35. Kondratenko, N. R. Application of Type-2 Membership Functions in Fuzzy Logic Systems [Text] / N. R. Kondratenko // Research Bulletin of the National Technical University of Ukraine "Kyiv Politechnic Institute". – 2016. – Vol. 2. – P. 43–50. doi: 10.20535/1810-0546.2016.2.51636
36. Karnik, N. Type-2 Fuzzy Logic Systems [Text] / N. Karnik, J. Mendel, Q. Liang // IEEE Transactions on Fuzzy Systems. – 1999. – Vol. 7, Issue 6. – P. 643–658. doi: 10.1109/91.811231
37. Kondratenko, N. R. Evolyucijnyj poshuk informatyvnykh oznak iz zaluchennyam eksperta v zadachi ocinky yakosti arteziansjkoi vody [Text] / N. R. Kondratenko, O. O. Snihur // Visnyk Vinnycjkogho politekhnichnogho instytutu. – 2015. – Vol. 3. – P. 96–101.
38. Teoriya informatsii i ee prilozheniya (sbornik perevodov) [Text] / A. A. Harkevich (Ed.). – Moscow: Fizmatgiz, 1959. – 328 p.

Розглянуто операцію додавання бінарних кодів без перенесення. Виявлено, що метод рекурсії забезпечує синтез системи бінарних кодів з кільцевою структурою при будь-якому початковому коді повної комбінаторної системи з повторенням, що й дозволяє використовувати обрану систему бінарних кодів для операції додавання без перенесення. Встановлена оцінка загальної складності обчислювального алгоритму суматора бінарних кодів

Ключові слова: суматор, комбінаторна система з повторенням, бінарні коди, додавання бінарних кодів, каскадна схема, клас комбінаторних систем, екземпляр класу, тезаурус, логарифмічна складність

Рассмотрена операция суммирования бинарных кодов без переноса. Выявлено, что метод рекурсии обеспечивает синтез системы бинарных кодов с кольцевой структурой при любом начальном коде полной комбинаторной системы с повторением, что и позволяет использовать выбранную систему бинарных кодов для операции суммирования без переноса. Установлена оценка общей сложности вычислительного алгоритма сумматора бинарных кодов

Ключевые слова: сумматор, комбинаторная система с повторением, бинарные коды, суммирование бинарных кодов, каскадная схема, класс комбинаторных систем, экземпляр класса, тезаурус, логарифмическая сложность

UDC 681.325
DOI: 10.15587/1729-4061.2016.75595

SUMMATION OF BINARY CODES WITHOUT CARRY

M. Solomko
PhD, Associate Professor*
E-mail: doctrinas@ukr.net
L. Zubyk
Senior Lecturer**
E-mail: labrob@ukr.net
P. Olshansky
Senior Lecturer*
E-mail: p.v.olszanski@nuwm.edu.ua
V. Nazaruk
PhD, Senior lecturer*
E-mail: v.d.nazaruk@nuwm.edu.ua
*Department of Computer Engineering***
Department of Computer Science*
***National University of Water and Environmental Engineering
Soborna str., 11, Rivne, Ukraine, 33028

1. Introduction

Binary code is a general designation of the code, by which messages can be transmitted in sequences that have two characters (for example, "0" and "1"). In general, the number of combinations (codes) of n-digit binary code is equal to the number of locations with repetition of n elements by m

$$\hat{P}(n,m)=n^m. \quad (1)$$

For a binary code, the number of combinations equals:

$$\hat{P}(2,n)=2^n, \quad (2)$$

where n is the digit capacity of a binary code.

The minimum possible number that can be written down by such a binary code equals 0. The maximum possible number that can be written down by such a binary code is determined by the formula

$$M=2^n-1. \quad (3)$$

Table 1

4-bit binary codes in lexicographical order

Numeric (literal) value	Binary code	Numeric (literal) value	Binary code
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

These two numbers fully determine the range of numbers that can be presented by a binary code (2). For example, for an 8-digit binary without a signed integer, the range of numbers is 0...255. For a 16-bit code, the range equals 0...65535.

The examples of binary codes are the code of Gray, Baudot code, Hamming code, ASCII, etc.

Development of the theory of universal and specialized processors is closely connected with the development of a binary number system, i.e. theoretical and numerical basis (TNB) of Rademacher [1].

The carry of one to a higher level in the basis of Rademacher leads to the decrease in fast performance of identifying correct signals at the outputs Si of the one-digit adders. The magnitude of this decline is proportional to the digit capacity of numbers and time delay of the signals in typical logical elements. The maximum time of the operation of summation occurs when the carry, which appeared in the first bit, passes through all the other bits (for example, when compiling codes 11...11 and 00...01). Modern achievements in the creation of high-performance processors are based on the designs of the theory of parallel computing.

The method of parallel computing in the Rademacher TNB, in particular, is based on the recurrent binary codes. Extension of the apparatus of obtaining these codes is one of the most central and practically important problems in this theory. The [2, 3] demonstrated, accordingly, receiving recurrent binary code using cyclic shift of the original 4-digit non-zero code fragment – 1111 and the original zero code fragment of 0000. Since the specified original code fragments are part of a complete combinatorial system with repetition (Table 1), the research is actual into the process of formation of recurrent binary codes with the help of the rest of the initial code fragments in Table 1, which allows extending the apparatus of obtaining recurrent binary codes, controlling the selection of a code at the stage of designing a computing device and the classification generalization of the binary codes with the aim of simplifying the structure of the subject area, increasing the diversity of the systems of binary codes, in particular, for arithmetic operations with binary numbers.

The relevance of the classification generalization of the binary codes for arithmetic operations is also in presenting the data of the systems of binary codes by a unified general table of the data, and, therefore, the unified general hardware costs at the level of an electronic device.

2. Literature review and problem statement

The [4] considered the design of the adder of binary numbers with a choice of carry (Carry Select Adder), which is one of the fast-performing versions of the parallel adder. The feature of the Carry Select Adder is in that the adder has linear complexity of the algorithm of the calculation, however, within the range of 16–128 bit scheme, it shows better efficiency of calculation compared with the scheme of the adder with logarithmic complexity of the calculation. The disadvantage of the Carry Select Adder is the organization of technology of selecting the carry through the split of the structure of the adder into separate groups of logical elements, each of which contributes to the delay of the carry signal (more groups – larger delay), which with increasing the digit capacity of the scheme reduces the productivity of computing.

The [5] demonstrated better efficiency of multiplying binary numbers for a 64-digit sign multiplier using the technology of Carry Select Adder (CSA), compared with the Carry Look-Ahead Adder (CLA). Thus, the results of the paper [5] confirm the specifics of the Carry Select Adder that were examined in the analysis of the work [4]. With the increase in the digit capacity of a sign multiplier, more promising is the technology of the CLA, since the latter uses a cascading scheme [6]. The calculations organized by a cascade scheme in the CLA demonstrate a significant advantage exactly while increasing the bit capacity of a device's scheme.

The [7] examined designing the adder of binary numbers with a skip in the carry (Carry Skip Adder), which is a modification of the parallel adder with the structure of lower hardware costs and energy consumption compared to Carry Look-ahead Adder. The feature of the Carry Skip Adder is in that the adder occupies a technological niche between the Ripple Carry Adder with greater productivity of computing and Carry Look-ahead Adder with lower hardware costs. The disadvantage of the Carry Skip Adder is the organization of the technology of skipping the carry through the breakdown of the structure of the adder into separate groups of logical elements, the half of which contributes to the delay of the transfer signal, which limits the productivity of computing, that is why the Carry Skip Adder technology achieves the complexity of the algorithm of calculation not less than linear.

The [8] demonstrated better performance in multiplying binary numbers for a 32-bit multiplier using the technology of Carry Save Adder (CSA) compared with Carry Look-Ahead Adder (CLA). The feature of using the CSA for the process of the multiplication by a multiplier is in the way of performing of addition of partial multiplications (Carry Save) and their final summation. The technology of Ripple Carry Adder is used only at the final stage. The multiplier with the CSA requires fewer complete adders than the multiplier with the CLA. Since the Carry Look-Ahead Adder uses a cascading scheme [6], then with increasing the digit capacity of binary numbers, the CLA becomes more promising. However, the analysis in [8] is limited by the consideration of calculations of 32-digit numbers while other data are not presented.

The [9] presents designing and implementation of an 8-bit Carry Look-ahead Adder with low power consumption based on the 180 nm CMOS technology. The peculiarity of the CLA is the lowest depth of the adder's scheme; in particular, for a 2048-bit scheme of the CLA, the depth is 26 logical elements. An adder has logarithmic complexity of the calculation algorithm. Therefore, the CLA is the main functional unit in arithmetic-logical devices due to its high performance speed. However, the lower depth of the adder's scheme is achieved by increasing the complexity of the scheme, which is a disadvantage of a Carry Look-ahead Adder. That is why 8-bit CLA is sufficient. In the practical context, 32-bit [8] and 64-bit CLA [5] are known.

The [10] presented a scheme of a serial adder with reverse logic gates. Reverse logic is able to effectively dissipate heat energy, which is the main requirement when designing VLSI with low power consumption. Reverse serial adder [10] is based on the serial adder of binary numbers which includes a complete single-digit adder and a trigger. The disadvantage of the latter adder is that it implements the technology of Ripple Carry Adder in the worst version, since the chain of carry consists of three logical elements, and, therefore, when adding n-bit numbers, the chain of carry will consist of 3n logical elements. Speeding up calculations is possible when using a serial adder without carry. In this case, a complete single-digit adder and a trigger will be replaced by one logical element – OR or AND, which is the advantage of the reviewed technology.

The [11] presents designing and implementation of a 16-bit Ripple Carry Adder (RCA) with low power consump-

tion based on the 45 nm CMOS technology. The peculiarity of the RCA is the least complexity of the scheme in the class of parallel adders that provides the scheme's performance with minimal consumption of power and thus it becomes possible, to certain extent, to solve primary problems in today's VLSI that arise due to two main reasons. One is the continued work of a battery for servicing mobile and portable devices, and the second is due to the increase in the number of transistors on a single-chip VLSI. The disadvantage of the RCA is the increase in the carry signal chain, which slows down the establishment of stable values of the signals at the Si outlets of the RCA one-digit adders.

The operation of summation of binary numbers in the digital technologies [4, 5, 7–10], with the exception of [11], implies a way to reduce the carry. Summation in a positional system without carry was first demonstrated using the Galois field codes that are obtained by using theoretical-numerical transformations over the Galois field and the initial non-zero code fragment.

However, the codes for the operation of summation without carry in a positional system can be obtained through the initial, zero and each of non-zero blocks of complete combinatorial system with repetition (Table 1). Generated codes for the operation of summation are categorized only as binary and in this case they have recurrent properties.

Assume P(2, n) is the class of combinatorial systems with initial blocks of complete combinatorial system with repetition (Table 1). Then P(2, 1111) – an instance of the class P(2, n) is a combinatorial system with a 4-bit initial block – 1111. P(2, 0000) – an instance of the class P(2, n) is a combinatorial system with a 4-bit initial block – 0000. P(2, b_i) – an instance of the class P(2, n) is a combinatorial system with a 4-bit initial block – b_i Table 1.

Unlike [2, 3], in this paper the Galois field codes, codes XAND are determined by the respective instances P(2, b_i) of the class P(2, n) by way of their selection on the ring structure using the original block b_i Table 1. This means that the principle of the construction of the system of binary codes by its code-beginning (block-beginning) is located within the range of the complete combinatorial system with repetition (Table 1). Therefore, all the blocks in Table 1 are equal in the principle of the synthesis of the corresponding system of binary codes P(2, b_i). In its turn, the chosen system of binary codes (an instance of P(2, b_i) in the class P(2, n)) is equal in its use, among other systems, for example, in arithmetic operations.

Since the use of binary codes for the operation of summation without carry is the task still unsolved, this paper demonstrates a new standard of the synthesis of binary codes, which comes down to that the set system of binary codes (instance of P(2, b_i) of the class P(2, n)) is selected on the ring structure with the help of corresponding initial block of complete combinatorial system with repetition. Similarly selected are the other systems of binary codes, with their blocks-beginnings, that can be applied to carry out the operation of summation without carry, thus expanding the apparatus of the synthesis of recurrent binary codes for their use in digital technologies.

3. The purpose and objectives of the study

The aim of this work is to construct a scheme of a parallel adder of binary codes without inter-digit carries and to determine the quality indicators of such an adder.

To achieve the set goal, the following tasks are to be solved:

– to determine the properties of recursive method of the synthesis of binary codes;

– to establish the validity of the use of any block-beginning of complete combinatorial system with repetition (Table 1) for the synthesis of recurrent binary codes P(2, b_i);

– to obtain an estimate of the complexity of the algorithm of calculation of signals of the sum of a parallel adder of binary codes without carry;

– to compile a protocol of computing of the operation of summation of binary codes without carry, to conduct the test of the synthesized adder to match the results of the operations of summation of binary numbers and the compiled protocol and to specify the range of adding the numbers of the adder of binary codes without carry;

– to perform a comparative performance analysis of the calculations of signals of the sum in the scheme of a parallel adder of binary codes without carry to the scheme of a parallel adder with a parallel way of the CLA carry (Carry Look-ahead Adder).

4. Recursive method of synthesis of binary codes

The method of recursion provides for a synthesis of binary codes with the necessary properties for the operation of summation without carry.

1. Recursion submits the next code (or a recurrent sequence element) by using logical operation on the previous code (element). Thus all n-digit codes are the result of a cyclic shift of the original code fragment with the key $x_j = x_{i-n} \oplus x_{i-1}$, (for the case of a 4-digit code).

2. Compiled in this way, the system of codes must possess the properties of a ring, which gives, in particular, the formation of the initial code of the system at the operation of the shift in the last code by one bit of a recurrent sequence.

3. The set of binary codes of any instance P(2, b_i) in the class P(2, n) with the properties of a ring relative to the operation of summation is the additive group of the instance P(2, b_i) with a ring structure that can be marked as – $P(2, b_i)^+$.

4. The system of codes will have a ring structure only when it contains one of the two combinations of binary numbers: 0000 or 1111 (for the case of 4-digit binary numbers).

5. It follows from the property 4 that, based on complete combinatorial system with repetition (Table 1), the formation of two ring structures, one of which contains the code 0000 (4), is possible

$$0\,0\,0\,0\,1\,0\,1\,0\,0\,1\,1\,0\,1\,1\,1, \quad (4)$$

and the other one contains the code 1111 (5)

$$1\,1\,1\,1\,0\,1\,0\,1\,1\,0\,0\,1\,0\,0\,0. \quad (5)$$

6. In the system of codes with a ring structure, made by using the XOR operation, the code 0000 is missing, in the system of codes with a ring structure, formed by using the XAND operations, the code 1111 is missing (for the case of 4-bit binary codes).

7. The order of alternation of recurrent binary codes in a ring structure is the same for all systems of binary codes (for all instances P(2, b_i) in the class P(2, n)). For example,

for a ring structure (4), the order of alternation of codes is as follows:

...
1000
0000
0001
...,

for a ring structure (5), the order of alternation of codes is as follows:

...
0111
1111
1110
....

8. It follows from the property 6 that the number of recurrent binary codes in the system $P(2, b_i)$ is determined by the number

$$b = 2^n - 1, \quad (6)$$

where n is the digit capacity of a binary code.

9. It follows from the property 8 that the range of numbers that can be presented by the binary recurrent codes (6) consists of

$$x_D + y_D = < 2^n - 2,$$

where n is the digit capacity of a binary code.

For example, for an 8-bit binary without a sign integer, the range of numbers, which can be presented by binary recurrent codes is 0...254. For a 16-bit, without a sign code, the range equals 0...65534.

5. Recurrent binary codes

For binary, such as 4-bit codes, at logical operation XOR, each of 2^n-1 n-digit non-zero code combination of recurrent sequence is the result of cyclic shift of any initial non-zero code fragment that belongs in complete combinatorial system with repetition (Table 1) with the key

$$x_j = x_{i-n} \oplus x_{i-1}.$$

Compiled in this way, the system of codes has a ring structure, which provides, in particular, for the formation of initial system code at the operation of the shift in the last code of the system by one bit of recurrent sequence.

Table 2 presents all 4-bit initial code fragments of complete combinatorial system with repetition (Table 1), with the exception of zero (0000) and corresponding recurrent sequences of the code elements, formed by the key

$$x_j = x_{i-4} \oplus x_{i-1}.$$

Logical operation XAND allows obtaining recurrent binary codes by a cyclic shift, starting at zero (0000) initial fragment [3].

All sequences (Table 2) present 4-bit binary recurrent codes shifted by one bit by to each other, the values of which are given in Tables 3, 4.

Table 2

Initial 4-bit code fragments of complete combinatorial system with repetition and corresponding recurrent sequence of code elements

#	Initial code fragment	Recurrent sequence
0	–	–
1	0001	000111101011001
2	0010	001000111101011
3	0011	001111010110010
4	0100	010001111010110
5	0101	010110010001111
6	0110	011001000111101
7	0111	011110101100100
8	1000	100011110101100
9	1001	100100011110101
10	1010	101011001000111
11	1011	101100100011110
12	1100	110010001111010
13	1101	110101100100011
14	1110	111010110010001
15	1111	111101011001000

Table 3

Recurrent 4-bit binary codes for initial code fragments – 1111, 1110, 0001, 0011, 1101, 1100, 0010, 0111

#	Initial code fragment							
	1111	1110	0001	0011	1101	1100	0010	0111
	Recurrent binary code							
0	1111	1110	0001	0011	1101	1100	0010	0111
1	1110	1101	0011	0111	1010	1001	0100	1111
2	1101	1010	0111	1111	0101	0010	1000	1110
3	1010	0101	1111	1110	1011	0100	0001	1101
4	0101	1011	1110	1101	0110	1000	0011	1010
5	1011	0110	1101	1010	1100	0001	0111	0101
6	0110	1100	1010	0101	1001	0011	1111	1011
7	1100	1001	0101	1011	0010	0111	1110	0110
8	1001	0010	1011	0110	0100	1111	1101	1100
9	0010	0100	0110	1100	1000	1110	1010	1001
10	0100	1000	1100	1001	0001	1101	0101	0010
11	1000	0001	1001	0010	0011	1010	1011	0100
12	0001	0011	0010	0100	0111	0101	0110	1000
13	0011	0111	0100	1000	1111	1011	1100	0001
14	0111	1111	1000	0001	1110	0110	1001	0011

Table 4

Recurrent 4-bit binary codes for initial code fragments – 1011, 1001, 1000, 0101, 0100, 0110, 1010

#	Initial code fragment						
	1011	1001	1000	0101	0100	0110	1010
	Recurrent binary code						
0	1011	1001	1000	0101	0100	0110	1010
1	0110	0010	0001	1011	1000	1100	0101
2	1100	0100	0011	0110	0001	1001	1011
3	1001	1000	0111	1100	0011	0010	0110
4	0010	0001	1111	1001	0111	0100	1100
5	0100	0011	1110	0010	1111	1000	1001
6	1000	0111	1101	0100	1110	0001	0010
7	0001	1111	1010	1000	1101	0011	0100
8	0011	1110	0101	0001	1010	0111	1000
9	0111	1101	1011	0011	0101	1111	0001
10	1111	1010	0110	0111	1011	1110	0011
11	1110	0101	1100	1111	0110	1101	0111
12	1101	1011	1001	1110	1100	1010	1111
13	1010	0110	0010	1101	1001	0101	1110
14	0101	1100	0100	1010	0010	1011	1101

For any given initial code fragment (Table 2), for example, x_1, x_2, x_3, x_4 with the key

$$x_j = x_{i-4} \oplus x_{i-1},$$

all other representations of the bits of corresponding recurrent sequence (Table 2) can be obtained through the initial:

$$x_5=x_1\oplus x_4,$$
$$x_6=x_2\oplus x_5=x_1\oplus x_2\oplus x_4,$$
$$x_7=x_3\oplus x_6=x_1\oplus x_2\oplus x_3\oplus x_4,$$
$$x_8=x_4\oplus x_7=x_1\oplus x_2\oplus x_3,$$
$$x_9=x_5\oplus x_8=x_2\oplus x_3\oplus x_4,$$
$$x_{10}=x_6\oplus x_9=x_1\oplus x_3,$$
$$x_{11}=x_7\oplus x_{10}=x_2\oplus x_4,$$
$$x_{12}=x_8\oplus x_{11}=x_1\oplus x_3\oplus x_4,$$
$$x_{13}=x_9\oplus x_{12}=x_1\oplus x_2,$$
$$x_{14}=x_{10}\oplus x_{13}=x_2\oplus x_3,$$
$$x_{15}=x_{11}\oplus x_{14}=x_3\oplus x_4,$$
$$x_{16}=x_{12}\oplus x_{15}=x_1\oplus x_3\oplus x_4\oplus x_3\oplus x_4=x_1,$$
$$x_{17}=x_{13}\oplus x_{16}=x_1\oplus x_2\oplus x_1=x_2,$$
$$x_{18}=x_{14}\oplus x_{17}=x_2\oplus x_3\oplus x_2=x_3,$$
$$x_{19}=x_{15}\oplus x_{18}=x_2\oplus x_4\oplus x_3=x_4,$$
$$x_{20}=x_{16}\oplus x_{19}=x_1\oplus x_4. \quad (7)$$

From the review of dependencies (7) we see that in the calculations of each bit of the recurrent sequence, starting with x_5, the first four bits participate.

6. Arithmetic operation of adding binary codes without carry

The operation of summation of binary codes A(x) and D(x) is the recursive shift in the selected sequence (Table 2), starting from the initial position of the code A(x) on a number of discrete positions defined by the decimal equivalent of the code D(x). Thus, the implementation of the indicated operation of summation boils down to simultaneous parallel mutually independent formation of each bit of the result of the calculation as the sum of the operation $\oplus$ without having to perform the operations of inter bit carries. The calculation of each result of the operation is carried out in one cycle. The process of calculation is invariant to the digit capacity of a word of data.

To perform the operation of summation, the codes-summands are presented by using dependencies (7).

Example 1. For the initial code fragment 1111, the dependency of the code A(x) – 0101 (4_{10}), code D(x) – 0110 (6_{10}), code of the sum C(x) – 0100 (10_{10}) are displayed in Table 5.

During the operation of summation, the code A(x) (Table 5, a) is exposed to the actions defined by the dependencies of the code D(x) in the recurrent sequence (Table 2), which correspond to the value of the code D(x) (Table 5, b). The dependencies of the code D(x) set a certain programming procedure (vector) over the code A(x) for the calculation of each digit of the sum C(x) (Table 5, c).

The dependencies (7) of the code of the number A(x) – 0101 (4_{10}) of the example 1 are in the 4th line of Table 8, the vector D'(x) is in the 6th line of Table 8 (Table 6).

Table 5

Expressions for the 4-bit codes: a – A(x), b – D(x), c – C(x), presented by dependencies (7)

A(x) – 0101 (4_{10})			D(x) – 0110 (6_{10})			C(x) – 0100 (10_{10})		
x_5	=	$x_1 \oplus x_4$	x_7	=	$x_1 \oplus x_2 \oplus x_3 \oplus x_4$	x_{11}	=	$x_2 \oplus x_4$
x_6	=	$x_1 \oplus x_2 \oplus x_4$	x_8	=	$x_1 \oplus x_2 \oplus x_3$	x_{12}	=	$x_1 \oplus x_3 \oplus x_4$
x_7	=	$x_1 \oplus x_2 \oplus x_3 \oplus x_4$	x_9	=	$x_2 \oplus x_3 \oplus x_4$	x_{13}	=	$x_1 \oplus x_2$
x_8	=	$x_1 \oplus x_2 \oplus x_3$	x_{10}	=	$x_1 \oplus x_3$	x_{14}	=	$x_2 \oplus x_3$
a			b			c		

Table 6

Dependencies of the code of the number A(x) and the vector D'(x) of the example 1 in algebraic representation of Table 8

#	D(x)	D'(x)			
		1st bit	2nd bit	3rd bit	4th bit
4	0101	$b_1 \oplus b_4$	$b_1 \oplus b_2 \oplus b_4$	$b_1 \oplus b_2 \oplus b_3 \oplus b_4$	$b_1 \oplus b_2 \oplus b_3$
6	0110	$b_1 \oplus b_2 \oplus b_3 \oplus b_4$	$b_1 \oplus b_2 \oplus b_3$	$b_2 \oplus b_3 \oplus b_4$	$b_1 \oplus b_3$

According to the program D'(x) (the 6th line of Table 8), for the calculation of the first digit of the sum C(x), all four dependencies of the code A(x) (Table 5) must participate. To calculate the sum C(x) in the second digit, the first three dependencies of the code A(x) (Table 5) must participate. To calculate the sum C(x) in the third digit, the dependencies of the second, third and fourth bits of the code A(x) (Table 5) must participate. To calculate the sum C(x) in the fourth digit, the dependencies of the first and the third bits of the code A(x) (Table 5) must participate.

The program procedure D'(x) over a 4-bit code A(x) can be presented by a table (Table 7). In the first line of Table 7, the dependencies of the code A(x) (Table 5) are recorded to calculate the first digit of the sum C(x) in accordance with program procedure D'(x) (Table 8). In the second line of Table 7, the dependencies of the code A(x) (Table 5) are recorded to calculate the second digit of the sum C(x) in accordance with the program procedure D'(x) (Table 8) and so on.

Table 7

Calculations over the 4-bit code A(x), which are defined by the program procedure D'(x)

–	–	x_5(A)	–	x_6(A)	–	x_7(A)	–	x_8(A)	–	The amount of code
x_7	=	$(x_1 \oplus x_4)$	$\oplus$	$(x_1 \oplus x_2 \oplus x_4)$	$\oplus$	$(x_1 \oplus x_2 \oplus x_3 \oplus x_4)$	$\oplus$	$(x_1 \oplus x_2 \oplus x_3)$	=	$x_2 \oplus x_4$
x_8	=	$(x_1 \oplus x_4)$	$\oplus$	$(x_1 \oplus x_2 \oplus x_4)$	$\oplus$	$(x_1 \oplus x_2 \oplus x_3 \oplus x_4)$	–	–	=	$x_1 \oplus x_3 \oplus x_4$
x_9	=	–	–	$(x_1 \oplus x_2 \oplus x_4)$	$\oplus$	$(x_1 \oplus x_2 \oplus x_3 \oplus x_4)$	$\oplus$	$(x_1 \oplus x_2 \oplus x_3)$	=	$x_1 \oplus x_2$
x_{10}	=	$(x_1 \oplus x_4)$	–	–	$\oplus$	$(x_1 \oplus x_2 \oplus x_3 \oplus x_4)$	–	–	=	$x_2 \oplus x_3$

Indexes at x in the far left column of Table 7 must be replaced with the corresponding indices for the sum C(x) – $x_{11}=x_2\oplus x_4$, $x_{12}=x_1\oplus x_3\oplus x_4$, $x_{13}=x_1\oplus x_2$, $x_{14}= x_2\oplus x_3$.

6. 1. Two general tables of representation of binary codes

Character representation of the expressions of the 4-bit codes a – A(x), b – D(x), c – C(x) of the instances

$P(2, b_i)$ of the class $P(2, n)$ allows, to all possible options of summation of binary codes, presenting the data about the vector of the code D(x) by two general tables (Table 8, 9).

6. 1. 1. General table of binary codes in algebraic representation

Table 8 displays the data on the vector of the code D(x) in algebraic representation by dependencies (7). Table 8 is a general table of representation of codes for all combinatorial systems $P(2, b_i)$.

For the code D(x) – 1111, for example, at the operation of summation with any code A(x), the program procedure (vector) D'(x) over the code A(x) in algebraic representation will take the form:

1^{st} bit 2^{nd} bit 3^{rd} bit 4^{th} bit
D'(x)=> $b_1\ b_2\ b_3\ b_4$

Table 8

Program procedure D'(x) over the 4-bit code A(x) for all combinatorial systems $P(2, b_i)$ in algebraic representation

#	Binary code	Digits of binary codes presented by dependencies (7), in algebraic representation			
–	D(x)	D'(x)			
0	xxxx	b_1	b_2	b_3	b_4
1	xxxx	b_2	b_3	b_4	$b_1 \oplus b_4$
2	xxxx	b_3	b_4	$b_1 \oplus b_4$	$b_1 \oplus b_2 \oplus b_4$
3	xxxx	b_4	$b_1 \oplus b_4$	$b_1 \oplus b_2 \oplus b_4$	$b_1 \oplus b_2 \oplus \oplus b_3 \oplus_4$
4	xxxx	$b_1 \oplus b_4$	$b_1 \oplus b_2 \oplus b_4$	$b_1 \oplus b_2 \oplus b_3 \oplus_4$	$b_1 \oplus b_2 \oplus b_3$
5	xxxx	$b_1 \oplus b_2 \oplus b_4$	$b_1 \oplus b_2 \oplus b_3 \oplus_4$	$b_1 \oplus b_2 \oplus b_3$	$b_2 \oplus b_3 \oplus b_4$
6	xxxx	$b_1 \oplus b_2 \oplus b_3 \oplus_4$	$b_1 \oplus b_2 \oplus b_3$	$b_2 \oplus b_3 \oplus b_4$	$b_1 \oplus b_3$
7	xxxx	$b_1 \oplus b_2 \oplus b_3$	$b_2 \oplus b_3 \oplus b_4$	$b_1 \oplus b_3$	$b_2 \oplus b_4$
8	xxxx	$b_2 \oplus b_3 \oplus b_4$	$b_1 \oplus b_3$	$b_2 \oplus b_4$	$b_1 \oplus b_3 \oplus b_4$
9	xxxx	$b_1 \oplus b_3$	$b_2 \oplus b_4$	$b_1 \oplus b_3 \oplus b_4$	$b_1 \oplus b_2$
10	xxxx	$b_2 \oplus b_4$	$b_1 \oplus b_3 \oplus b_4$	b1⊕b_2	$b_2 \oplus b_3$
11	xxxx	$b_1 \oplus b_3 \oplus b_4$	$b_1 \oplus b_2$	$b_2 \oplus b_3$	$b_3 \oplus b_4$
12	xxxx	$b_1 \oplus b_2$	$b_2 \oplus b_3$	$b_3 \oplus b_4$	b_1
13	xxxx	$b_2 \oplus b_3$	$b_3 \oplus b_4$	b_1	b_2
14	xxxx	$b_3 \oplus b_4$	b_1	b_2	b_3

6. 1. 2. General table of binary codes in bitmap representation

Table 9 displays the data on the vector of the code D(x) by the dependencies (7) using the numeric symbols (bits). Table 9 is a general table of representation of codes for all combinatorial systems $P(2, b_i)$, where the presence of a character in the dependencies (7) is indicated by one, and the absence of a character is indicated by zero.

For the code D(x) – 1111, for example, during the operation of summation with any code A(x), the program procedure (vector) D'(x) over the code A(x) in the bitmap representation will take the form:

1^{st} bit 2^{nd} bit 3^{rd} bit 4^{th} bit
D'(x)=> 1 1 1 1

Table 9

Program procedure D'(x) over the 4-bit code A(x) for all combinatorial systems $P(2, b_i)$ in the bitmap representation

#	Binary code	Digits of binary codes presented by dependencies (7), in the bitmap representation			
		$d_{44}\ d_{34}\ d_{24}\ d_{14}$	$d_{43}\ d_{33}\ d_{23}\ d_{13}$	$d_{42}\ d_{32}\ d_{22}\ d_{12}$	$d_{41}\ d_{31}\ d_{21}\ d_{11}$
–	D(x)	D'(x)			
0	xxxx	1⊕0⊕0⊕0	0⊕1⊕0⊕0	0⊕0⊕1⊕0	0⊕0⊕0⊕1
1	xxxx	0⊕1⊕0⊕0	0⊕0⊕1⊕0	0⊕0⊕0⊕1	1⊕0⊕0⊕1
2	xxxx	0⊕0⊕1⊕0	0⊕0⊕0⊕1	1⊕0⊕0⊕1	1⊕1⊕0⊕1
3	xxxx	0⊕0⊕0⊕1	1⊕0⊕0⊕1	1⊕1⊕0⊕1	1⊕1⊕1⊕1
4	xxxx	1⊕0⊕0⊕1	1⊕1⊕0⊕1	1⊕1⊕1⊕1	1⊕1⊕1⊕0
5	xxxx	1⊕1⊕0⊕1	1⊕1⊕1⊕1	1⊕1⊕1⊕0	0⊕1⊕1⊕1
6	xxxx	1⊕1⊕1⊕1	1⊕1⊕1⊕0	0⊕1⊕1⊕1	1⊕0⊕1⊕0
7	xxxx	1⊕1⊕1⊕0	0⊕1⊕1⊕1	1⊕0⊕1⊕0	0⊕1⊕0⊕1
8	xxxx	0⊕1⊕1⊕1	1⊕0⊕1⊕0	0⊕1⊕0⊕1	1⊕0⊕1⊕1
9	xxxx	1⊕0⊕1⊕0	0⊕1⊕0⊕1	1⊕0⊕1⊕1	1⊕1⊕0⊕0
10	xxxx	0⊕1⊕0⊕1	1⊕0⊕1⊕1	1⊕1⊕0⊕0	0⊕1⊕1⊕0
11	xxxx	1⊕0⊕1⊕1	1⊕1⊕0⊕0	0⊕1⊕1⊕0	0⊕0⊕1⊕1
12	xxxx	1⊕1⊕0⊕0	0⊕1⊕1⊕0	0⊕0⊕1⊕1	1⊕0⊕0⊕0
13	xxxx	0⊕1⊕1⊕0	0⊕0⊕1⊕1	1⊕0⊕0⊕0	0⊕1⊕0⊕0
14	xxxx	0⊕0⊕1⊕1	1⊕0⊕0⊕0	0⊕1⊕0⊕0	0⊕0⊕1⊕0

In a shortened record of the vector D'(x), the sign of the operation is omitted, for example:

–	D(x)	D'(x)			
14	0111	0011	1000	0100	0010

In the end Table 9 will take a compact view.

6. 2. Adding binary codes in bitmap representation without carry

Since arithmetic operations in electronic schemes are carried out by using physical signals, which, in turn, are defined by substitution, in accordance with the bits of binary codes

$$A = \begin{pmatrix} i_1 & i_2 & \dots & i_k \\ \alpha_{i_1} & \alpha_{i_2} & \dots & \alpha_{i_k} \end{pmatrix},$$

we will present binary codes of the numbers A(x), D(x), C(x) in bits, where the presence of a character of the sequence (7) is denoted by one, while the absence of a character is denoted by zero. For example, the 4-bit code $x_1 \oplus x_4$ for any initial code fragment of combinatorial systems $P(2, b_i)$, in the bitmap representation will look like $1 \oplus 0 \oplus 0 \oplus 1$, the 4-bit code $x_1 \oplus x_2 \oplus x_3 \oplus x_4$ in the bitmap representation will take the form $1 \oplus 1 \oplus 1 \oplus 1$.

Example 2. For the initial code fragment 1111 (as for other initial code fragments), the vector D'(x) and the code A(x) from the example 1 can be presented in bits (Table 11).

The program procedure D'(x) over the code A(x) from the example 1 (Table 5) will look like (Table 10).

Table 10

Program procedure (vector) D'(x) over the 4-bit code A(x) – 0101 (4_{10}) from the example 1

	1^{st} bit	2^{nd} bit	3^{rd} bit	4^{th} bit
D'(x)	$1_1 \oplus 1_2 \oplus 1_3 \oplus 1_4$	$1_1 \oplus 1_2 \oplus 1_3 \oplus 0_4$	$0_1 \oplus 1_2 \oplus 1_3 \oplus 1_4$	$1_1 \oplus 0_2 \oplus 1_3 \oplus 0_4$

Table 11

Calculations defined by the program procedure D'(x) (the 6th line of Table 8) over the 4-bit code A(x) from the example 1 (Table 5), presented in bits

–	–	$x_5(A)$	–	$x_6(A)$	–	$x_7(A)$	–	$x_8(A)$	–	Sum of codes
x_7	=	0	⊕	1	⊕	0	⊕	1	=	0
x_8	=	0	⊕	1	⊕	0	⊕	0	=	1
x_9	=	0	⊕	1	⊕	0	⊕	1	=	0
x_{10}	=	0	⊕	0	⊕	0	⊕	0	=	0

The obtained values of bits in the lines of Table 11 x_7=0, x_8=1, x_9=0, x_{10}=0 correspond to the bits of the code of the sum C(x). Indices at x must be replaced with the corresponding indices for the sum C(x) – x_{11}=0, x_{12}=1, x_{13}=0,x_{14}=0.

Dependencies (7) are used to construct the vector D'(x), so in general one can consider missing the character of the vector D'(x), which is denoted by 0_D. The vector D'(x) with regard to the missing characters will look like (Table 12).

Table 12

Vector D'(x) (the 6th line of Table 8) with regard to the missing characters

#	D(x)	D'(x)			
		1st bit	2nd bit	3rd bit	4th bit
4	0101	0	1	0	1
6	0110	$b_1 \oplus b_2 \oplus \oplus b_3 \oplus b_4$	$b_1 \oplus b_2 \oplus \oplus b_3 \oplus 0_D$	$0_D \oplus b_2 \oplus \oplus b_3 \oplus b_4$	$b_1 \oplus 0_D \oplus \oplus b_3 \oplus 0_D$

The calculations in Table 11 will be written down by equations (8), each of which is formed by substituting the bits of the code A(x) in the structure of the corresponding digit of the vector D'(x), given the missing character 0_D. For another option of adding codes it is necessary to perform another substitution of the A(x) code bits in the structure of the corresponding digit of the vector D'(x) and to repeat the considered order of arithmetic operation.

$$x_7=0_A\oplus 1_A\oplus 0_A\oplus 1_A=\mathbf{0},$$

$$x_8=0_A\oplus 1_A\oplus 0_A\oplus 0_D=\mathbf{1},$$

$$x_9=0_D\oplus 1_A\oplus 0_A\oplus 1_A=\mathbf{0},$$

$$x_{10}=0_A\oplus 0_D\oplus 0_A\oplus 0_D=\mathbf{0}. \quad (8)$$

The obtained values of the equations x_7=0, x_8=1, x_9=0, x_{10}=0 correspond to the bits of the code of the sum C(x). Indices at x must be replaced with the corresponding indices for the sum C(x) – x_{11}=0, x_{12}=1, x_{13}=0,x_{14}=0.

Substitution at the level of the scheme requires a process of logical operation, which for the missing characters of the vector D'(x) gives the following options matching the wild-card characters:

$0_A \wedge 0_D$,

$1_A \wedge 0_D$.

The last option of the substitution inverts one into zero (excludes the bit-unit of the code A from logical process), which requires a logical element I for the scheme. Substitution with the present characters of the vector D'(x) gives other variants of the logical substitution of the variables:

$0_A \wedge 1_D$,

$1_A \wedge 1_D$,

that is, zero becomes zero, one becomes one.

The process of the substitution with the specified logic is represented by the equations (9), in which the A code is written down by the initial values of the bits –0101 (4_{10}):

$$x_7=(0_A\wedge 1_D)\oplus(1_A\wedge 1_D)\oplus(0_A\wedge 1_D)\oplus(1_A\wedge 1_D)=\mathbf{0},$$

$$x_8=(0_A\wedge 1_D)\oplus(1_A\wedge 1_D)\oplus(0_A\wedge 1_D)\oplus(1_A\wedge 0_D)=\mathbf{1},$$

$$x_9=(0_A\wedge 0_D)\oplus(1_A\wedge 1_D)\oplus(0_A\wedge 1_D)\oplus(1_A\wedge 1_D)=\mathbf{0},$$

$$x_{10}=(0_A\wedge 1_D)\oplus(1_A\wedge 0_D)\oplus(0_A\wedge 1_D)\oplus(1_A\wedge 0_D)=\mathbf{0}. \quad (9)$$

The obtained values of the equations x7=1, x8=0, x9=1, x10=1 correspond to the bits of the code of the sum C(x). Indices at x must be replaced with the corresponding indices for the sum C(x) – x_{11}=1, x_{12}=0, x_{13}=1, x_{14}=1.

6. 3. Scheme of the adder of binary codes without carry

By the equations (9) a combination scheme of the 4-bit adder of binary codes is synthesized. For the first digit of the code of the sum C(x) x_{11} such a scheme, which is built based on the adder of the Galois field codes [12] is presented in Fig. 1.

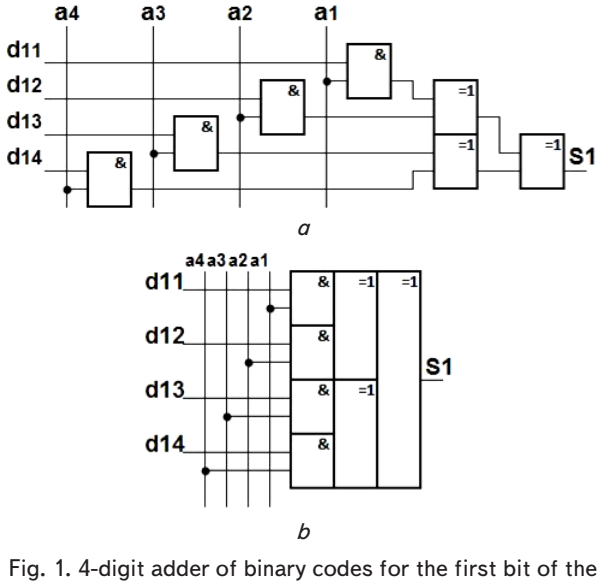

Fig. 1. 4-digit adder of binary codes for the first bit of the sum C(x): *a* – combination scheme of the first digit of the adder; *b* – scheme of the adder of the first bit presented by complex logic

Similarly to the scheme in Fig. 1, the schemes of other digits of the adder of binary codes are built, for the synthesis of which the appropriate logical equations are used similar to (9).

Fig. 2 presents the scheme of the first digit of the 8-bit adder of binary codes without carry on the logical elements AND and XOR.

Fig. 3 presents the scheme of the first digit of the 16-bit adder of binary codes without carry on the logical elements OR and XAND [3].

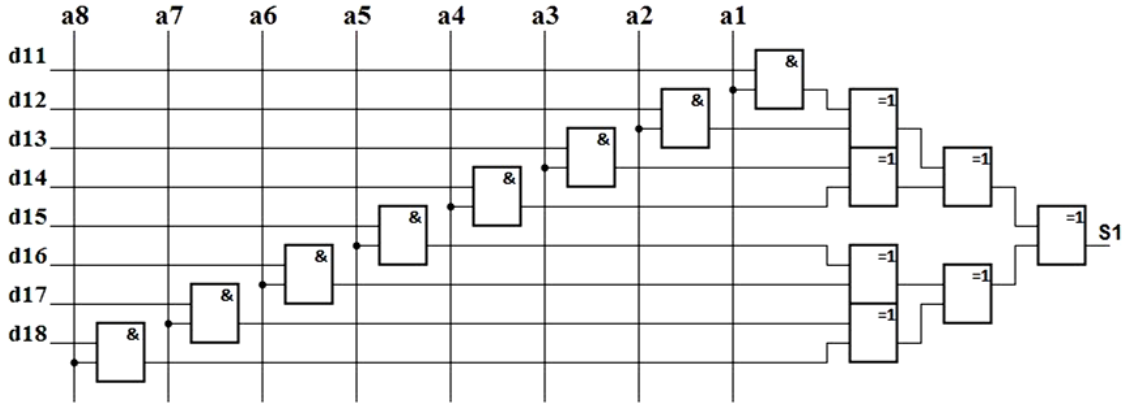

Fig. 2. Scheme of the first digit of the 8-bit adder of binary codes without carry on the logical elements AND and XOR

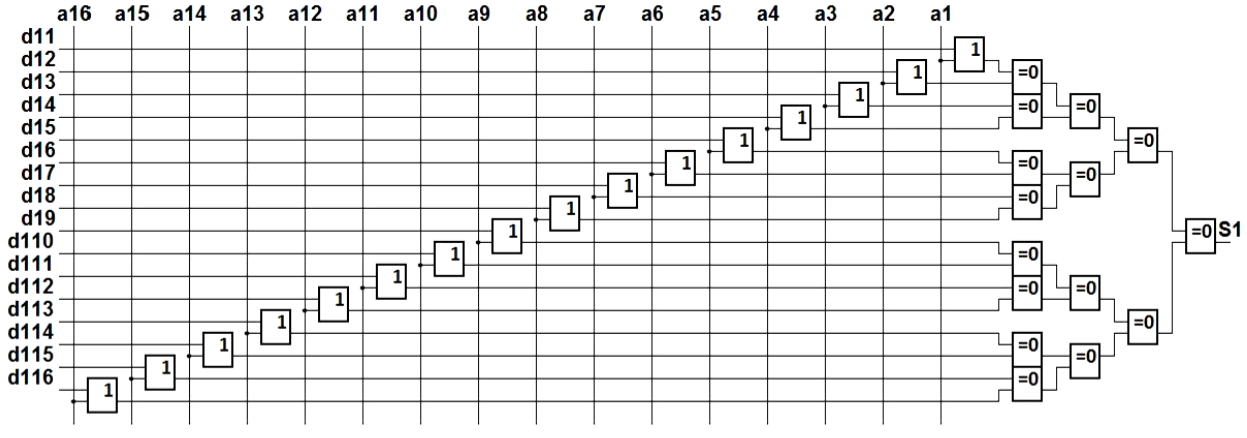

Fig. 3. Scheme of the first digit of the 16-bit adder of binary codes without carry on the logical elements OR and XAND

6. 4. Other examples of adding binary codes without carry

Example 3. For the codes with the original code fragment 1000, A(x) – 1111 (4_{10}), D(x) – 1101 (6_{10}), the code of the sum C(x) – 0110 (10_{10}) (Table 4), to conduct the operation of summation in the bitmap representation.

Table 13

Calculations defined by the program procedure D'(x) (the 6th line of Table 9) over the 4-bit code A(x) – 1111 (4_{10}) of the initial code fragment 1000 represented in bits

–	–	x_5(A)	–	x_6(A)	–	x_7(A)	–	x_8(A)	–	Sum of codes
x_7	=	1	⊕	1	⊕	1	⊕	1	=	0
x_8	=	1	⊕	1	⊕	1	⊕	0	=	1
x_9	=	0	⊕	1	⊕	1	⊕	1	=	1
x_{10}	=	1	⊕	0	⊕	1	⊕	0	=	0

The obtained values of bits in the lines of Table 13 x_7=0, x_8=1, x_9=1, x_{10}=0 correspond to the digits of the code of the sum C(x). Indices at x must be replaced with the corresponding indices for the sum C(x) – x_{11}=0, x_{12}=1, x_{13}=1, x_{14}=0.

Example 4. For the codes with the original code fragment 1011, A(x) – 1100 (2_{10}), D(x) – 0001 (7_{10}), the code of the sum C(x) – 0111 (9_{10}) (Table 4), to conduct the operation of summation in the bitmap representation.

Table 14

Vector D'(x) (7th line of Table 9) with regard to the missing characters

#	D(x)	D'(x)			
		1st bit	2nd bit	3rd bit	4th bit
2	1100	1	1	0	0
7	0001	$1_1 \oplus 1_2 \oplus \oplus 1_3 \oplus 0_4$	$0_1 \oplus 1_2 \oplus \oplus 1_3 \oplus 1_4$	$1_1 \oplus 0_2 \oplus \oplus 1_3 \oplus 0_4$	$0_1 \oplus 1_2 \oplus \oplus 0_3 \oplus 1_4$

Table 15

Calculations defined by the program procedure D'(x) (7th line of Table 8) over the 4-bit code A(x) – 1100 (2_{10}) of the initial code fragment 1011, presented in bits

–	–	x_3(A)	–	x_4(A)	–	x_5(A)	–	x_6(A)	–	Sum of codes
x_8	=	1	⊕	1	⊕	0	⊕	0	=	0
x_9	=	0	⊕	1	⊕	0	⊕	0	=	1
x_{10}	=	1	⊕	0	⊕	0	⊕	0	=	1
x_{11}	=	0	⊕	1	⊕	0	⊕	0	=	1

The obtained values of the digits in the rlines of Table 15 x_8=0, x_9=1, x_{10}=1, x_{11}=1 correspond to the digits of the code of the sum C(x). Indices at x must be replaced with the corresponding indices for the sum C(x) – x_{10}=0, x_{11}=1, x_{12}=1, x_{13}=1.

7. The structure of functional connection of the control output of a decoder with a string of intermediate coefficients d_{ji} of the logical vector D'(x) in the scheme of the adder of binary codes

The calculation of the 4-bit code of the sum C(x) requires submitting to one of the input of the adder 16 (2^k, where k is the digit capacity of the adder) of the bits of intermediate coefficients d_{ji} of the logical vector D'(x). For this, one needs a functional connection of the control output of the decoder with a string of intermediate coefficients d_{ji} of the vector D'(x) belonging to the code D(x). One of the devices that can provide such a functional connection is a memory device. Another solution might be a multiplexer, which, however, gives a linear total complexity of computing in the adder's scheme.

Functional connection of the adder with a string of intermediate coefficients d_{ji} of the logical vector D'(x) uses the decryption of the k-digit code D(x) into a 2^k-digit unitary code, in which one determines current vector D'(x) in the line of Table 9.

For the operation of summation of binary codes, controlling output of the decoder (one in unitary code) connects with the corresponding line in Table 9, after which all 2^k bits of the line are submitted onto 2^k inputs d_{11}, d_{12}, d_{13}, d_{14} ... d_{41}, d_{42}, d_{43}, d_{44} of the adder (Fig. 4).

The structure of the functional connection of the adder with a string of intermediate coefficients d_{ji} of the logical vector D'(x) consists of the circuit of a memory device, a decoder and the scheme of the adder of binary codes of 4 digits. To the inputs d_1, d_2, d_3, d_4 of the structure, the bits of the code D(x) are submitted; to the inputs a_1, a_2, a_3, a_4 the bits of the code A(x) are submitted. The bits of the code of the sum C(x) receive the adder S_1, S_2, S_3,S_4 at the outputs.

With increasing digit capacity of an adder, the principle of constructing the structure of the functional connection of the control output of the decoder with a string of intermediate coefficients d_{ji} of the logical vector D'(x) does not change.

8. Computation protocol of the 4-bit adder of binary codes

The range of adding numbers of an adder of binary codes without carry is:

$$x_D+y_D=<2^n-2,$$

where n is the digit capacity of a number. A number of options to add a multi digit parallel adder of binary codes without carry is:

$$b=\sum_{k=0}^{2^n-1}2^n-k-1$$

or

$$b=\sum_{k=1}^{2^n-1}k,$$

where n is the digit capacity of a number.

Having computed the values b, we determine the number of strings of the computation protocol of the 4-bit parallel adder of binary codes without carry, which is 120 lines (Table 16).

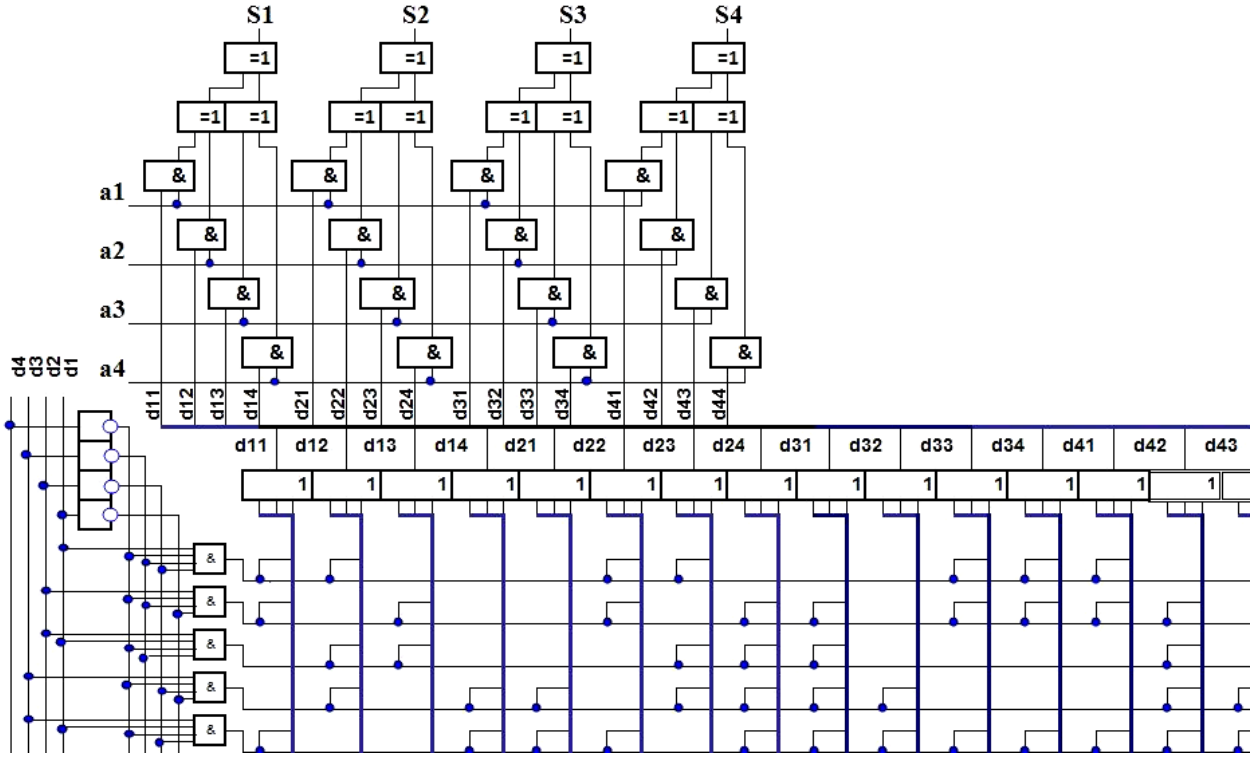

Fig. 4. Structure of the functional connection of the control output of the decoder with a string of intermediate coefficients d_{ji} of the logical vector D'(x) using the multiplexer in the scheme of the 4-bit adder of binary codes

Table 16

Computation protocol of the 4-bit adder of binary codes without carry for the initial code fragment — 1111

#	input								output			
	a_1	a_2	a_3	a_4	d_1	d_2	d_3	d_4	g_1	g_2	g_3	g_4
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
2	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**0**
3	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**1**
4	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**
5	**1**	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**
6	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**1**
7	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**0**
8	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**0**
9	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**
10	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**0**
11	**1**	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**0**
12	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**0**
13	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**1**
14	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**
15	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**1**	**1**
16	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**
17	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**1**
18	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0**
19	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**1**
20	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**
21	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**0**
22	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**0**
23	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**1**
24	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**
25	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**
26	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**
27	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
28	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**
29	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**1**
30	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**
31	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**0**
32	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**1**
33	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**
34	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**
35	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**0**	**0**
36	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**1**
37	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**0**
38	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**
39	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**0**
40	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**1**
41	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**
42	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**1**
43	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**0**
44	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**
45	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**1**
46	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**1**	**0**
47	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**0**
48	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**1**
49	**1**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**
50	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**
51	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**
52	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**
53	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**1**
54	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**1**
55	**0**	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**0**	**1**
56	**0**	**1**	**0**	**1**	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**1**
57	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**0**
58	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**0**
59	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**1**

Ccontinuation of Table 16

1	2	3	4	5	6	7	8	9	10	11	12	13
60	0	1	0	1	1	0	1	1	0	0	1	0
61	0	1	0	1	0	1	1	0	0	1	0	0
62	0	1	0	1	1	1	0	0	1	0	0	0
63	0	1	0	1	1	0	0	1	0	0	0	1
64	0	1	0	1	0	0	1	0	0	0	1	1
65	0	1	0	1	0	1	0	0	0	1	1	1
66	1	0	1	1	1	1	1	1	1	0	1	1
67	1	0	1	1	1	1	1	0	0	1	1	0
68	1	0	1	1	1	1	0	1	1	1	0	0
69	1	0	1	1	1	0	1	0	1	0	0	1
70	1	0	1	1	0	1	0	1	0	0	1	0
71	1	0	1	1	1	0	1	1	0	1	0	0
72	1	0	1	1	0	1	1	0	1	0	0	0
73	1	0	1	1	1	1	0	0	0	0	0	1
74	1	0	1	1	1	0	0	1	0	0	1	1
75	1	0	1	1	0	0	1	0	0	1	1	1
76	0	1	1	0	1	1	1	1	0	1	1	0
77	0	1	1	0	1	1	1	0	1	1	0	0
78	0	1	1	0	1	1	0	1	1	0	0	1
79	0	1	1	0	1	0	1	0	0	0	1	0
80	0	1	1	0	0	1	0	1	0	1	0	0
81	0	1	1	0	1	0	1	1	1	0	0	0
82	0	1	1	0	0	1	1	0	0	0	0	1
83	0	1	1	0	1	1	0	0	0	0	1	1
84	0	1	1	0	1	0	0	1	0	1	1	1
85	1	1	0	0	1	1	1	1	1	1	0	0
86	1	1	0	0	1	1	1	0	1	0	0	1
87	1	1	0	0	1	1	0	1	0	0	1	0
88	1	1	0	0	1	0	1	0	0	1	0	0
89	1	1	0	0	0	1	0	1	1	0	0	0
90	1	1	0	0	1	0	1	1	0	0	0	1
91	1	1	0	0	0	1	1	0	0	0	1	1
92	1	1	0	0	1	1	0	0	0	1	1	1
93	1	0	0	1	1	1	1	1	1	0	0	1
94	1	0	0	1	1	1	1	0	0	0	1	0
95	1	0	0	1	1	1	0	1	0	1	0	0
96	1	0	0	1	1	0	1	0	1	0	0	0
97	1	0	0	1	0	1	0	1	0	0	0	1
98	1	0	0	1	1	0	1	1	0	0	1	1
99	1	0	0	1	0	1	1	0	0	1	1	1
100	0	0	1	0	1	1	1	1	0	0	1	0
101	0	0	1	0	1	1	1	0	0	1	0	0
102	0	0	1	0	1	1	0	1	1	0	0	0
103	0	0	1	0	1	0	1	0	0	0	0	1
104	0	0	1	0	0	1	0	1	0	0	1	1
105	0	0	1	0	1	0	1	1	0	1	1	1
106	0	1	0	0	1	1	1	1	0	1	0	0
107	0	1	0	0	1	1	1	0	1	0	0	0
108	0	1	0	0	1	1	0	1	0	0	0	1
109	0	1	0	0	1	0	1	0	0	0	1	1
110	0	1	0	0	0	1	0	1	0	1	1	1
111	1	0	0	0	1	1	1	1	1	0	0	0
112	1	0	0	0	1	1	1	0	0	0	0	1
113	1	0	0	0	1	1	0	1	0	0	1	1
114	1	0	0	0	1	0	1	0	0	1	1	1
115	0	0	0	1	1	1	1	1	0	0	0	1
116	0	0	0	1	1	1	1	0	0	0	1	1
117	0	0	0	1	1	1	0	1	0	1	1	1
118	0	0	1	1	1	1	1	1	0	0	1	1
119	0	0	1	1	1	1	1	0	0	1	1	1
120	0	1	1	1	1	1	1	1	0	1	1	1

The logic of the work of the adder of binary codes (Fig. 4) corresponds to the computation protocol of the adder for the initial code fragment – 1111 (Table 16). For other initial code fragments, the codes in the computation protocol will have different location, corresponding to the initial block of the chosen system of binary codes.

9. The complexity of the algorithm of calculation of the adder of binary codes without carry

The schemes in Fig. 2, 3 represent a structure that implements multi operand summation [13, 14], when at the same time the neighboring pairs of terms are added, and then their sums (Table 17).

Table 17

The algorithm for pairing ($n=2^3=8$)

Steps	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
1	e_1+e_2		e_3+e_4		e_5+e_6		e_7+e_8	
2	$e_1+e_2+e_3+e_4$				$e_5+e_6+e_7+e_8$			
3	$e_1+e_2+e_3+e_4+e_5+e_6+e_7+e_8$							

If $n=2^k$, where n is the number of terms, then the algorithm for pairing consists of k steps (cycles): the first step includes n/2 addition, the second – n/4, ..., the last –one addition. The number of steps k is determined by the formula:

$$k = \log_2 n. \quad (10)$$

This variant of multi operand addition is implemented by a cascade scheme ("pyramid") [13–16], and it has logarithmic complexity.

Assume as one computing step the calculation on one logical element, in the serial connection of the elements of the scheme. Given the fact that the logical XOR elements in the scheme in Fig. 2, 3 are connected by a cascade scheme, the complexity of the algorithm of calculation of signals of the sum will look like

$$O(\log_2 n+1), \quad (11)$$

where n is the digit capacity of binary codes, which equals the number of terms in the cascade scheme of the summation. In the (11), $\log_2 n$ reflects a cascade connection of XOR logic elements; one displays the logical item I, included sequentially.

The adder of binary codes (Fig. 4) includes a decoder, a memory device and the circuit of summing of the coefficients d_{ji} with bits of the code of the number A.

The connection of multi-pass logic elements AND (Fig. 5) of the decoder can also be organized according to the cascade scheme.

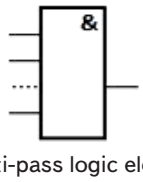

Fig. 5. Multi-pass logic element AND

Then the complexity of calculating of the decoder can be presented by the estimation (11), in which one reflects the Invertor, connected in series (Fig. 4). The estimation of the total complexity of calculation by logical elements XOR and the decoder will look like

$$O(\log_2 n+1+\log_2 n+1)=O(2\log_2 n+2). \quad (12)$$

For quick selection of bits of intermediate coefficients d_{ji} of the logical vector D'(x) from the string in Table 9, one requires a device of constant memory, the cells of which will store the values of the bits of the logical vector D'(x) after their recording. These requirements are met by, for example, static memory. Fig. 6 presents a cell of static memory, in which the key of the cell is modeled by a trigger.

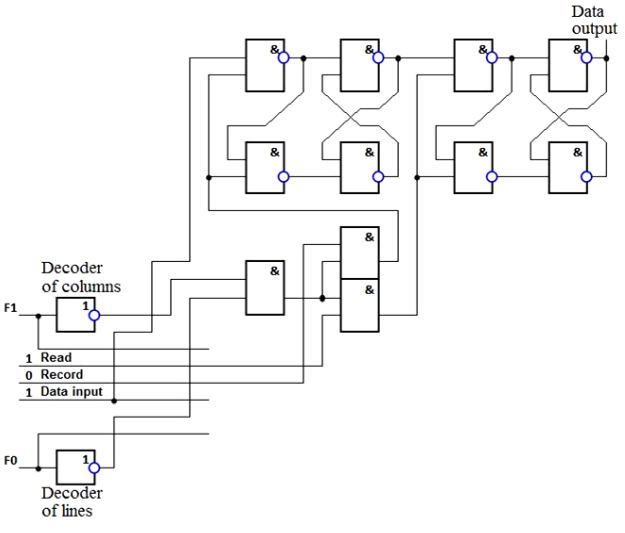

Fig. 6. Cell of static memory

We see in Fig. 6 that the unit of the unitary code of a decoder chooses recorded bit of the vector D'(x) (Table 9) by using the chain of the depth in eight (including Invertor on the elements AND) logical elements, connected in series. This number of logical elements does not depend on the digit capacity of the adder of binary numbers. That is why the estimation (12) will increase by 8 elements more, connected in series and, therefore, will manifest the overall complexity of the algorithm of calculation in the parallel adder of binary codes without carry.

$$O(2\log_2 n+10)=O(\log n). \quad (13)$$

The estimation (13) specifies the logarithmic growth of the complexity of the algorithm of the calculation – doubling the digit capacity of the adder n increases the time of determining correct signals of the sum by a stable value.

10. Comparison of the structures of parallel adder without inter digit carry and parallel adder with a parallel way of carry

Table 18 presents the dynamics of increasing the depth of the scheme of parallel adder with a parallel way of carry CLA (Carry Look-ahead Adder), the synthesis of which is based on the model of calculation of the adder in the form of oriented acyclic graph that represents a binary tree [6]. Since the acyclic graph provides a cascading scheme, then, therefore, the number of computational steps of the graph optimizes

(indicates the minimum sufficient) number of carries for the operation of addition of multi digit binary numbers in the scheme of the parallel adder with a parallel way of carry CLA.

Table 18

Dynamics of increasing the depth of the scheme of parallel adder with a parallel way of carry with increase in the digit capacity of the adder

n	8	16	32	64	128	256	512	1014	2048
CLA	10	12	14	16	18	20	22	24	26

The estimation of the complexity of computing in the scheme of the adder with increasing dynamics of the depth of the scheme of the adder in Table 15, with increasing the digit capacity of the adder looks like

$$O(2\log_2 n+4). \quad (14)$$

Fig. 7 presents the dynamics of increasing the depth of the scheme of the parallel adder without inter digit carry (AWC) and the parallel adder with a parallel way of carry CLA.

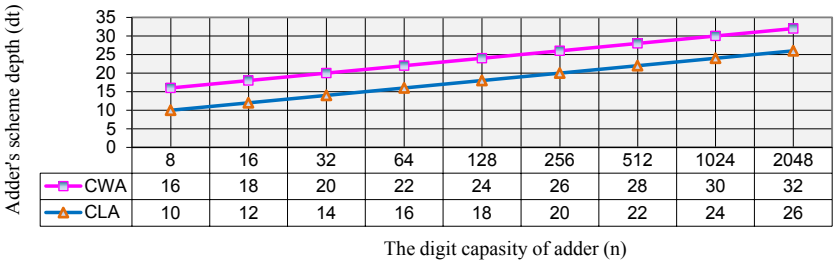

Fig. 7. Dynamics of increasing the depth of the scheme of the parallel adder without inter digit carry (AWC) and parallel adder with a parallel way of carry CLA

Given the Fig. 7, we see that the complexity of the algorithm of the calculation of both adders obeys the logarithmic law.

11. Discussion of the results of the study of the operation of summation without inter digit carry for binary codes

The studies of this work demonstrate that:

1. The codes known in the literature for the operation of summation – for example, Galois field codes [2, 4], XAND codes [3], are determined by the combinatorial systems with initial blocks of complete combinatorial system with repetition P(2, n) (Table 1). In its turn, the combinatorial systems (instances $P(2, b_i)$ of the class P(2, n)) are the systems of binary codes and therefore belong in the same object. Thus, the only basis of the specified binary codes indicates the usefulness of their classification generalization, within the range of operation of summation, on the basis of a single criterion – an object of binary codes. And appropriateness here is the necessity, so the usefulness of this study lies in the fact that they predetermine generalization of classification of binary codes, in order to simplify the structure of the subject area and to increase the diversity of binary codes, in particular, for arithmetic operations with binary numbers.

2. The data about the vector of the code D(x) that are presented by dependencies (7) in algebraic or bitmap view are general representation of binary codes for all combinatorial systems $P(2, b_i)$ that allows changing the system of binary codes using a single universal table of the data on the vector of the code D(x).

Thus, the studies of this work may become a component of the technology of designing electronic computing systems, because:

– they expand the apparatus of obtaining recurrent binary codes for their application in the information technology;

– they provide a possibility to control the selection of the code at the stage of designing a computing device;

– they help predict the impact of the implementation of the selected code in the solution of problems of the information systems;

– they minimize hardware costs associated with the selection of the system of binary code for the calculation.

Reduction of thesaurus of parallel adder of binary codes without inter digit carry

An instance $P(2, b_i)$ of the class P(2, n) is selected on a ring structure using original block b_i of complete combinatorial system with repetition (Table 1). This means that the principle of setting up the system of binary codes with its code-beginning is within the range of complete combinatorial system with repetition (Table 1). Since the location of the principle of obtaining binary codes is defined, the thesaurus of parallel adder of binary codes without carry is necessary to rewrite (Table 19).

We see from Table 17 that the number of concepts of thesaurus of the adder of binary codes is less in comparison with the number of concepts of thesaurus of the adder of the Galois field codes, which, however, does not affect the quality of the construction of the adder of binary codes without carry and reliability of the computational results in this adder.

Table 19

Comparison of the thesaurus of the Galois field codes and the adder of binary codes

#	Thesaurus of the Galois field codes adder	Thesaurus of the adder of binary codes
1	Galois field	combinatorial system
2	irreducible polynomial	–
3	generating vector	–
4	initial code fragment	initial code
5	key	key
6	recursion	recursion
7	logic operation XOR, XAND	logic operation XOR, XAND

The prospect of further review of the operation of summation of binary codes without carry is to use it in other digital technologies, in particular, in the methods of cryptography.

12. Conclusions

1. It was established that the properties of recursive method of the synthesis of binary codes allow focusing the principle of building codes in the range of complete combinatorial system with repetition, which ensures reduction of the thesaurus of the parallel adder of binary codes without carry.

2. It was found that the system of binary codes, formed by means of any initial code of complete combinatorial system with

repetition, has a ring structure, which allows using any system of binary codes in the operation of adding codes without carry.

3. We discovered that the calculation of the signals of the sum in the scheme of a parallel adder of binary codes without carry is performed by the script of the algorithm of pairing. Thus, the complexity of the algorithm of calculation of the signals of the sum of a parallel adder of binary codes without carry is O(log n) and it is logarithmic – the time of algorithm realization increases by the logarithmic law with the increase in the digit capacity of numbers n.

4. It was established that the logic of the work of the adder of binary codes without carry corresponds to the computation protocol of the parallel adder without carry. A number of options of adding b of multi digit parallel adder of binary codes without carry is $b=\sum_{k=1}^{2^n-1} k$, where n is the digit capacity of a number.

5. It was established that the range of adding of numbers of the adder of binary codes without carry is:

$$x_D+y_D=<2^n-2,$$

where n is the digit capacity of a number.

6. It was established that the productivity of computing of signals of the sum by the parallel adder of binary codes without carry and by the parallel adder with a parallel way of carry CLA (Carry Look-ahead Adder) is approximately the same. Thus, the complexity of the algorithm of calculation of signals of the sum and the carry of the CLA adder is also subject to the logarithmic law. And since the adders of binary codes have no hardware costs for the carries between the digits, obvious is reduction in energy consumption, decrease in the heat release by a computing device (integrated circuit) based on such adders.

References

1. Nikolaichuk, Y. M. Teoriya djerel informatsii [Text]: monograph / Y. M. Nikolaichuk. – 2-nd ed., cor. – Ternopil: TzOv Terno-Graf, 2010. – 534 p.
2. Nikolaichuk, Y. M. Theoretical foundations and principles of arithmetic logic unit vertically through information technology [Text] / Y. M. Nikolaichuk, O. M. Zastavna, P. V. Gumen // News of Khmelnytsky national university. – 2012. – Issue 2. – P. 190–196. – Available at: http://www.nbuv.gov.ua/old_jrn/natural/Vchnu_tekh/2012_2/49nic.pdf
3. Solomko, M. Parallel adder carry no transfer in logic elements XAND [Text] / M. Solomko, B. Krulikovskyi, Y. M. Nikolaichuk // Proceedings of the National University «Lviv Polytechnic» Computer systems and networks. – 2015. – Issue 830. – P. 145–158. – Available at: http://ena.lp.edu.ua:8080/xmlui/bitstream/handle/ntb/32480/21-145-158.pdf?sequence=4&isAllowed=y
4. Gopinath, B. Design and Implementation of High Speed Carry Select Adder [Text] / B. Gopinath, N. Sangeetha, S. Jenifer nancy, T. Umarani // International Journal of Engineering Research & Technology (IJERT). – 2015. – Vol. 4, Issue 02. – P. 419–422. – Available at: https://zenodo.org/record/33085/files/Design_and_Implementation_of_High_Speed_Carry_Select_Adder.pdf
5. Deepthi, E. Performance Analysis of a 64-bit signed Multiplier with a Carry Select Adder Using VHDL [Text] / E. Deepthi, V. M. Rani, K. Manasa // IJCSNS International Journal of Computer Science and Network Security. – 2015. – Vol. 15, Issue 11. – P. 91–94. – Available at: http://paper.ijcsns.org/07_book/201511/20151118.pdf
6. Solomko, M. Study of carry optimization while adding binary numbers in the rademacher number-theoretic basis [Text] / M. Solomko, B. Krulikovskyi // Eastern-European Journal of Enterprise Technologies. – 2016. – Vol. 3, Issue 4 (81). – P. 56–63. doi: 10.15587/1729-4061.2016.70355
7. Maity, S. Design and Implementation of Low-Power High-Performance Carry Skip Adder [Text] / S. Maity, B. Prasad De, A. Kr. Singh // International Journal of Engineering and Advanced Technology (IJEAT). – 2012. – Vol. 1, Issue 4. – P. 212–218. – Availabnle at: http://202.120.43.103/Downloads4/20150616101558803.pdf
8. Singh, R. P. P. Performance Analysis of 32-Bit Array Multiplier with a Carry Save Adder and with a Carry-Look-Ahead Adder [Text] / R. P. P. Singh, P. Kumar, B. Singh // International Journal of Recent Trends in Engineering. – 2009. – Vol. 2, Issue 6. – P. 83–86. – Available at: http://searchdl.org/public/journals/2009/IJRTET/2/6/307.pdf
9. Sajid, A. Design and Implementation of Low Power 8-bit Carry-look Ahead Adder Using Static CMOS Logic and Adiabatic Logic [Text] / A. Sajid, A. Nafees, S. Rahman // International Journal of Information Technology and Computer Science. – 2013. – Vol. 5, Issue 11. – P. 78–92. doi: 10.5815/ijitcs.2013.11.09
10. Srinivasa Rao, N. Serial Adder using Reversible Gates [Text] / N. Srinivasa Rao, P. Satyanarayana // International Journal of Advanced Research in Computer and Communication Engineering. – 2015. – Vol. 4, Issue 5. – P. 498–501. – Available at: http://www.ijarcce.com/upload/2015/may-15/IJARCCE%20105.pdf
11. Joshi, D. D. Design and Implementation of 16-bit Ripple Carry Adder for Low Power in 45nm CMOS Technology [Text] / D. D. Joshi, J. K. Singh // International Journal of Emerging Technology and Advanced Engineering. – 2014. – Vol. 4, Issue 1. – P. 216–220.
12. Nykolajchuk, Ja. M. Kody polja Galua: teorija ta zastosuvannja [Text]: monografija / Ja. M. Nykolajchuk. – Ternopil': TzOV Terno-Graf, 2012. – 576 p.
13. Martynjuk, T. B. Rekursyvni algorytmy bagatooperandnoi' obrobky informacii' [Text]: monografija / T. B. Martynjuk. – Vinnycja: «UNIVERSUM-Vinnycja», 2000. – 216 p.
14. Martynjuk, T. B. Metody ta zasoby paralel'nyh peretvoren' vektornyh masyviv danyh [Text]: monografija / T. B. Martynjuk, V. V. Homjuk. – Vinnycja: «UNIVERSUM-Vinnycja», 2005. – 202 p.
15. Hamaiun, V. P. On the development of computational structures mnogooperandnyh [Text] / V. P. Hamaiun // Control systems and machines. – 1990. – Vol. 4. – P. 31–33.
16. Hamaiun, V. P. Theoretical bases, algorithms and structures in operational processing [Text]: author. dis. ... dr. tehn. Sciences / V. P. Hamaiun; National Academy of Sciences of Ukraine. Institute of Cybernetics named after. V. M. Hlushkova. – Kiev, 1999. – 33 p.