p-ISSN 1607-3274 Радіоелектроніка, інформатика, управління. 2025. № 4
e-ISSN 2313-688X Radio Electronics, Computer Science, Control. 2025. № 4

UDC 004.94 : 004.2

PSEUDO-RANDOM ENCODING OF STATES IN THE ALGORITHM FOR ALGEBRAIC SYNTHESIS OF A FINITE STATE MACHINE

Babakov R. M. – Dr. Sc., Associate Professor, Professor of the Information Technologies Department, Vasyl' Stus Donetsk National University, Vinnytsia, Ukraine.

Barkalov A. A. – Dr. Sc., Professor, Professor of the Institute of Computer Science and Electronics, University of Zielona Gora, Zielona Gora, Poland.

Titarenko L. A. – Dr. Sc., Professor, Professor of the Institute of Computer Science and Electronics, University of Zielona Gora, Zielona Gora, Poland.

ABSTRACT

Context. The problem of algebraic synthesis of finite state machine with datapath of transitions is considered. The circuit of this state machine may require less hardware expenses and have a lower cost compared to circuits of other classes of digital control units. The object of research is the process of finding complete and partial solutions of the problem of algebraic synthesis of finite state machine using specialized algorithms. One of such algorithms is the previously known algorithm of complete sequential enumeration of state coding variants with a fixed set of transition operations. In the vast majority of cases, complete sequential enumeration is performed too long, which makes its practical application in the process of synthesizing of finite state machines with operational transformation of state codes impossible. This paper proposes a new approach, which consists in replacing complete sequential enumeration of state coding variants with pseudo-random coding. This allows you to increase the number of state codes that change in each iteration of the algorithm and can contribute to a faster search for satisfactory solutions to the algebraic synthesis problem.

Objective. Development and research of an algorithm for finding solutions to the algebraic synthesis problem of a finite state machine with datapath of transitions based on pseudo-random selection of state codes.

Method. The research is based on the structure of finite state machine with datapath of transitions. The synthesis of the finite state machine circuit involves a mandatory stage of algebraic synthesis, the result of which is the combination of a certain way of states encoding with the assignment of arithmetic-logical operations to state machine transitions. Such combination is called the solution to the algebraic synthesis problem. In the general case, there are many solutions for a given finite state machine, each of which can be either complete (when operations are mapped to all transitions) or partial (when part of transitions cannot be implemented using any of the given operations). The more transitions are implemented by given operations, the less hardware expenses will be required to implement the state machine circuit and the better solution found. The search for the best solution requires consideration of a large number of possible variants of states encoding. The paper includes a modification of a previously known algorithm, which consists in replacing the complete sequential enumeration of variants of states encoding with pseudo-random code generation. Both algorithms were implemented in the form of software using the Python language and tested on the example of a finite state machine that implements an abstract control algorithm. In the course of the experiments, it was investigated which of the algorithms would find the best solution to algebraic synthesis problem in a fixed time. The experiments were repeated for different sets of transition operations. The purpose of the experiments was to evaluate which of state code assignment strategies is more effective: sequential enumeration of state codes or their pseudo-random generation.

Results. Using the example of an abstract control algorithm, it is demonstrated that in general, pseudo-random assignment of state codes allows finding better solutions to the algebraic synthesis problem in the same time than sequential enumeration of state codes. Factors such as computer speed or the method of pseudo-random generation of state codes do not have a significant impact on the results of the experiments. The advantage of pseudo-random generation of state codes is preserved when using different sets of transition operations.

Conclusions. The basis of the algebraic synthesis of finite state machine with datapath of transitions is an algorithm for finding solutions to the algebraic synthesis problem. The article proposes an algorithm for finding such solutions based on pseudo-random encoding of finite state machine states. The software implementation of this algorithm has proven that such approach is generally better than sequential enumeration for state encoding variants, since it allows finding better solutions (solutions with fewer operationally unimplemented transitions) in the same time. The pseudo-random assignment of state codes can be the basis of future algorithms for the algebraic synthesis of finite state machines.

KEYWORDS: finite state machine, datapath of transitions, arithmetic and logical operations, algebraic synthesis algorithm, sequential enumeration, pseudo-random generation of state codes.

DOI 10.15588/1607-3274-2025-4-3

ABBREVIATIONS

FSM is a finite state machine;
CU is a control unit
DT is a datapath of transitions;
GSA is a graph-scheme of algorithm;
TO is a transition operation.

NOMENCLATURE

A is a sets of FSM states;
X is a sets of logical conditions;
Y is a sets of microoperations;
M is a number of FSM states;
L is a number of logical conditions analyzed by FSM;
P is a number of microoperations formed by FSM;
R is a bit capacity of state code;
O is a set of transitions operations;
O_i is an element of set O;
I is a number of TO;
B is a number of uncovered FSM transitions;
$B_{\min}$ is a minimal number of uncovered transitions;
a^t is a current state of FSM;
$K(a^t)$ is a code of current state;
a^{t+1} is a state if transition;
$K(a^{t+1})$ is a code of state of transition;
G is a graph-scheme of implemented control algorithm.

INTRODUCTION

Any digital system includes a CU, which coordinates the joint operation of all system blocks [1–3]. Structurally, CU can be implemented in the form of a FSM, where a given control algorithm is implemented in a circuit way [4–6]. The FSM circuit allows for FSM transitions that depend on several input signals (the so-called multidirectional transitions) to be performed in one cycle of operation, while other types of CUs spend several cycles on this. Due to this, FSM has the highest speed among all types of CUs and is used in critical application systems, where speed is a decisive factor. At the same time, the FSM circuit is characterized by the highest hardware expenses compared to other types of CUs. This makes the scientific problem of reducing hardware expenses in the FSM circuit relevant [4, 7]. Reducing hardware expenses allows improving such characteristics of the circuit as cost, dimensions, reliability, etc. [8–10].

In [11], an approach to constructing an FSM circuit is proposed, which involves the transformation of state codes during FSM transitions using a certain set of arithmetic-logical operations. This approach, known as operational transformation of state codes, leads to FSM structure with datapath of transitions, in which some of transitions are implemented using a set of operations, and some are implemented in a canonical way using a system of Boolean equations [12, 13]. Each arithmetic-logical operation (called in this structure as transition operation) is implemented in the datapath by a separate combinational circuit with fixed hardware expenses. The structure of FSM with DT allows you to use any TO to implement any number of FSM transitions (provided that the states are appropriately coded). This allows not to increase (or to increase to a lesser extent) the hardware expenses while increasing the number of transitions implemented by given TOs compared to other FSM structures.

One of stages of synthesis of FSM with DT is so-called algebraic synthesis, which involves a special coding of states, which is coordinated in a certain way with formation of set of transition operations [12]. The set of state codes together with a given set of OPs is called a formal solution to the algebraic synthesis problem. For a given FSM with a fixed set of OPs, there are as many formal solutions as there are different variants of states encoding. If we consider a separate solution, we can determine that part of FSM transitions can be implemented using a given set of OPs (is covered by the given OPs), and the other part cannot be implemented by any of the given OPs (is not covered by the OPs) and must be implemented in a canonical way using a system of Boolean equations. Among the formal solutions found, the best is the one that covers a larger number of FSM transitions. Formal solutions to the algebraic synthesis problem, in which all transitions are covered, are called as complete solutions, otherwise they are called as partial solutions [12, 13].

Currently, there are no known algorithms, rules, or approaches that allow algebraic synthesis to be performed in an optimal way in a short time. In [14], an algorithm for algebraic synthesis is proposed, based on a complete sequential enumeration for state coding options for a fixed set of transition operations. The application of this algorithm to FSM of even low complexity requires a lot of time, which does not allow finding the best possible formal solution. Therefore, the algorithm provides for a time limit in its operation, during which it searches for a formal solution with a minimal number of uncovered FSM transitions. The result of the algorithm can be either a complete or a partial solution to the algebraic synthesis problem, but only within the range of solutions searched for a certain time. Usually, the longer the complete enumeration algorithm runs, the better solutions to the algebraic synthesis problem will be found.

This article is devoted to the development and research of the algorithm for finding formal solutions to the problem of algebraic synthesis that uses other method of states encoding than sequential enumeration of variants.

The object of the study is the process of finding complete and partial solutions to the problem of algebraic synthesis of an FSM using specialized state coding algorithms.

The subject of the study is algorithms for finding complete and partial solutions to the problem of algebraic synthesis of FSM with DT based on a complete sequential enumeration or pseudo-random generation of FSM state codes.

DOI 10.15588/1607-3274-2025-4-3

The purpose of the work is to develop and study an algorithm for finding solutions to the problem of algebraic synthesis of finite state machine with datapath of transitions based on a pseudo-random generation of state codes.

1 PROBLEM STATEMENT

Let a finite state machine with datapath of transitions be given in the form of GSA [10, 15, 16]. According to GSA, a set of states $A=\{a_0,...,a_{M-1}\}$, a set of input signals $X=\{x_0,...,x_L\}$ and a set of microoperations $Y=\{y_1,...,y_P\}$ are formed. The set of transition operations $O=\{O_1,...,O_I\}$ is also given. Each element of $O_i \in O$ represents a certain arithmetic-logical operation, which implies the appropriate interpretation of the binary codes of the machine states. The GSA and the set O are the input data for the algebraic synthesis of the FSM with DT.

This article solves the problem of developing and studying algorithm for the algebraic synthesis of FSM with DT according to a given GSA, which uses pseudo-random generation of state codes.

2 REVIEW OF THE LITERATURE

The known methods of optimizing the logic circuit of a finite state machine usually lead to changes in its structure [3, 6, 10]. In this article, we consider the structure of an FSM with DT [11]. In it, the transition function is realized by datapath of transitions, which implements arithmetic and logic operations to convert state codes. The synthesis of FSM with DT based on a given GSA is considered in [12].

In [14], the algorithm for finding formal solutions to the problem of algebraic synthesis of FSM with DT according to a given GSA was proposed. In this paper, this algorithm will be denoted by A1. It assumes a fixed set of transition operations and uses a sequential enumeration of all possible state coding variants. For each considered variant of state coding, the algorithm determines the number of uncovered transitions. This number allows the algorithm to choose the coding variant in which the number of uncovered transitions is minimal among all considered variants.

Let a given GSA be denoted by M states of a Mealy or Moore finite state machine, for which $R=\lceil \log_2 M \rceil$ binary bits are sufficient to encode [3, 10]. In total, there are 2^R various codes that are usually interpreted in FSM with DT as binary vectors (in logical operations) or as scalar numbers without a sign (in arithmetic operations). The number of variants for encoding M states by 2^R codes is determined by the combinatorial formula for the number of permutations in accordance with expression (1).

$$N=\frac{(2^R)!}{(2^R-M)!}. \quad (1)$$

As an example, consider the GSA G shown in Fig. 1. The GSA is abstract and is denoted by $M=18$ states a_0-a_{17} of the Moore state machine. Since the operational transformation of state codes concerns only the transition function of the FSM and is not related to the output function, the GSA in Fig. 1 does not contain microoperations (output signals). Instead, the operational nodes indicate the corresponding states of the Moore state machine. The initial and final nodes are marked with a same state a_0, which indicates the cyclicity of the control algorithm [3]. To encode $M=18$ states, $R=5$ binary digits are sufficient. Five binary digits allow for 32 different codes that can be used to encode the states.

According to (1), when $M=18$, $R=5$, the number N of state coding variants is approximately equal to 3×10^{24}. The authors have implemented algorithm A1 in Python, which on i5–13500 processor allows analyzing about 200,000 variants in one second. Under such conditions, it would take 50×10^{10} years to go through all 3×10^{24} variants sequentially. Even a theoretical increase in the program's performance by a million times would require 500,000 years for a complete search, which is unpleasant for practical use. It should be understood that the GCA G given as an example is relatively small, while the graph-schemes of real control algorithms can have a number of states several times or dozens of times larger.

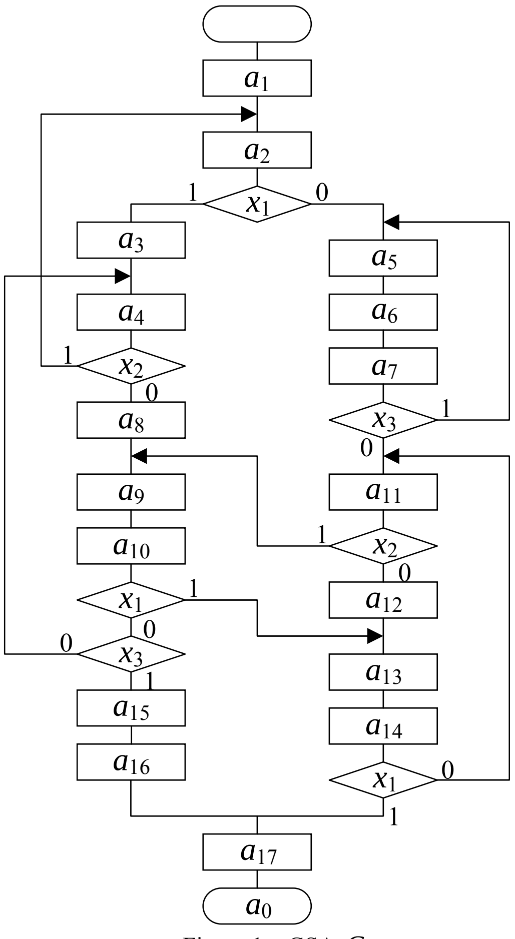


Figure 1 – GSA G

It should also be noted that the complete enumeration is performed for a given set of transition operations. Even if we do manage to perform complete enumeration of state encoding options, there is no guarantee that a complete solution to the algebraic synthesis problem will be found (i.e., a solution with all transitions covered). Indeed, the best way to encode states will be found, but it may turn out to be a partial solution with a certain number of uncovered transitions, and the resulting state machine circuit may not have a gain in hardware expenses.

In this case, we will be forced to choose another set of transition operations and re-execute complete enumeration for state coding variants. We will repeat this process until we select a set of TOs such that the found solution to the algebraic synthesis problem will allow us to obtain a state machine circuit with satisfactory characteristics.

Thus, it is obvious that it is impossible to use complete enumeration of state coding variants in practice. In this aspect, the A1 algorithm proposed in [14] provides for a time limit on execution. At the beginning of the work, a certain time t_A is set, upon reaching which the algorithm stops working and produces the best solution to the algebraic synthesis problem among the found solutions. This can be either a complete solution or a partial solution with a minimum number of uncovered transitions among all solutions considered during time t_A. It is quite possible that the best solution found in a limited time will allow you to design a state machine circuit with better (lower) hardware expenses. Thus, limiting the algorithm to the running time is currently the only way to obtain practically useful results.

3 MATERIALS AND METHODS

During complete enumeration of coding variants, the codes of some states change more often, the codes of other states change less often. For example, in the case of GSA *G*, states $a_0 - a_{17}$ initially acquire codes equal to their indices:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17

Next, a certain algorithm for permutation (rearrangement) of state codes works. In the author's software implementation, the *permutations* function from the *itertools* module of the Python language is used to permute the codes. Here are the first few generated sets of state codes:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 19
...
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 28
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 29
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 30
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 31
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 16
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 18
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 19
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 20
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 21
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 22
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 23
...

On the sets shown, the code of the last state a_{17} is continuously changing and the code of the penultimate state a_{16} is changing once. The other codes remain unchanged. If we take any relatively short period of time (a minute or an hour), we can assume that the state codes hardly change during this time (except for the codes of the last few states). This is especially true in the case of large-sized GSAs (with tens or hundreds of states). The algorithm repeatedly processes almost the same set of codes for some period of time, repeatedly performing almost the same work. As a result, during this time, the algorithm will produce mostly similar results. It's good if the algorithm immediately finds a fairly "efficient" solution (with zero or few uncovered transitions). But if the first solutions found have a large number of uncovered transitions, you should expect that in the near future the algorithm will find solutions with approximately the same number of uncovered transitions.

From the above, we can conclude that the algorithm of complete sequential enumeration of state coding variants is generally inefficient. Even if we modify the process of code permutation so that the codes of the first rather than the last states are changed most often, this will not lead to any fundamental changes in the results, since within any small period of time, the vast majority of state codes will still remain unchanged.

The authors put forward the following hypothesis: an increase in the number of states that change their codes during each iteration of the process of enumeration for state coding variants contributes to finding better solutions (solutions with fewer uncovered transitions) for a fixed algorithm running time. To test the hypothesis, this article proposes a modification of algorithm A1, in which instead of a complete sequential enumeration for state coding variants, a pseudo-random selection of them is used. The modified algorithm is shown in Fig. 2 and is denoted by A2 in this article. Let us consider its features.

1. Similar to algorithm A1, algorithm A2 runs continuously for the allotted time t_A, after which it terminates and returns a formal solution (a state encoding variant and a set of transition operations) that has the smallest number of uncovered transitions among all solutions viewed during the runtime.

2. The algorithm can terminate ahead of schedule if a complete solution to the algebraic synthesis problem is found. Here, it is conventionally assumed that all complete solutions (if there are several for a given GSA and a set of TOs) lead to the same hardware expenses for implementing the FSM circuit. Thus, finding the first complete solution makes it unnecessary to search for other complete solutions.

DOI 10.15588/1607-3274-2025-4-3

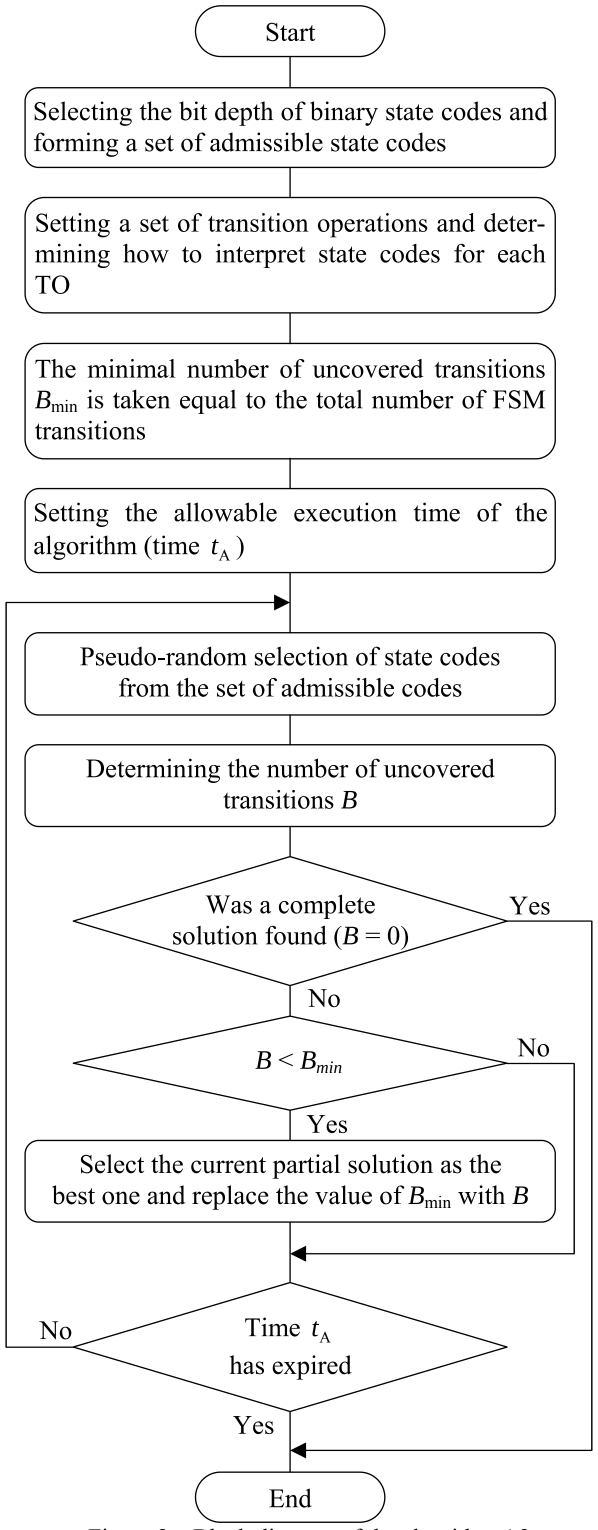

Start
Selecting the bit depth of binary state codes and forming a set of admissible state codes
Setting a set of transition operations and determining how to interpret state codes for each TO
The minimal number of uncovered transitions B_{min} is taken equal to the total number of FSM transitions
Setting the allowable execution time of the algorithm (time t_A)
Pseudo-random selection of state codes from the set of admissible codes
Determining the number of uncovered transitions B
Was a complete solution found ($B = 0$)
Yes
No
$B < B_{min}$
No
Yes
Select the current partial solution as the best one and replace the value of B_{min} with B
No
Time t_A has expired
Yes
End

Figure 2 – Block diagram of the algorithm A2

3. At each iteration of the algorithm, the codes of all states are selected in a pseudo-random way among the set of admissible state codes. In Python, for example, the *shuffle* function of the standard *np.random* module can be used for this purpose, applied to the list of admissible state codes.

4. The search for the best partial solution is similar to algorithm A1. So, the number of uncovered transitions is determined by the matrix method proposed in [13].

5. Unlike algorithm A1, algorithm A2 allows for the re-generation of previously generated code sets. For example, if a sequence of codes was generated:
15, 5, 6, 25, 9, 0, 26, 28, 2, 18, 11, 3, 1, 7, 23, 22, 10, 12,
nothing prevents this sequence from being generated repeatedly later. This fact reduces the performance of the algorithm to some extent, since the number of coding variants considered by algorithm A2 in time t_A will generally be less than the number considered under the same conditions by algorithm A1.

In practice, this can show itself as follows. Let's imagine a small GSA containing $M = 8$ states. To encode them, $R = 3$ binary digits are enough. With these values, expression (1) gives $N = 40320$ possible state encoding variants. Algorithm A1, capable of processing up to 200,000 coding variants per second, will process 40320 variants in about 0.2 seconds. During this time, it will find all complete solutions that exist for a given GSA and a set of TOs.

Algorithm A2 will work differently under the same conditions. Since the value of t_A is given, the algorithm will work for exactly that long, or until the first complete solution is found. If, for example, there is only one complete solution for a given GSA and a set of TOs, algorithm A2, when randomly selecting for state coding variants, may not generate the needed coding variant at all and fail to find a complete solution in the allotted time t_A. In this case, the user will be provided with the best of the partial solutions found, which may not be the absolutely best.

So, sometimes algorithm A2 can produce a worse result than algorithm A1. For practical applications, it is important to know which of these algorithms usually produces a better result on the same input data for the same running time. Since the answer to this question is not obvious, the authors conducted experimental studies of the comparative effectiveness of both algorithms. Let's consider these experiments.

4 EXPERIMENTS

The purpose of the experiments is to compare the efficiency of the algorithm A2 proposed in this paper and the prototype algorithm A1 considered in [14]. Since we are interested in finding formal solutions with fewer uncovered transitions, we will consider the algorithm that finds a solution with fewer uncovered transitions in the same time to be better. We will compare the algorithms using their software implementations in Python. The hardware used is a computer with i5–13500 processor running the Windows 11 operating system.

As you know, the search for solutions is influenced by a given GSA and a set of transition operations. When designing an FSM, its graph-scheme of algorithm is usually considered to be given and cannot be changed, while the set of transition operations remains variable. So, we will conduct experiments using the example of a GSA G (Fig. 1), changing the set of transition operations in each experiment.

The experiment includes the following stages:

1. Setting the set of transition operations and operation time t_A.

2. Continuous execution of algorithm A1 for five minutes with fixing the best solution found (solution with minimal number of uncovered transitions).

3. Repeat step 2 for algorithm A2.

4. Printing the final results.

Let's consider the result of the experiment for the GSA G and five transition operations $O_1 - O_5$ given by expressions (2) – (6).

$$O_1: \ K(a^{t+1}) = K(a^t) + 3_{10}; \qquad (2)$$

$$O_2: \ K(a^{t+1}) = K(a^t) \oplus 10011_2; \qquad (3)$$

$$O_3: \ K(a^{t+1}) = K(a^t) \,\&\, 01111_2; \qquad (4)$$

$$O_4: \ K(a^{t+1}) = K(a^t) \vee 01010_2; \qquad (5)$$

$$O_5: \ K(a^{t+1}) = K(a^t) \div 4_{10}. \qquad (6)$$

In this experiment, all the TOs are chosen at random. Each TO is a certain arithmetic or logic operation performed between two operands: code of current state of the FSM and a constant. In arithmetic operations (O_1, O_5), operands and result are interpreted as unsigned decimal numbers in range [0; 31]. In logical operations ($O_2 - O_4$), operands and result are interpreted as 5-bit binary vectors. In order to shorten the writing, we will denote these transition operations similarly to the notation in [14]: "+3", "⊕19", "&15", "∨10", "÷4".

Fig. 3 shows a screenshot of the program results. The results of algorithm A1 are located after the line "Algorithm A1", the results of algorithm A2 are located after the line "Algorithm A2". The result of each algorithm contains the following information:

– a list of used transition operations (written by the corresponding Python operators);

– the best solution found (state encoding variant);

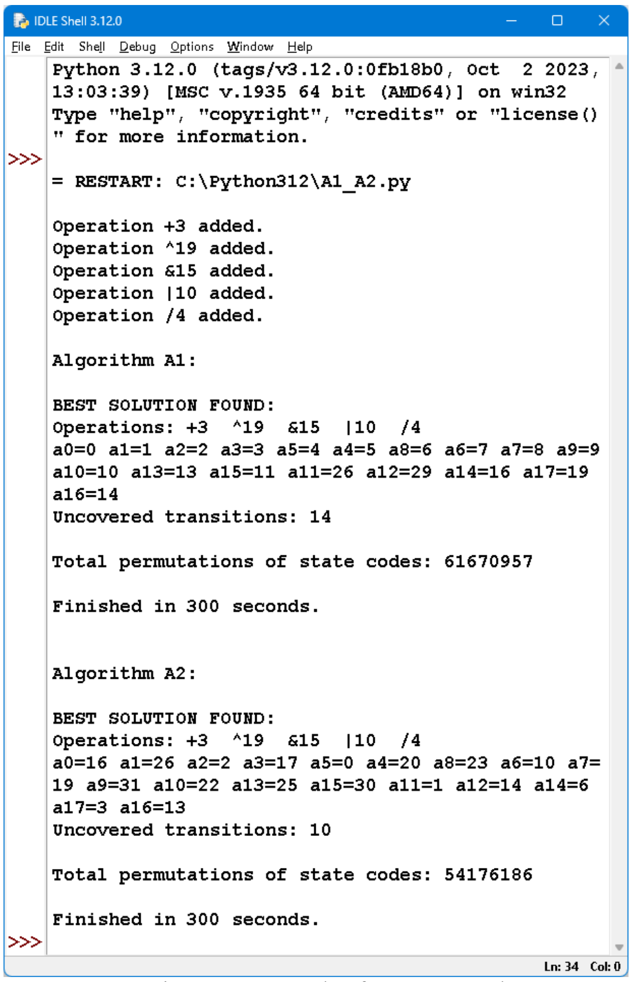
– the number of uncovered transitions;

– the number of considered state encoding variants;

– the algorithm running time (300 seconds or 5 minutes).

Fig. 3 shows that in this case, algorithm A2 performed better than algorithm A1. It found a solution with 10 uncovered transitions, while algorithm A1 found a solution with 14 uncovered transitions. It also can be seen that algorithm A2 performs the state encoding variants more slowly. In 300 seconds, it managed to process about 54 million variants, while algorithm A1 managed to process about 62 million. This is probably due to the slower speed of the pseudo-random state code mixing function. Nevertheless, the slower speed did not prevent algorithm A2 from finding a better solution to the algebraic synthesis problem compared to algorithm A1.

The software implementation of the algorithms is designed so that slightly different results can be obtained at each run. For algorithm A1, the number of considered state encoding variants can differ, since in the conditions of a multitasking operating system, the processor access time allocated to the program will be slightly different each time. For algorithm A2, a completely different result can be obtained, since each time the program is run, the initial state of the pseudorandom number generator of the Python interpreter will be different. However, in terms of the ratio of the number of uncovered transitions, each new result of the program will be close to the one considered above.



```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023,
13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()
" for more information.
>>>
= RESTART: C:\Python312\A1_A2.py

Operation +3 added.
Operation ^19 added.
Operation &15 added.
Operation |10 added.
Operation /4 added.

Algorithm A1:

BEST SOLUTION FOUND:
Operations: +3  ^19  &15  |10  /4
a0=0 a1=1 a2=2 a3=3 a5=4 a4=5 a8=6 a6=7 a7=8 a9=9
a10=10 a13=13 a15=11 a11=26 a12=29 a14=16 a17=19
a16=14
Uncovered transitions: 14

Total permutations of state codes: 61670957

Finished in 300 seconds.


Algorithm A2:

BEST SOLUTION FOUND:
Operations: +3  ^19  &15  |10  /4
a0=16 a1=26 a2=2 a3=17 a5=0 a4=20 a8=23 a6=10 a7=
19 a9=31 a10=22 a13=25 a15=30 a11=1 a12=14 a14=6
a17=3 a16=13
Uncovered transitions: 10

Total permutations of state codes: 54176186

Finished in 300 seconds.
>>>
```

Figure 3 – Example of program results

5 RESULTS

The experiment considered above was repeated for 20 different sets of TOs. The number of uncovered transitions obtained for each set of OPs by algorithms A1 and A2 is given in Table 1. Column "N" contains the row numbers of the table; columns $O_1 - O_5$ contain used TOs (one row of the table corresponds to one set of TOs); columns A1, A2 contain the number of uncovered transitions obtained by the corresponding algorithms for each set of TOs.

Among the values given in columns A1 and A2 of each row of the table, the smaller value is considered the better. For clarity, the smaller values are highlighted in bold and marked with a dark background. As can be seen, algorithm A1 had an advantage over algorithm A2 only in five cases, that is, in five out of twenty considered sets of OPs. In most cases, algorithm A2 found solutions with fewer uncovered transitions in the same running time. This may indicate a higher overall efficiency of pseudo-random encoding of states compared to continuous sequential enumeration of encoding variants.

Table 1 – Comparison of algorithms A1 and A2 by the number of uncovered transitions for different sets of OPs

N	Transition operations					Number of uncovered transitions	
	O_1	O_2	O_3	O_4	O_5	A1	A2
1	+ 3	+ 12	⊕ 19	& 10	∨ 11	14	**10**
2	+ 2	+ 19	⊕ 6	⊕ 27	& 15	14	**10**
3	+ 5	+ 8	+ 17	⊕ 23	∨ 19	16	**10**
4	⊕ 1	⊕ 3	⊕ 6	⊕ 18	⊕ 29	**7**	10
5	+ 5	+ 12	+ 17	+ 25	+ 30	15	**8**
6	+ 1	⊕ 12	⊕ 18	⊕ 25	⊕ 31	9	9
7	+ 2	+ 10	& 19	∨ 22	∨ 29	15	**11**
8	+ 3	⊕ 9	⊕ 17	× 2	÷ 2	**8**	10
9	+ 1	+ 2	+ 3	+ 4	+ 5	**7**	8
10	⊕ 15	⊕ 28	÷ 4	& 22	∨ 26	15	**10**
11	+ 2	+ 9	+ 15	+ 18	– 7	14	**9**
12	+5	& 7	& 23	∨ 13	∨ 23	18	**11**
13	+ 14	+ 15	⊕ 10	÷ 2	× 4	13	**10**
14	+ 1	+ 2	– 1	– 2	× 8	**9**	12
15	+10	+ 15	⊕ 11	⊕ 18	⊕ 26	17	**10**
16	+ 17	– 9	× 4	& 20	∨ 25	17	**10**
17	+ 1	– 1	× 2	× 4	÷ 2	**6**	11
18	+ 21	– 6	⊕ 14	∨ 10	∨ 21	17	**10**
19	– 2	– 5	– 11	– 20	– 27	12	**9**
20	+ 2	– 5	& 12	∨ 23	÷ 4	14	**11**

In the experiment, algorithm A1 performed a sequential enumeration for state encoding variants for each set of OPs, always starting from the initial variant (when the state codes correspond to their indices):
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17

This allowed us to analyze only the very beginning of the complete enumeration and did not allow us to "peek" into its other intervals.

In order to investigate the efficiency of sequential search of state encoding variants at different intervals of complete enumeration, a modification of algorithm A1 was performed, which consists in the following. The entire list of admissible state codes (in the case of GSA *G*, these are codes 0, 1, 2, ..., 30, 31) is shuffled in a pseudo-random manner once at the beginning of the algorithm. Then the algorithm generates sequential state encoding variants, starting from this pseudo-random set of codes. This approach allows us to investigate the random "lifetime" of a complete sequential enumeration of state encoding variants and to evaluate the efficiency of algorithm A1 regardless of the moment of its work.

Let us denote the modified algorithm A1 by A1* and compare its efficiency with algorithm A2. To do this, we will use the same sets of TOs and the same conditions as in the previous experiment. The results obtained are given in Table 2. As can be seen, in all cases considered, algorithm A2 showed better results compared to algorithm A1*. This is additional evidence of its effectiveness.

Table 2 – Comparison of algorithms A1* and A2

N	Transition operations					Number of uncovered transitions	
	O_1	O_2	O_3	O_4	O_5	A1*	A2
1	+ 3	+ 12	⊕ 19	& 10	∨ 11	15	**10**
2	+ 2	+ 19	⊕ 6	⊕ 27	& 15	14	**11**
3	+ 5	+ 8	+ 17	⊕ 23	∨ 19	14	**9**
4	⊕ 1	⊕ 3	⊕ 6	⊕ 18	⊕ 29	15	**9**
5	+ 5	+ 12	+ 17	+ 25	+ 30	14	**8**
6	+ 1	⊕ 12	⊕ 18	⊕ 25	⊕ 31	16	**10**
7	+ 2	+ 10	& 19	∨ 22	∨ 29	17	**10**
8	+ 3	⊕ 9	⊕ 17	× 2	÷ 2	14	**10**
9	+ 1	+ 2	+ 3	+ 4	+ 5	14	**10**
10	⊕ 15	⊕ 28	÷ 4	& 22	∨ 26	16	**9**
11	+ 2	+ 9	+ 15	+ 18	– 7	13	**9**
12	+5	& 7	& 23	∨ 13	∨ 23	17	**12**
13	+ 14	+ 15	⊕ 10	÷ 2	× 4	14	**9**
14	+ 1	+ 2	– 1	– 2	× 8	17	**13**
15	+10	+ 15	⊕ 11	⊕ 18	⊕ 26	15	**8**
16	+ 17	– 9	× 4	& 20	∨ 25	15	**10**
17	+ 1	– 1	× 2	× 4	÷ 2	17	**10**
18	+ 21	– 6	⊕ 14	∨ 10	∨ 21	14	**10**
19	– 2	– 5	– 11	– 20	– 27	14	**9**
20	+ 2	– 5	& 12	∨ 23	÷ 4	14	**9**

6 DISCUSSION

The proposed algorithm for algebraic synthesis of FSM with DT (algorithm A2) differs from the prototype algorithm (algorithm A1) in the way of searching for variants of FSM states encoding. Due to the pseudo-random selection of state codes, the average number of states whose codes change in each subsequent iteration of searching increases. As a result, the new algorithm in most of the analyzed situations allowed finding better solutions to the algebraic synthesis problem compared to the prototype algorithm.

The higher efficiency of algorithm A2 was proven only experimentally and could not be predicted theoretically. The results obtained were influenced by factors such as the structure of the given GSA, the used sets of transition operations and the program running time. However, according to the authors, made restrictions made it possible to determine the general trend, which is the greater efficiency of algorithm A2 compared to algorithm A1 when searching for solutions with a smaller number of uncovered automaton transitions.

The numbers in last two columns of Table 1 and Table 2 should be considered only in the context of their mutual comparison. One should not look at the fact that no complete solution was found in the experiments conducted. This was not the goal, but if necessary, a complete solution could perhaps be found by selecting suitable TOs and increasing the algorithm runtime.

DOI 10.15588/1607-3274-2025-4-3

CONCLUSIONS

The article proposes a new algorithm for algebraic synthesis of finite state machine with datapath of transitions. Its feature is the enumeration of state coding variants by pseudo-random selection of codes. The goal of the algorithm is to find complete or partial solutions to the algebraic synthesis problem for a given graph-scheme of algorithm and a set of transition operations.

The scientific novelty of the article lies in the fact that for the first time an algorithm for searching for complete and partial solutions to the algebraic synthesis problem with pseudo-random generation of state codes has been developed, implemented in software, and studied. Experimental research of the test GSA had shown that the new algorithm is generally capable of finding more efficient solutions compared to the previously known prototype algorithm based on a complete sequential enumeration of state coding variants. This proves the usefulness of its development and the feasibility of using it as part of FSM synthesis algorithms.

The practical use of the obtained results is possible in the development of methods and algorithms for the synthesis of finite state machine with datapath of transitions within the framework of specialized CADs of digital control units.

Prospects for further research are to solve a range of scientific and practical problems related to optimize the running time of the proposed algorithm by parallelizing the complete enumeration algorithm on multi-core computing systems. Also, the authors see a separate direction in studying the effectiveness of the algorithm on large-sized GSAs, in particular on pseudo-randomly generated GSAs. As a result, this will allow a more accurate assessment of the effectiveness of FSM with operational transformation of state codes according to the criterion of hardware expenses in the device circuit by modeling the synthesis of FSM circuits using hardware description languages.

ACKNOWLEDGEMENTS

The work was conducted in the direction of the state budget scientific research project of Vasyl' Stus Donetsk National University "Methods, algorithms and tools of computer-aided design of control units of computing systems" (state registration number 0122U200085).

REFERENCES

1. Bailliul J., Samad T. Encyclopedia of Systems and Control. Springer, London, UK, 2015, 1554 p. DOI: https://doi.org/10.1007/978-1-4471-5058-9
2. Czerwinski R., Kania D. Finite state machines logic synthesis for complex programmable logic devices. Berlin, Springer, 2013, 172 p. DOI: https://doi.org/10.1007/978-3-642-36166-1
3. Baranov S. Logic and System Design of Digital Systems. Tallin, TUTPress, 2008, 267 p.
4. Micheli G. Synthesis and Optimization of Digital Circuits. McGraw-Hill, Cambridge, MA, USA, 1994, 579 p.
5. Minns P., Elliot I. FSM-Based Digital Design Using Verilog HDL. JohnWiley and Sons, Hoboken, NJ, USA, 2008, 408 p. DOI: https://doi.org/ 10.1002/9780470987629
6. Skliarova I., Sklyarov V., Sudnitson A. Design of FPGA-based circuits using hierarchical finite state machines. Tallinn, TUT Press, 2012, 240 p.
7. Klimovich A. S., Solov'ev V. V. Minimization of Mealy finite-state machines by internal states gluing, *Journal of Computer and Systems Science International*, 2012, Volume 51, pp. 244–255. DOI: https://doi.org/10.1134/S1064230712010091
8. Grout I. Digital Systems Design with FPGAs and CPLDs. Elsevier Science, Amsterdam, The Netherlands, 2008, 784 p. DOI: https://doi.org/10.1016/B978-0-7506-8397-5.X0001-3
9. Kubica M., Opara A., and Kania D. Technology Mapping for LUT-Based FPGA, *Lecture Notes in Electrical Engineering,* 2021, Vol. 713, 207 p. DOI: https://doi.org/10.1007/978-3-030-60488-2
10. Baranov S. Logic Synthesis for Control Automata. Dordrecht, Kluwer Academic Publishers, 1994, 312 p.
11. Barkalov A. A. Babakov R. M. Operational formation of state codes in microprogram automata, *Cybernetics and Systems Analysis*, 2011, Volume 47 (2), pp. 193–197. DOI: https://doi.org/10.1007/s10559-011-9301-y
12. Barkalov A. A., Titarenko L. A., Babakov R. M. Synthesis of Finite State Machine with Datapath of Transitions According to the Operational Table of Transitions, *Radio Electronics, Computer Science, Control,* 2022, Volume 3 (62), pp. 109–119. DOI: https://doi.org/10.15588/1607-3274-2022-3-11
13. Barkalov A. A., Babakov R. M. A Matrix Method for Detecting Formal Solutions to the Problem of Algebraic Synthesis of a Finite-State Machine with a Datapath of Transitions, *Cybernetics and Systems Analysis*, 2023, Volume 59 (2), pp. 190–198. DOI: https://doi.org/10.1007/s10559-023-00554-6
14. Babakov R. M., Barkalov A. A., Titarenko L. A., Voitenko M. O. Algorithmic Differences of Complete and Partial Algebraic Synthesis of a Finite State Machine with Datapath of Transitions, *Radio Electronics, Computer Science, Control,* 2024, Volume 4, pp. 143–152. DOI: https://doi.org/10.15588/1607-3274-2024-4-14
15. Kubica M., Kania D., Kulisz J. A Technology Mapping of FSMs Based on a Graph of Excitations and Outputs, *IEEE Access*, 2019, Volume 7, pp. 16123–16131. DOI: https://doi.org/10.1109/ACCESS.2019.2895206
16. Baranov S. Finite State Machines and Algorithmic State Machines. Seattle, WA, USA: Amazon, 2018, 185 p.

Received 14.07.2025.
Accepted 29.10.2025.

УДК 004.94 : 004.2

ПСЕВДОВИПАДКОВЕ КОДУВАННЯ СТАНІВ В АЛГОРИТМІ АЛГЕБРАЇЧНОГО СИНТЕЗУ МІКРОПРОГРАМНОГО АВТОМАТА

Бабаков Р. М. – д-р техн. наук, доцент, професор кафедри інформаційних технологій Донецького національного університету імені Василя Стуса, м. Вінниця, Україна.

Баркалов О. О. – д-р техн. наук, професор, професор Інституту комп'ютерних наук та електроніки університету Зеленогурського, м. Зельона Гура, Польща.

Тітаренко Л. О. – д-р техн. наук, професор, професор Інституту комп'ютерних наук та електроніки університету Зеленогурського, м. Зельона Гура, Польща.

АНОТАЦІЯ

Актуальність. Розглянуто задачу алгебраїчного синтезу мікропрограмного автомата з операційним автоматом переходів. Схема цього автомата може потребувати менших витрат апаратури і мати меншу вартість у порівнянні зі схемами інших класів цифрових пристроїв керування. Об'єктом дослідження є процес пошуку повних і часткових розв'язків задачі алгебраїчного синтезу автомата із використанням спеціалізованих алгоритмів. Одним із таких алгоритмів є раніше відомий алгоритм повного послідовного перебору варіантів кодування станів при фіксованій множині операцій переходів. У переважній більшості випадків повний послідовний перебір виконується надто довго, що унеможливлює його практичне застосування в процесі синтезу автоматів з операційним перетворенням кодів станів. В даній роботі пропонується новий підхід, який полягає у заміні повного послідовного перебору варіантів кодування станів на псевдовипадкове кодування. Це дозволяє збільшити кількість кодів станів, що змінюються у кожній ітерації алгоритму, і може сприяти більш швидкому пошуку задовільних розв'язків задачі алгебраїчного синтезу.

Мета. Розробка і дослідження алгоритму пошуку розв'язків задачі алгебраїчного синтезу мікропрограмного автомата з операційним автоматом переходів на основі псевдовипадкового вибору кодів станів.

Метод. В основу дослідження покладено структуру мікропрограмного автомата з операційним автоматом переходів. Синтез схеми автомата передбачає обов'язковий етап алгебраїчного синтезу, результатом якого є поєднання певного способу кодування станів із заставленням арифметико-логічних операцій автоматним переходам. Таке поєднання називається розв'язком задачі алгебраїчного синтезу. В загальному випадку для заданого автомата існує багато розв'язків, кожен з яких може бути як повним (коли операції зіставлені усім переходам) або частковим (коли частина переходів не може бути реалізована за допомогою жодної із заданих операцій). Чим більше переходів реалізуються заданими операціями, тим менше апаратурних витрат потребуватиме реалізація схеми автомата і тим кращим є знайдений розв'язок. Пошук кращих розв'язків потребує розгляду великої кількості можливих варіантів кодування станів. В роботі здійснено модифікацію раніше відомого алгоритму, яка полягає у заміні повного послідовного перебору варіантів кодування станів на псевдовипадкову генерацію кодів. Обидва алгоритми були реалізовані програмно за допомогою мови Python і протестовані на прикладі мікропрограмного автомата, що імплементує абстрактний алгоритм керування. У процесі експериментів досліджувалось, який із алгоритмів знайде кращий розв'язок задачі алгебраїчного синтезу за фіксований час. Експерименти повторювались для різних наборів операцій переходів. Метою експериментів було оцінити, яка із стратегій завдання кодів станів є більш ефективною: послідовний перебір кодів станів чи їх псевдовипадкова генерація.

Результати. На прикладі абстрактного алгоритму керування продемонстровано, що загалом псевдовипадкове завдання кодів станів дозволяє за однаковий час знайти кращі розв'язки задачі алгебраїчного синтезу, ніж послідовний перебір кодів станів. Такі фактори, як швидкодія комп'ютера чи метод псевдовипадкової генерації кодів станів, не мають суттєвого впливу на результат експериментів. Перевага псевдовипадкової генерації кодів станів зберігається при використанні різних наборів операцій переходів.

Висновки. В основі алгебраїчного синтезу мікропрограмного автомата з операційним автоматом переходів лежить алгоритм пошуку розв'язків задачі алгебраїчного синтезу. В роботі запропоновано алгоритм пошуку таких розв'язків, оснований на псевдовипадковому кодуванні станів автомата. Програмна реалізація цього алгоритму довела, що такий підхід в загальному випадку є кращим за послідовний перебір варіантів кодування станів, оскільки дозволяє знайти кращі розв'язки (розв'язки з меншою кількістю операційно нереалізованих переходів) за той самий час. Псевдовипадкове завдання кодів станів може бути покладене в основу майбутніх алгоритмів алгебраїчного синтезу мікропрограмних автоматів.

КЛЮЧОВІ СЛОВА: мікропрограмний автомат, операційний автомат переходів, арифметико-логічні операції, алгоритм алгебраїчного синтезу, послідовний перебір, псевдовипадкова генерація кодів станів.

ЛІТЕРАТУРА

1. Bailliul J. Encyclopedia of Systems and Control / J. Bailliul, T. Samad. – Springer: London, UK, 2015. – 1554 p. DOI: https://doi.org/10.1007/978-1-4471-5058-9
2. Czerwinski R. Finite state machines logic synthesis for complex programmable logic devices / R. Czerwinski, D. Kania. – Berlin : Springer, 2013. – 172 p. DOI: https://doi.org/10.1007/978-3-642-36166-1
3. Baranov S. Logic and System Design of Digital Systems / S. Baranov. – Tallin : TUTPress, 2008. – 267 p.
4. Micheli G. Synthesis and Optimization of Digital Circuits / G. Micheli. – McGraw-Hill : Cambridge, MA, USA, 1994. – 579 p.

DOI 10.15588/1607-3274-2025-4-3

5. Minns P. FSM-Based Digital Design Using Verilog HDL / P. Minns, I. Elliot. – JohnWiley and Sons : Hoboken, NJ, USA, 2008. – 408 p. DOI: https://doi.org/10.1002/9780470987629
6. Skliarova I. Design of FPGA-based circuits using hierarchical finite state machines / I. Skliarova, V. Sklyarov, A. Sudnitson. – Tallinn : TUT Press, 2012. – 240 p.
7. Klimovich A. S. Minimization of Mealy finite-state machines by internal states gluing / A. S. Klimovich, V. V. Solov'ev // Journal of Computer and Systems Science International. – 2012. – Volume 51. – P. 244–255. DOI: https://doi.org/10.1134/S1064230712010091
8. Grout I. Digital Systems Design with FPGAs and CPLDs / I. Grout. – Elsevier Science: Amsterdam, The Netherlands, 2008. – 784 p. DOI: https://doi.org/10.1016/B978-0-7506-8397-5.X0001-3
9. Kubica M. Technology Mapping for LUT-Based FPGA / M. Kubica, A. Opara, and D. Kania // Lecture Notes in Electrical Engineering. – 2021. – Vol. 713. – 207 p. DOI: https://doi.org/10.1007/978-3-030-60488-2
10. Baranov S. Logic Synthesis for Control Automata / S. Baranov. – Dordrecht : Kluwer Academic Publishers, 1994. – 312 p.
11. Barkalov A. A. Operational formation of state codes in microprogram automata / A. A. Barkalov, R. M. Babakov // Cybernetics and Systems Analysis. – 2011. – Volume 47 (2). – P. 193–197. DOI: https://doi.org/10.1007/s10559-011-9301-y
12. Barkalov A. A. Synthesis of Finite State Machine with Datapath of Transitions According to the Operational Table of Transitions / A. A. Barkalov, L. A. Titarenko, R. M. Babakov // Radio Electronics, Computer Science, Control. – 2022. – Volume 3 (62). – P. 109–119. DOI: https://doi.org/10.15588/1607-3274-2022-3-11
13. Barkalov A. A. A Matrix Method for Detecting Formal Solutions to the Problem of Algebraic Synthesis of a Finite-State Machine with a Datapath of Transitions / A. A. Barkalov, R. M. Babakov // Cybernetics and Systems Analysis. – 2023. – Volume 59 (2). – P. 190–198. DOI: https://doi.org/10.1007/s10559-023-00554-6
14. Algorithmic Differences of Complete and Partial Algebraic Synthesis of a Finite State Machine with Datapath of Transitions / [R. M. Babakov, A. A. Barkalov, L. A. Titarenko, M. O. Voitenko] // Radio Electronics, Computer Science, Control. – 2024. – Volume 4. – P. 143–152. DOI: https://doi.org/10.15588/1607-3274-2024-4-14
15. Kubica M. A Technology Mapping of FSMs Based on a Graph of Excitations and Outputs / M. Kubica, D. Kania, J. Kulisz // IEEE Access. – 2019. – Volume 7. – P. 16123–16131. DOI: https://doi.org/10.1109/ACCESS.2019.2895206
16. Baranov S. Finite State Machines and Algorithmic State Machines / S. Baranov. – Seattle, WA, USA : Amazon, 2018. – 185 p.

DOI 10.15588/1607-3274-2025-4-3