_6) = 11$. We use (15) to show that a state a_m does not belong to a set $A^k \in \Pi_A$. If $T_1 = T_2 = 0$, then $a_m \notin A^1$; if $T_3 = T_4 = 0$, then $a_m \notin A^2$.

Let us form a DST of FSM S_1. Let us use the state codes from the previous step. It leads to Table 2.

Let us encode the terms $F_h \in F$ in a manner minimizing the numbers of arguments $NA(f_i)$, where $f_i \in \Phi \cup Y$. To do it, we can use methods from [52].

Using Table 2, we can derive the following SBFs:

$$\begin{aligned}
D_1 &= F_2 \vee F_3 \vee F_6 \vee F_{10} \vee F_{14};\\
D_2 &= F_6 \vee F_{14} \vee F_{15} \vee F_{17};\\
D_3 &= F_4 \vee F_7 \vee F_8 \vee F_{11} \vee F_{12} \vee F_{13};\\
D_4 &= F_1 \vee F_5 \vee F_9 \vee F_{12} \vee F_{13} \vee F_{16}.
\end{aligned} \tag{16}$$

$$\begin{aligned}
y_1 &= F_1 \vee F_3 \vee F_{12};\\
y_2 &= F_1 \vee F_6 \vee F_{11};\\
y_3 &= F_2 \vee F_8 \vee F_{16};\\
y_4 &= F_1 \vee F_2 \vee F_{12} \vee F_{17};\\
y_5 &= F_4 \vee F_{10} \vee F_{16};\\
y_6 &= F_5 \vee F_9 \vee F_{13};\\
y_7 &= F_{14};\\
y_8 &= F_2 \vee F_7 \vee F_{10} \vee F_{16}.
\end{aligned} \tag{17}$$

Table 2
DST of Mealy FSM S_1

a_m	$K(a_m)$	a_s	$K(a_s)$	X_h	Y_h	Φ_h	h
a_1	01	a_2	0001	x_1	$y_1y_2y_4$	D_4	1
		a_3	1000	$\bar{x}_1$	$y_3y_4y_8$	D_1	2
a_2	01	a_3	1000	x_3	y_1	D_1	3
		a_5	0010	$\bar{x}_3x_4$	y_5	D_3	4
		a_6	0011	$\bar{x}_3\bar{x}_4$	y_6	D_3D_4	5
a_3	10	a_4	1100	x_1x_2	y_2	D_1D_2	6
		a_5	0010	$x_1\bar{x}_2$	y_8	D_3	7
		a_5	0010	$\bar{x}_1$	y_3	D_3	8
a_4	11	a_2	0001	x_1x_2	y_6	D_4	9
		a_3	1000	$x_1\bar{x}_2$	y_5y_8	D_1	10
		a_5	0010	$\bar{x}_1x_3$	y_2	D_3	11
		a_6	0011	$\bar{x}_1\bar{x}_3$	y_1y_4	D_3D_4	12
a_5	10	a_6	0011	x_4	y_6	D_3D_4	13
		a_4	1100	$\bar{x}_4x_5$	y_7	D_1D_2	14
		a_1	0100	$\bar{x}_4\bar{x}_5$	–	D_2	15
a_6	11	a_2	0001	x_3	$y_3y_5y_8$	D_4	16
		a_1	0100	$\bar{x}_3$	y_4	D_2	17

There is $H = 17$ for S_1. Using (6) gives $R_H = 5$ and $Z = \{z_1, \dots, z_5\}$. Using methods [52] gives the codes $K(F_h)$ shown in Fig. 5.

z_4z_5 \ $z_1z_2z_3$	000	001	011	010	110	111	101	100
00	F_1	F_{12}	F_{17}	F_2	F_7	*	F_5	*
01	*	F_3	*	F_8	*	F_{14}	F_9	*
11	F_{11}	F_{15}	*	*	F_4	*	F_{13}	*
10	F_6	*	*	F_{16}	F_{10}	*	*	*

Fig. 5. Codes of product terms for FSM S_1

Using (16) and codes (Fig. 5) leads to SBF having $NA(f_i) = 5$ for $f_i \in \Phi$. But the system (17) is transformed into the following system:

$$\begin{aligned}
y_1 &= \bar{z}_1\bar{z}_2\bar{z}_4;\\
y_2 &= \bar{z}_2\bar{z}_3;\\
y_3 &= \bar{z}_1z_2\bar{z}_3;\\
y_4 &= \bar{z}_1\bar{z}_4\bar{z}_5;\\
y_5 &= z_2\bar{z}_3z_4;\\
y_6 &= z_1\bar{z}_2;\\
y_7 &= z_2\bar{z}_3\bar{z}_5;\\
y_8 &= z_2z_2z_3.
\end{aligned} \tag{18}$$

There are 22 literals in SBF (18). In the general case, it should be $N \cdot R_H = 40$ literals in this system. Each literal is equal to an interconnection between the blocks $LUTerZ$ and $LUTerY$. So, using [52] allows decreasing for the number of interconnections. In turn, it reduces the consumed power [24, 25].

A DST is a base for constructing tables for $LUTer1$ - $LUTerK$. To form a table for $LUTerk$, it is necessary:

1. To eliminate subtables corresponding to transitions from $a_m \notin A^k$.
2. To replace the columns a_s, $K(a_s)$, Y_h and Φ_h by column Z_h. This column contains variables $z_r \in Z$ equal to 1 in codes $K(F_h)$.

In the discussed case, Table 3 represents the $LUTer1$, Table 4 the $LUTer2$. We use the same numbers in the column h of Tables 2–4.

Table 3
Table of $LUTer1$

a_m	$K(a_m)$	X_h^1	Z_h^1	h
a_1	01	x_1	–	1
		$\bar{x}_1$	z_2^1	2
a_3	10	x_1x_2	z_4^1	6
		$x_1\bar{x}_2$	$z_1^1z_2^1$	7
		$\bar{x}_1$	$z_2^1z_5^1$	8
a_4	11	x_1x_2	$z_1^1z_3^1z_5^1$	9
		$x_1\bar{x}_2$	$z_1^1z_2^1z_4^1$	10
		$\bar{x}_1x_3$	$z_4^1z_5^1$	11
		$\bar{x}_1\bar{x}_3$	z_3^1	12

Table 4
Table of $LUTer2$

a_m	$K(a_m)$	X_h^2	Z_h^2	h
a_2	01	x_3	$z_3^2z_5^2$	3
		$\bar{x}_3x_4$	$z_1^2z_2^2z_4^2z_5^2$	4
		$\bar{x}_3\bar{x}_4$	$z_1^2z_3^2$	5
a_5	10	x_4	$z_1^2z_3^2z_4^2z_5^2$	13
		$\bar{x}_4x_5$	$z_1^2z_2^2z_3^2z_5^2$	14
		$\bar{x}_4\bar{x}_5$	$z_3^2z_4^2z_5^2$	15
a_6	11	x_3	$z_2^2z_4^2$	16
		$\bar{x}_3$	$z_2^2z_3^2$	17

Using Tables 3 and 4, we can obtain SBF (13). For example (after minimization), we can get the following:

$$\begin{aligned} z_1^1 &= T_1\bar{T}_2x_1\bar{x}_2 \vee T_1T_2x_1; \\ z_1^2 &= \bar{T}_3T_4\bar{x}_3 \vee T_3\bar{T}_4x_4 \vee \bar{T}_3T_4x_5 . \end{aligned} \tag{19}$$

The functions (19) are used as arguments of SBF (14). The equations (14) are obtained in the trivial way:

$$z_1 = z_1^2 \vee z_1^2; \ldots z_5 = z_5^1 \vee z_5^2 . \tag{20}$$

The system $Y(Z)$ is already found. It is the system (18). The system $\Phi(Z)$ is constructed using the system (16) and the codes $K(F_h)$ from Karnaugh map (Fig. 5). Our analysis shows that each function $D_r \in \Phi$ depends on 5 variables.

There are all variables $z_r \in Z$ in the columns Z_h^1 and Z_h^2. It means that there are 10 LUTs in the circuits

of $LUTer1$–$LUTer2$. Also, it means that there are $R_H = 5$ LUTs in the circuit of $LUTerZ$. As follows from (18), there are $N = 8$ LUTs in the circuit of $LUTerY$. At last, there are $R_T = 4$ LUTs in the circuit of $LUTerT$.

The logic circuit of FSM $U_3(S_1)$ is shown in Fig. 6. As follows from Fig. 6, there are 27 LUTs with $S_L = 5$ in the circuit. There are three levels of LUTs in the circuit. Also, the circuit has a regular system of interconnections.

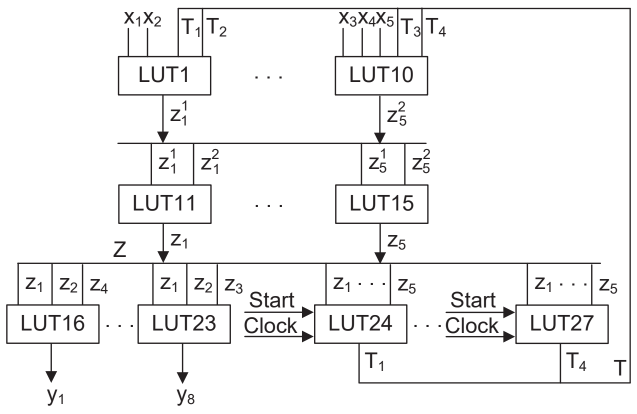

Fig. 6. Logic circuit of FSM $U_3(S_1)$

The last step of the design method assumes using standard industrial tools such as Vivado by Xilinx [50]. We do not discuss this step for our example.

6. Constructing partition Π_A

We propose a simple sequential algorithm for finding the partition Π_A satisfying (10) and having the minimum possible numbers of blocks.

Let us characterize the state $a_m \in A$ by two sets. A set $X(a_m)$ includes inputs determining transitions from state $a_m \in A$. A set $Z(a_m)$ includes variables $z_r \in Z$ equal to 1 in codes of terms $F_h \in F$ corresponding to transitions from state $a_m \in A$. If $a_m \in A^k$, then $X(a_m) \subseteq X^k$ and $Z(a_m) \subseteq Z^k$.

To find Π_A, we use two evaluations. The first of them determines how many new elements will be added to X^k due to including the state a_m into the class A^k. It is the following evaluation:

$$N(a_m, X^k) = |X(a_m) \backslash X^k|. \tag{21}$$

In (21), the symbol "\" stands for the subtraction of sets.

Let A^* be a set of states which were not included into classes $A^1, \ldots, A^{k-1}$. Let X^* be a set including inputs $x_l \in X$ from sets $X(a_m)$ where $a_m \in A^*$. The second evaluation, $N(a_m, X^*)$, shows how many inputs are excluded from the further analysis due to including a_m into A^k:

$$N(a_m, X^*) = |X(a_m) \backslash \underset{j}{\cup} X(a_j)|. \tag{22}$$

In (22), there is the following relation: $a_j \in A^* \backslash a_m$.

There are two stages in generating a block $A^k \in \Pi_A$. At the first stage, a base element (BE) of A^k is selected. A BE should satisfy the following relation:

$$|X(a_m)| = \max_j |X(a_j)|,\ a_j \in A^* \backslash \{a_m\}. \tag{23}$$

If condition (23) takes place for states a_m and a_s, let us choose a state a_m with $m > s$.

The second stage is a multistep one. During each step, the next state is successively added to the block A^k. The process is terminated when: 1) all states are distributed or 2) it is not possible to add states into A^k due to violation of (10).

Let A^* include all unallocated states $a_m \in A$. We use the following rule to add a state into A^k. Let us construct a set $P(A^k)$ including all states $a_m \in A^*$ whose including into A^k does not violate the condition (10). Let us find a state $a_m \in P(A^k)$ such as

$$N(a_m, X^k) = \min_j\ N(a_j, X^k), a_j \in P(A^k) \backslash \{a_m\}. \tag{24}$$

If the property (24) takes places for more than a single state, then let us choose a state with the following property:

$$N(a_m, X^*) = \max_j\ N(a_j, X^*), a_j \in P(A^k) \backslash \{a_m\}. \tag{25}$$

If the property (25) is true for several states, then one of them is included into A^k. Next, we should make $P(A^k)$ empty.

The proposed design method may be used if $|X(a_m)| < S_L$. If there is $|X(a_m)| \geq S_L$, condition (10) is violated even for this single state $a_m \in A$. In this case, other design methods should be used. We do not discuss this situation in this article.

To better understand the method of constructing the partition Π_A, we show the constructing process as a flowchart (Fig. 7). So, we should check if the set of states contains a state $a_m \in A$ with $|X(a_m)| \geq S_L$. A loop consisting of *blocks*1-3 is used for checking (Fig. 7). If $|X(a_m)| < S_L$ (output "No" of *block*1), then the next state is selected (*block*2). Otherwise, the process is terminated (output "Yes" of *block*1). If not all states are checked yet (output "No" of *block*3), then the checking process is continued. The constructing process begins if all states are checked (output "Yes" of block3). This part of the flowchart is clear from the description of the method. The process is terminated, if all states $a_m \in A$ are distributed among different classes of Π_A (outputs "Yes" of either *block*4 or *block*5).

Let us discuss an example of constructing the partition Π_A for FSM $U_3(S_1)$. The process is shown in Table 5.

We assume that $S_L = 5$. As follows from (10), the following relation should take place: $L_i + R_i = 5$. It means that the following pairs $\langle L_i, R_i \rangle$ are possible: $\langle 0,5 \rangle$ (the class A^k contains up to 31 states with unconditional transitions), $\langle 0,4 \rangle$ (the class A^k contains up to 15 states), $\langle 2,3 \rangle$, $\langle 3,2 \rangle$, $\langle 4,1 \rangle$ (the class A^k contains a single state whose transitions depend on 4 inputs).

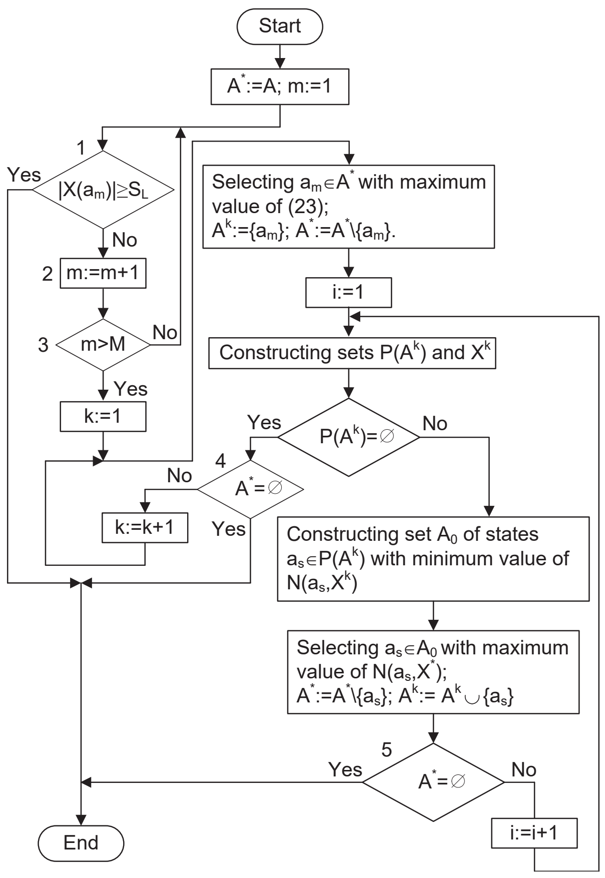
Start
A*:=A; m:=1
1
Yes
|X(a_m)|≥S_L
No
2
m:=m+1
3
m>M
No
Yes
k:=1
Selecting a_m∈A* with maximum value of (23); A^k:={a_m}; A*:=A*\{a_m}.
i:=1
Constructing sets P(A^k) and X^k
Yes
P(A^k)=∅
No
4
No
A*=∅
Yes
k:=k+1
Constructing set A_0 of states a_s∈P(A^k) with minimum value of N(a_s,X^k)
Selecting a_s∈A_0 with maximum value of N(a_s,X*); A*:=A*\{a_s}; A^k:= A^k∪{a_s}
5
Yes
A*=∅
No
i:=i+1
End

Fig. 7. Flowchart of constructing partition Π_A

Table 5
Constructing partition Π_A

a_m	$\lvert X(a_m)\rvert$	BE1	I/II		BE2	I/2	
			1	2		1	2
a_1	1		0/1 ⊕	–	–	–	–
a_2	2		2	2	⊕	–	–
a_3	2		0/0	0 ⊕	–	–	–
a_4	3	⊕	–	–	–	–	–
a_5	2		2	2		0/0 ⊕	–
a_6	1		0/0	0		0/0	0
A^k		a_4	a_1	a_3	a_2	a_5	a_6

Let us explain the columns of Table 5.

The column a_m contains states of FSM. The column $|X(a_m)|$ contains the numbers of inputs $x_l \in X(a_m)$. The columns BE_k ($k \in \{1,2\}$) contain basic elements for step k. The symbol "I" stands for $N(a_m, X^k)$, the symbol "II" for $N(a_m, X^*)$. The sign "⊕" means that a state from the corresponding row is included into A^k. The sign "–" means that the corresponding state is eliminated from further consideration. There are states $a_m \in A^k$ in the row A^k. They are listed in the order of selection.

As follows from Table 5, there are $M = 6$ steps in the constructing Π_A. As a result, the following partition is constructed: $\Pi_A = \{A^1, A^2\}$ with $A^1 = \{a_1, a_3, a_4\}$ and $A^2 = \{a_2, a_5, a_6\}$. So, it is the same as we used in Section 5.

Using rules (24)–(25) allows reducing the number of the same inputs $x_l \in X$ in different sets $X^k \subseteq X$. It leads to a decrease in the number of LUTs compared to when inputs are duplicated in different sets $X^k \subseteq X$. Also, it leads to regularization of the system of interconnections.

7. Results of experiments

To compare the proposed method with some other known design methods, we need some standard benchmarks. As many other researchers, we use the library [53]. There are 48 different benchmarks in this library represented in KISS2 format. We show the characteristics of these FSMs in Table 6.

The system Vivado [50] was used to implement FSM circuits and estimate such their characteristics as the number of LUTs in the circuit, the maximum operating frequency and consumed power. The VHDL-based files should be created to connect benchmarks and Vivado. To translate KISS2 files into VHDL, we used the CAD system K2F [33]. The Active-HDL environment was used for synthesis and simulation of benchmark FSMs. To execute the technology mapping, placement and routing, we used the industrial package Vivado [50] by Xilinx. As a target platform, we used the FPGA chip XC7VX690tffg1761-2 by Virtex 7. The configurable logic blocks of this chip include LUTs with up to 6 inputs.

We took four different methods to compare with our approach. Two of them are methods used directly in Vivado 2019.1. They are the following methods: 1) Auto (when the best encoding approach is chosen by the CAD system) and 2) One-hot (this approach is very popular in FSM design). We used JEDI-based FSMs as a third approach. It is known that JEDI [54] produces state codes optimizing the numbers of arguments in SBFs representing FSM circuits. At last, we used the system DEMAIN [55] to produce circuits for benchmark FSMs. This system uses really good methods of functional decomposition. Three tables represent the outcomes of experiments. We found such characteristics as the number of LUTs (Table 7), maximum operating frequency (Table 8) and power consumption (Table 9). All design methods were based on the model U_1 shown in Fig. 2.

We used the same organization for Tables 7–9. The columns of each table include names of different methods. We name these methods as Auto, One-hot, JEDI, DEMAIN and U_3. The rows of each table include the names of benchmarks. There are two additional rows in each table. The results of the summation of corresponding values are shown in the row "Total". This row includes the results of summation for numbers of LUTs (Table 7), maximum operating frequency (Table 8) and power consumption (Table 9). We took the summarized characteristics of U_3-based FSMs as 100%. The percentage of summarized characteristics respectively to benchmarks based on U_3 is shown in the row "Percentage" of Tables 7–9.

Table 6
Characteristics of benchmarks

Benchmark	L	N	H	M	R
bbara	4	2	60	10	4
bbsse	7	7	56	16	4
bbtas	2	2	24	6	3
bbcount	3	4	28	7	3
cse	7	7	91	16	4
dk14	3	5	56	7	3
dk15	3	5	32	4	2
dk16	2	3	108	27	5
dk17	2	3	32	8	3
dk27	1	2	14	7	3
dk512	1	3	30	15	4
donfile	2	1	96	24	5
ex1	9	19	138	20	5
ex2	2	2	72	19	5
ex3	2	2	36	10	4
ex4	6	9	21	14	4
ex5	2	2	32	9	4
ex6	5	8	34	8	3
ex7	2	2	36	10	4
keyb	7	7	170	19	5
kirkman	12	6	370	16	4
lion	2	1	11	4	2
lion9	2	1	25	9	4
mark1	5	16	22	15	4
mc	3	5	10	4	2
modulo12	1	1	24	12	4
opus	5	6	22	10	4
planet	7	19	115	48	6
planet1	7	19	115	48	6
pma	8	8	73	24	5
s1	8	6	107	20	5
s1488	8	19	251	48	6
s1494	8	19	250	48	6
s1a	8	6	107	20	5
s208	11	2	153	18	5
s27	4	1	34	6	3
s386	7	7	64	13	4
s8	4	1	20	5	3
sand	11	9	184	32	5
shifreg	1	1	16	8	3
sse	7	7	56	16	4
styr	9	10	166	30	5
tma	7	6	44	20	5
s420	19	2	137	18	5
s510	19	7	77	47	6
s820	18	19	232	25	5
s832	18	19	245	25	5

Table 7
Experimental results (the number of LUTs)

Benchmark	Auto	One-Hot	JEDI	DEMAIN	U_3
bbara	17	17	10	9	12
bbsse	33	37	24	26	23
bbtas	5	5	5	5	6
beecount	19	19	14	16	13
cse	40	66	36	38	33
dk14	10	27	10	12	13
dk15	5	16	5	6	7
dk16	15	34	12	14	11
dk17	5	12	5	6	6
dk27	3	5	4	4	6
dk512	10	10	9	10	9
donfile	31	31	22	26	20
ex1	70	74	53	57	43
ex2	9	9	8	9	9
ex3	9	9	9	9	9
ex4	15	13	12	13	11
ex5	9	9	9	9	9
ex6	24	36	22	23	21
ex7	4	5	4	4	5
keyb	43	61	40	42	38
kirkman	42	58	39	41	36
lion	2	5	2	2	3
lion9	6	11	5	5	6
mark1	23	23	20	21	19
mc	4	7	4	5	5
modulo12	7	7	7	7	8
opus	28	28	22	26	24
planet	131	131	88	94	81
planet1	131	131	88	94	81
pma	94	94	86	91	79
s1	65	99	61	64	58
s1488	124	131	108	112	93
s1494	126	132	110	117	95
s1a	49	81	43	54	42
s208	12	31	10	11	10
s27	6	18	6	6	7
s386	26	39	22	25	19
s420	10	31	9	10	9
s510	48	48	32	39	30
s8	9	9	9	9	11
s820	88	82	68	76	59
s832	80	79	62	70	55
sand	132	132	114	121	102
shiftreg	2	6	2	2	5
sse	33	37	30	32	27
styr	93	120	81	88	74
tma	45	39	39	41	34
Total	1792	2104	1480	1601	1377
Percentage	130.10%	152.80%	107.50%	116.30%	100%

Table 8
Experimental results (the operating frequency, MHz)

Benchmark	Auto	One-Hot	JEDI	DEMAIN	U_3
bbara	193.39	193.39	212.21	198.46	153.52
bbsse	157.06	169.12	182.34	178.91	117.31
bbtas	204.16	204.16	206.12	208.32	164.28
beecount	166.61	166.61	187.32	184.21	126.98
cse	146.43	163.64	178.12	174.19	106.34
dk14	191.64	172.65	193.85	187.32	151.62
dk15	192.53	185.36	194.87	188.54	152.51
dk16	169.72	174.79	197.13	189.83	129.35
dk17	199.28	167	199.39	172.19	159.22
dk27	206.02	201.9	204.18	205.1	166.04
dk512	196.27	196.27	199.75	197.49	156.45
donfile	184.03	184	203.65	194.83	144.65
ex1	150.94	139.76	176.87	186.14	110.71
ex2	198.57	198.57	200.14	199.75	158.58
ex3	194.86	194.86	195.76	193.43	154.89
ex4	180.96	177.71	192.83	178.14	140.92
ex5	180.25	180.25	181.16	181.76	140.42
ex6	169.57	163.8	176.59	174.12	129.34
ex7	200.04	200.84	200.6	200.32	160.58
keyb	156.45	143.47	168.43	157.16	116.29
kirkman	141.38	154	156.68	143.76	101.06
lion	202.43	204	202.35	201.32	162.58
lion9	205.3	185.22	206.38	205.86	165.51
mark1	162.39	162.39	176.18	169.65	122.32
mc	196.66	195.47	196.87	192.53	156.12
modulo12	207	207	207.13	207.37	167.45
opus	166.2	166.2	178.32	168.79	126.27
planet	132.71	132.71	187.14	185.73	92.43
planet1	132.71	132.71	187.14	185.73	92.92
pma	146.18	146.18	169.83	153.57	106.53
s1	146.41	135.85	157.16	149.17	106.73
s1488	138.5	131.94	157.18	153.12	98.94
s1494	149.39	145.75	164.34	159.42	109.19
s1a	153.37	176.4	169.17	158.12	113.67
s208	174.34	176.46	178.76	172.87	134.42
s27	198.73	191.5	199.13	198.43	158.76
s386	168.15	173.46	179.15	169.21	128.2
s420	173.88	176.46	177.25	172.87	133.07
s510	177.65	177.65	198.32	183.18	97.39
s8	180.02	178.95	181.23	180.39	140.22
s820	152	153.16	176.58	166.29	112.18
s832	145.71	153.23	173.78	160.03	105.82
sand	115.97	115.97	126.82	120.63	75.18
shiftreg	262.67	263.57	276.26	276.14	222.95
sse	157.06	169.12	174.63	169.69	117.21
styr	137.61	129.92	145.64	138.83	97.07
tma	163.88	147.8	164.14	168.19	123.24
Total	8126.95	8173.06	8719.07	8103.27	6246.64
Percentage	130.10%	130.80%	139.60%	129.70%	100%

Table 9
Experimental results (the consumed power, Watts)

Benchmark	Auto	One- Hot	JEDI	DEMAIN	U_3
bbara	0.569	0.569	0.488	0.482	0.599
bbsse	2.22	1.206	1.713	1.824	1.722
bbtas	0.533	0.533	0.533	0.533	0.682
beecount	1.631	1.631	1.021	1.236	1.035
cse	0.958	1.019	0.891	0.911	0.883
dk14	2.959	3.33	2.952	2.998	3.102
dk15	1.403	1.905	1.399	1.402	1.512
dk16	2.967	2.742	2. 512	2.715	2.535
dk17	1.901	1.935	1.891	1.938	2.007
dk27	1.168	0.854	1.158	1.161	1.272
dk512	1.496	1.496	1.345	1.498	1.365
donfile	0.709	0.709	0.603	0.638	0.678
ex1	4.102	2.968	2.342	2.416	2.028
ex2	0.368	0.386	0.342	0.365	0.467
ex3	0.391	0.391	0.391	0.394	0.474
ex4	1.562	1.241	1.187	1.198	1.223
ex5	0.387	0.387	0.385	0.383	0.426
ex6	2.269	3.85	2.242	2.258	2.275
ex7	0.992	1.181	0.994	0.996	1.098
keyb	1.093	1.071	1.075	1.082	1.096
kirkman	1.693	1.844	1.439	1.498	1.427
lion	0.542	0.629	0.547	0.544	0.649
lion9	0.733	0.97	0.728	0.73	0.884
mark1	1.445	1.445	1.227	1.301	1.287
mc	0.447	0.561	0.443	0.492	0.562
modulo12	0.559	0.559	0.563	0.532	0.648
opus	1.344	1.344	1.283	1.334	1.321
planet	4.122	4.122	2.456	3.002	2.428
planet1	4.122	4.122	2.456	3.002	2.338
pma	1.37	1.37	1.253	1.361	1.103
s1	2.685	3.13	2.518	2.612	2.448
s1488	3.982	4.096	3.548	3.629	2.183
s1494	3.079	3.178	2.982	3.011	2.758
s1a	1.322	2.01	1.208	1.602	1.185
s208	1.367	2.82	1.249	1.302	1.357
s27	0.756	1.95	0.765	0.769	0.864
s386	1.251	1.393	1.121	1.187	1.198
s420	1.337	2.82	1.286	1.334	1.392
s510	1.543	1.543	1.091	1.218	1.102
s8	0.736	0.805	0.732	0.734	0.982
s820	2.054	1.801	1.463	1.612	1.243
s832	2.096	2.087	1.828	1.512	1.332
sand	1.149	1.149	0.988	1.017	0.917
shiftreg	0.523	0.603	0.512	0.503	0.812
sse	1.22	1.296	1.089	1.193	1.107
styr	4.044	4.771	3.187	3.612	3.032
tma	1.589	1.314	1.321	1.427	1.218
Total	85.479	89.585	65.935	68.498	63.346
Percentage	134.90%	141.40%	104.10%	108.10%	100%

Analysis of Table 7 shows that the U_3-based FSMs have better LUT counts than their counterparts used in our experiments. There is the following decrease: 1) 34% in relation to results obtained for Auto-based FSMs; 2) 41.4% in relation to results obtained using One-hot state assignment; 3) 4.1% in relation to results obtained for JEDI-based FSMs and 4) 8.1% in relation to results obtained for DEMAIN-based FSMs. As a rule, our approach gives better results for FSMs having more than 15 states. If $M < 15$, then the better results are produced by either JEDI or DEMAIN.

Analysis of Table 8 shows that the U_3-based FSMs have a lower operating frequency as compared to circuits produced by their counterparts used in our experiments. This can be explained by the fact that the circuits of U_3-based FSMs have at least three levels of logic. Because the library [53] includes rather simple FSMs, other design methods can produce FSM circuits with fewer logic levels. We think that for complex FSMs, our method can give a gain in frequency, too. To test this assumption, we have generated several FSMs having $M = 200$, $H = 2000$, $L = 30$ and $N = 50$. For all these complex FSMs, we have around 10% gain in operating frequency.

Our approach allows producing FSM circuits with a more regular interconnected system. Interconnections are known to be responsible for 60%–70% of power losses in FPGA-based circuits [24]. Due to this, our approach gives a gain in power consumption compared to other investigated methods. As follows from Table 9, our approach gives the following gain: 1) 30.1% in comparison with Auto; 2) 52.8% in comparison with One-hot; 3) 7.5% in comparison with JEDI and 4) 16.3% in comparison with DEMAIN.

So, our approach produces better results for rather complex FSMs (for the number of LUTs and consumed power). Of course, this conclusion is true only for the benchmarks [53] and the device XC7VX690tffy1761-2. It is almost impossible to make similar conclusion for the general case. Research [31] shows that the value $S_L = 6$ is optimal from the point of view of the area/productivity/power tradeoff for LUTs. So, the proposed approach can be successfully used both now and in the near future.

8. Conclusions

If FPGA chips are used to implement digital systems, then the circuits of sequential logic blocks can be implemented as networks of LUTs. Complex sequential blocks are represented by systems of Boolean functions having a lot of arguments. But a LUT has a very limited amount of inputs. This value cannot be increased due to the violation of the balance of such characteristics as the chip area occupied by a LUT, the performance and consumed power. This contradiction leads to multi-level circuits of complex sequential blocks. In turn, it leads to the necessity of using different methods of functional decomposition to implement the circuits of digital systems.

The structural decomposition is a good alternative to the functional decomposition. In the case of FSMs, these methods allow elimination of the direct dependence between inputs and outputs. Also, the structural decomposition allows obtaining circuits with regular system of interconnections [33]. Due to it, FSM circuits based on the methods of structural decomposition can have fewer numbers of LUTs, a higher operating frequency and lower power consumption then equivalent FSM circuits designed with methods of functional decomposition.

A novel method of structural decomposition is proposed in the current article. The method is aimed at LUT-based Mealy FSMs. The essence of this method is the encoding of the product terms corresponding to rows of direct structure table representing FSMs. The proposed method is technology depended because it forms a partition of the set of terms based on amount of inputs of LUTs.

To analyse the efficiency of the proposed method, we have conducted experiments using standard benchmarks from the library [53]. The experiments show that the proposed method leads to reduction in both the number of LUTs and power consumption in comparison with FSM circuits obtained by the Xilinx CAD tool Vivado 2019.1, JEDI- and DEMAIN-based FSMs. All methods chosen for comparison are based on functional decomposition. But our approach produces slower circuits. But this disadvantage is reduced with the growth of the complexity of sequential blocks.

Till now, our methods were aimed at FPGAs produced by Xilinx. Next, we are going to apply the proposed approach for FPGAs produced by other companies different from Xilinx [20, 21]. Also, we will try to apply this method to optimize sequential blocks based on counters.

References

[1] J. Baillieul and T. Samad, *Encyclopedia of Systems and Control*, Springer, 2015.

[2] M. Arora, *Embedded System Design, Introduction to SoC System Architecture*, Learning Bytes Publishing, 2016.

[3] V. Chakravarthi, *A Practical Approach to VLSI System on Chip (SoC) Design, A Comprehensive Guide*, Springer, 2020.

[4] P. Minns and I. Elliot, *FSM-based digital design using Verilog HDL*, John Wiley and Sons, 2008.

[5] S. Baranov, *Logic and System Design of Digital Systems*, Tallinn: TUT Press, 2008.

[6] B.D. Brown and H.C. Card, "Stochastic neural computation. I computational elements", *IEEE Trans. Comput.* 50(9), 891–905 (2001).

[7] O. Barkalov, L. Titarenko, and M. Mazurkiewicz, *Foundations of Embedded Systems*, Springer, 2019.

[8] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu, and W.J. Gross, "VLSI Implementation of Deep Neural Network Using Integral Stochastic Computing", *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* 25(10), 26882699 (2017).

[9] P. Li, D.J. Lilja, W. Qian, M.D. Riedel, and K. Bazargan, "Logical computation on stochastic bit streams with linear finitestate machines", *IEEE Trans. Comput.* 63(6), 1474–1486 (2014).

[10] Y. Xie, S. Liao, B. Yuan, Y. Wang, and Z. Wang, "Fully-parallel area-efficient deep neural network design using stochastic computing", *IEEE Trans. Circuits Syst. II-Express Briefs* 64(12), 1382–1386 (2017).

[11] N. Das and P.A. Priya, "FPGA Implementation of Reconfigurable Finite State Machine with Input Multiplexing Architecture Using Hungarian Method", *Int. J. Reconfigurable Comput.* 2018, 6831901 (2018).

[12] J. Glaser, M. Damm, J. Haase, and C. Grimm, "TR-FSM: Transition-Based Reconfigurable Finite State Machine", *ACM Trans. Reconfigurable Technol. Syst.* 4, 23:1–23:14 (2011).

[13] R. Czerwinski and D. Kania, *Finite State Machine Logic Synthesis for Complex Programmable Logic Devices*, Springer, 2013.

[14] V. Sklyarov, I. Skliarova, A. Barkalov, and L. Titarenko, *Synthesis and optimization of FPGA-based systems*, Springer, 2014.

[15] M. Kubica, D. Kania, and J. Kulisz, "A Technology Mapping of FSMs Based on a Graph of Excitations and Outputs", *IEEE Access* 7, 16123–16131 (2019).

[16] A. Opara, M. Kubica, and D. Kania, "Methods of Improving Time Efficiency of Decomposition Dedicated at FPGA Structures and Using BDD in the Process of Cyber-Physical Synthesis", *IEEE Access* 7, 20619–20631 (2019).

[17] M. Kubica and D. Kania, "Area-oriented technology mapping for LUT-based logic blocks", *Int. J. Appl. Math. Comput. Sci.* 27(1), 207–222 (2017).

[18] M. Kubica, A. Opara, and D. Kania, "Logic Synthesis for FPGAs Based on Cutting of BDD", *Microprocess. Microsyst.* 52, 173–187 (2017).

[19] I. Skliarova, V. Sklyarov, and A. Sudnitson, *Design of FPGAbased circuits using Hierarchical Finite State Machines*, Tallinn: TUT Press, 2012.

[20] Altera, [Online]. http://www.altera.com (accesed: May, 2020).

[21] Atmel, [Online]. http://www.atmel.com (accesed: May, 2020).

[22] Xilinx, [Online]. http://www.xilinx.com (accesed: May, 2020).

[23] I. Kuon, R. Tessier, and J. Rose, "FPGA architecture: Survey and Challenges", *Found. Trends Electron. Design Automat.* 2(2), 135–253 (2008).

[24] I. Grout, *Digital Systems Design with FPGAs and CPLDs*, Elsevier Science, 2011.

[25] S. Kilts, *Advanced FPGA Design: Architecture, Implementation, and Optimization*, Wiley-IEEE Press, 2007.

[26] Intel, "Intel SoC FPGA Embedded Development Suite User Guide". [Online]. https://www.intel.com/content/www/us/en/programmable/documentation/lro1402536290550.html (accesed: May, 2020).

[27] Xilinix, "Zynq UltraScale+MPSoC". [Online]. https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html#productTable (accesed: May, 2020).

[28] G. Stringham, *Hardware/firmware Interface Design: Best Practices for Improving Embedded Systems Development*, Newnes, 2010.

[29] I. Skliarova and V. Sklyarov, *FPGA-BASED hardware accelerators*, Springer, 2019.

[30] T. Łuba, M. Rawski, and Z. Jachna, "Functional Decomposition as a universal method for logic synthesis of digital circuits", in *Proceedings of IX International Conference MIXDES'02*, 2002, p. 285290.

[31] A. Ling, D.P. Singh, and S.D. Brown, "FPGA technology mapping: a study of optimality", in *Proceedings 42nd Design Automation Conference (DAC05)*, 2005, pp. 427–432.

[32] M. Kubica and D. Kania, "Technology mapping oriented to adaptive logic modules", *Bull. Pol. Acad. Sci. Tech. Sci.* 67(5), 947–956 (2019).

[33] O. Barkalov, L. Titarenko, K. Mielcarek, and S. Chmielewski, *Logic Synthesis for FPGA-Based Control Units: Structural Decomposition in Logic Design*, Springer, 2020.

[34] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide", tech. rep., Microelectronic Center of North Carolina, 1991.

[35] G.D. Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.

[36] E. Testa, L. Amaru, M. Soeken, A. Mishchenko, P. Vuillod, J. Luo, C. Casares, P. Gaillardon, and G.D. Micheli, "Scalable boolean methods in a modern synthesis flow", in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 1643–1648.

[37] R. Brayton and A. Mishchenko, "ABC: An Academic Industrial-Strength Verification Tool", in *Computer Aided Verification*, pp. 24–40 eds. T. Touili, B. Cook, and P. Jackson, Springer, 2010.

[38] A. Opara, M. Kubica, and D. Kania, "Methods of Improving Time Efficiency of Decomposition Dedicated at FPGA Structures and Using BDD in the Process of Cyber-Physical Synthesis", *IEEE Access* 7, 20619–20631 (2019).

[39] O. Barkalov, L. Titarenko, and K. Mielcarek, "Hardware reduction for LUT-based Mealy FSMs", *Int. J. Appl. Math. Comput. Sci.* 28(3), 595–607 (2018).

[40] Xilinix, "Virtex-7 family overview". [Online]. https://www.xilinx.com/products/silicon-devices/fpga/virtex-7.html (accesed: May, 2020).

[41] A. Mishchenko, R.K. Brayton, J.H.R. Jiang, and S. Jang, "Scalable Don't-Care-Based Logic Optimization and Resynthesis", *ACM Trans. Reconfigurable Technol. Syst.* 4(4), 34(1–23) (2011).

[42] A. Mishchenko, S. Chatterjee, and R.K. Brayton, "Improve-ments to Technology Mapping for LUT-based FPGAs", *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 26(2), 240253 (2007).

[43] C. Scholl, *Functional Decomposition with Application to FPGA Synthesis*, Kluwer Academic Publishers, 2001.

[44] L. Machado and J. Cortadella, "Support-Reducing Decomposition for FPGA Mapping", *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 39(1), 213–224 (2020).

[45] R. Czerwinski, D. Kania, and J. Kulisz, "FSMs state encoding targeting at logic level minimization", *Bull. Pol. Acad. Sci. Tech. Sci.* 54(4), 479–487 (2006).

[46] R. Czerwinski and D. Kania, "Synthesis method of high speed finite state machines", *Bull. Pol. Acad. Sci. Tech. Sci.* 58(4), 635–644 (2010).

[47] A. Opara and D. Kania, "Decomposition-based logic synthesis for PAL-based CPLDs," *Int. J. Appl. Math. Comput. Sci.* 20(2), 367–384 (2010).

[48] E. Sentowich, et al., "SIS: a system for sequential circuit synthesis", in *Proc. of the Inter. Conf. of Computer Design (ICCD'92)*, 1992, p.328333.

[49] Xilinx, "XST User Guide. V. 11.3". [Online]. http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/xst.pdf (accesed: May, 2020).

[50] Vivado. [Online]. https://www.xilinx.com/products/design_tools/vivado.html (accesed: May, 2020).

[51] A. Barkalov, L. Titarenko, M. Mazurkiewicz, and K. Krzywicki, "Encoding of terms in EMB-Based Mealy FSMs", *Appl. Sci.* 10(8), 21 (2020).

[52] S. Achasova, *Synthesis algorithms for automata with PLAs*, M: Soviet radio, 1987.

[53] LGSynth93, "International Workshop on logic synthesis benchmark suite (LGSynth93)". [Online]. https://people.engr.ncsu.edu/brglez/CBL/benchmarks/LGSynth93/LGSynth93.tar, 1993 (accesed: February, 2018).

[54] B. Lin, "Synthesis of multiple-level logic from symbolic high-level description languages", in *IFIP International Conference on Very Large Scale Integration*, 1989, pp. 187–196).

[55] M. Rawski, L. Jozwiak, M. Nowicka, and T. Łuba, "Nondisjoint decomposition of boolean functions and its application in FPGA-oriented technology mapping", in *EUROMICRO 97. Proceedings of the 23rd EUROMICRO Conference: New Frontiers of Information Technology (Cat. No.97TB100167)*, 1997, pp. 24–30.