Synthesis of High-Speed Finite State Machines in FPGAs by State Splitting

Valery Salauyou

Synthesis of High-Speed Finite State Machines in FPGAs by State Splitting

Valery Salauyou

Faculty of Computer Science, Bialystok University of Technology, Bialystok, Poland

valsol@mail.ru

Abstract. A synthesis method of high-speed finite state machines (FSMs) in field programmable gate arrays (FPGAs) based on LUT (Look Up Table) by internal state splitting is offered. The method can be easily included in designing the flow of digital systems in FPGA. Estimations of the number of LUT levels are presented for an implementation of FSM transition functions in the case of sequential and parallel decomposition. Split algorithms of FSM internal states for the synthesis of high-speed FSMs are described. The experimental results showed a high efficiency of the offered method. FSM performance increases by 1.52 times on occasion. In conclusion, the experimental results were considered, and prospective directions for designing high-speed FSMs are specified.

Keywords: synthesis, finite state machine, high-speed, high performance, state splitting, field programmable gate array, look up table

1 Introduction

Large-size functional blocks and nodes of a digital system and also the digital system itself, as a rule, include a control device or a controller. The speed of a digital system and functional blocks depends directly on the speed of their control devices. The mathematical model for the majority of control devices and controllers is a finite state machine (FSM). Because of this, the synthesis methods of high-speed FSMs are necessary for designing high-performance digital systems. Note that an implementation cost can be ignored in the synthesis of high-speed FSMs, because an FSM area takes a small part compared with other system components (for example, memory or transceivers).

Now, programmable logic devices (PLDs) are widely used for designing digital systems. Two types of PLD architectures are widely used: on the basis of two programmed matrixes (AND and OR), and on the basis of functional generators, an LUT (Look Up Table). The first PLD type is called Complex Programmable Logic Devices (CPLDs), and the second PLD type is called Field Programmable Gate Arrays (FPGAs). It is possible to represent an FPGA structure as a great quantity of LUTs united by interconnections. Every LUT allows realizing any Boolean function from a

small number of arguments (as a rule, from 4 to 6). The methods of FSM synthesis on CPLD have been considered in [1].

Many authors considered the synthesis problem of high-speed FSMs on PLD. Their methods were characterized by a large variety of approaches to deciding on a given task. In [2], a technique for improving the performance of a synchronous circuit configured as an FPGA-based look-up table without changing the initial circuit configuration is presented. Only the register location is altered. This improves clock speed and data throughput at the expense of latency. In [3], the methods and tools for state encoding and combinational synthesis of sequential circuits based on new criteria of information flow optimization are considered. In [4], the timing optimization technique for a complex FSM that consists of not only random logic but also data operators is proposed. The technique, based on the concept of a catalyst**,** adds a functionally redundant block (which includes a piece of combinational logic and several other registers) to the circuits under consideration so that the timing critical paths are divided into stages. In [5, 6], the styles of FSMs description in VHDL language and known methods of state assignment for the implementation of FSMs are researched. In [7], evolutionary methods are applied to the synthesis of FSMs. At the first stage, the task of state assignment by means of genetic algorithms is resolved. Then evolutionary algorithms are applied to the minimization of chip area and time delay of FSM output signals. In [8], the task of state assignment and optimization of the combinational circuit at implementation of high-speed FSMs in CPLD is considered. In [9], a novel architecture that is specifically optimized for implementing reconfigurable FSMs, Transition-based Reconfigurable FSM (TR-FSM), is presented. The architecture shows a considerable reduction in area, delay, and power consumption compared to FPGA architectures. In [10], a new model of the automatic machine named the virtual finite state machine (Finite Virtual State Machine - FVSM) is offered. For implementation of the FVSM, architecture based on storage and a technique of FVSM generation from traditional FSMs is offered. FVSM implemented on new architecture have an advantage on high-speed performance compared with traditional implementation of FSMs on storage RAM. In [11], an implementation of FSMs in FPGA with the use of integral units of storage ROM is considered. Two pieces of FSMs architecture with multiplexers on inputs of ROM blocks which allow reducing the area and increasing high-speed FSM performance are offered. In [12], the reduction task of arguments of transition functions by state splitting is considered; this allows reducing an area and time delay in the implementation of FSMs on FPGA.

This paper also uses splitting of FSM states, but the purpose of splitting is an increase of FSMs performance in LUT-based FPGA. Splitting of FSM states belongs to operations of equivalent conversions of an FSM and does not change the algorithm of its functioning. During splitting of FSM states, the machine type (Mealy or Moore) is saved, the general structure of the FSM does not change, and embedded memory blocks of FPGAs are not used. In the course of state splitting, the hierarchy of state names is saved, which simplifies the analysis and debugging of the project. Because of this, the offered synthesis method of high-speed FSMs in FPGA is aimed at practical usage and can be easily included in the general flow of digital system design.

This paper is organized as follows. Section 2 describes estimations of the number of LUT levels in the implementation of FSM transition functions in the case of se-

quential and parallel decomposition. Section 3 considers the synthesis method of high-speed FSMs, which includes two algorithms: a general algorithm and an algorithm for the decomposition of the concrete state. A detailed example shows the method. The experimental results are reported in Section 4. The paper concludes with a summary in Section 5.

2 Estimations for the number of LUT levels for transition functions

Let $A = \{a_1, ..., a_M\}$ be the set of internal states, $X = \{x_1, ..., x_L\}$ be the set of input variables, $Y = \{y_1, ..., y_N\}$ the set of output variables, and $D = \{d_1, ..., d_R\}$ the set of transition functions of an FSM.

A one-hot state assignment is traditionally used for the synthesis of high-speed FSMs in FPGAs. Thus, each internal state a_i ($a_i \in A$) corresponds to a separate flip-flop of FSM's memory. A setting of this flip-flop in 1 signifies that the FSM is in the given state. The data input of each flip-flop is controlled by the transition function d_i, $d_i \in D$, i.e. any internal state a_i ($a_i \in A$) of the FSM corresponds with its own transition function d_i, $i = \overline{1,M}$.

Let $X(a_m,a_i)$ be the set of FSM input variables, whose values initiate the transition from state a_m to state a_i (a_m, $a_i \in A$). To implement some transition from state a_m to state a_i, it is necessary to check the value of the flip-flop output for the active state a_m (one bit) and the input variable values of the $X(a_m,a_i)$ set, which initiates the given transition. To implement the transition function d_i, it is necessary to check the values of the flip-flop outputs for all states, such that transitions from which lead to state a_i, i.e. $|B(a_i)|$ values, where $B(a_i)$ is the set of states from which transitions terminate in state a_i, where $|A|$ is the cardinality of set A. Besides, it is necessary to check the values of all input variables, which initiate transitions to state a_i, i.e. $|X(a_i)|$ values, where $X(a_i)$ is the set of input variables, whose values initiate transitions to state a_i, $X(a_i) = \bigcup_{a_m \in B(a_i)} X(a_m, a_i)$.

Let r_i be a rank of the transition function d_i, where

$$r_i = \left|B(a_i)\right| + \left|X(a_i)\right|. \quad (1)$$

Let n be the number of inputs of LUTs. If the rank r_i for transition function d_i ($i = \overline{1,M}$) exceeds n, there is a necessity to decompose the transition function d_i and its implementation on several LUTs.

Note that by splitting internal states it is impossible to lower the rank of the transition functions below the value

$$r^* = \max(|X(a_m, a_s)|) + 1, m, s = \overline{1,M}. \quad (2)$$

In this method, the value r^* is used as an upper boundary of the ranks of the transition functions in splitting the FSM states.

It is well-known that there are two basic approaches to the decomposition of Boolean functions: sequential and parallel. In the case of sequential decomposition, all the LUTs are sequentially connected in a chain (fig. 1).

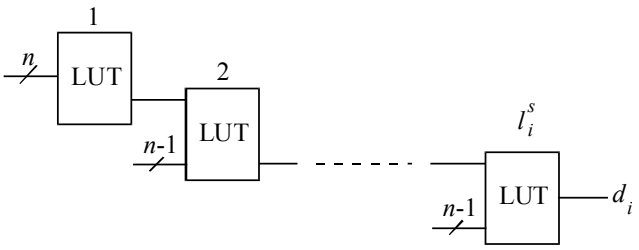

Fig. 1. Sequential decomposition of Boolean function

The n arguments of function d_i arrive on inputs of the first LUT, and the (n-1) arguments arrive on inputs of all remaining LUTs. So the number l_i^s of the LUT's levels (in the case a sequential decomposition of the transition function d_i having the rank r_i) is defined by the expression:

$$l_i^s = \text{int}\left(\frac{r_i - n}{n-1}\right) + 1, \tag{3}$$

where int(A) is the least integer number more or equal to A.

In the case of parallel decomposition, the LUTs incorporate in the form of a hierarchical tree structure (fig. 2).

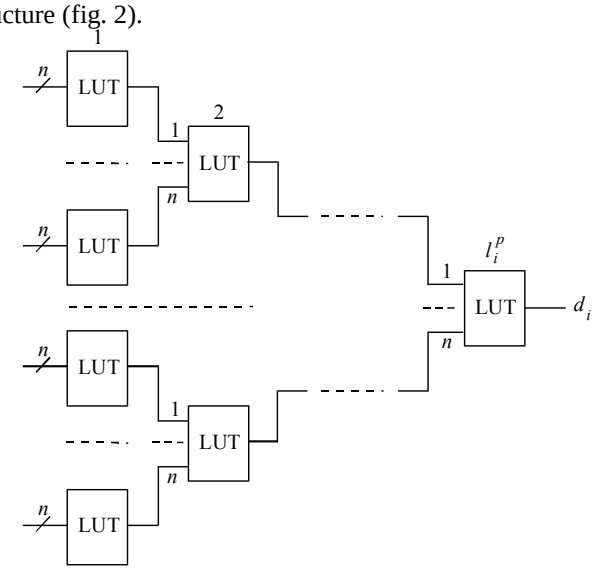

Fig. 2. Parallel decomposition of Boolean function

The values of the function arguments arrive on LUTs inputs of the first level, and the values of the intermediate functions arrive on LUTs inputs of all next levels. So the number of LUT's levels (in the case parallel decomposition the transition function d_i having the rank r_i) is defined by the following expression:

$$l_i^p = \text{int}(\log_n r_i). \quad (4)$$

It is difficult to predict what type of decomposition (sequential or parallel) is used by a concrete synthesizer. The preliminary research showed that, for example, the Quartus II design tool from Altera simultaneously uses both sequential and parallel decomposition. The number l_i levels of LUTs in the implementation on FPGA transition function d_i with the rank r_i can be between values l_i^s and l_i^p, $i=\overline{1,M}$.

Let k be an integer coefficient ($k \in [0,10]$) that allows adapting the offered algorithm in defining the number of LUT's levels for the specific synthesizer. In this case the number l_i of LUT's levels for the implementation of the transition function d_i having the rank r_i will be defined by following expression:

$$l_i = \text{int}\left(\frac{10-k}{10} l_i^p + \frac{k}{10} l_i^s\right). \quad (5)$$

The specific value of coefficient k depends on the architecture of the FPGA and the used synthesizer.

The following problem is the answer to the question: when is it necessary to stop splitting the FSM states? The matter is that in splitting state a_i ($i=\overline{1,M}$), except for the increase of the number M of the FSM states, the number of transitions in the states of set $A(a_i)$ is also increased, where $A(a_i)$ is the set of states in which the transitions from state a_i terminate. When splitting state a_i, the cardinalities of sets $B(a_m)$ ($a_m \in A(a_i)$) are increased for the states of set $A(a_i)$. Therefore, according to (1) for the states of set $A(a_i)$ the ranks of the transition functions grow, which can lead to an increase of the values and l_i^s, l_i^p, and l_i.

In this algorithm, the process of state splitting is finished, when the following condition is met:

$$l_{\max} \le \text{int}(l_{mid}), \quad (6)$$

where l_{max} is the number of LUT levels, which is necessary for the implementation of the most "bad" function having the maximum rank; l_{mid} is the arithmetic mean value of the number of LUT levels for all transition functions. Note that in the process of splitting the FSM internal states, the value l_{mid} will increase and the value l_{max} will decrease, therefore the algorithm execution always comes to an end.

3 Method for high-speed FSM synthesis

According to the above discussion, the algorithm of state splitting for high-speed FSM synthesis is described as follows.

Algorithm 1.

1. The coefficient k ($k \in [0,10]$) is determined, which reflects the method used by the synthesis tool for the decomposition of Boolean functions.
2. According to (1) ranks r_i ($i=\overline{1,M}$) for all FSM transition functions are defined.
3. On the basis of (3), (4), and (5), for each transition function d_i the number l_i of LUT levels is defined.
4. The values l_{max} and l_{mid} are determined. If condition (6) is met, then go to step 7, otherwise go to step 5.
5. The state a_i, for which r_i = max, is selected. If there are several such states, from them the state for which $|A(a_i)|$ = min is selected.
6. The state a_i (which was selected in step 5) is split by means of algorithm 2 on the minimum number H of states $a_{i\_1},\ldots,a_{i\_H}$ so that for each state a_{i_h} ($h=\overline{1,H}$) was fulfilled $r_{i\_h} \le r^*$, where r^* is defined according to (2); go to step 2.
7. End.

Further synthesis of the FSM is performed using traditional techniques, for example, automatically by means of using a design tool synthesizer. For this purpose, it is enough to describe the FSM received after splitting internal states in one of the design languages (Verilog or VHDL). The value of coefficient k (step 1 of algorithm 1) is defined empirically by means of synthesis of the test examples in the used design tool.

For splitting some a_i state, $i=\overline{1,M}$, which is executed in step 6 of algorithm 1, Boolean matrix W is constructed as follows. Let $C(a_i)$ be the set of transitions to state a_i. Rows of matrix W correspond to the elements of set $C(a_i)$. Columns of matrix W are divided on two parts according to types of arguments of transition function d_i. The first part of matrix W columns correspond to set $B(a_i)$ of FSM states, the transitions from which terminate in state a_i, and the second part of matrix W columns correspond to set $X(a_i)$ of input variables, whose values initiate the transitions in state a_i. A one is put at the intersection of row t ($t=\overline{1,T}$, $T = |C(a_i)|$) and column j of the first part of matrix W if the transition c_t ($c_t \in C(a_i)$) is executed from state a_j ($a_j \in B(a_i)$). A one is put at the intersection of row t and column j of the second part of matrix W if input variable x_j ($x_j \in X(a_i)$) accepts a significant value (0 or 1) on transition c_t ($c_t \in C(a_i)$). Now the task is reduced to a partition of matrix W on a minimum number H of row minors $W_1,\ldots,W_H$ so that the number of columns, which contain ones in each minor W_h ($h=\overline{1,H}$), do not exceed value r* defined according to (2). The rows of each minor W_h will define transitions in state a_{i_h} ($h=\overline{1,H}$).

Let w_t be some row of matrix W. For finding the row partition of matrix W on a minimum number H of row minors $W_1,\ldots,W_H$, the following algorithm can be used.

Algorithm 2.

1. Put $h := 0$.
2. Put $h := h + 1$. A formation of minor W_h begins. The row w_t, which has the maximum number of ones, is selected in minor W_h as a reference row. The row w_t is included in minor W_H and the row w_t is eliminated from further reviewing, put $W_h := \{w_t\}$, $W := W \backslash \{w_t\}$.
3. The rows are added in minor W_h. For this purpose, among rows of matrix W, the row w_t is selected, for which the next inequality is satisfied $|W_h \cup \{w_t\}| \le r^*$, where $|W_h \cup \{w_t\}|$ is the total number of ones in the columns of minor W_h and the row w_t after their joining on OR. If such rows can be selected from several among them, row w_t is selected, which has the maximum number of common ones with minor W_h, i.e. $|W_h \cap \{w_t\}| = \max$. The row w_t is included in minor W_h and row w_t is eliminated from further reviewing, put $W_h := W_h \cup \{w_t\}$, $W := W \backslash \{w_t\}$.
   Step 3 repeats until at least a single row can be included in minor W_h.
4. If in matrix W all the rows are distributed between the minors, then go to step 5, otherwise go to step 2.
5. End.

We show the operation of the offered synthesis method in the example. It is necessary to synthesize the high-speed FSM whose state diagram is shown in fig. 3.

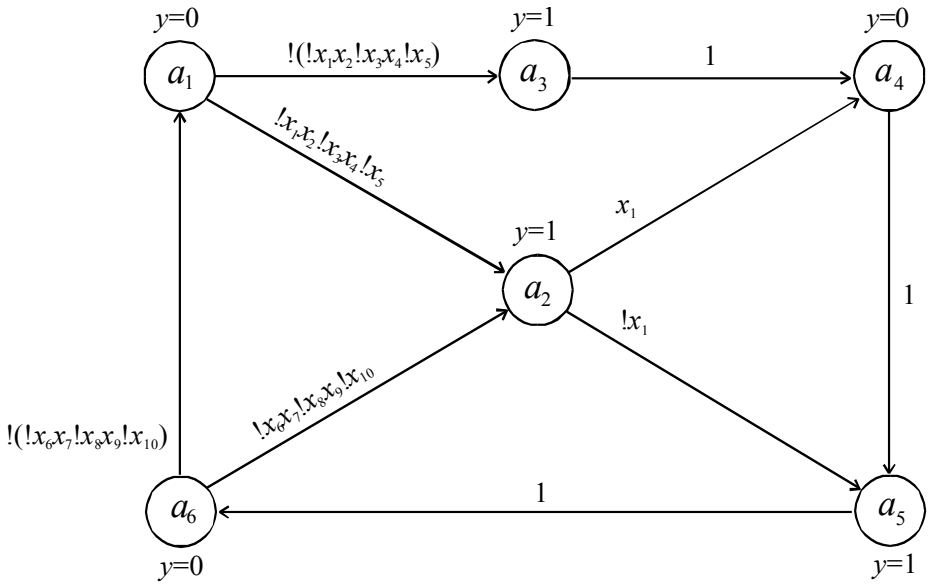

Fig. 3. State diagram of the initial FSM

This FSM represents the machine Moore, which has 6 states $a_1,\ldots,a_6$, 10 input variables $x_1,\ldots,x_{10}$, and one output variable y. The transitions from states a_3, a_4, and a_5 are unconditional, therefore the logical value 1 is written on these transitions as a transition condition. The values of sets $B(a_i)$ and $X(a_i)$, and also ranks r_i of the transition

functions for the initial FSM are presented in Table 1, where Ø is an empty set. Since for this example we have max(|$X(a_m,a_s)$|)=5, then (according to (2)) the value $r^* = 6$. It is necessary to construct the FSM on FPGA with 6-input LUT, i.e. we have n = 6.

Table 1. Values of $B(a_i)$, $X(a_i)$, r_i, l_i^s, and l_i^p for the initial FSM

State	$B(a_i)$	$X(a_i)$	r_i	l_i^s	l_i^p
a_1	$\{a_6\}$	$\{x_6,x_7,x_8,x_9,x_{10}\}$	6	1	1
a_2	$\{a_1,a_6\}$	$\{x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8,x_9,x_{10}\}$	12	3	2
a_3	$\{a_1\}$	$\{x_1,x_2,x_3,x_4,x_5\}$	6	1	1
a_4	$\{a_2,a_3\}$	$\{x_1\}$	3	1	1
a_5	$\{a_2,a_4\}$	$\{x_1\}$	3	1	1
a_6	$\{a_5\}$	Ø	1	1	1

According to (3) and (4), the values l_i^s and l_i^p are defined for each state (they are presented in the appropriate columns of Table 1). We do not know how the compiler performs a decomposition of Boolean functions, therefore we assume the sequential decomposition (a worst variant) and the value of coefficient k in expression (5) is equal to 10, i.e. we have $k = 10$. As a result, the number of LUT levels (which are necessary for the implementation of each transition function) is defined by the value $l_i = l_i^s$. Thus, for our example we have int(l_{mid}) = int(8/6) = 2. In other words, splitting FSM internal states stops as soon as each transition function can be implemented in two levels of LUTs.

For this example, we have $l_{max} = l_2^s = 3$, i.e. the condition (9) does not meet for state a_2, since $l_{max} = l_2^s = 3 > \text{int}(l_{mid}) = 2$. For this reason, state a_2 is split by means of algorithm 2. Matrix W is constructed for splitting state a_2 (fig. 4).

	a_1	a_6	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
w_1	1	0	1	1	1	1	1	0	0	0	0	0
w_2	0	1	0	0	0	0	0	1	1	1	1	1

Fig. 4. Matrix W for splitting state a_2

Matrix W has two rows. Row w_1 corresponds to the transition from state a_1 to state a_2, and row w_2 corresponds to the transition from state a_6 to state a_2. The execution of algorithm 2 leads to a partition of rows of matrix W into two subsets: $W_1 = \{w_1\}$ and $W_2 = \{w_2\}$. So, state a_2 is split into two states a_{2_1} and a_{2_2}, as shown in fig. 5.

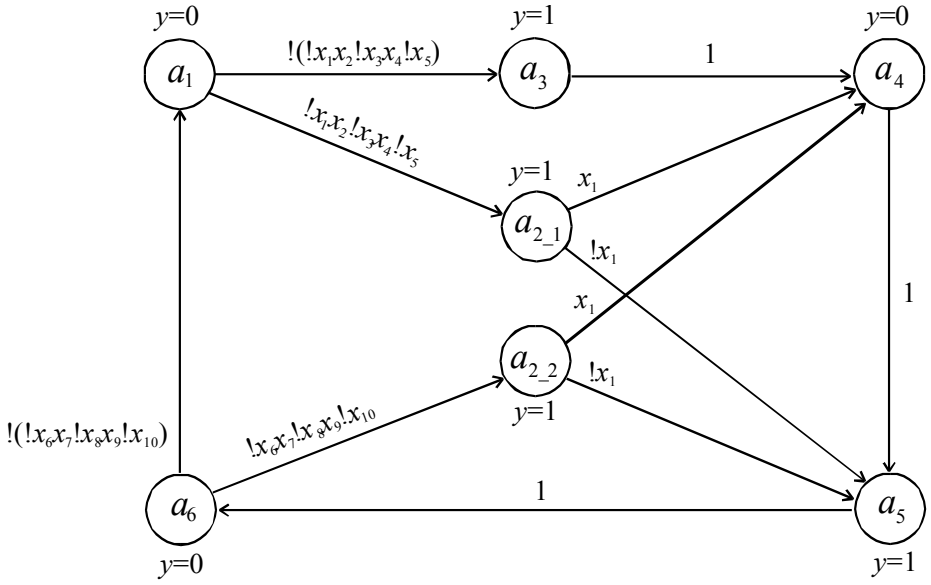

Fig. 5. State diagram of the FSM after splitting state a_2

The new values of $B(a_i)$, $X(a_i)$, r_i, l_i^s, and l_i^p are presented in Table 2. Now we have $l_{max} = l_{mid} = 1$ and (according to (6)) running of algorithm 1 is completed.

Table 2. Values of $B(a_i)$, $X(a_i)$, r_i, l_i^s, and l_i^p after splitting state a_2

State	$B(a_i)$	$X(a_i)$	r_i	l_i^s	l_i^p
a_1	$\{a_6\}$	$\{x_6,x_7,x_8,x_9,x_{10}\}$	6	1	1
a_{2_1}	$\{a_1\}$	$\{x_1,x_2,x_3,x_4,x_5\}$	6	1	1
a_{2_2}	$\{a_6\}$	$\{x_6,x_7,x_8,x_9,x_{10}\}$	6	1	1
a_3	$\{a_1\}$	$\{x_1,x_2,x_3,x_4,x_5\}$	6	1	1
a_4	$\{a_{2\_1},a_3\}$	$\{x_1\}$	3	1	1
a_5	$\{a_{2\_2},a_4\}$	$\{x_1\}$	3	1	1
a_6	$\{a_5\}$	$\emptyset$[1]	1	1	1

Thus, for the given FSM by splitting state a_2 we reduced the number of LUT levels from 3 to 1, in the case of sequential decomposition, and from 2 to 1, in the case of parallel decomposition.

4 Experimental Results

The efficiency of the offered synthesis method was checked in the implementation of the initial FSM (fig. 1) and the FSM after splitting state a_2 (fig. 2) on FPGAs from Altera by means of the design tool Quartus II version 15.0. The main optimization

criterion had been selected as the parameter «speed». The «one-hot» method of state assignment was selected for the initial FSM, and the «user» method of state assignment was selected for the FSM after synthesis (the state codes are defined from the FSM description).

Table 3 presents the results of the experimental research of the offered method for various FPGA families, where $nLUT_1$ and $nLUT_2$ are the number of LUTs used in the implementation of the initial and the synthesized FSM, respectively; F1 and F2 are the clock frequency (in MHz) for the initial and the synthesized FSM, respectively; F1/F2 is the relation of the appropriate parameters.

Table 3. Results of the experimental researches

Family	$nLUT_1$	F_1	$nLUT_2$	F_2	F_2/F_1
Arria II GX	8	1307	7	1269	0.97
Cyclone IV E	9	778	10	793	1.02
Cyclone IV GX	9	729	10	802	1.10
Cyclone V	6	686	6	925	1.35
MAX 10	10	800	11	816	1.02
MAX V	10	343	9	314	0.92
MAX II	10	389	9	593	1.52

Analysis of Table 3 shows that the application of the offered method increased the performance of the FSM for 5 FPGA families from 7. Thus, for the family MAX II performance was increased by 1.52 times, and for the family Cyclone V performance increased by 1.35 times. In addition, the number of used LUTs decreased for the following families: Arria II GX, MAX V, and MAX II.

Conclusions

The presented results of the experimental research showed the following. Despite the fact that in the considered example the rank of transition function was reduced from 12 to 6, which allowed to reduce the number of LUT levels from 3 to 1 in the case of sequential decomposition, and from 2 to 1 in the case of parallel decomposition; however, the performance of the FSM did not increase for all FPGA families. This is a sign of the complexity of the synthesis task of high-speed FSMs. FSM performance depends not only on the results of logical synthesis, but also on the results of placing and routing. The reduction of the number of used LUTs for some FPGA families (as a result of the application of the offered method) can be accounted simply: with the reduction of the number of LUT levels, the LUT amount also decreases.

The present study was supported by a grant S/WI/1/2013 from Bialystok University of Technology and founded from the resources for research by Ministry of Science and Higher Education.

References

1. Salauyou, V.V., Klimowicz, A.S.: Logic Design of Digital Systems on Programmable Logic Devices. Hot Line – Telecom, Moscow (2008) (in Russian)
2. Miyazaki, N., Nakada, H., Tsutsui, A., Yamada, K., Ohta, N.: Performance Improvement Technique for Synchronous Circuits Realized as LUT-Based FPGA's. IEEE Transactions on Very Large Scale Integration (VLSI) systems. 3, 455-459 (1995)
3. Jozwiak L, Slusarczyk A, Chojnacki A. Fast and compact sequential circuits through the information-driven circuit synthesis // In: Euromicro Symposium on Digital Systems Design, pp. 46-53. IEEE Press, Warsaw. Poland (2001)
4. Huang S.-Y. On speeding up extended finite state machines using catalyst circuitry // In: Asia and South Pacific Design Automation Conf. (ASAP-DAC), pp. 583-588 Yokohama. Jan (2001)
5. Kuusilinna, K., Lahtinen, V., Hamalainen, T., Saarinen, J.: Finite State Machine Encoding for VHDL Synthesis. Computers and Digital Techniques. IEE Proceedings. 1, 23-30 (2001)
6. Rafla N. I., Davis B. A Study of finite state machine coding styles for implementation in FPGAs. In: 49th IEEE International Midwest Symposium on Circuits and Systems. pp. 337-341. San Juan. USA (2006)
7. Nedjah N., Mourelle L. Evolutionary synthesis of synchronous finite state machines // In: Int. Conf. on Computer Engineering and Systems, pp.19-24. Cairo. Egypt. (2006)
8. Czerwiński, R., Kania, D.: Synthesis Method of High Speed Finite State Machines. Bulletin of the Polish Academy of Sciences: Technical Sciences. 4, 635–644. (2010)
9. Glaser, J., Damm, M., Haase, J., Grimm, C.: TR-FSM: Transition-based Reconfigurable Finite State Machine. ACM Transactions on Reconfigurable Technology and Systems (TRETS). 3, 23:1-23:14 (2011)
10. Senhadji-Navarro, R., Garcia-Vargas, I.: Finite Virtual State Machines. IEICE Transactions on information and systems. 10, 2544-2547 (2012)
11. Garcia-Vargas, I., Senhadji-Navarro, R.: Finite State Machines With Input Multiplexing: a Performance Study. IEEE Transaction on computer-aided design of integrated circuits and systems. 5, 867-871 (2015)
12. Solov'ev, V.V.: Splitting the Internal States in Order to Reduce the Number of Arguments in Functions of Finite Automata. J. of Computer and Systems Sciences International. 5, 777-783 (2005)