sciendo

Int. J. Appl. Math. Comput. Sci., 2018, Vol. 28, No. 3, 595–607
DOI: 10.2478/amcs-2018-0046

amcs

HARDWARE REDUCTION FOR LUT–BASED MEALY FSMs

ALEXANDER BARKALOV [a], LARYSA TITARENKO [a], KAMIL MIELCAREK [a,*]

[a] Institute of Metrology, Electronics and Computer Science
University of Zielona Góra, ul. prof. Z. Szafrana 2, 65-516 Zielona Góra, Poland
e-mail: {A.Barkalov,L.Titarenko,K.Mielcarek}@imei.uz.zgora.pl

A method is proposed targeting a decrease in the number of LUTs in circuits of FPGA-based Mealy FSMs. The method improves hardware consumption for Mealy FSMs with the encoding of collections of output variables. The approach is based on constructing a partition for the set of internal states. Each state has two codes. It diminishes the number of arguments in input memory functions. An example of synthesis is given, along with results of investigations. The method targets rather complex FSMs, having more than 15 states.

Keywords: Mealy FSM, synthesis, FPGA, LUT, partition, encoding collections of output variables.

1. Introduction

A lot of digital systems include control units (Baranov, 2008; Gajski *et al.*, 2009). As follows from the works of Czerwiński and Kania (2013) or Minns and Elliot (2008), different models of finite state machines (FSMs) are used very often for representing and designing control units. In many practical cases, the model of a Mealy FSM is used for these purposes (Sklyarov *et al.*, 2014; Micheli, 1994). That is why we choose the Mealy FSM model in this research.

It is very important to diminish the amount of hardware consumed by an FSM logic circuit (Gajski *et al.*, 2009; Czerwiński and Kania, 2013). Solution methods for this problem strongly depend on specific features of logic elements used for implementing the circuits (Czerwiński and Kania, 2013; Sklyarov *et al.*, 2014). In our article, we discuss a case when field programmable gate arrays (FPGAs) are used to implement Mealy FSM logic circuits. We chose FPGAs because they are very popular and are used very often for implementing FSM logic circuits (Maxfield, 2004; Grout, 2008).

It is enough to use only two components of FPGA fabric to implement any logic circuit. These components are logic elements (LEs) and a matrix of programmable interconnections (Altera, 2018; Xilinx, 2018). An LE includes a look-up table (LUT) element, a programmable flip-flop and multiplexers. The LUT is a memory block having S_L address inputs and a single output. The LUT can keep a truth table of an arbitrary Boolean function having up to S_L arguments. It is possible to bypass the flip-flop of an LE. Consequently, the output of the LE could be either combinational or registered.

The LUT has a rather small amount of inputs ($S_L \leq 6$) (Altera, 2018; Xilinx, 2018). This peculiarity leads to applying functional decomposition (Scholl, 2001; Kam *et al.*, 1997; Nowicka *et al.*, 1999) of Boolean functions having more than S_L arguments. The decomposition leads to multilevel circuits with complex interconnections. In turn, it leads to increasing the propagation time and power consumption of the circuit (Barkalov *et al.*, 2015). It is very important to decrease the power consumption for FSM circuits (Kubica and Kania, 2017) as well as for other digital systems (Sajewski, 2017).

To improve the characteristics of FSM circuits, it is necessary to reduce the number of arguments in Boolean functions representing an FSM logic circuit (Sklyarov *et al.*, 2014). As a rule, various methods of state assignment are used to solve this problem (Minns and Elliot, 2008; Kam *et al.*, 1997). JEDI (Lin and Newton, 1989) is one of the best among these methods. JEDI is used, for example, in CAD tools such as SIS (Sentowich *et al.*, 1992) and ABC (ABC System, 2018).

Also, a hardware reduction can be obtained due to a structural decomposition of the FSM circuit (Barkalov *et al.*, 2012). In this case, the designers use methods such as the replacement of logical conditions (Sklyarov

*Corresponding author

et al., 2014; Baranov, 1994), the encoding of collections of microoperatios (Sklyarov *et al.*, 2014; Baranov, 1994), the transformation of object codes (Barkalov and Barkalov, Jr., 2005). These methods are based on the representation of an FSM circuit as a multi-level circuit. Each level of the FSM circuit is represented by a system of additional functions. They are much simpler than the functions implemented by a single-level circuit. The composition of additional functions represents the system of functions of a single-level circuit

In this article, we propose a design method targeting a hardware reduction in LUT-based Mealy FSMs. The method is based on a three-level structure of an FSM circuit and an encoding of collections of output variables.

2. Background of Mealy FSMs

A Mealy FSM is defined as the sextuple $S = (X, Y, A, \delta, \lambda, a_1)$ (Baranov, 2008; Micheli, 1994), where $X = \{x_1, \dots, x_L\}$ is a finite set of inputs, $Y = \{y_1, \dots, y_N\}$ is a finite set of outputs, $A = \{a_1, \dots, a_M\}$ is a finite set of states, $\delta : A \times X \to A$ is the transition function, $\lambda : A \times X \to Y$ is the output function, $a_1 \in A$ is the initial state.

The sextuple S can be represented by a state transition table (STT) (Micheli, 1994). The STT includes the following columns: a_m is the current state; a_s is the state of the transition; X_h is a conjunction of some elements of the set X (or their complements) determining the transition from a_m into a_s; Y_h is the collection of outputs generated during the transition $\langle a_m, a_s \rangle$; h is the number of the transition.

Consider the STT of a Mealy FSM S_1 (Table 1). It has $H = 20$ rows. The following sets can be derived from Table 1: $A = \{a_1, \dots, a_{10}\}$, $X = \{x_1, \dots, x_5\}$, $Y = \{y_1, \dots, y_8\}$. This gives $M = 10$, $L = 5$ and $N = 8$.

When the set of states is constructed, the state assignment should be executed (Micheli, 1994; Baranov, 1994). During this step, each state $a_m \in A$ is represented by its code $K(a_m)$ having R bits. The variables $T_r \in T$ are used for state assignment, where T is a set of state variables. The method of one-hot state assignment is very popular in the FPGA-based design of FSMs (Garcia-Vargas *et al.*, 2007; Tiwari and Tomko, 2004). But if embedded memory blocks (EMB) are used, then a binary assignment is more preferable, when we have

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

A register (RG) is used to keep the state codes. It includes flip-flops with the mutual synchronization pulse *Clock* and mutual clearing pulse *Start*. As a rule, D flip-flops are used for implementing the RG (Baranov, 2008). To change the content of the RG, input memory functions $D_r \in \Phi$ are used, where $\Phi = \{D_1, \dots, D_R\}$.

To design an FSM logic circuit, it is necessary to construct a structure table (ST) of the Mealy FSM. It is the extension of the initial STT by the following three columns: $K(a_m)$ is the code of the current state; $K(a_s)$ is the code of the state of transition; Φ_h is a collection of input memory functions equal to 1 to load into RG the code $K(a_s)$. The ST forms a basis for deriving the functions

$$\Phi = \Phi(T, X), \tag{2}$$

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

They are used for implementing the FSM logic circuit.

Let us construct the ST for Mealy FSM S_1. We have $M = 10$, so that $R = 4$. Let us form an ST for the Mealy FSM represented by Table 1. Since $M = 10$, we see that $R = 4$. This yields the sets $T = \{T_1, \dots, T_4\}$ and $\Phi = \{D_1, \dots, D_4\}$. Use the trivial state assignment resulting in the following state codes: $K(a_1) = 0000, \dots, K(a_{10}) = 1001$. These codes are used in the structure table (Table 2).

An ST is used to derive functions (2) and (3). For example, observe the symbol D_1 in rows 14–16 of Table 2. This gives the equation $D_1 = A_6\bar{x_1}\bar{x_3} \vee A_7 \vee A_8x_5 = \bar{T_1}T_2\bar{T_3}T_4\bar{x_1}\bar{x_3} \vee \bar{T_1}T_2T_3\bar{T_4} \vee \bar{T_1}T_2T_3T_4x_5$.

Functions (2) and (3) depend on terms

$$F_h = A_mX_h \quad (h = \overline{1, H}). \tag{4}$$

In (4), the symbol A_m stands for the conjunction of state variables $T_r \in T$ corresponding to the code $K(a_m)$ from the row h of the ST.

Table 1. State transition table of Mealy FSM S_1.

a_m	a_s	X_h	Y_h	h
a_1	a_2	1	y_1y_2	1
a_2	a_3	x_1	y_3	2
	a_3	$\bar{x_1}$	y_2y_4	3
a_3	a_4	x_2	y_1y_2	4
	a_4	$x_1\bar{x_2}$	y_3y_5	5
	a_3	$\bar{x_1}\bar{x_2}$	y_2y_4	6
a_4	a_5	1	y_6y_7	7
a_5	a_6	x_3x_4	y_6y_7	8
	a_6	$x_3\bar{x_4}$	y_3	9
	a_7	$\bar{x_3}x_5$	y_2y_4	10
	a_7	$\bar{x_3}\bar{x_5}$	y_7	11
a_6	a_8	x_1	y_2y_5	12
	a_8	$\bar{x_1}x_3$	y_2y_8	13
	a_9	$\bar{x_1}\bar{x_3}$	y_3	14
a_7	a_9	1	y_3	15
a_8	a_{10}	x_5	y_2y_4	16
	a_1	$\bar{x_5}$	y_2y_8	17
a_9	a_1	1	–	18
a_{10}	a_4	x_6	y_1y_2	19
	a_1	$\bar{x_6}$	–	20

Functions (2) and (3) determine the trivial structural diagram of LUT-based Mealy FSM U_1 (Fig. 1). We use the symbol LUTer for circuits implemented with LUTs.

In U_1, the block LUTerΦ implements the system (2). If a function $D_r \in \Phi$ is generated as an output function of some LUT, then this output is connected with D flip-flops. These flip-flops form a distributed register (RG) keeping the state codes. Pulse $Start$ is used for zeroing the RG. The pulse $Clock$ allows changing the content of the RG. The block LUTerY implements the system (3).

Let us analyse the design methods targeting the hardware reduction in FPGA-based Mealy FSM circuits.

3. State of the art

Four basic optimization problems arise in the process of FSM design (Sklyarov *et al.*, 2014). They are the following: (i) a decrease in the chip area occupied by an FSM circuit (the problem of hardware reduction); (ii) a reduction in signal propagation time; (iii) a reduction in power consumption; (iv) an improvement in testability. In this article, we consider the first of these problems.

Table 2. Structure table of Mealy FSM S_1.

a_m	$K(a_m)$	a_s	$K(a_s)$	X_h	Y_h	Φ_h	h
a_1	0000	a_2	0001	1	y_1y_2	D_4	1
a_2	0001	a_3	0010	x_1	y_3	D_3	2
		a_3	0010	$\bar{x_1}$	y_2y_4	D_3	3
a_3	0010	a_4	0011	x_2	y_1y_2	D_3D_4	4
		a_4	0011	$x_1\bar{x_2}$	y_3y_5	D_3D_4	5
		a_3	0010	$\bar{x_1}\bar{x_2}$	y_2y_4	D_3	6
a_4	0011	a_5	0101	1	y_6y_7	D_2	7
a_5	0100	a_6	0101	x_3x_4	y_6y_7	D_2D_4	8
		a_6	0101	$x_3\bar{x_4}$	y_3	D_2D_4	9
		a_7	0110	$\bar{x_3}x_5$	y_2y_4	D_2D_3	10
		a_7	0110	$\bar{x_3}\bar{x_5}$	y_7	D_2D_3	11
a_6	0101	a_8	0111	x_1	y_2y_5	$D_2D_3D_4$	12
		a_8	0111	$\bar{x_1}x_3$	y_2y_8	$D_2D_3D_4$	13
		a_9	1000	$\bar{x_1}\bar{x_3}$	y_3	D_1	14
a_7	0110	a_9	1000	1	y_3	D_1	15
a_8	0111	a_{10}	0010	x_5	y_2y_4	D_1D_4	16
		a_1	0000	$\bar{x_5}$	y_2y_8	–	17
a_9	1000	a_1	0000	1	–	–	18
a_{10}	1001	a_4	0011	x_6	y_1y_2	D_3D_4	19
		a_1	0000	$\bar{x_6}$	–	–	20

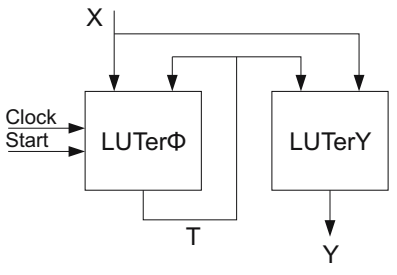

Fig. 1. Structural diagram of Mealy FSM U_1.

Functions (2) and (3) could depend on up to $R + L$ arguments. Our analysis of the library LGSynth93 (LGSynth93, 1993) shows that for some benchmarks we have $L + R > 15$. At the same time, we get $S_L \leq 6$ for modern LUTs (Altera, 2018; Xilinx, 2018). Therefore, the following condition could be met for FSM U_1:

$$L + R \gg S_L. \tag{5}$$

If (5) is fulfilled, the problem of hardware reduction arises for a particular FSM. There are four main groups of methods for solving this problem:

(a) the appropriate state assignment (Baranov, 2008; Micheli, 1994; Kam *et al.*, 1997);

(b) the functional decomposition of Boolean functions (2) and (3) representing an FSM circuit (Scholl, 2001; Nowicka *et al.*, 1999; Rawski *et al.*, 2005a; 2005b; Sasao, 2011);

(c) the replacement of LUTs by embedded memory blocks (Sklyarov *et al.*, 2014; Barkalov *et al.*, 2015; Sutter *et al.*, 2002; Cong and Yan, 2000; Sklyarov, 2000; Garcia-Vargas *et al.*, 2007; Tiwari and Tomko, 2004; Rawski *et al.*, 2011);

(d) the structural decomposition of the FSM circuit (Sklyarov *et al.*, 2014; Barkalov and Titarenko, 2009; Kołopieńczyk *et al.*, 2017).

Known methods of state assignment target obtaining state codes making it possible to diminish the number of arguments in functions (2) and (3). Modern FPGAs have a lot of flip-flops. Therefore the one-hot state assignment is very popular in FPGA-based design (Sklyarov *et al.*, 2014). In this case, we have $R = M$ and only a single variable $T_r \in T$ forms a conjunction $A_m(m = \overline{1, M})$. It allows reducing the number of arguments in terms (4). But results in (Sklyarov, 2000) show that the binary encoding of states produces better results than the one-hot if $M > 10$.

It seems to us that JEDI is the best among the known state assignment algorithms (Czerwiński and Kania, 2013). It is distributed with the system SIS (Sentowich *et al.*, 1992). JEDI targets a multi-level implemented FSM circuits. In the case of the input dominant version of JEDI, it maximizes the size of common cubes in functions (2) and (3). The output dominant version of JEDI maximizes the number of common cubes in these functions.

There are different strategies of state assignment used in standard industrial packages. For example, seven different methods are used in the design tool XST of Xilinx (Xilinx, 2018). Among them, there are one-hot,

compact, Gray, Johnson and other. It is really difficult to say which would be the best for a particular FSM.

In the case of functional decomposition (Scholl, 2001; Rawski *et al.*, 2005a; 2011; Sasao, 2011), an original function is broken down into smaller and smaller components. The process is terminated when each component depends on no more than S_L arguments. Three main approaches are used for the decomposition: serial, parallel and balanced. Each step of serial decomposition leads to an increase in the number of circuit levels. In turn, this results in a reduction in the maximum operating frequency of the FSM circuit. In the parallel decomposition, these characteristics are minimized. The balanced decomposition leads to a solution minimizing disadvantages and maximizing strong sides of the previous two approaches. This approach is used, for example, by the systems DEMAIN (DEMAIN, 2018) and PKmin (PKmin, 2018).

There are a lot of EMBs in modern FPGA chips (Sentowich *et al*., 1992). Using EMBs allows improving characteristics of FSM circuits (Sklyarov, 2000). A lot of EMB-based design methods can be found in the literature (Sklyarov *et al.*, 2014; Cong and Yan, 2000; Sklyarov, 2000; Garcia-Vargas *et al.*, 2007; Tiwari and Tomko, 2004; Rawski *et al.*, 2005a; 2011).

All these methods use the property of the configurability of EMBs (Nowicka *et al*., 1999). This property allows changing the numbers of cells and their outputs (Grout, 2008). Consequently, the modern EMBs are very flexible and could be tuned to meet the requirements of a particular FSM.

Let V_0 be the number of memory cells having only a single output. Assume that

$$2^{L+R}(R+N) \leq V_0. \tag{6}$$

In this case, only a single EMB is necessary to implement an FSM logic circuit. Our investigations (Kołopieńczyk *et al*., 2017) show that the condition (6) is met for 68% of benchmarks from the library LGSynth93 (LGSynth93, 1993).

If (6) is violated, then an FSM circuit could be implemented as: (i) a network of EMBs or (ii) a network of LUTs and EMBs. A survey of various approaches to EMB-based design is provided by Garcia-Vargas and Senhadji-Navarro (2015). But these methods could be used only if there are "free" EMBs, which are not used for implementing other parts of a digital system.

Our article is connected with structural decomposition of FSM circuits. In this case, an FSM circuit is represented by several blocks (Barkalov *et al*., 2015). Some blocks implement functions different from (2) or (3). We discuss the encoding of collections of output variables (COVs). Let us explain this approach.

Each row of ST includes a COV. The following COVs could be derived from Table 2:

$$\begin{aligned} &Y_1 = \emptyset, && Y_2 = \{y_1, y_2\},\\ &Y_3 = \{y_3\}, && Y_4 = \{y_2, y_4\},\\ &Y_5\{y_3, y_5\}, && Y_6 = \{y_6, y_7\},\\ &Y_7 = \{y_1, y_7\}, && Y_8 = \{y_7\},\\ &Y_9 = \{y_2, y_5\}, && Y_{10} = \{y_2, y_8\}. \end{aligned}$$

The COV Y_1 corresponds to the transition from a_{10} into a_1 when no output variables are generated. Therefore, there are $Q = 10$ different COVs in the case of S_1.

Encode each COV $Y_q \subseteq Y$ by a binary code $K(Y_q)$ having R_Q bits,

$$R_Q = \lceil \log_2 Q \rceil. \tag{7}$$

Use variables $z_r \in Z$ for encoding COVs, where $|Z| = R_Q$.

This approach leads to LUT-based Mealy FSM U_2 with a decomposed output block (Fig. 2). In this FSM, the LUTerΦ implements the system (2). The LUTerZ implements functions

$$Z = Z(T, X). \tag{8}$$

The LUTerY implements the functions

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

Let us compare FSMs U_1 and U_2. Two FSMs are called equivalent if they are designed using the same STT. Obviously, there are the same amounts of LUTs in the blocks LUTerΦ in equivalent FSMs U_1 and U_2. Assume that

$$N \gg Q. \tag{10}$$

In this case, the number of LUTs in LUTerZ is much lower than the numbers of LUTs in LUTerY of U_1. Assume that

$$R_Q \leq S_L. \tag{11}$$

In this case, only N LUTs are sufficient to implement the circuit of LUTerY of U_2.

Obviously, the method should be applied if the number of elements in the block LUTerY of U_1 significantly exceeds the total number of LUTs in the

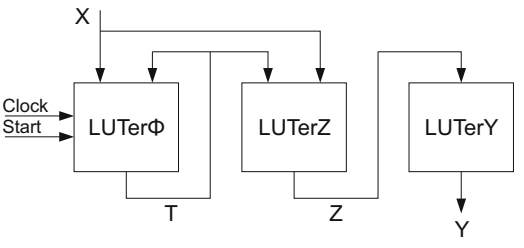

Fig. 2. Structural diagram of Mealy FSM U_2.

blocks LUTerZ and LUTerY of equivalent U_2. Our investigations of the library LGSynth93 (LGSynth93, 1993) show that circuits of FSMs U_2 always require fewer LUTs than the circuits of equivalent FSMs U_1. However, the circuits of U_2 have more structural levels than their counterparts U_1. This may lead to a decreased performance of FSMs U_2 compared with equivalent FSMs U_1.

An overview of various methods of structural decomposition in presented in the works of Sklyarov *et al*. (2014) and Barkalov *et al*. (2015). All the known methods are based on introduction of additional variables and reducing the number of functions depending on both state and input variables.

To design an FSM U_2, it is necessary to transform its initial ST. To do it, the column Y_h should be replaced by a column Z_h. The column Z_h includes variables $z_r \in Z$ equal to 1 in the code $K(Y_q)$ of COV $Y_q \subseteq Y$ from the h-th row of ST.

Using (7), we can find that $R_Q = 4$ for FSM $U_2(S_1)$. We use the symbol $U_i(S_j)$ to show that an FSM U_i is designed using an STT of the Mealy FSM S_j. Consequently, there is $Z = \{z_1, \dots, z_4\}$ in the discussed case. Let us encode COVs $Y_q \subseteq Y$ in the trivial way: $K(Y_1) = 0000, K(Y_2) = 0001$ and so on. The transformed ST of Mealy FSM $U_2(S_1)$ is represented by Table 3.

Let us explain how the column Z_h is obtained. For example, observe COV Y_1 in the first row of Table 2. It has code 0001. Thus is the variable z_4 is included in the first row of Table 3, and so on.

Table 3. Transformed ST of Mealy FSM $U_2(S_1)$.

a_m	$K(a_m)$	a_s	$K(a_s)$	X_h	Z_h	Φ_h	h
a_1	0000	a_2	0001	1	z_4	D_4	1
a_2	0001	a_3	0010	x_1	z_3	D_3	2
		a_3	0010	$\bar{x_1}$	z_3z_4	D_3	3
a_3	0010	a_4	0011	x_2	z_4	D_3D_4	4
		a_4	0011	$x_1\bar{x_2}$	z_2	D_3D_4	5
		a_3	0010	$\bar{x_1}\bar{x_2}$	z_3z_4	D_3	6
a_4	0011	a_5	0101	1	z_2z_4	D_2	7
a_5	0100	a_6	0101	x_3x_4	z_2z_4	D_2D_4	8
		a_6	0101	$x_3\bar{x_4}$	z_3	D_2D_4	9
		a_7	0110	$\bar{x_3}x_5$	z_3z_4	D_2D_3	10
		a_7	0110	$\bar{x_3}\bar{x_5}$	$z_2z_3z_4$	D_2D_3	11
a_6	0101	a_8	0111	x_1	z_1	$D_2D_3D_4$	12
		a_8	0111	$\bar{x_1}x_3$	z_1z_4	$D_2D_3D_4$	13
		a_9	1000	$\bar{x_1}\bar{x_3}$	z_3	D_1	14
a_7	0110	a_9	1000	1	z_3	D_1	15
a_8	0111	a_{10}	0010	x_5	z_3z_4	D_1D_4	16
		a_1	0000	$\bar{x_5}$	z_1z_4	–	17
a_9	1000	a_1	0000	1	–	–	18
a_{10}	1001	a_4	0011	x_6	z_4	D_3D_4	19
		a_1	0000	$\bar{x_6}$	–	–	20

In this article we propose a design method allowing us to reduce the number of LUTs in blocks LUTerΦ and LUTerZ of Mealy FSM U_2. The method is based on introducing new variables $\tau_r \in \mathcal{T}$. Let us discuss the proposed method.

4. Main idea of the proposed method

Let us find a partition $\Pi_A = \{A^1, \dots, A^I\}$ of the set A such that

$$R_i + L_i \le S_L \quad (i = \overline{1, I}). \tag{12}$$

In (12), the symbol R_i stands for the number of additional state variables $\tau_r \in \mathcal{T}$ necessary for encoding the states $a_m \in A^i$. The symbol L_i stands for the number of input variables $x_e \in X^i$ determining transitions from states $a_m \in A^i$.

Each state $a_m \in A^i$ has its code $C(a_m)$ having R_i bits

$$R_i = \lceil \log_2(|A^i| + 1) \rceil. \tag{13}$$

It is necessary to have a code showing that $a_m \notin A^i$. This necessity explains 1 in (13). Codes $C(a_m)$ are generated based on codes $K(a_s)$.

There are R_0 variables in the set $\mathcal{T}$, where

$$R_0 = R_1 + R_2 + \dots + R_I. \tag{14}$$

The first R_1 variables are used to encode the states $a_m \in A^1$, the next R_2 variables encode the states $a_m \in A^2$ and so on.

Each class $A^i \in \Pi_A$ determines a structure table ST_i with transitions from the states $a_m \in A^i$. The table ST_i could be constructed for both the initial and the transformed ST of the Mealy FSM. In the second case, it is possible to derive the sets X^i, Z^i and Φ^i from the table ST_i $(i = \overline{1, I})$.

The set $X^i \subseteq X$ includes input variables from the column X_h of ST_i. The set $Z^i \subseteq Z$ includes additional variables from the column Z_h of ST_i. The set $\Phi^i \in \Phi$ includes input memory functions from the column Φ_h of ST_i. Let us point out that current states $a_m \in A$ have codes $C(a_m)$, whereas the states of transitions $a_s \in A$ have codes $K(a_m)$.

Using this preliminary information, we propose the structural diagram of Mealy FSM U_3 (Fig. 3). It includes three levels of logic.

Each block LUTeri corresponds to the table ST_i $(i = \overline{1, I})$. The LUTeri generates the systems of functions:

$$Z^i = Z^i(\mathcal{T}^i, X^i), \tag{15}$$

$$\Phi^i = \Phi^i(\mathcal{T}^i, X^i). \tag{16}$$

In (15) and (16), the symbol $\mathcal{T}^i$ stands for the subset of $\mathcal{T}$ whose elements are used to encode the states $a_m \in A^i$.

The block LUTerTZ generates variables $z_r \in Z$ and $D_r \in \Phi$. Each LUT of this block executes function OR:

$$z_r = \bigvee_{i=1}^{I} z_r^i \quad (r = \overline{1, R_Q}), \tag{17}$$

$$D_r = \bigvee_{i=1}^{I} D_r^i \quad (r = \overline{1, R}). \tag{18}$$

In (17), the symbol z_r^i means that $z_r \in Z^i$. In (18), the symbol D_r^i means that $D_r \in \Phi^i$.

The block LUTer$\mathcal{T}$ implements the system

$$\mathcal{T} = \mathcal{T}(T). \tag{19}$$

This block transforms codes $K(a_s)$ into codes $C(a_s)$.

At each instant, only a single LUTeri is "active." This means that there are 1's at some of its outputs. At the same time, there are only 0's at the outputs of other blocks. These blocks are "idle." The following relation is used to show that a block LUTeri is idle:

$$\tau_r \in \mathcal{T}^i \rightarrow \tau_r = 0. \tag{20}$$

If (12) is true, then a single LUT is sufficient to implement any function $D_r \in \Phi^i$ and $z_r \in Z^i$. Assume that

$$I \leq S_L. \tag{21}$$

In this case, there are exactly $R + R_Q$ LUTs in the circuit of LUTerTZ.

Assume that

$$R \leq S_L. \tag{22}$$

In this case, there are only R_0 LUTs in the circuit of LUTer$\mathcal{T}$.

If conditions (11), (21) and (22) are fulfilled, then there are only $R + R_Q + N + R_0$ LUTs in the blocks LUTerTZ, LUTerY and LUTer$\mathcal{T}$ of FSM U_3.

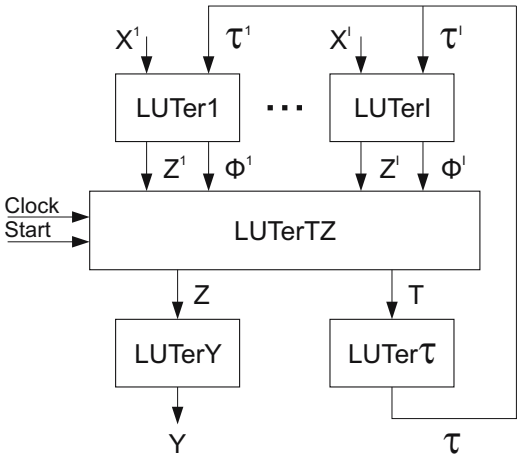

Fig. 3. Structural diagram of Mealy FSM U_3.

Accordingly, this is the best case for applying our approach. In this case, it is important to reduce the number of functions implemented by each block LUTeri. This is possible through finding an appropriate partition Π_A. In what follows, we discuss an approach to find this partition.

5. Construction of partition Π_A

The problem could be formulated as the following one. It is necessary to find a partition Π_A of the set A having a minimum number of blocks I and such that the condition (12) is met for each block.

In this article we propose a sequential algorithm to solve this problem. The algorithm minimizes appearance of the same input variables into different sets $X^i \subset X$. In the best case, the following relation takes place:

$$X^i \cap X^j = \emptyset \quad (i \neq j, i, j \in \{1, \ldots, I\}). \tag{23}$$

Each state $a_m \in A$ is characterized by two sets. The set $X(a_m)$ includes input variables determining transitions from state $a_m \in A$. The set $Z(a_m)$ includes variables $z_r \in Z$ equal to 1 in codes $K(Y_q)$ for COVs generated during transitions from the state $a_m \in A$. If $a_m \in A^i$, then $X(a_m) \subseteq X^i$ and $Z(a_m) \subseteq Z^i$.

We use two evaluations in the proposed algorithm. The first of them regards the difference between the number of shared input variables and the number of different input variables for class A^i and state $a_m \in A$:

$$N(a_m, X^i) = |X(a_m) \cap X^i| - |X(a_m) \setminus X^i|. \tag{24}$$

The second evaluation counts to the number of variables $z_r \in Z$ common for $Z(a_m)$ and Z^i:

$$N(a_m, Z^i) = |Z(a_m) \cap Z^i|. \tag{25}$$

There are two stages in generating each block. At the first stage, we choose a basic element (BE) for the block A^i. We take the state $a_m \in A^*$ as a BE, if the following condition takes place

$$|X(a_m)| = \max |X(a_j)|, \quad a_j \in A^* \setminus \{a_m\}. \tag{26}$$

In (26), the symbol A^* stands for the set of states which are not distributed after constructing the block $A^{i-1} \in \Pi_A$. If (26) is fulfilled for states a_m and a_s, choose the state with the lower value of the subscript.

The second stage is a multi-step one. During each step, the next state is successively added to the block A^i. The rules of inclusion are explained below. The process of forming the block is terminated if: (i) all states are already distributed among the blocks or (ii) it is not possible to include any state in A^i without violation of (12).

We use the following rule for including the next successive element in A^i. Let A^* include all unallocated states $a_m \in A$. Choose all states $a_m \in A^*$ whose

inclusion into A^i does not violate the restriction (12). Collect these states in the set $P(A^i)$. Select a state $a_m \in P(A^i)$ with the following property:

$$N(a_m, X^i) = \max N(a_j, X^i), \quad a_j \in P(A^i) \setminus \{a_m\}. \quad (27)$$

If there are more than one such state, then we should choose a state with the following property:

$$N(a_m, Z^i) = \max N(a_j, Z^i), \quad a_j \in P(A^i) \setminus \{a_m\}. \quad (28)$$

If the evaluations (28) are the same for several states from $P(A^i)$, then the state with the minimum value of the subscript is selected.

Let us find the partition Π_A for $U_3(S_1)$. The process is represented by Table 4. The partition is constructed for $S_L = 5$.

Let us explain the columns of Table 4. The column a_m contains states of the FSM. There are a number of input variables for states $a_m \in A$ in the column $|X(a_m)|$. There are basic elements for each stage shown in the columns BE_i $(i \in \{1, 2, 3\})$. The symbol "I" stands for (24), the symbol "II" for (25). The sign $\oplus$ means that the state from the corresponding row is included in the set A^i. The sign "–" means that $a_m \notin A^*$, where a_m is a state from the corresponding row. The row A^i includes states $a_m \in A^i$. The states are listed in the order of selection.

As follows from Table 4, there are $M = 10$ steps of selection. As a result, there is a partition $\Pi_A = \{A^1, A^2, A^3\}$ with $A^1 = \{a_4, a_5, a_8\}, A^2 = \{a_2, a_3, a_6\}$ and $A^3 = \{a_1, a_7, a_9, a_{10}\}$. Using Table 3, one could find the following sets: $Z^1 = \{z_1, \ldots, z_4\}, X^2 = \{x_1, x_2, x_3\}, Z^2 = \{z_1, \ldots, z_4\}, X^3 = \{x_6\}, Z^3 = \{z_3, z_4\}$. Now we can depict the block diagram of the FSM $U_3(S_1)$ (Fig. 4).

Using (13), we find that $R_1 = R_2 = 2$ and $R_3 = 3$. It gives $R_0 = 7$ and $\mathcal{T} = \{\tau_1, \ldots, \tau_7\}$. There are 8 LUTs in LUTer1, 8 LUTs in LUTer2 and 5 LUTs in LUTer3. Since $I = 3$, the condition (21) is satisfied. Thus, there are $R + R_0 = 8$ LUTs in LUTerTZ. There are $N = 8$ LUTs in LUTerY, because the condition (11) is satisfied. The condition (22) is met. Accordingly, there are $R_0 = 7$ LUTs in LUTer$\mathcal{T}$. Then, there are 44 LUTs with $S_L = 5$ in the circuit of Mealy FSM $U_3(S_1)$.

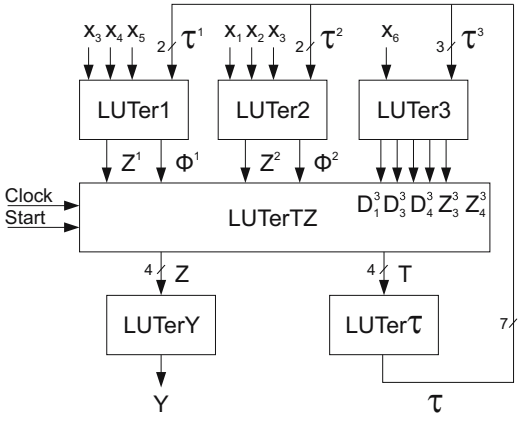

Fig. 4. Block diagram of Mealy FSM $U_3(\Gamma_1)$.

As follows from Fig. 4, only x_3 is shared between LUTer1 and LUTer2. Therefore, our approach allows obtaining circuits with more regular connections than, e.g., for FSM U_1. The same is true for pulses $Start$ and $Clock$, which are distributed only among the LUTs of LUTerTZ.

6. Proposed design method and an example of synthesis

In this article, we propose a design method for Mealy FSM U_3. It includes the following steps:

1. Finding set A from the state transition table.
2. Executing the state assignment.
3. Encoding collections of output variables.
4. Constructing the structure table of FM U_1.
5. Constructing the transformed structure table.
6. Constructing the partition Π_A.
7. Constructing tables ST_i for classes $A^i \in \Pi_A$.
8. Finding systems (15) and (16) for each class A^i.
9. Finding systems (17) and (18) for LUTerTZ.
10. Constructing tables for LUTerY and LUTer$\mathcal{T}$.
11. Implementing FSM circuit with particular LUTs.

Let us discuss an example of synthesis for Mealy FSM $U_3(S_1)$. We have already executed the first six design steps of this example. Table 2 represents the FSM structure table; Table 3 represents the transformed ST; partition Π_A follows from Table 4.

Use state variables $\tau_1, \tau_2 \in \mathcal{T}^1$ for encoding states $a_m \in A^1, \tau_3, \tau_4 \in \mathcal{T}^2$ for $a_m \in A^2$ and $\tau_5, \tau_6, \tau_7 \in \mathcal{T}^3$ for $a_m \in A^3$. There are state codes $C(a_m)$ shown in Table 5.

Tables ST_1–ST_3 are constructed using the transformed ST (Table 3). Each table ST_i includes only transitions from the states $a_m \in A^i$. But there is column $C(a_m)$ instead of $K(a_m)$. There are superscripts i for functions $z_r \in Z$ and $D_r \in \Phi$ in ST_i. This means that columns Z_h, Φ_h of ST are replaced by columns Z_h^i, Φ_h^i in tables ST_i $(i = \overline{1, I})$.

Table 4. Constructing partition Π_A for FSM $U_3(S_1)$.

a_m	$\lvert X(a_m)\rvert$	BE_1	I/II		BE_2	I/II		BE_3	I/II		
			1	2		1	2		1	2	3
a_1	0		0/1	0/1		0/1	0/1		0/0⊕	–	–
a_2	1		-1/2	-1/2		1/2⊕	–		–	–	–
a_3	2		-2/3	-2/3	⊕	–	–		–	–	–
a_4	1		0/2	0/2⊕		–	–		–	–	–
a_5	3	⊕	–	–		–	–		–	–	–
a_6	2		0/2	0/2		1/2	1/2⊕		–	–	–
a_7	0		0/1	0/1		0/1	0/1		0/0	0/0⊕	–
a_8	1		1/2⊕	–		–	–		–	–	–
a_9	0		0/0	0/0		0/0	0/0		0/0	0/0	0/0⊕
a_{10}	1		-1/0	-1/0		-1/0	-1/0	⊕	–	–	–
A^2		a_5	a_8	a_4	a_3	a_2	a_6	a_{10}	a_1	a_7	a_9

For example, Table 6 represents ST_1. It has $H_1 = 7$ rows. The following minimized Boolean functions could be derived from it:

$$z_4^1 = \bar{\tau_1}\tau_2 \vee \tau_1\bar{\tau_2}x_3x_4 \vee \tau_1\bar{\tau_2}\bar{x_3} \vee \tau_1\tau_2,$$

$$D_2^1 = \bar{\tau_1}\tau_2 \vee \tau_1\bar{\tau_2}.$$

Acting in the same manner, it is possible to construct other tables (ST_2 and ST_3) and functions. These functions are used to construct systems (17) and (18).

Each LUT of LUTerTZ executes the function OR. It clearly follows from equations (17) and (18). For example, $D_2 \notin \Phi^3$, and then $D_2 = D_2^1 \vee D_2^2$. Next, we have $z_1 \notin Z^3$. Therefore, $z_1 = z_1^1 \vee z_1^2$. Accordingly, it is a trivial thing to find equations for output functions of LUTerTZ.

Functions (9) depend on variables $z_r \in Z$. Therefore, the following columns are present in the table of LUTerY: $Y_q, K(Y_q), Y, q$. There are $Q = 10$ rows in table of LUTerY for the discussed example (Table 7).

This table could be viewed as N truth tables for output functions. Let us point out that all output functions are equal to zero for codes 1010–1111.

The table of LUTer$\mathcal{T}$ is constructed based on the table of state codes $C(a_m)$. It has columns $a_m, K(a_m), \mathcal{T}, m$. In the discussed case, there are $M = 10$ rows in this table (Table 8). Let us explain how to fill this table. For example, we have $C(a_4) = 01$ (Table 5). So, $\tau_1 = 0, \tau_2 = 1, \tau_3 = \tau_4 = 0\,(a_4 \notin A^2)$ and $\tau_5 = \tau_6 = \tau_7 = 0\,(a_4 \notin A^3)$. All other rows are filled in the same manner. The table of LUTer$\mathcal{T}$ corresponds to R_0 tables of LUTs.

To implement an FSM circuit, it is necessary to use standard CAD tools (Altera, 2018; Xilinx, 2018). They form bit-streams for each LUT based on the technology mapping of the FSM circuit (Maxfield, 2004; Grout, 2008). We do not discuss this step for our example.

Table 5. State codes $C(a_m)$ of Mealy FSM $U_3(S_1)$.

$a_m \in A^1$	$C(a_m)$	$a_m \in A^2$	$C(a_m)$	$a_m \in A^3$	$C(a_m)$
	$\tau_1\tau_2$		$\tau_3\tau_4$		$\tau_5\tau_6\tau_7$
a_4	01	a_2	01	a_1	001
a_5	10	a_3	10	a_7	010
a_8	11	a_6	11	a_9	011
–	–	–	–	a_{10}	100

Table 6. Structure table ST_1 of Mealy FSM $U_3(S_1)$.

a_m	$C(a_m)$	a_s	$K(a_s)$	X_h	Z_h^1	Φ_h^1	h
a_4	01	a_5	0100	1	$z_2^1z_4^1$	D_2^1	1
a_5	10	a_6	0101	x_3x_4	$z_2^1z_4^1$	$D_2^1D_4^1$	2
		a_6	0101	$x_3\bar{x_4}$	z_3^1	$D_2^1D_4^1$	3
		a_7	0110	$\bar{x_3}x_5$	$z_3^1z_4^1$	$D_2^1D_3^1$	4
		a_7	0110	$\bar{x_3}\bar{x_5}$	$z_2^1z_3^1z_4^1$	$D_2^1D_3^1$	5
a_8	11	a_{10}	1001	x_5	$z_3^1z_4^1$	$D_1^1D_4^1$	6
		a_1	0000	$\bar{x_5}$	$z_1^1z_4^1$	–	7

Table 7. Table of LUTerY for Mealy FSM $U_3(S_1)$.

Y_q	$K(Y_q)$	Y	q
	$z_1z_2z_3z_4$	$y_1y_2y_3y_4y_5y_6y_7y_8$	
y_1	0000	00000000	1
y_2	0001	11000000	2
y_3	0010	00100000	3
y_4	0011	01010000	4
y_5	0100	00101000	5
y_6	0101	00000110	6
y_7	0110	10000010	7
y_8	0111	00000010	8
y_9	1000	01001000	9
y_{10}	1001	01000001	10

7. Results

To investigate the efficiency of the proposed method, we use standard benchmarks from the LGSynth93 library. It includes 48 benchmarks related to the practice of FSM design. These benchmarks are presented in the KISS2 format.

We choose this set of benchmarks because it includes both simple ($M < 10$) and quite complex ($M > 50$) FSMs. Also, it is very often used to study the efficiency of various methods of FSM design (Czerwiński and Kania, 2013; Tiwari and Tomko, 2004).

To use these benchmarks, we applied the CAD tool named K2F. It translates the KISS2 file into VHDL model of an FSM. To synthesize and simulate the FSM, we use the Active-HDL environment. To get the FSM circuit, we use the Xilinx ISE package. Its version 14.1 was used for synthesis and implementation of the FSM for a given control algorithm.

We compared our approach with four other methods, namely: (i) Auto of ISE 14.1, (ii) Compact of ISE 14.1, (iii) JEDI, (iv) DEMAIN. The results of investigations are shown in Table 9. The system DEMAIN is used for synthesis of combinational circuits. We use this system to decompose the Boolean functions representing circuits of benchmarks FSMs.

For each method, we found two characteristics of benchmark FSMs. They are the number of LUTs in the FSM circuit (columns "LUTs") and the FSM maximum operating clock frequency (column "Freq.") measured in MHz.

The results of summation for both the numbers of LUTs and frequency in are included the row "Total." We have taken the summarized characteristics of U_3 as 100%. The row "Percentage" shows the percentage of summarized characteristics with respect to the benchmarks synthesized as U_3.

As can be seen in Table 9, the proposed method allows minimizing the number of LUTs in FSM circuits in comparison with other investigated methods. There are the following gains: (i) 23% in comparison with U_1Auto, (ii) 29% in comparison with $U_1Compact$, (iii) 9% in comparison with JEDI-based FSMs, and (iv) 13% in comparison with FSMs, designed by DEMAIN.

Table 8. Table of LUTer$\mathcal{T}$ for Mealy FSM $U_3(S_1)$.

a_m	$K(a_m)$	$\mathcal{T}$	m
	$T_1T_2T_3T_4$	$\tau_1\tau_2\tau_3\tau_4\tau_5\tau_6\tau_7$	
a_1	0000	0000001	1
a_2	0001	0001000	2
a_3	0010	0010000	3
a_4	0011	0100000	4
a_5	0100	1000000	5
a_6	0101	0011000	6
a_7	0110	1000010	7
a_8	0111	1100000	8
a_9	1000	0000011	9
a_{10}	1001	0000100	10

The following conclusion can be made. There are more LUTs in FSM circuits designed by ISE 14.1 in comparison with their counterparts designed using either JEDI or DEMAIN or K2F. If $M < 15$, then the best results are obtained using JEDI. Our approach yields better results for rather complex FSMs having more than 15 states. Sometimes, DEMAIN produces better results than JEDI (for rather simple automata).

To support this conclusion, Table 10 was included. It contain the results for 10 most complex benchmarks of the library LGSynth93 (LGSynth93, 1993). Our approach requires 19% fewer LUTs in comparison with JEDI and 25% fewer in comparison with DEMAIN. Thus, the gain practically doubled for complex benchmarks of LGSynth93 with respect to the average gain for all benchmarks.

As follows from Table 9, our approach produces FSMs which are a bit slower than the FSMs produced by U_1Auto (2%), JEDI (5%) and DEMAIN (5%). But this drawback is diminished for complex benchmarks (Table 10). For the complex benchmarks, our approach obtains the operating frequency only 4% lower than JEDI and practically the same as DEMAIN.

To show the benefits of usage of different methods, we use Table 9 to construct two line graphs shown in Figs. 5 and 6. We show the difference in the numbers of LUTs for different methods from Table 9 in Fig. 5. Figure 6 shows the difference of frequencies. For both the graphs, the number of states is shown in the x-axis. In both graphs, we used the methods with the minimum value of LUTs (Fig. 5) or the maximum value of frequency (Fig. 6) as a reference. As can be seen, the differences for benchmarks with the numbers of states up to about 15 are minor and the models are quite similar (Fig. 5). In this range, model U_3 is not the winner (the differences between U_3 and the smallest one are up to about 5 LUTs). Since then U_3 is the best model (at the bottom of the chart). For benchmarks with more than 15 states, U_3 has the lowest number of LUTs used.

Differences between models are quite significant and start from about 10 LUTs and end (owing to the lack of benchmarks with more states) at about 75 LUTs.

As can be seen in Fig. 6, the JEDI model is the fastest, the U_3 model is slower for about 50–150 MHz for benchmarks having less than 20 states and from 5 to 35 MHz slower for benchmarks having more than 20 states. One can observe an interesting property of the U_3 model for LGSynth benchmarks with over 20 states. The differences of maximal frequency are surprisingly small.

To sum up, according to Figs. 5 and 6, the U_3 model

Table 9. Results of investigations.

Benchmark	$U_1 Auto$		$U_1 Com$		JEDI		DEMAIN		U_3	
	LUTs	Freq.	LUTs	Freq.	LUTs	Freq.	LUTs	Freq.	LUTs	Freq.
bbara	11	639	13	635	10	690	9	702	12	650
bbsse	29	559	29	582	24	592	26	580	21	620
bbtas	5	962	5	966	5	980	5	978	8	860
bbcount	7	952	7	952	6	989	5	1022	9	920
cse	49	480	46	463	42	498	44	482	40	480
dk14	8	545	8	945	7	982	6	996	12	860
dk15	7	1062	7	1062	6	1090	7	1066	11	920
dk16	16	556	15	625	14	582	16	578	12	594
dk17	6	952	6	952	6	952	5	964	8	920
dk27	5	900	5	897	5	900	4	912	9	880
dk512	17	730	7	899	13	789	14	776	12	760
donfile	15	558	14	612	11	596	13	574	10	580
ex1	64	586	74	447	51	620	53	608	46	620
ex2	14	940	16	985	11	1002	12	988	12	980
ex3	12	980	13	986	12	982	11	998	14	960
ex4	15	962	16	626	12	1003	13	996	15	920
ex5	14	986	15	986	13	998	12	1003	16	890
ex6	29	553	20	621	26	579	24	599	28	580
ex7	14	988	15	990	12	1002	11	1060	14	992
keyb	56	384	65	358	48	410	50	398	42	420
kirkman	51	874	53	569	43	901	49	898	41	890
lion	3	1084	3	1080	3	1080	3	1080	6	920
lion9	6	980	5	996	5	996	5	998	7	910
mark1	27	726	19	708	22	798	24	749	26	744
mc	5	1071	5	1071	5	1071	5	1071	7	980
modulo12	26	612	28	632	19	710	22	678	24	640
opus	22	596	21	628	17	688	20	642	22	622
planet	100	888	138	389	88	989	92	921	64	940
planet1	100	888	138	389	88	989	92	921	64	940
pma	73	554	72	438	67	596	68	574	52	580
s1	77	550	75	447	70	598	76	582	61	570
s1488	140	425	141	432	131	470	136	452	101	460
s1494	124	412	143	442	112	492	118	478	93	472
s1a	77	550	75	447	70	598	76	586	64	580
s208	28	559	23	669	23	670	25	582	20	590
s386	26	577	28	581	24	598	22	621	24	590
s8	4	962	4	962	4	962	4	962	8	920
sand	99	569	121	426	89	612	91	598	81	602
shiftreg	3	1584	3	1584	3	1584	3	1584	6	1220
sse	29	559	28	543	24	610	26	588	21	562
styr	118	430	127	369	109	476	111	462	92	440
tav	6	1556	6	911	6	1560	6	1560	8	1402
tbk	55	406	71	465	48	492	49	484	36	441
tma	30	440	32	438	26	476	28	461	21	458
train11	28	560	26	580	25	598	27	572	28	568
train4	8	416	10	466	8	416	7	470	9	401
s27	4	962	4	962	4	962	4	962	6	890
s298	362	406	330	313	320	438	334	429	286	410
Total	2028	35870	2125	33526	1797	37686	1863	37245	1650	35151
Percentage	123%	102%	129%	95%	109%	107%	113%	105%	100%	100%

Table 10. Results of investigations for the most complex benchmarks.

Benchmark	M	JEDI		DEMAIN		U_3	
		LUTs	Freq.	LUTs	Freq.	LUTs	Freq.
s298	218	320	438	339	429	286	410
planet	48	88	989	92	921	64	940
planet1	48	88	989	92	921	64	940
s1488	48	131	470	136	452	101	460
s1494	48	112	492	118	478	93	472
sand	32	89	612	91	598	81	602
tbk	32	48	492	49	484	36	441
styr	30	109	976	111	462	92	440
dk16	27	14	582	16	578	12	594
donfile	24	11	596	13	574	10	580
Total		1010	6116	1062	5898	849	5879
Percentage		111%	104%	125%	100,3%	100%	100%

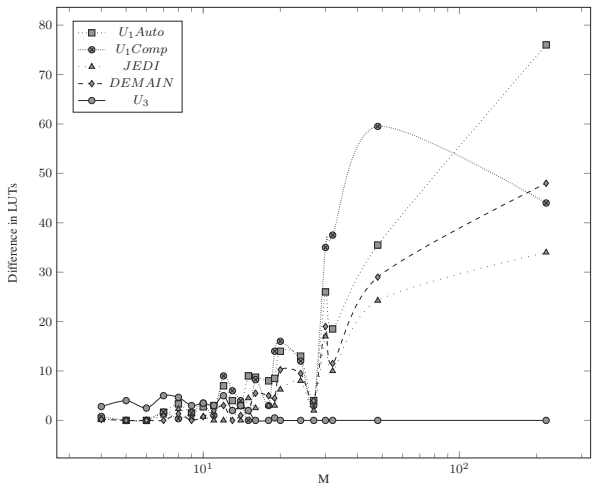

Fig. 5. Differences in the number of LUTs for benchmarks in comparison to minimum values.

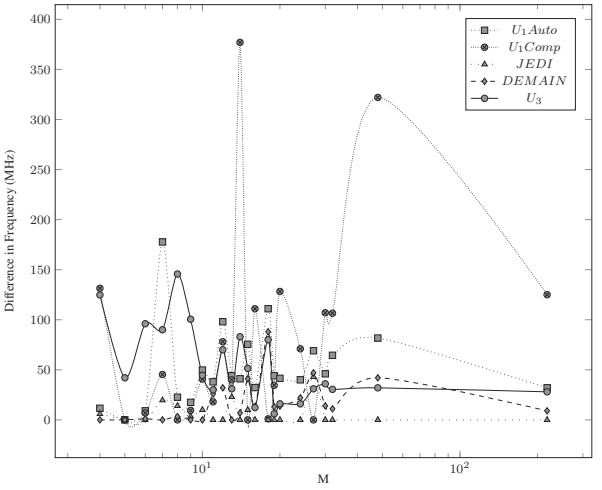

Fig. 6. Differences in frequencies for benchmarks in comparison with the maximum value.

is better for rather complex FSMs structures.

Let us point out that these conclusions are valid only for LGSynth93 benchmarks and the device XC5VLX30FF324 used for implementing FSM circuits. In the case of FPGA-based design, it is almost impossible to make some predictions for the common case. However, it is evident from out investigations that our approach could give better results for FSMs with $M > 15$.

8. Conclusion

The paper presents an original approach targeting LUT-based Mealy FSMs. The method is based on structural decomposition of the FSM circuit. As a result, the circuit has three levels of logic. The proposed method also uses an encoding of output variable collections.

The initial structure table of Mealy FSM is divided by sub-tables. To encode the states for each sub-table, use of additional state variables has been proposed. It leads to a reduction in the number of arguments in input memory functions in comparison with known design methods. As a result, a single LUT is sufficient to implement any function for any sub-table.

References

ABC System (2018). `https://people.eecs.berkeley.edu/~alanmi/abc/`.

Altera (2018). Cyclone IV Device Handbook, `http://www.altera.com/literature/hb/cyclone-iv/cyclone4-handbook.pdf`.

Baranov, S. (1994). *Logic Synthesis of Control Automata*, Kluwer, Boston, MA.

Baranov, S. (2008). *Logic and System Design of Digital Systems*, TUT Press, Tallinn.

Barkalov, A.A. and Barkalov, Jr., A.A. (2005). Design of Mealy finite-state machines with the transformation of object codes, *International Journal of Applied Mathematics and Computer Science* **15**(1): 151–158.

Barkalov, A. and Titarenko, L. (2009). *Logic Synthesis for FSM-based Control Units*, Springer, Berlin.

Barkalov, A., Titarenko, L. and Barkalov Jr., A. (2012). Structural decomposition as a tool for the optimization of an FPGA-based implementation of a Mealy FSM, *Cybernetics and Systems Analysis* **48**(2): 313–322.

Barkalov, A., Titarenko, L., Kołopieńczyk, M., Mielcarek, K. and Bazydło, G. (2015). *Logic Synthesis for FPGA-Based Finite State Machines*, Springer, Cham.

Cong, J. and Yan, K. (2000). Synthesis for FPGAs with embedded memory blocks, *Proceedings of the 2000 ACM/SIGDA 8th International Symposium on FPGAs, New York, NY, USA*, pp. 75–82.

Czerwiński, R. and Kania, D. (2013). *Finite State Machine Logic Synthesis for Complex Programmable Logic Devices*, Springer, Berlin.

DEMAIN (2018). `http://zpt2.tele.pw.edu.pl/Files/demain/demain.htm`.

Gajski, D.D., Abdi, S., Gerstlauer, A. and Schirner, G. (2009). *Embedded System Design: Modeling, Synthesis and Verification*, Springer, Berlin/Heidelberg.

Garcia-Vargas, I. and Senhadji-Navarro, R. (2015). Finite state machines with input multiplexing: A performance study, *IEEE Transactions a Computer-Aided Design of Integrated Circuits and Systems* **34**(5): 867–871.

Garcia-Vargas, I., Senhadji-Navarro, R., Jiménez-Moreno, G., Civit-Balcells, A. and Guerra-Gutierrez, P. (2007). ROM-based finite state machine implementation in low cost FPGAs, *Proceedings of the IEEE International Symposium on Industrial Electronics, ISIE 2007, Toronto, Canada*, pp. 2342–2347.

Grout, I. (2008). *Digital Systems Design with FPGAs and CPLDs*, Elsevier, Oxford.

Kam, T., Villa, T., Brayton, R. and Sangiovanni-Vincentelli, A. (1997). *A Synthesis of Finite State Machines: Functional Optimization*, Springer, Boston, MA.

Kołopieńczyk, M., Titarenko, L. and Barkalov, A. (2017). Design of EMB-based Moore FSMs, *Journal of Circuits, Systems, and Computers* **26**(7): 1–23.

Kubica, M. and Kania, D. (2017). Area-oriented technology mapping for LUT-based logic blocks, *International Journal of Applied Mathematics and Computer Science* **27**(1): 207–222, DOI: 10.1515/amcs-2017-0015.

LGSynth93 (1993). Benchmarks test, `http://people.engr.ncsu.edu/brglez/CBL/benchmarks/LGSynth93/LGSynth93.tar`.

Lin, B. and Newton, A. (1989). Synthesis of multiple level logic from symbolic high-level description languages, *Proceedings of the International Conference on VLSI, Taipei, Taiwan*, pp. 187–196.

Maxfield, C. (2004). *The Design Warrior's Guide to FPGAs*, Academic Press, Orlando, FL.

Micheli, G.D. (1994). *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, New York, NY.

Minns, P. and Elliot, I. (2008). *FSM-Based Digital Design Using Verilog HDL*, Wiley, Hoboken, NJ.

Nowicka, M., Łuba, T. and Rawski, M. (1999). FPGA-based decomposition of Boolean functions: Algorithms and implementation, *Proceedings of the 6th International Conference on Advanced Computer Systems, Szczecin, Poland,* pp. 502–509.

PKmin (2018). `http://pkmin.za.pl/`.

Rawski, M., Selvaraj, H. and Łuba, T. (2005a). An application of functional decomposition in ROM-based FSM implementation in FPGA devices, *Journal of System Architecture* **51**(6–7): 423–434.

Rawski, M., Selvaraj, H., Luba, T. and Szotkowski, P. (2005b). Application of symbolic functional decomposition concept in FSM implementation targeting FPGA devices, *Proceedings of the 6th International Conference on Computational Intelligence and Multimedia Applications (ICCIMA'05), Las Vegas, NV, USA*, pp. 153–158.

Rawski, M., Tomaszewicz, P., Borowski, G. and Łuba, T. (2011). Logic synthesis method of digital circuits designed for implementation with embedded memory blocks on FPGAs, *in* M. Adamski *et al.* (Eds.), *Design of Digital Systems and Devices*, Springer, Berlin, pp. 121–144.

Sajewski, Ł. (2017). Minimum energy control of descriptor fractional discrete-time linear systems with two different fractional orders, *International Journal of Applied and Computer Science* **27**(1): 33–41, DOI: 10.1515/amcs-2017-0003.

Sasao, T. (2011). *Memory-Based Logic Synthesis*, Springer, New York, NY.

Scholl, C. (2001). *Functional Decomposition with Application to FPGA Synthesis*, Kluwer, Boston, MA.

Sentowich, E., Singh, K., Lavango L., Moon, C., Murgai, R., Saldanha, A., Savoj, H., P, P.S., Bryton, R. and Sangiovanni-Vincentelli, A. (1992). SIS: A system for sequential circuit synthesis, *Technical report*, University of California, Berkeley, CA.

Sklyarov, V. (2000). Synthesis and implementation of RAM-based finite state machines in FPGAs, *Proceedings of the 10th International Conference on Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing, Villach, Austria*, pp. 718–728.

Sklyarov, V., Skliarova, I., Barkalov, A. and Titarenko, L. (2014). *Synthesis and Optimization of FPGA-Based Systems*, Springer, Berlin.

Sutter, G., Todorovich, E., López-Buedo, S. and Boemo, E. (2002). Low-power FSMs in FPGA: Encoding alternatives, *Proceedings of the 12th International Workshop on Power and Timing Modelling Optimization and Simulation, Sevilla, Spain*, pp. 363–370.

Tiwari, A. and Tomko, K. (2004). Saving power by mapping finite-state machines into embedded memory blocks in FPGAs, *Proceedings of the Conference on Design, Automation and Test in Europe, Paris, France*, pp. 916–921.

Xilinx (2018). Virtex-5 Family Overview, `http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf`.

Alexander A. Barkalov worked as a tutor at Donetsk National Technical University (DNTU) from 1976 to 1996. He cooperated actively with the Kiev Institute of Cybernetics (IC), named after Victor Glushkov. He obtained his DSc degree in informatics from the IC in 1995. From 1996 to 2003 he worked as a professor at DNTU. Since 2003 he has been working as a professor at the Faculty of Computer, Electrical and Control Engineering, University of Zielona Góra, Poland.

Larysa Titarenko obtained her DSc degree in telecommunications in 2005 from the Kharkov National University of Radioelectronics (KNURE). Up to 2003 she worked as a professor of the KNURE. Since 2005 she has been working as a professor at the Faculty of Computer, Electrical and Control Engineering of the University of Zielona Góra.

Kamil Mielcarek received his MSc in computer engineering from the Technical University of Zielona Góra in 1995 and his PhD in computer science from the University of Zielona Góra in 2010. Since 2001 he has been a lecturer there. His current interests include synthesis and optimization of control units in field-programmable logic devices, hardware description languages, perfect graphs and Petri nets, algorithmics, and safety of UNIX and network systems.

Received: 9 October 2017
Revised: 13 February 2018
Accepted: 28 April 2018