UDC 004.6

Oleksandr K. Teslenko[1], Candidate of Technical Sciences, Associate Professor of System Software and Specialized Computer Systems Faculty, E-mail: teslenko@scs.kpi.ua, ORCID: 0000-0002-5891-4345
Maksym Y. Bondarchuk[1], PhD student of System Software and Specialized Computer Systems Faculty, E-mail: bondarchuk.m.y@gmail.com, ORCID: 0000-0002-4861-6627
[1]National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", 37 Peremohy Ave., Kyiv, Ukraine 03056

IMPLEMENTATION OF ARBITRARY BITNESS PERMUTATIONS IN ONE OF THE CLASSES OF LINEAR STRUCTURES

Abstract: *Speed of transformation and simplicity of implementation are one of the key contributors in permutation researches. The paper reviews the implementation of arbitrary bitness permutation in the field of computer engineering on one of the classes of combination structures of linear complexity from the number of variables – one-dimensional cascades of structural units. The fact that the reflection formed by the specified linear structure is completely the same as the reflection of the corresponding Mealy finite state machine as a prototype of the structural module of the cascade is used. This allowed us to explore the properties of structural units and the cascade as a whole in the context of the concepts of the theory of digital automata. The implementation of arbitrary bitness permutations is based on usage of the connected graphs for state table and on usage of unique combinations without repeats for each row of output table. The purpose of this permutation is to convert large volumes of data in fast and simple way using hardware or software with the ability to be used in multiple areas of researches. The study of providing the bijectivity of the reflection and the equivalence analysis of permutations was performed. The algorithm of construction of finite-state machines for implementation of direct and inverted permutations is shown, as well as examples of state and output tables construction. Examples of hardware implementation using field-programmable gate arrays are given. The size of state and output tables for the software implementation is estimated. The number of unique bijective reflections and amount of key information for the investigated permutation in cryptographic transformations has been estimated. The theoretical speed of transformations of the bijective reflection is estimated for both field-programmable gate arrays and software implementation according to the modern indicators of types of computing devices memory speed. The practical verification of processing speed is made with software implementation. Areas of application of the investigated arbitrary bitness permutation are proposed.*

Keywords*: permutation functions; structural synthesis of finite state machines; Mealy machine; bijective reflection; field-programmable gate arrays*

Introduction

Permutations (substitutes) are considered as functions of a single variable that provide a bijective reflection of input data at the output. Permutations are used when considering theoretical questions in different sections of mathematics (e.g. finite group theory, finite fields, combinatorics, etc.), and in practical development (for example in cryptographic transformation). Despite the considerable results of studies of permutations in mathematics [1], in computer engineering the implementation of substitutions has been studied to a lesser extent. In mathematics it is ordinary to characterize permutation by its degree – the number of elements of the input (output) set. In computer engineering, permutation can be characterized by the bitness b of the binary input data. In this case the degree of permutation is 2^b. The purpose of the work is creation and research of hardware and software means for implementation of permutations of arbitrary bitness. The problem is cost optimization and processing time while ensuring that the values of the result bits depend on all the input data bits, or at least on all previous input data bits. The usage of such implementation of permutation gives the prospect of increasing efficiency in creating various data transformations in one or another criterion set.

© Teslenko, O. K., Bondarchuk, M. Y., 2020

Analysis of recent scientific publications and achievements

Hardware implementation methods of limited bitness permutations have been investigated in a number of papers [2-4] in particular [3] and [4] have the implementation results for any 8-bit permutations, as well as for 8-bit permutations class with special properties, both made on field-programmable gate arrays (FPGA). The software implementation of permutations of a limited bitness is simple and is used in cryptographic transformations [5-6]. The considered papers investigate the "permutation-implementation" approach: one or another class of permutations is formed and then the implementation of permutations of this class is determined. [7] proposes "implementation-permutation" approach to the implementation of permutations, which could be

 DOI: 10.15276/hait 01.2020.7 ISSN 2663-0176 (Print)
ISSN 2663-7731 (Online)

described as follows: determines the linear complexity combinational structure from the bit of the input data, one-dimensional cascade of structural units (OCSU), and defines the classification of such structures: the classes of the simplest, simple, complex, unidirectional, bidirectional and regular.

Obviously, it is theoretically possible to implement permutations of arbitrary bitness with OCSU. This raises the following tasks: what should be the structural unit (SU) and how many and what permutations can be implemented on the OCSU of the appropriate class? In general, these problems are not solved.

The [8] shows that with the simplest unidirectional regular OCSU, 48 different substitutions of arbitrary bitness (with 2^b degree) ($b > 1$) can be implemented by changing the SU. 850 simple substitutions with 2^b degree ($b > 1$) can be implemented on the simplest bidirectional regular OCSU, as presented in [9]. The significant increase in the number of different permutations with a slight complication of OCSU allows predicting the effect-tiveness of further complications.

Problem formulation

This paper discusses the implementation of arbitrary bitness permutations in the class of unidirectional complex regular OCSUs. The structural unit of the cascade consists of two combination schemes: the first one implements the value of signals at the primary outputs, and the second one implements the value of the signals at the side outputs. The inputs of both combination circuits receive signals from the primary and side inputs of the SU (Fig. 1) Obviously, this class of OCSU implements the reflection of the input data, which is the same with the reflection of the input sequences of the corresponding Mealy finite-state machine (FSM or FSA – finite-state automata). This allows us to use the automata theory [10] to solve the following problems:

1) define combinational circuits for the signals' formation at the outputs of the SU and implementation of permutations of arbitrary bitness (bijective reflections) on the OCSU;

2) define conditions for different OCSUs to implement the same permutations;

3) determine the number of different permutations that can be implemented on the OCSU of the selected class;

4) define inverted SU, that would provide the implementation of inverted permutation for given direct SU.

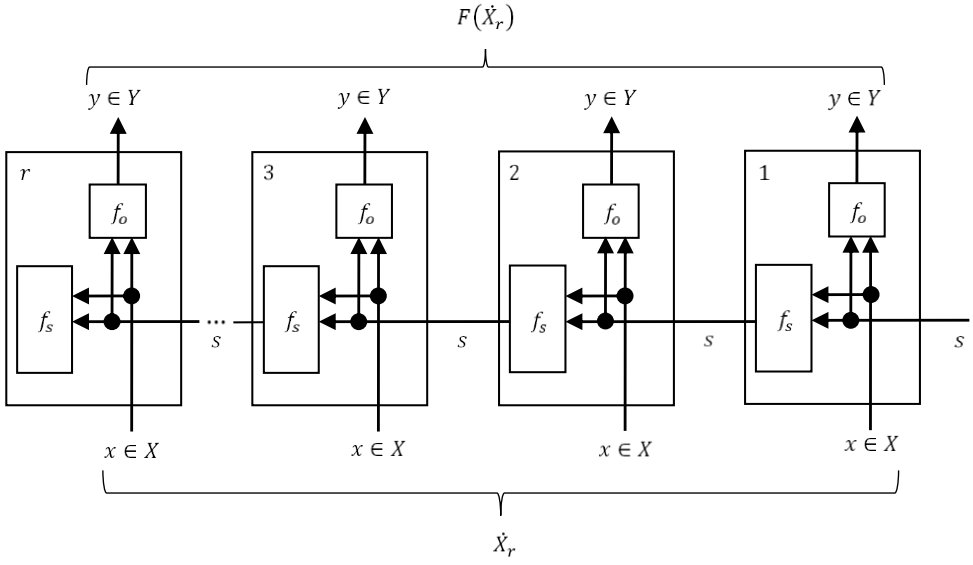

Fig. 1. Unidirectional OCSU for $F(\dot{X}_r)$ permutation implementation

Research Results

Terminology

Suppose we have a finite set of symbols called the input alphabet X of dimension n: $\{x_1, x_2, \dots, x_n\}$, a finite set of symbols called the output alphabet Y of dimension n: $\{y_1, y_2, \dots, y_n\}$, a finite set of states S of dimension m, a state transition function $\dot{f}_s(x,s): S \times X \to S$, the output transition function $\dot{f}_o(x,s): S \times X \to Y$ and the initial state $s_0 \in S$.

In the case when the machine is considered as a SU prototype, the input and output alphabets are the same, the values of n and m are degrees of two and the functions of outputs and states are implemented by the corresponding combination circuits. We will assume that state and output functions are fully defined. We will also assume that the machine is connected graph, which means, that it does not contain unreachable states and none of its states is equivalent to another [10]. In addition, an initial value is set at the side inputs of the first OCSU design module. This value corresponds to the initial state of the machine (Fig. 1).

Let us denote the set of all sequences of length r from the elements of the input alphabet by $\dot{X}_r$ and by Y_r for the output one. The Mealy machine (respectively OCSU) maps the set $\dot{X}_r$ to the set Y_r of OCSU ($\dot{X}_r \to \dot{Y}_r$).

Reflection bijectivity analysis

Machine implementation of the permutation (according to OCSU) means the implementation of a bijective reflection ($\dot{X}_r \to \dot{Y}_r$) at any value of r. In other words, different sequences of $\dot{X}_r$ correspond to different sequences of $\dot{Y}_r$.

To eliminate the ambiguity of the permutation (bijective reflection) of arbitrary bitness, which is implementation by the FSA (or OCSU), we will hereafter denote it as $F(\dot{X}_r)$.

Two sequences of characters considered different if they differ at least in one character. If machine's reflection is not bijective for some r, it will not be bijective for any $r_1 > r$.

The loss state is called the state s_l for which exist $x_1, x_2 \in X$, where $x_1 \neq x_2$ and $f_o(s_l, x_1) = f_o(s_l, x_2)$. Taking this into account, the following theorems are given.

Theorem 1. The machine's reflection $f_A: \dot{X}_r \to \dot{Y}_r$ is bijective on $\dot{X}$ if and only if the machine A does not contain loss states reached from the initial state s_0 in r steps [11-12].

Theorem 2. In order for OCSU to implement the $F(\dot{X}_r)$ permutation, it is required and sufficient that, for any signal at the side inputs of the SU, at the primary (not side) outputs, the combination circuit implements any substitution of the values of the signals at the primary inputs.

The bitness of the $F(\dot{X}_r)$ permutation is $b = r \log n$. The degree of permutation is n^r.

Equivalence analysis

Let $p(s)$ be some permutation of elements of the S set and let $p^{-1}(s)$ be the inverse permutation, that is, $p(p^{-1}(s)) = p^{-1}(p(s)) = s$.

Theorem 3. If for any machine A we create an automatic machine A_1 with the same alphabets X, Y, S and with the following functions $v_o(x,s) = f_o(x, p(s))$, $v_s(x,s) = p^{-1}(f_s(x, p(s)))$, and with the initial state $s_{0_A} = p^{-1}(s_0)$, then machines A and A_1 will be equivalent.

Prove. Let x symbol be the input of A and A_1. A_1 machine generates $v_o(x,s) = f_o(x, p(s))$ output symbol, and $v_s(x,s) = p^{-1}(f_s(x, p(s)))$ state. In the first iteration, the initial state of the machine A_1 is $s_{0_A} = p^{-1}(s_0)$. So, we have

$$v_o(x,s) = f_o(x, p(s)) = f_o\left(x, p(s_{0_A})\right) = f_o(x, p^{-1}(s_0)) = f_o(x,s).$$

A_1 will go into the state

$$p^{-1}\left(f_s\left(x, p(s_{0_A})\right)\right) = p^{-1}\left(f_s\left(x, p(p^{-1}(s_0))\right)\right) = p^{-1}(f_s(x, s_0)).$$

The same happens for any number of the following input characters. Thus, both machines implement the same reflection, which means, they are equivalent [10].

Thus, in the OCSU class under consideration, there are $m!$ OCSUs that implement the same $F(\dot{X}_r)$ permutation, which is important when estimating the number of different permutations.

Structural unit construction algorithm

The basic algorithm for construction output tables is as follows. Output table rows are states. A random number generator is used to get a random character from the set of outputs. The generator produces the next output symbol. The character is added to the row of the table that does not contain this character. If the character produced by the generator exists in all rows, it is skipped. The algorithm ends when all rows are filled. The basic algorithm for forming the state table is as follows. A random number generator is used to get a random

ISSN 2663-0176 (Print)
ISSN 2663-7731 (Online)

state from the set of states. Produced states are placed in the free cells of the state table. The algorithm ends after all cells in the state table are filled. If we want to get the connected graph, then the respective connectivity checks must be performed. If machine happens not to be a connected graph, the state table construction algorithm repeats.

Direct reflection state and output tables generation algorithm code snippet using C#:

```
InitialState = _random.Next(M);
for (int i = 0; i < M; i++)
{
  StateMatrix.Add(new List<int>());
  OutputMatrix.Add(new List<int>());
  for (int j = 0; j < N; j++)
  {
    StateMatrix[i].Add(_random.Next(M));
  }

  while (OutputMatrix[i].Count < N)
  {
    int y = _random.Next(N);
    if (!OutputMatrix[i].Contains(y))
    {
      OutputMatrix[i].Add(y);
    }
  }
}
```

So, let's consider the example of the machine for $m = 12$ and $n = 8$.

In each cell of the output table (Table 1) is one of the symbols of the output alphabet, and in each cell of the state table (Table 2) is one of the states.

The numeration of state and output table is made from top to bottom, starting with the first state and ending with the last one. The numeration of columns of conversion tables and exits is from left to right, starting with the first character of the input alphabet and ending with the last.

Table 1. Output table of $F(\dot{X}_r)$ FSA

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
s_1	y_1	y_4	y_6	y_7	y_2	y_5	y_3	y_8
s_2	y_6	y_3	y_4	y_8	y_5	y_7	y_2	y_1
s_3	y_4	y_8	y_3	y_6	y_2	y_5	y_1	y_7
s_4	y_1	y_3	y_8	y_4	y_5	y_7	y_6	y_2
s_5	y_1	y_7	y_6	y_3	y_2	y_5	y_4	y_8
s_6	y_3	y_1	y_6	y_5	y_4	y_8	y_7	y_2
s_7	y_4	y_7	y_3	y_1	y_6	y_2	y_5	y_8
s_8	y_4	y_1	y_8	y_5	y_2	y_6	y_3	y_7
s_9	y_5	y_7	y_4	y_8	y_2	y_3	y_6	y_1
s_{10}	y_4	y_1	y_7	y_8	y_5	y_2	y_3	y_6
s_{11}	y_6	y_3	y_2	y_1	y_5	y_4	y_8	y_7
s_{12}	y_6	y_2	y_7	y_1	y_3	y_8	y_5	y_4

Table 2. State table of $F(\dot{X}_r)$ FSA

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
s_1	s_1	s_{12}	s_5	s_{11}	s_4	s_{11}	s_5	s_4
s_2	s_7	s_{11}	s_4	s_4	s_8	s_{12}	s_8	s_3
s_3	s_{10}	s_{12}	s_1	s_7	s_{10}	s_5	s_9	s_6
s_4	s_6	s_7	s_1	s_5	s_8	s_8	s_{10}	s_7
s_5	s_{11}	s_{12}	s_5	s_3	s_4	s_7	s_{10}	s_4
s_6	s_9	s_7	s_3	s_9	s_3	s_6	s_3	s_6
s_7	s_8	s_8	s_8	s_9	s_{11}	s_7	s_7	s_1
s_8	s_{10}	s_2	s_{11}	s_5	s_6	s_1	s_4	s_{12}
s_9	s_7	s_2	s_8	s_6	s_2	s_{12}	s_{12}	s_{10}
s_{10}	s_9	s_{10}	s_{12}	s_{11}	s_{12}	s_7	s_2	s_9
s_{11}	s_9	s_{11}	s_{12}	s_1	s_{10}	s_9	s_1	s_{11}
s_{12}	s_4	s_{12}	s_9	s_4	s_5	s_6	s_5	s_5

Software implementation of reflection

$\dot{X}_r \to \dot{Y}_r$ reflection implementation algorithm code snippet using C#:

```
int state = InitialState;
var outputs = new List<int>();
for (int i = 0; i < r; i++)
{
  outputs.Add(OutputMatrix[state][input[i]]);
  state = StateMatrix[state][input[i]];
}
return outputs;
```

It's easy to see simplicity of implementation when using conversion tables, which can be stored in either the memory or processor cache and high speed of direct and inverse transformations.

Note that in general case it may be necessary to reformat the result in $\dot{Y}_r$.

Inverted reflections

In computer engineering, the usage of direct permutations is in many cases accompanied by the inverted permutations. The inverted bijective reflection $\dot{Y}_r \to \dot{X}_r$ (inverted permutation $F^{-1}(\dot{Y}_r)$) is defined as $F^{-1}(F(\dot{X}_r)) = \dot{X}_r$ and is implemented by the inverted machine. The inverted machine is determined based on the direct machine.

Inverted permutation $F^{-1}(\dot{Y}_r)$ FSA implementation construction algorithm

The output function is generated using following algorithm. The symbols of the direct machine's output table (Table 3) are rearranged for each state s independently from other states. For each output table cell from left to right that corresponds to y_a output alphabet character we put x_b input alphabet character, that was responsible for the current symbol y_a in the direct permutation

output table. Indices a and b can have any value or can be the same.

State table (Table 4) is generated based on the inverted machine's outputs function and direct machine's output table. The symbols of the direct machine's state table are rearranged for each row independently from other. In the cell corresponding to the y_a output alphabet symbol from left to right we put s_b state that has the same coordinates in the state table as the corresponding to y_a output symbol inverted permutation output table's symbol x_b. Indices a and b can have any value or can be the same.

Inverted reflection state and output tables generation algorithm snippet using C#:

```
InitialState = directMachine.InitialState;
for (int i = 0; i < M; i++)
{
  StateMatrix.Add(new List<int>());
  OutputMatrix.Add(new List<int>());

  for (int j = 0; j < N; j++)
  {
    for (int k = 0; k < N; k++)
    {
      if (directMachine
        .OutputMatrix[i][k] == j)
      {
        OutputMatrix[i].Add(k);
        StateMatrix[i].Add(
            directMachine.StateMatrix[i][k]);
        break;
      }
    }
  }
}
```

Inverted bijective reflection $\dot{Y}_r \to \dot{X}_r$ formation algorithm is completely the same as previous algorithm for direct machine. The difference is only in the usage of appropriate tables.

Table 3. Output table of $F^{-1}(\dot{Y}_r)$ FSA

	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8
s_1	x_1	x_5	x_7	x_2	x_6	x_3	x_4	x_8
s_2	x_8	x_7	x_2	x_3	x_5	x_1	x_6	x_4
s_3	x_7	x_5	x_3	x_1	x_6	x_4	x_8	x_2
s_4	x_1	x_8	x_2	x_4	x_5	x_7	x_6	x_3
s_5	x_1	x_5	x_4	x_7	x_6	x_3	x_2	x_8
s_6	x_2	x_8	x_1	x_5	x_4	x_3	x_7	x_6
s_7	x_4	x_6	x_3	x_1	x_7	x_5	x_2	x_8
s_8	x_2	x_5	x_7	x_1	x_4	x_6	x_8	x_3
s_9	x_8	x_5	x_6	x_3	x_1	x_7	x_2	x_4
s_{10}	x_2	x_6	x_7	x_1	x_5	x_8	x_3	x_4
s_{11}	x_4	x_3	x_2	x_6	x_5	x_1	x_8	x_7
s_{12}	x_4	x_2	x_5	x_8	x_7	x_1	x_3	x_6

Table 4. State table of $F^{-1}(\dot{Y}_r)$ FSA

	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8
s_1	s_1	s_4	s_5	s_{12}	s_{11}	s_5	s_{11}	s_4
s_2	s_3	s_8	s_{11}	s_4	s_8	s_7	s_{12}	s_4
s_3	s_9	s_{10}	s_1	s_{10}	s_5	s_7	s_6	s_{12}
s_4	s_6	s_7	s_7	s_5	s_8	s_{10}	s_8	s_1
s_5	s_{11}	s_4	s_3	s_{10}	s_7	s_5	s_{12}	s_4
s_6	s_7	s_6	s_9	s_3	s_9	s_3	s_3	s_6
s_7	s_9	s_7	s_8	s_8	s_7	s_{11}	s_8	s_1
s_8	s_2	s_6	s_4	s_{10}	s_5	s_1	s_{12}	s_{11}
s_9	s_{10}	s_2	s_{12}	s_8	s_7	s_{12}	s_2	s_6
s_{10}	s_{10}	s_7	s_2	s_9	s_{12}	s_9	s_{12}	s_{11}
s_{11}	s_1	s_{12}	s_{11}	s_9	s_{10}	s_9	s_{11}	s_1
s_{12}	s_4	s_{12}	s_5	s_5	s_5	s_4	s_9	s_6

Hardware implementation using FPGA

Regular unidirectional OCSU is a combination circuit where each SU implements $d + w$ boolean functions from $d + w$ variables, where $d = log\, n$, $w = log\, m$. In modern FPGA [13], any boolean function of 6 variables can be implemented on a single lookup table (LUT). If $d + w < 7$, then $d + w$ LUTs are required to implement a single SU, and $r(d + w)$ LUTs are required to implement the OCSU reflection ($\dot{X}_r \to \dot{Y}_r$). Two 5-variables functions can be implemented using one LUT. Then for $d + w < 6$ (e.g. $= 2$, $w = 2$) the SU is implemented using two LUTs, and byte transformations are implemented using 8 LUTs.

The 8-inputs circuit is implemented using 4 sequentially connected SUs, each with 2 inputs and 2 outputs (Fig. 1). In this way, the byte reflection is implemented. The number of different byte reflections is equal to the number of different SUs, considering graph's connectivity of the machines and their equivalence. According to (1), the number is over 20 million, but the number of different 8-bit permutations is much larger: 1.26887E+89. However, there is a problem of *permutation's belongings*, which are implemented on four such or other SUs, given the requirements for one or another application, which is the subject of further research.

Let t_{LUT} be the delay of one LUT. Then the processing speed is rt_{LUT}.

An example of a unidirectional OCSU is shown in Fig. 1.

SU's state and output tables size estimations

An alphabet of length n requires $d = log\, n$ bits for each character. Dimensions of state and output tables are $n \times m$. $w = log\, m$ bits are required to

ISSN 2663-0176 (Print)
ISSN 2663-7731 (Online)

store the current state. Thus, the total amount of memory for the presented function is

$$Q = nmd + (nm + 1)w.$$

The results of this calculation are presented in Table 5. The value of Q accurately reflects the amount of memory required in the case of software implementations. In the case of bit-stream implementation for FPGA debugging, Q can be considered as the lower bound.

Table 5 provides a visual representation of the volumes of data for SU formation. We see that large values of n and m require terabytes of data, which complicates hardware implementation but is not a problem for modern software implementations.

Table 5. Memory amount required for the FSA software implementation

n	d, bits	m	w, bits	Q, bytes	Q, MB	Q, GB
8	3	8	3	48	0,00005	0,00000005
16	4	16	4	257	0,00024	0,00000024
64	6	64	6	6145	0,00586	0,00000572
64	6	128	7	13313	0,01270	0,00001240
128	7	128	7	28673	0,02734	0,00002670
128	7	256	8	61441	0,05859	0,00005722
256	8	256	8	131073	0,1250	0,00012207
256	8	512	9	278529	0,266	0,00025940
1024	10	1024	10	2621441	2,50	0,00244141
1024	10	2048	11	5505025	5,3	0,00512695
4096	12	4096	12	50331650	48	0,04687500
16384	14	16384	14	939524098	896	1
16384	14	32768	15	1946157058	1856	2
65536	16	65536	16	17179869186	16384	16
65536	16	131072	17	35433480194	33792	33
1048576	20	1048576	20	5497558138883	5242880	5120
1048576	20	2097152	21	11269994184707	10747904	10496
2097152	21	2097152	21	23089744183299	22020096	21504
2097152	21	4194304	22	47278999994371	45088768	44032
4194304	22	4194304	22	96757023244291	92274688	90112
4194304	22	8388608	23	197912092999683	188743680	184320
8388608	23	8388608	23	404620279021571	385875968	376832
8388608	23	16777216	24	826832744087555	788529152	770048
16777216	24	16777216	24	1688849860263940	1610612736	1572864
16777216	24	33554432	25	3448068464705540	3288334336	3211264

Bijective reflections number estimations

Obviously, the number of reflections of any length is determined by the number of different SUs and depends on the initial state and the number of different states and output functions. Thus, the number of possible permutations $F(\dot{X}_r)$ is

$$O(n, m) = L_{f_s} L_{f_o} m,$$

where: L_{f_s} is the number of state functions; L_{f_o} is the number of output functions and m is the initial states number.

Output table's number is calculated as

$$L_{f_s} = m^{nm},$$

which is any case of a state table without any restrictions. The development of the adjusting parameters of this number is the subject of further research and can be considered as the number of connected graphs, in accordance with the accepted restrictions.

Using the formula for the number of possible connected graphs with m vertices of A001349 sequence [14], the number of state functions is reduced to

$$L_{f_s} = 1 + log \sum_{k=0}^{\infty} \frac{2^{\frac{m(m-1)}{2}}}{m!} k^m.$$

Given this sequence, the number of conversion functions can be represented by the following range

$$1 + log \sum_{k=0}^{\infty} \frac{2^{\frac{n(n-1)}{2}}}{n!} k^n \le L_{f_s} \le m^{nm}.$$

Output functions number L_{f_o} is calculated as the variation of output table rows, each of which has $n!$ variants according to the bijectivity requirements at m possible places [15]. That is

$$L_{f_o} = A_{n!}^{m} = \frac{(n!)!}{(n!-m)!},33333$$

where $m \le n!$.

Taking this into account the number of $F(\dot{X}_r)$ permutations can be approximated as

$$\left(1 + log \sum_{k=0}^{\infty} \frac{2^{\frac{m(m-1)}{2}}}{m!} k^m\right) \frac{\mathrm{m}(n!)!}{(n!-m)!} \leq O(n,m) \leq \frac{m^{nm+1}(n!)!}{(n!-m)!}. (1)$$

Note that the number of different permutations $F(\dot{X}_r)$ will be at least $m!$ times smaller, according to Theorem 2.

Software implementation processing speed estimations

To convert one character of the input sequence, three operations must be performed: two memory reads using indices from state and output tables and status change. The index is determined by the value of the current state and the input alphabet character as stated above. Status change is a memory write operation. Since no calculations are used, the direct and inverse transformations times are directly proportional to the speed of used memory (CPU cache levels L1, L2, L3, L4, random-access memory (RAM), read-only memory (ROM), external device, etc.).

Let's denote the speed of performing one operation with memory as v (MB/s).

Then the conversion time T of the input message with the length r of the input alphabet is calculated as

$$T = 4vr,$$

where: the factor 4 consists of one read operation from state table, one read operation from output table, one read operation of the current state and one write operation of the new state.

Table 6 lists the memory capacity and maximum speed of different types of memory according to recent studies [16]. The amount of storage should be considered when evaluating the permutation appliance in accordance with Table 5.

Table 6. Speed and volume of memory types

Memory type	Volume	Speed
L1 cache	128 KB	700 Gbps
L2 cache	1 MB	200 Gbps
L3 cache	6 MB	100 Gbps
L4 cache	128 MB	40 Gbps
RAM	Gigabytes	10 Gbps
ROM	Terabytes	160 MB/s

Table 7 shows the permutation processing time, depending on the type of memory, bitness of the input alphabet and the number of states. For cases when the maximum amount of memory at the time of article writing exceeds the requirements of Table 5, the time is marked by *.

Table 7. Theoretical time of permutation processing

r	T L1, ns	T L2, ns	T L4, ns	T L5, ns	T RAM, ns	T ROM, ns
8	4,26E-02	1,49E-01	2,98E-01	7,45E-01	2,98E+00	4,66E+01
16	8,51E-02	2,98E-01	5,96E-01	1,49E+00	5,96E+00	9,31E+01
32	1,70E-01	5,96E-01	1,19E+00	2,98E+00	1,19E+01	1,86E+02
64	3,41E-01	1,19E+00	2,38E+00	5,96E+00	2,38E+01	3,73E+02
128	6,81E-01	2,38E+00	4,77E+00	1,19E+01	4,77E+01	7,45E+02
256	1,36E+00	4,77E+00	9,54E+00	2,38E+01	9,54E+01	1,49E+03
512	2,72E+00	9,54E+00	1,91E+01	4,77E+01	1,91E+02	2,98E+03
1024	5,45E+00	1,91E+01	3,81E+01	9,54E+01	3,81E+02	5,96E+03
2048	1,09E+01	3,81E+01	7,63E+01	1,91E+02	7,63E+02	1,19E+04
4096	2,18E+01	7,63E+01	1,53E+02	3,81E+02	1,53E+03	2,38E+04
8192	4,36E+01	1,53E+02	3,05E+02	7,63E+02	3,05E+03	4,77E+04
16384	8,72E+01	3,05E+02	6,10E+02	1,53E+03	6,10E+03	9,54E+04
32768	1,74E+02	6,10E+02	1,22E+03	3,05E+03	1,22E+04	1,91E+05
65536	3,49E+02	1,22E+03	2,44E+03	6,10E+03	2,44E+04	3,81E+05
131072	6,98E+02	2,44E+03	4,88E+03	1,22E+04	4,88E+04	7,63E+05
262144	1,40E+03	4,88E+03	9,77E+03	2,44E+04	9,77E+04	1,53E+06
524288	2,79E+03	9,77E+03	1,95E+04	4,88E+04	1,95E+05	3,05E+06
1048576	5,58E+03*	1,95E+04	3,91E+04	9,77E+04	3,91E+05	6,10E+06
2097152	1,12E+04*	3,91E+04	7,81E+04	1,95E+05	7,81E+05	1,22E+07
4194304	2,23E+04*	7,81E+04	1,56E+05	3,91E+05	1,56E+06	2,44E+07
8388608	4,46E+04*	1,56E+05	3,13E+05	7,81E+05	3,13E+06	4,88E+07

ISSN 2663-0176 (Print)
ISSN 2663-7731 (Online)

Appliance

The proposed implementations of permutations can be used in data compression [17], combination circuits optimization [8; 18], non-algorithmic implementation of encoders and decoders of fault-tolerant coding [19] and to increase efficiency of algorithms' software implementation [20]. The studied permutation has a wide range of applications and can theoretically be used in any device that requires, for example, high speed encryption and decryption.

Results comparison with existing counterparts

The effectiveness of the proposed solutions follows from the comparisons with known results.

In hardware implementations, for example, byte permutations in our case, it is enough to use $C = 8(d + w)/d$ LUTs, at the same time, the best implementations of the byte substitutions proposed in [4] require 19 LUTs. If we take $d = 4$, $w = 2$ ($n = 16$, $m = 4$), then we have $C = 12$ LUTs. If we want to implement two boolean functions with 5 variables and take $d = 2$, $w = 2$ ($n = 4$, $m = 4$), we will have $C = 8$ LUTs.

About time characteristics of software implementations. In [21], the implementation of high-bitness permutations using simple transformations in Galois fields is considered. The theoretical speed calculations are substantially inferior to those given in Table 7, since in our case we achieve speed of up to one million Mbps. There are no experimental results in [21].

Table 8 shows the results of the experiments (processing time T_r and processing speed ϑ_r) for the implementation of the bijective reflections for the $d = w = 8$ ($n = m = 256$) case, and the r value actually corresponds to the number of bytes of the file. The experiments were performed using DDR3 RAM and an Intel Core i7-6700K processor.

Experimental time of processing is lower than theoretical (especially when converting small amounts of data), because in modern multi-threaded operating systems it is difficult to gain monopoly access to system resources, especially cache when processing long-length files.

In [5] experimental data of time of cryptographic transformations by the "Kalyna" standard are given. The best results have a speed of about 2500 Mbps. Speeds in Table 8 in general exceed the results of "Kalyna". However, it should be noted that the conditions for conducting the experiment in [5] actually correspond to theoretical calculations. This is evidenced by usage of a cache that contained only one block (16 to 64 bytes) of the original message and encrypted only that block (in fact, electronic code book (ECB) mode). In our case, the theoretical speed is more than one million Mbps.

Table 8. Permutation's software implementation processing time and speed

r	T_r ОЗП, ns	ϑ_r, Mbps
1024	3,06E+04	255,00
2048	3,46E+04	451,26
4096	4,26E+04	732,92
8192	5,14E+04	1216,55
16384	7,54E+04	1657,00
32768	1,18E+05	2110,82
65536	2,09E+05	2396,50
131072	3,94E+05	2541,13
262144	7,65E+05	2615,53
524288	1,50E+06	2672,66
1048576	3,06E+06	2612,61
2097152	5,78E+06	2769,95
4194304	1,17E+07	2743,38
8388608	2,12E+07	3023,83

Conclusions and prospects for further research

The obtained results allow conversion of files of any finite length. The number of different transformations increases faster than the exponent of the n and m parameters and does not depend on the file size. For example, at $n = m = 8$ (bitness of the SU data is only 3), the number of different $F(\dot{X}_r)$ permutations are estimated to be at least 10^{50}.

The studied implementation of the permutations provides good performance, although it does require memory space for the conversion table and the outputs. The results show that, in terms of the required amount of data, the output and state tables can be embedded even in the processor cache to provide high-speed conversion.

The subject of further research is the analysis of the strong cryptography of the proposed implementations of arbitrary bitness permutations and comparison of results with block cipher algorithms (it is clear that the studied function is faster, but has worse strong cryptography values, but it is not clear how much values will deiffer) and development of algorithms for different levels of cryptanalysis.

The prospect of further study may also be the consideration of a class of bidirectional OCSUs (Fig. 2 and Fig. 3), which are not considered in this paper because of too high memory requirements for large graph sizes.

Regarding providing bijection reflection in bidirectional SUs. For two states in both directions theoretically and practically the question is solved [9]. For more than two states, the solutions are not yet known.

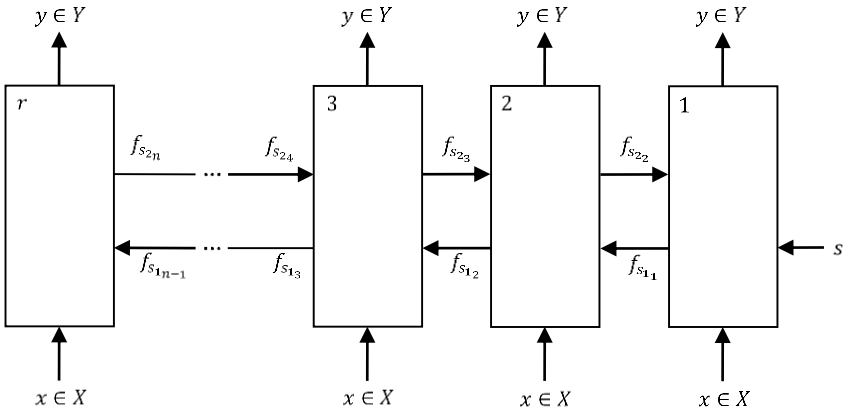

Fig. 2. Bidirectional OCSU for permutation

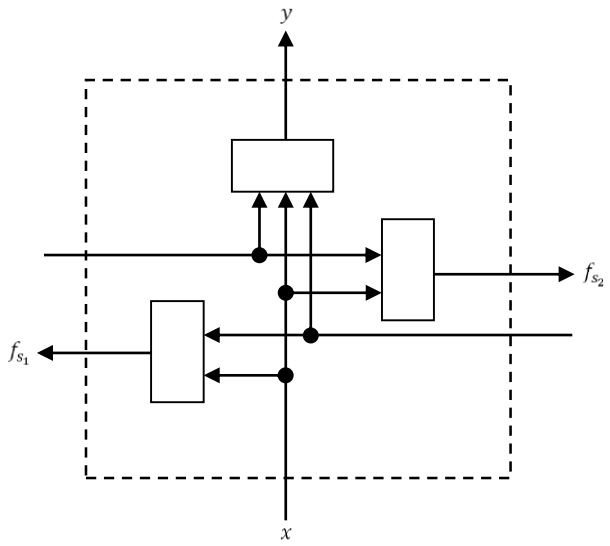

Fig. 3. One element of bidirectional OCSU for permutation

References

1. Hazewinkel, Michiel. (2000). "Permutation", *Encyclopedia of Mathematics, Springer Science+Business Media B. V. Kluwer Academic Publishers.* ISBN 9789048153787. DOI 10.1007/978-94-015-1279-4.

2. Peng, Li. (2019). "The Generation of (n,n(n−1),n−1) Permutation Group Codes for Communication Systems", *Communications IEEE Transactions on*, Vol. 67, No. 7, pp. 4535-4549. DOI 10.1109/TCOMM.2019.2902149.

3. Boss, E., Grosso, V. & Guneysu, T. (2017). "Strong 8-bit sboxes with ecient masking in hardware", *J. Cryptographic Engineering*. No. 7(2), pp.149-165. DOI 10.1007/s13389-017-0156-7.

4. Fomin, D. B. & Trifonov, D. B. (2019). "Ob aparatnoy realisazii odnogo klassa baytovyh podstanovok". [About hardware implementation of one class of byte permutations]. *Prikladnaya diskretnaya matematika*, Appendix 12, pp. 134-137 (in Russian).

5. Oliynykov, R., Gorbenko, R., Gorbenko, I., Kazymyrov, O., Ruzhevcev, Y. & Gorbenko Y. (April-June 2015). "Pryntsypy pobudovy i osnovni vlastyvosti novoho natsional'noho standartu blokovoho shyfruvannya ukrayiny" [Construction principles and basic properties of Ukraine's new national block cypher encryption standard]. *Zakhyst informatsiyi*, Vol. 2, pp. 142-157 (in Ukrainian).

6. (November 2001). "Advanced Encryp-tion Standard (AES)". (PDF). *Federal Information*

ISSN 2663-0176 (Print)
ISSN 2663-7731 (Online)

Processing Standards. DOI 10.6028/NIST.FIPS.197.

7. Tarasenko, V. P., Teslenko, O. K. & Yanovska, O. Y. (2007). "Problemy aparatnoi realisacii pidstanovok". [Problems of hardware implementation of permutations]. *Naykovi zapysky UNDIZ*, No. 2, pp. 52-58 (in Ukrainian).

8. Tarasenko, V. P., Teslenko, O. K. & Yanovska, O. Y. (2010). "Vlastyvosti povnykh pidstanovok, yaki realizuyut'sya nayprostishym odnonapravlenym rehulyarnym OKKM". [Properties of complete permutations implemented by the simplest unidirectional regular OCSU]. *Radioelektronni i komp'yuterni systemy*. No. 6, pp. 123-128 (in Ukrainian).

9. Tarasenko, V. P., Teslenko, O. K. & Yanovska, O. Y. (2012). "Mozhlyvosti naiprostishym dvonapravlenyh reguliarnyh odnovymirnyh kaskadiv konstruktyvnyh moduliv schodo realizacii riznyh povnyh pidstanovok". [Features of the simplest bidirectional regular one-dimensional cascades of structural units for the implementation of various complete permutations]. *Radioelektronni i kompyuterni systemy*. No. 7 (59), pp. 147-153 (in Ukrainian).

10. Glushkov, V. M. (1967). "Syntes cyfrovyh avtomatov". [Synthesis of Digital Automata]. Moskov, Russian Federation (in Russian).

11. Karandashov, M. V. (2014). "Issledovanie biektivnykh avtomatnykh otobrazhenii na kol'tse vychetov po moduliu 2^k". [Research bijective automaton mappings on the ring of residues modulo 2^k]. *Computer Science and Information Technologies: Proc. Intern. Sci. Conf.* Saratov, Russian Federation, *Publ. Center "Nauka*", pp. 148-152 (in Russian).

12. Gill, A. (1962). "Introduction to the theory of finite-state machines". New York, Toronto, Ontario, London, McGraw-Hill Book Co., Inc., 207 p. (Russ. ed.: "Gill A. Vvedenie v teoriiu konechnykh avtomatov". Moscow, Russian Federation, *Publ. Nauka*, 272 p.) (in Russian).

13. "Summary of Virtex-6 FPGA Features. Virtex-6 Family Overview". XILINX DS150 (v2.5). [Electronic resource]. – Available at: https://www.xilinx.com/support/documentation/data_sheets/ds150.pdf. – Active link: 20.08.2015.

14. "Sequence A001349 – Number of connected graphs with n nodes". The On-Line Encyclopedia of Integer Sequences® (OEIS®). [Electronic resource]. – Available at: http://oeis.org/A001349. – Active link: 10.01.2020.

15. J. H. van Lint & Wilson R. M. (1992). "A Course in Combinatorics", *Cambridge University Press*.

16. SiSoftware Official Live Ranker "SiSoftware Zone". [Electronic resource]. – Available at: https://ranker.sisoftware.co.uk/top_device_all.php?q=d6ebdb. – Active link: 31.07.2014.

17. Tarasenko, V. P., Teslenko, O. K. & Yanovska, O. Y. (2007). "Vykorystannya pryamykh ta obernenykh pidstanovok dovil′noyi rozryadnosti dlya pidvyshchennya efektyvnosti isnuyuchykh zasobiv ushchil′nennya danykh". [The use of direct and inverse arbitrary bit permutations to improve the performance of existing data comperssors]. *Fourth International Scientific and Practical Conference* "*Metody ta zasoby koduvannya, zakhystu y ushchil'nennya informatsiyi*", Vinnytsia, Ukraine, pp. 223-226 (in Ukrainian).

18. Tarasenko, V. P., Teslenko, O. K. & Yanovska, O. Y. (2010). "Analiz vplyvu pidstanovok na dekompozytsiyu bulevykh funktsiy". [Analysis of the effect of permutations on the decomposition of boolean functions]. *Herald of University "Ukrayina" Informatyka, obchyslyuval'na tekhnika ta kibernetyka*, No. 8, pp. 40-47 (in Ukrainian).

19. Klyatchenko, Y. M., Teslenko, O. K. & Yanovska, O. Y. (2011), "Vykorystannya pidstanovok dlya nealhorytmichnoyi realizatsiyi koderiv ta dekoderiv zavadostiykoho koduvannya" [Use of permutations for non-algorithmic implementation of encoders and decoders of fault-tolerant coding]. *International Scientific Conference "Suchasni komp'yuterni systemy ta merezhi: rozrobka ta vykorystannya"*, L'viv, Ukraine, pp. 191-193 (in Ukrainian).

20. Mel'nykova, O. A. & Maslyennikova, A. O. (2017). "Pidstanovky dlya pidvyshchennya efektyvnosti prohramnoyi realizatsiyi alhorytmiv, yaki vykorystovuyut' znakovo-tsyfrovi predstavlennya". [Permutations for increasing the efficiency of software implementation of algorithms that use character-to-digital representations]. "*Tekhnichni nauky" Series*, Vol. 15 Kharkiv National University of Radioelectronics, Kharkiv, Ukraine, pp. 126-132 (in Ukrainian).

21. Abornev, A. V. (2014). "Nelineynyye podstanovki na vektornom prostranstve, rekursivno-porozhdonnyye nad kol'tsom Galua

kharakteristiki 4" [Nonlinear permutations on a recursively generated vector space over a Galois ring of characteristic 4], *Prikladnaya diskretnaya matematika*. Vol. 7, pp .40-41 (in Russian)

Received. 31.01.2020
Received after revision 17.02.2020
Accepted 20.02.2020

УДК 004.6

[1]Тесленко, Олександр Кирилович, кандидат техніч. наук, старший науковий співробітник, доцент кафедри системного програмування і спеціалізованих комп'ютерних систем, E-mail: teslenko@scs.kpi.ua, ORCID: 0000-0002-5891-4345
[1]Бондарчук, Максим Юрійович, аспірант кафедри системного програмування і спеціалізованих комп'ютерних систем, E-mail: bondarchuk.m.y@gmail.com, ORCID: 0000-0002-4861-6627
[1]Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», проспект Перемоги 37, Київ, Україна, 03056

РЕАЛІЗАЦІЯ ПІДСТАНОВОК ДОВІЛЬНОЇ РОЗРЯДНОСТІ В ОДНОМУ ІЗ КЛАСІВ ЛІНІЙНИХ СТРУКТУР

Анотація: *Швидкість перетворення і простота реалізації є одними з ключових факторів підстановок. У статті розглянуто реалізацію підстановки довільної розрядності в області комп'ютерної інженерії на одному із класів комбінаційних структур лінійної складності від кількості змінних – одновимірних каскадів конструктивних модулів. Використано той факт, що відображення, яке формує вказана лінійна структура, повністю збігається з відображенням відповідного скінченного автомата Мілі як прототипу конструктивного модуля каскаду. Це дозволило досліджувати властивості конструктивних модулів та каскаду в цілому у розрізі понять теорії цифрових автоматів. Реалізація підстановок довільної розрядності полягає у використанні приведених автоматів для таблиці станів і використанні унікальних комбінацій без повторів для кожного рядку таблиці виходів. Метою реалізації даної підстановки є швидке перетворення даних великих об'ємів з можливістю застосування в кількох напрямках досліджень при простій реалізації на апаратному або програмному рівні. Виконано дослідження забезпечення бієктивності відображення та проведено аналіз еквівалентності відображень. Показано алгоритми формування автоматів для реалізації прямих та обернених підстановок, а також приклади формування таблиць переходів та виходів. Наведено приклади апаратної реалізації на програмованих логічних інтегральних схемах. Виконано оцінку об'єму таблиць переходів та виходів для апаратної та програмної реалізації. Виконано оцінку кількості унікальних бієктивних відображень. Проведено теоретичну оцінку швидкості бієктивних відображень при реалізації на програмованих логічних інтегральних схемах, а також при програмній реалізації згідно з сучасними показниками швидкості видів пам'яті обчислювальних пристроїв для кожного виду. Наведено експериментальну оцінку, а також проведено практичну перевірку швидкості перетворення за допомогою програмної реалізації. Запропоновано області застосування досліджених реалізацій підстановок довільної розрядності.*

Ключові слова: *функції підстановок; автомат Мілі; бієктивне відображення, програмовані логічні інтегральні схеми*

УДК 004.6

[1]Тесленко, Александр Кириллович, кандидат технич. наук, старший научный сотрудник, доцент кафедры системного программирования и специализированных компьютерных систем, E-mail: teslenko@scs.kpi.ua, ORCID: 0000-0002-5891-4345
[1]Бондарчук, Максим Юрьевич, аспирант кафедры системного программирования и специализированных компьютерных систем, E-mail: bondarchuk.m.y@gmail.com, ORCID: 0000-0002-4861-6627
[1]Национальный технический университет Украины «Киевский политехнический институт имени Игоря Сикорского», проспект Победы 37, Киев, Украина, 03056

ISSN 2663-0176 (Print)
ISSN 2663-7731 (Online)

РЕАЛИЗАЦИЯ ПОДСТАНОВОК ПРОИЗВОЛЬНОЙ РАЗРЯДНОСТИ В ОДНОМ ИЗ КЛАССОВ ЛИНЕЙНЫХ СТРУКТУР

Аннотация: *Скорость преобразования и простота реализации являются одними из ключевых факторов подстановок. В статье рассмотрены реализацию подстановки произвольной разрядности в области компьютерной инженерии на одном из классов комбинационных структур линейной сложности от количества переменных – одномерных каскадов конструктивных модулей. Использован тот факт, что отображение, которое формирует указанная линейная структура, полностью совпадает с отображением соответствующего конечного автомата Мили как прототипа конструктивного модуля каскада. Это позволило исследовать свойства конструктивных модулей и каскада в целом в разрезе понятий теории цифровых автоматов. Реализация подстановок произвольной разрядности заключается в использовании приведенных автоматов для таблицы состояний и использовании уникальных комбинаций без повторов для каждого строке таблицы выходов. Целью реализации данной подстановки является быстрое преобразование данных больших объемов с возможностью применения в нескольких направлениях исследований при простой реализации на аппаратном или программном уровне. Выполнены исследования обеспечения биективности отражения и проведен анализ эквивалентности отражений. Показано алгоритмы формирования автоматов для реализации прямых и обратных подстановок, а также примеры формирования таблиц переходов и выходов. Приведены примеры аппаратной реализации на программируемых логических интегральных схемах. Выполнена оценка объема таблиц переходов и выходов для аппаратной и программной реализации. Выполнена оценка количества уникальных биективных отражений. Проведено теоретическую оценку скорости биективных отражений при реализации на программируемых логических интегральных схемах, а также при программной реализации согласно современным показателям скорости видов памяти вычислительных устройств для каждого вида. Приведены экспериментальную оценку, а также проведено практическую проверку скорости преобразования с помощью программной реализации. Предложено области применения исследованных реализаций подстановок произвольной разрядности.*

Ключевые слова: *функции подстановок; автомат Мили; биективное отображения, программируемые логические интегральные схемы.*

Oleksandr Teslenko, Candidate of Technical Sciences
Research field: system programming, cryptography, design of specialized computer systems

Maksym Bondarchuk, PhD student
Research field: cryptography, automata theory, heuristics, artificial intelligence