Проведеними дослідженнями встановлена перспектива збільшення продуктивності обчислювальних компонентів, зокрема комбінаційних суматорів, на основі використання принципів обчислення цифрових сигналів ациклічної моделі.

Застосування ациклічної моделі розраховано на:

– процес послідовного (для молодших розрядів схеми суматора) та паралельного (для решти розрядів) обчислення сигналів суми і перенесення. Завдяки зазначеному підходу стає можливим, у підсумку, зменшити складність апаратної частини пристрою та не збільшити глибину схеми;

– встановлення оптимального числа обчислювальних кроків.

Експериментально доведено припущення про те, що число обчислювальних кроків орієнтованого ациклічного графа з двома логічними операціями AND і XOR визначає оптимальне число перенесень у схемі n-bit паралельного суматора бінарних кодів. Зокрема, це підтверджується наявністю 8-bit паралельного ациклічного суматора з глибиною схеми 8 типових 2-входових логічних елементів. Зв'язок між числом обчислювальних кроків ациклічного графа і числом перенесень одиниці до старшого розряду спричиняє процес співставлення структури суматора з відповідним ациклічним графом. Метою зазначеного співставлення є встановлення мінімально достатнього числа перенесень для операції додавання бінарних кодів у схемі паралельного суматора з паралельним способом перенесення.

Використання ациклічної моделі вигідніше у порівнянні з аналогами за такими чинниками:

– меншою вартістю розробки, оскільки ациклічна модель визначає простішу структуру суматора;

– наявністю критерію оптимізації – число обчислювальних кроків ациклічного графа вказує на мінімально достатнє число перенесень одиниці до старшого розряду.

Завдяки цьому забезпечується можливість отримання оптимальних значень показників складності структури та глибини схеми суматора. У порівнянні з аналогами відомих структур 8-bit префіксних суматорів це забезпечує збільшення показника якості 8-bit ациклічних суматорів, наприклад, за енергоспоживанням, площею чипа, у залежності від обраної структури, на 14–31 %.

Є підстави стверджувати про можливість збільшення продуктивності обчислювальних компонентів, зокрема суматорів бінарних кодів, шляхом використання принципів обчислення цифрових сигналів ациклічної моделі

Ключові слова: ациклічна модель додавання бінарних кодів, префіксна модель, Ling Adder, Kogge-Stone Adder, Han-Carlson Adder

UDC 681.325
DOI: 10.15587/1729-4061.2019.157150

REDUCTION AND OPTIMAL PERFORMANCE OF ACYCLIC ADDERS OF BINARY CODES

M. Solomko
PhD, Associate Professor*
E-mail: doctrinas@ukr.net
P. Tadeyev
PhD, Doctor of Pedagogical Sciences, Professor
Department of Higher Mathematics**
E-mail: ptadeyev@gmail.com
Ya. Zubyk
Senior Lecturer
Department of Applied Mathematics**
E-mail: j.j.zubyk@nuwm.edu.ua
O. Hladka
PhD, Associate Professor*
E-mail: o.m.hladka@nuwm.edu.ua
*Department of Computer Science**
**National University of Water and Environmental Engineering
Soborna str., 11, Rivne, Ukraine, 33028

1. Introduction

Efficiency of addition of binary codes is greatly dependent on the adder design and the methodology of computation of the sum and carry signals.

Binary addition is the main arithmetic operation in the systems of very large-scale integration (VLSI) circuits. Binary adder is one of the most important elements in processor chips, ALU, counters, memory addressing methods or as a part of filters, e. g. DSP grid filter, etc. The adder structure with series carry is one of the first and most fundamental structures for performing binary addition operations. Its speed depends on the number of input operands, hence, signal delay increases with increase in their number. Parallel prefix adders (PPA) [1–4] provide better performance compared to the adders with a series carry. Besides, any reduction in delay directly concerns the increase in bandwidth [5].

In the nanometer range, development of an addition algorithm with a small area occupied by the chip, low power consumption and high productivity in its realization is urgent for today.

The mathematical apparatus of directed acyclic graphs (Fig. 1, 2) makes it possible to unambiguously obtain values of the sum and carry signals in one stage of digital signal processing [6], so it is capable of effectively replacement of the three-stage prefix model of computation.

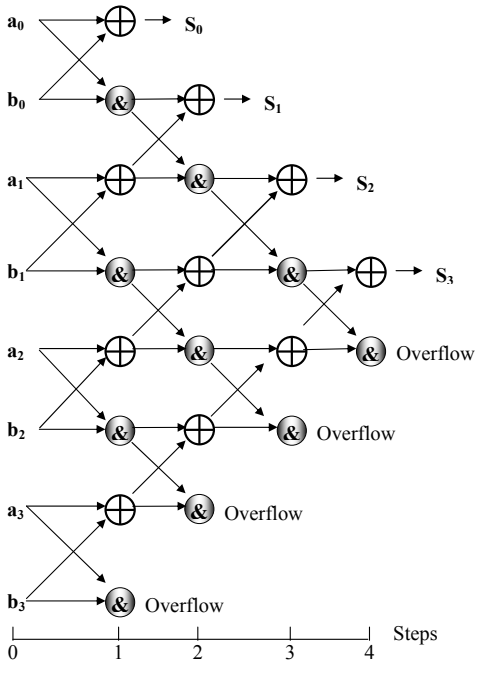

Fig. 1. Directed acyclic graph. Model of a computing circuit of a 4-bit parallel acyclic adder using the parallel carry method

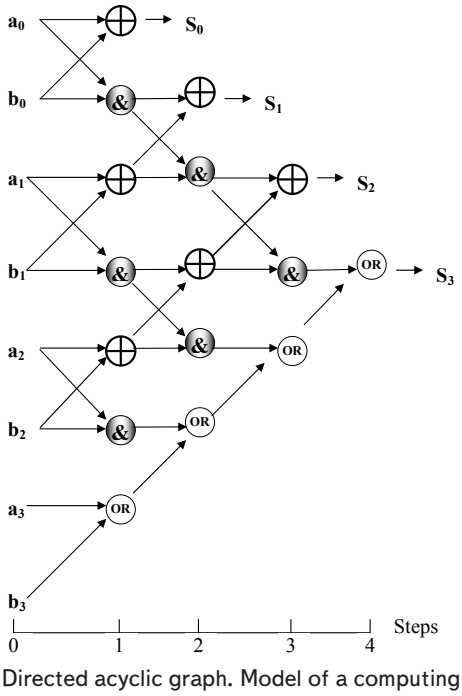

Fig. 2. Directed acyclic graph. Model of a computing circuit of a parallel 4-bit acyclic adder with OR logic elements in the last digit

Arithmetic operations are realized by means of gate circuits with functional elements in the bases consisting of the logical algebra functions. Speed of digital devices, their reliability and power consumption depend on the adder structure. In this connection, minimization of complexity and depth of the logic circuits is one of the central and practically important problems in this theory arising when digital devices are designed.

Processor evolution is the result of tireless optimization, so the studies directed, in particular, at improvement of the following factors are still urgent:

– manufacturing technologies;
– structural implementation;
– speed and power consumption;
– cost of digital devices.

2. Literature review and problem statement

Configuration of computing structures in the field programmable gate arrays (FPGA) in which parallel-prefix adders have better performance was presented in [7]. Parallel expansion of the computing process is a fundamental operation in modern digital circuits and is vital in most computer technologies including ALU units, microchips and in DSP development. In this regard, the Kogge-Stone Adder (KSA) study as well as additional studies of Ripple Carrier Adder, Carry Look Adder and Carry Select Adder were conducted [7]. It is noted that the Kogge-Stone adder is the fastest among the parallel-prefix adders, however, this adder has a high complexity and a huge number of connecting wires. Study [7] has demonstrated signs of improvement of the computational process by means of reverse logic gates (RLG). Reverse circuits that control data through bit splitting in contrast to bit drop will soon offer a physically possible basic approach to continue computing productivity buildup.

Models of parallel prefix adders developed with the Tanner tool for 130 nm technology were presented in [9]. In a nanometer range, it is important to develop addition algorithms ensuring small chip area, low power consumption and high performance. Parallel prefix adders are suitable for VLSI realization due to their simple logical structure and regular connections between the groups of logic elements. Each prefix can be defined in terms of logical level of sweeping and connection of tracks. Comparative analysis of 8-bit Kogge-Stone and Han-Carlson parallel adders has shown that in terms of cost or area and power, the Han-Carlson adder is better choice between these PPAs. The Kogge-Stone adder is better in terms of signal lag. Thus, the Kogge-Stone adder is better for quick addition than the Han-Carlson adder. However, it is advisable to use the Han-Carlson structure for better use of space and lower power consumption. A similar benchmarking can also be done for 16-, 24- and 32-bit PPAs.

Problems of implementation of the high-speed VLSI style in nanoscale technologies where working voltages of transistors are potentially subject to changes under the influence of environment are discussed in [10]. In particular, to augment performance of the DSP processor, a high-speed Kogge-Stone parallel prefix was developed with the help of Xilinx ISE. KSA is a parallel prefix adder, a form of a tracking management unit. This is the fastest adder used in the digital industry for high-performance arithmetic circuits. Fast computing process in the KSA is realized in parallel due to the bigger chip area. To reduce power supply voltage, the author's methods were used. Study [10] has presented a new architectural system for reducing signal delays and occurrence of computational errors through testing the already processed signals.

In order to reduce complexity of the Kogge-Stone adder, so-called almost true adder (variable-delay adder) was proposed in [11]. The variable-delay adder based on the Han-Carlson parallel-prefix topology uses speculation: the true arithmetic function is replaced by approximated values. It is faster and gives correct results for most cases, however,

not always. An error detection net is used. Approximated adder is complemented by an error detection net which confirms an error signal when speculation fails. Speculative variable-delay adders reduce average delay compared to the conventional architectures. A number of speculative variable-delay adders were synthesized with the help of Xilinx 14.3 for various lengths of operands using topology of Han-Carlson and Kogge-Stone. The results obtained indicate that the proposed variable delay in the Han-Carlson adder is used in high-speed applications.

Despite the fact that the Kogger-Stone and Han-Carlson adders are more or less effective, they cannot be used for inputs with higher bit numbers. As it is said, they use more space when the number of input bits increases. Besides, power consumption increases. Thereupon, a parallel prefix procedure applied in development of effective adders in which computation results are determined by one cycle of a synchronous pulse was presented in [12]. The overall chip area and overall delay are reduced without compromising parameters such as performance and power consumption. The developed adders use the Quantum-dot Cellular Automata (QCA) techniques intensively applied for further improvement. Various synchronization circuits are used to observe the adder operation.

Two different approaches to the choice of the adder structure to achieve a minimum delay and reduce chip area were considered in [13]. The study method involved comparison of parameters of the built-in Carry Skip Adder and the Kogge-Stone adder with and without a multiplexer. Removal of the multiplexer reduces area and power consumption. The presented adders were developed with the help of Verilog HDL in VIVADO IDE software environment and implemented on the Zynq board.

A stage circuit as a computation mechanism being a part of the prefix model of the adder using logical structure of three-stage computation of the sum and carry signals was presented in [14]. Note that the acyclic model of computation of the sum and carry signals (Fig. 1, 2) was meant for the logical structure of the adder with series-parallel method of prefix computation and uses the structure of one-stage computation. Thus, prefix and acyclic models are different objects as they have different principles of computation and therefore different capabilities in terms of computing speed, chip area and power consumption.

Design of the adders implemented with the use of memristors was presented in [15] where designs based on memristors for standard adder architectures (ripple carry adder, carry lookahead adder and parallel prefix adder) were explained and chip areas and delays compared. It was noted that the carry lookahead adder has complexity similar to the parallel prefix adders. It was also shown that the Kogge-Stone design has better (among the parallel prefix adders) metric in terms of delays and area.

A new methodology for designing fuzzy adders meant for image processing accelerators was considered in [16]. In particular, the proposed methodology uses the parallel prefix architecture and methods for ensuring low power consumption due to fuzzy adders. Two examples for evaluating the proposed methodology were considered:

1) image blurring filter which uses normal Gaussian distribution;

2) Sobel's operator.

The results were demonstrated on the 45 nm technology where reduction of power consumption varied from 7.7 % to 73.2 % for several image quality levels.

Development and analysis of various types of adders using CMOS technology and transistor logic (DPL) were presented in [17]. Computation of a conditional adder was developed using CMOS, CPL and Dual Transistor Logic (DTL). 16-bit and 32-bit adders, their speed, area and power consumption were compared.

A single-bit adder as a high-speed component of multi-bit adders, matrix multipliers, arithmetic-logical units of microprocessors and components of problem-directed processors of data encryption was presented in the patent [18]. The technical novelty of the patent is an additional introduction of inverse inputs, outputs and AND-NOT logic elements with a multiplex connection by outputs. This makes it possible to maintain high-speed performance when used as a component of structurally more complex multi-bit matrix and multistage computing devices in which operations of adding binary numbers in the number-theoretical basis of Rademacher are provided.

References [7, 9–18] show that the models of parallel prefix computation, in particular, the Kogge-Stone and Han-Carlson architecture are basically output objects for increasing efficiency of processing signals in digital devices. These architectures use parallel computations of the digital signal prefix starting from the low-order bits. This is the actual path (method) of the prefix. However, such a principle of computing the sum and carry signals ultimately results in piling and the digital device complication.

The acyclic model (Fig. 1, 2) is designed for the logic structure of adders with a series-parallel method of digital signal computation. The series carry method is the most fundamental in terms of minimum costs of the hardware part of digital components. Thus, prefix and acyclic models are different objects as they have different principles of computation and therefore different capabilities in terms of performance, chip area and power consumption.

Therefore, there are reasons to believe that the parallel prefix structure, in particular, the model of the Kogge-Stone and Han-Carlson adders is not optimal enough which necessitates studies with the acyclic model of digital signal processing.

3. The aim and objectives of the study

The study objective is to synthesize optimal structures of 4- and 8-bit parallel binary adders with OR and XOR logic elements in the last digit using the acyclic model of digital signal processing. This will make it possible to increase speed, reduce power consumed by the adder compared with counterparts and extend the principle of synthesis to a larger bit size of acyclic adders utilizing the series-parallel carry method.

To achieve this objective, the following tasks have to be addressed:

– set the range of numbers of acyclic adders with OR and XOR logical elements in the last digit, compare their speed and power consumption;

– evaluate speed and power consumption of the acyclic adder structure with OR logical element in the last digit for processing modified codes of input arguments;

– conduct comparative analysis of speed and complexity of the 8-bit acyclic adder structures with XOR logical elements in the last digit and the 8-bit adders of the prefix model of computation of the sum and carry signals.

4. The scale of measurement of the combinational circuit of the adder

Combinational circuit of the adder as a theoretical object will be measured with the help of parameters of depth and complexity of the device logical structure by recounting the number of corresponding logical elements. The following measurement units are taken: logical elements 2-In AND, 2-In OR, Inventor: one logical element; 2-In XOR: four elements (Fig. 3, *a*). To establish complexity of the adder circuit, assume that the conventional graphical designation of 2-In XOR in Fig. 3, *b*, also consists of four elements.

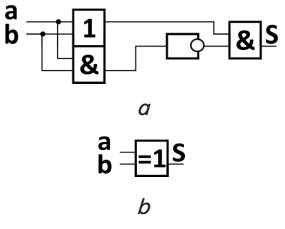

Fig. 3. Logical element 2-In XOR: a variant of the open structure of 2-In XOR (*a*); conditional graphic designation of 2-In XOR (*b*)

Circuit complexity is the number of functional elements in the circuit. This notion is very close to the notion of *bit complexity* of computation and the notion of the circuit proper is close to the notion of a *program without branching*.

The *circuit depth* is the number of functional elements in the longest chain that connects inputs of the circuit with its outputs. Assume that the signal at the output of the element does not appear immediately after the signal injection to the inputs but with some delay, then the circuit depth determines overall delay. For the circuit of adding *n*-bit numbers, the circuit depth is proportional to the number of bits, *n*.

Speed of the combinational circuit is assessed by the maximum delay of the signal when it passes from the input of the circuit to its output, i. e. it is determined by the time interval from the moment of arriving of input signals to the moment of detecting corresponding values at the output. The signal delay is proportional to the number of elements through which the signal passes from the input to the output of the circuit. Therefore, speed of the circuit is characterized by the value $r\,t$ where t is the time of signal delay in one element. The value of r is determined by the number of levels of the combinational circuit (CC) which is calculated as follows. *Zero level* is assigned to the CC inputs. The logical elements related only to the inputs of the circuit belong to the *first level*. An element refers to the level k if it is associated with elements of levels k-1, k-2, etc. The maximum level of elements, r, is determined by the number of CC levels which is called the *circuit rank* [19]. An example of determining the rank r of the circuit is shown in Fig. 4.

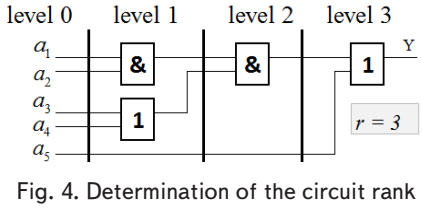

Fig. 4. Determination of the circuit rank

The method of computation of the circuit rank is, in fact, a technology for unambiguous establishment of the circuit depth.

The circuit rank computation is analogous to the steps of computation of the sum and carry signals in the combinational circuit. Taking into account the computation steps, speed of the circuit is characterized by the value kt where t is the time of signal delay in one element. The value of k is defined as follows. A *zero step* is assigned to the CC inputs. Logical elements related only to the circuit inputs relate to the *first step*. An element refers to step k if it is related to the elements of steps k-1, k-2, etc. The maximum step k is determined by the number of computation steps in the circuit. Each step is numbered. An example of computational steps of the circuit is shown in Fig. 5.

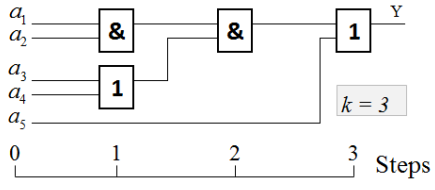

Fig. 5. Determination of computational steps of the circuit

The signal delay time for each of the elements 2-In AND, 2-In OR, Inventor is taken the same. Fig. 3 presents one of options of the open structure of the logical element 2-In XOR which consists of four logical elements including Inventor. Series connection of the logical elements 2-In AND, Inventor, 2-In AND forms the longest chain of the 2-In XOR structure. Therefore, the 2-In XOR depth is three computational steps (three logical elements). To establish depth of the adder circuit, take depth of three computation steps (three logical elements) for the open structure of the 2-In XOR (Fig. 3, *a*) as well as for conditional graphical designation of the 2-In XOR (Fig. 3, *b*).

The adder circuit is optimized either at the level of logical elements (for example, using the fastest elements in the carry chain, in particular, logical element AND-OR-NO has a smaller delay time compared to the logical element AND-OR if the latter is realized by the structure of AND-OR-NO-NO) or at the circuit level (for example, using structural methods for accelerating passage of the carry signal). The method of minimizing the logic function is a procedure for optimizing the circuit of computer logic, however, in the general case, for the majority of Boolean functions with n variables, minimal DNFs have size exponential from n. Quite often, the task of optimizing the logic circuit for the entire time of computer existence was solved empirically. At present, circuit optimization is solved to some extent by software, for example, the Logic Friday program optimizes circuits in several bases from the range of available elements [20].

5. The results of application of the acyclic model to reduce complexity and boost speed of the binary code adders

5. 1. The number range of the acyclic model

The result of the adding operation with bits a_i and b_i in the i-th bit of the binary code is expressed by two parameters: c_i is the result of operation of adding bits of the current digit of the binary code and the result of p_{i+1} by carrying a unit to the high-order digit. The results of the digit-by-digit execution of operations c_i and carry p_{i+1} are formed according to the following rules:

$$c_i = \begin{cases} a_i + b_i, & \text{at } a_i + b_i < q; \\ a_i + b_i - q, & \text{at } a_i + b_i > q. \end{cases} \quad (1)$$

$$p_{i+1} = \begin{cases} 0, & \text{at } a_i + b_i < q; \\ 1, & \text{at } a_i + b_i > q. \end{cases}$$

For the acyclic model in Fig. 1 (with XOR logical elements in the last digit) rules (1) should be executed. This provides a range of numbers in the binary code for n-bit grid in the range from 0 to $2^n - 1$. For example, for an 8-bit grid, the number range in the binary code of the acyclic model (Fig. 1) will be from 0 to 255.

Note that the number of all n-bit pairs of N arguments that can take part in the addition operation is

$$N = 2^n \times 2^n = 2^{2n}.$$

For example, the number of pairs N=256 for 4-bit arguments. Of these, 136 pairs provide a range of adding numbers in the binary code for a 4-bit grid in the range from 0 to $2^4 - 1$. The rest of the pairs will give overflow of the digital grid of the adder circuit (Fig. 6).

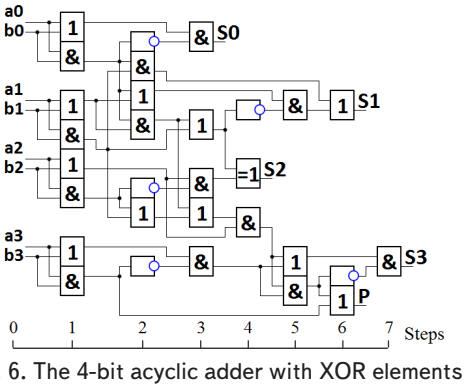

Fig. 6. The 4-bit acyclic adder with XOR elements in the last digit

Logical equations of the 4-bit acyclic adder in Fig. 6 are as follows:

$$S_0 = a_0\overline{b_0} + \overline{a_0}b_0;$$

$$S_1 = a_0b_0\overline{a_1}\,\overline{b_1} + \overline{a_0}a_1\overline{b_1} + \overline{a_0}\,\overline{a_1}b_1 + a_0b_0a_1b_1 + \overline{b_0}a_1\overline{b_1} + \overline{b_0}\,\overline{a_1}b_1;$$

$$S_2 = \overline{a_0}\,\overline{b_1}a_2\overline{b_2} + \overline{a_0}a_1a_2\overline{b_2} + a_0\overline{b_1}\,\overline{a_2}b_2 + \overline{a_0}a_1a_2b + \\ + \overline{b_0}\,\overline{b_1}a_2\overline{b_2} + \overline{b_0}\,\overline{a_1}a_2\overline{b_2} + \overline{b_0}\,\overline{b_1}\,\overline{a_2}b_2 + \overline{b_0}a_1\overline{a_2}b_2 + \\ + a_0b_0\overline{b_1}\,\overline{a_2}b_2 + a_0b_0a_1\overline{a_2}\,\overline{b_2} + a_0b_0\overline{b_1}a_2b_2 + a_0b_0a_1a_2b_2 + \\ + \overline{a_1}\,\overline{b_1}a_2\overline{b_2} + \overline{a_1}\,\overline{b_1}\,\overline{a_2}b_2 + a_1\overline{b_1}\,\overline{a_2}\,\overline{b_2} + a_1b_1a_2b_2;$$

$$S_3 = \overline{a_0}\,\overline{b_1}\,\overline{b_2}a_3\overline{b_3} + \overline{a_0}a_1\overline{b_2}a_3\overline{b_3} + \overline{a_0}\,\overline{b_1}a_2a_3\overline{b_3} + \overline{a_0}a_1a_2a_3\overline{b_3} + \\ + \overline{a_0}\,\overline{b_1}\,\overline{b_2}a_3b_3 + \overline{a_0}a_1\overline{b_2}a_3b_3 + \overline{a_0}\,\overline{b_1}a_2a_3b_3 + \overline{a_0}a_1a_2a_3b_3 + \\ + \overline{b_0}\,\overline{b_1}\,\overline{b_2}a_3\overline{b_3} + \overline{b_0}a_1\overline{b_2}a_3\overline{b_3} + \overline{b_0}\,\overline{b_1}a_2a_3\overline{b_3} + \overline{b_0}a_1a_2a_3\overline{b_3} + \\ + \overline{b_0}\,\overline{b_1}\,\overline{b_2}a_3b_3 + \overline{b_0}a_1\overline{b_2}a_3b_3 + \overline{b_0}\,\overline{b_1}a_2a_3b_3 + \overline{b_0}a_1a_2a_3b_3 + \\ + \overline{a_1}\,\overline{b_1}\,\overline{b_2}a_3\overline{b_3} + \overline{a_1}\,\overline{b_1}a_2a_3\overline{b_3} + \overline{a_1}\,\overline{b_1}\,\overline{b_2}a_3b_3 + \overline{a_1}\,\overline{b_1}a_2a_3b_3 + \\ + a_0b_0\overline{b_1}\,\overline{b_2}a_3\overline{b_3} + a_0b_0a_1\overline{b_2}a_3\overline{b_3} + a_0b_0\overline{b_1}a_2a_3\overline{b_3} + a_0b_0a_1a_2a_3\overline{b_3} + \\ + a_0b_0\overline{b_1}\,\overline{b_2}a_3b_3 + a_0b_0a_1\overline{b_2}a_3b_3 + a_0b_0\overline{b_1}a_2a_3b_3 + a_0b_0a_1a_2a_3b_3 + \\ + a_1b_1\overline{b_2}a_3\overline{b_3} + a_1b_1a_2\overline{a_3}\,\overline{b_3} + a_1b_1\overline{b_2}a_3b_3 + a_1b_1a_2a_3b_3 + \\ + \overline{a_2}\,\overline{b_2}a_3\overline{b_3} + \overline{a_2}\,\overline{b_2}\,\overline{a_3}b_3 + a_2\overline{b_2}\,\overline{a_3}\,\overline{b_3} + a_2b_2a_3b_3;$$

$$P = a_0b_0b_1b_2b_3 + a_0b_0a_1b_2b_3 + a_0b_0b_1a_2b_3 + a_0b_0a_1a_2b_3 + \\ + a_0b_0b_1b_2a_3 + a_0b_0a_1b_2a_3 + a_0b_0b_1a_2a_3 + a_0b_0a_1a_2a_3 + \\ + a_1b_1b_2b_3 + a_1b_1a_2b_3 + a_1b_1b_2a_3 + a_1b_1a_2a_3 + \\ + a_2b_2b_3 + a_2b_2a_3 + a_3b_3.$$

For the acyclic model in Fig. 2 (with logical OR elements in the last digit), rules (1) are not executed in the last digit of adding binary codes. However, when the rule (1) is not executed in the last digit, logic of the acyclic adder (Fig. 7) gives overflow of the grid of the adder circuit. Thus, non-execution of rule (1) in the last digit of the acyclic model of the adder is registered by the signal of overflow of the digital grid. In this case, 136 pairs of 4-bit arguments provide a range of adding numbers of the acyclic adder with the OR logical elements in the last digit in the range from 0 to 2^4–1, and in a general case, from 0 to $2^n - 1$. The rest of the pairs will give overflow in the binary code for the n-bit digital grid of the adder circuit.

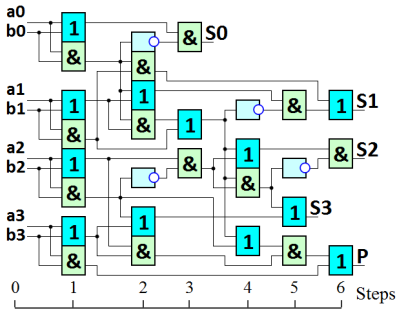

Fig. 7. 4-bit acyclic adder with OR elements in the last digit

Logical equations of the 4-bit acyclic adder in Fig. 7 are as follows:

$$S_0 = a_0\overline{b_0} + \overline{a_0}b_0;$$

$$S_1 = a_0b_0\overline{a_1}\,\overline{b_1} + \overline{a_0}a_1\overline{b_1} + \overline{a_0}\,\overline{a_1}b_1 + b_0a_1\overline{b_1} + \overline{b_0}\,\overline{a_1}b_1 + a_0b_0a_1b_1;$$

$$S_2 = \overline{a_0}\,\overline{b_1}a_2\overline{b_2} + \overline{a_0}a_1a_2\overline{b_2} + \overline{a_0}\,\overline{b_1}\,\overline{a_2}b_2 + \overline{a_0}\,\overline{a_1}a_2b + \overline{b_0}\,\overline{b_1}a_2\overline{b_2} + \\ + \overline{b_0}a_1a_2\overline{b_2} + \overline{b_0}\,\overline{b_1}\,\overline{a_2}b_2 + \overline{b_0}a_1\overline{a_2}b_2 + a_0b_0\overline{b_1}\,\overline{a_2}\,\overline{b_2} + \\ + a_0b_0a_1\overline{a_2}\,\overline{b_2} + a_0b_0\overline{b_1}a_2b_2 + a_0b_0a_1a_2b_2 + a_1\overline{b_1}\,\overline{a_2}\,\overline{b_2} + a_1b_1a_2b_2;$$

$$S_3 = +a_0b_0\overline{b_1}\,\overline{b_2} + a_0\overline{b_0}a_1\overline{b_2} + a_0\overline{b_0}\,\overline{b_1}a_2 + \\ + a_0b_0a_1a_2 + a_1\overline{b_1}\,\overline{b_2} + a_1\overline{b_1}a_2 + a_2\overline{b_2} + a_3 + b_3;$$

$$P = a_0b_0b_1b_2b_3 + a_0b_0a_1b_2b_3 + a_0b_0b_1a_2b_3 + a_0b_0a_1a_2b_3 + \\ + a_0b_0b_1b_2a_3 + a_0b_0a_1b_2a_3 + a_0b_0b_1a_2a_3 + a_0b_0a_1a_2a_3 + \\ + a_1b_1b_2b_3 + a_1b_1a_2b_3 + a_1b_1b_2a_3 + \\ + a_1b_1a_2a_3 + a_2b_2b_3 + a_2b_2a_3 + a_3b_3.$$

It can be seen from Table 1 that both 4-bit adders with OR and XOR logical elements provide the same range of number addition in the range from 0 to $2^4 - 1$. However, the adder with OR logical elements in the last digit is faster (the circuit depth of 6 elements) and has a simpler structure (the circuit complexity is 29 elements) compared to the adder circuit with XOR logical elements in the last digit.

Table 1

Comparative table of parameters of the 4-bit acyclic adder with OR and XOR elements in the last digit

Parameters	4-bit adder with OR logical elements in the last digit	4-bit adder with XOR logical elements in the last digit
Circuit complexity	29	33
Circuit depth	6	7
Range of digit adding	2^4-1	2^4-1
Number of all pairs of 4-bit arguments which can take part in the adding operation	256	256
Number of pairs of 4-bit arguments ensuring the adder operation with no overfill	136	136
Percentage of pairs of 4-bit arguments ensuring the adder operation with no overfill	53.13 %	53.13 %

5. 2. Processing of modified codes by acyclic adders with OR elements in sign digits of the input arguments

As an example of using the acyclic model of the adder with OR logical elements in the last digit, arithmetic operations with modified codes at different signs of numbers can be considered.

Two digits are used in the modified codes for assigning sign to the number (Table 2).

Table 2

Coding of the number sign in modified codes

Bits of the sign digits	Comments
00	Sign «+»
11	Sign «–»
01	Positive overflow digit
10	Negative overflow digit

Modified codes have turned out to be convenient (in terms of constructing arithmetical-logical transducers (ALT) to detect overflow of the digital grid. If the sign digits of the result take the value of 00 or 11, then there was no overflow of the digital grid and if 01 or 10, overflow took place.

In algebraic addition, sign digits are considered as the high-order digits of the number. If there is a carry from a high-order digit during the addition operation, the carry bit is added to the low-order digit of the sum in the case of applying a modified inverse code or rejected when applying a modified complement code.

The shortcoming inherent in the reverse code passes into the modified inverse code. The modified direct code retains all shortcomings of the direct code [21]. Therefore, we will focus on considering only the modified complement code (MCC).

The MCC is obtained from the complement code by simple duplication of the sign digit of the number. Moreover, numbers can be stored in the computer memory in the complement code and converted into the modified code when they are forwarded to an executing device.

Performance of arithmetic operations in MCC looks as follows.

Example 1. Add the numbers given in MCC: A=0010101 (21_{10}) and B=0010010 (18_{10}). A>0, B>0.

$$\begin{array}{rl} & 0010101 \\ + & \underline{0010010} \\ & \mathbf{01}00111 \end{array}$$

Positive overflow takes place.

Example 2. Add numbers given in MCC: A=0010101 (21_{10}) and B=0001010 (10_{10}). A>0, B>0.

$$\begin{array}{rl} & 0010101 \\ + & \underline{0001010} \\ & \mathbf{00}11111 \end{array}$$

Positive sum was obtained in MCC.

Example 3. Add the numbers given in the MCC: A= =0010101 (21_{10}) and B=1110011 (-13_{10}). A>0, B<0. $A>|B|$.

$$\begin{array}{rl} & 0010101 \\ + & \underline{1110011} \\ & \mathbf{00}01000 \end{array}$$

Positive sum was obtained in MCC.

Example 4. Add numbers given in MCC: A=0010101 (21_{10}) and B=1100110 (-26_{10}). A>0, B<0 $A<|B|$.

$$\begin{array}{rl} & 0010101 \\ + & \underline{1100110} \\ & \mathbf{11}11011 \end{array}$$

Negative sum was obtained in MCC.

Example 5. Add numbers given in MCC: A=1101011 (-21_{10}) and B=1101110 (-18_{10}). A<0, B<0.

$$\begin{array}{rl} & 1101011 \\ + & \underline{1101110} \\ & \mathbf{10}11001 \end{array}$$

Negative overflow takes place.

Example 6. Add numbers given in the MCC: A=1101011 (-21_{10}) and B=1110110 (-10_{10}). A<0, B<0.

$$\begin{array}{rl} & 1101011 \\ + & \underline{1110110} \\ & \mathbf{11}00001 \end{array}$$

Negative sum was obtained in MCC.

Example 7. Add fractional numbers 1.010100 and 1.110000 given in the supplement code using the MCC.

Solution:

x = 1.010100 (comp.) -> 11.010100 (mod. comp.)
y = 1.110000 (comp.) -> 11.110000 (mod. comp.)

s=x+y=11.000100 (mod. comp.) (discard carry from the high-order digit)
$\Rightarrow$ 1.000100 (comp.)

Thus, s = 1.000100 (comp.) = -(0.111011 + 1 low-order digit) = =-0.111100 = 1.111100 (direct).

Check:

s = 1.000100 (comp.) = -(0.111011 + 1 low-order digit) = =-0.111100 = -15/16

x = 1.010100 (comp.) = -(0.101011 + 1 low-order digit) = =-0.101100 = -11/16

y = 1.110000 (comp.) = -(0.001111 + 1 low-order digit) = =-0.010000 = -4/16

-11/16 + (-4/16) = -15/16 – the result of addition is correct.

Modified codes are used to obtain algebraic addition which, in the end, greatly simplifies the hardware part of digital components. Algebraic addition in the adder's circuit is provided by comparing the sign digits of the modified code with the high-order digits of the device circuit. Since the sign digits of the modified codes of the two input arguments with different signs always have structure 00 and 11 or 11 and 00, carry in the two high-order digits of the adder circuit can only be transitive. In another way, structure 00 and 11 or 11 and 00 of two sign digits creates condition for a transitive carry and does not generate its own carry signal.

The condition for transitive carry is determined by the logical function (2).

$$p_i = a_i \vee b_i \text{ or } p_i = a_i + b_i. \quad (2)$$

If $p_i = 1$, then transitive carry to the next digits is possible, in a case when $p_i = 0$, transitive carry to the next digits is impossible. Arguments $a_i = 0$ and $b_i = 1$ or $a_i = 1$ and $b_i = 0$ of the logical function (2) always give the value of $p_i = 1$. Therefore, to provide logic of setting sign of the sum code at different signs of the input arguments and fixation of the overflow signal, it is sufficient to use one input element OR (Fig. 8) in each of the two high-order digits of the adder circuit. It simplifies logical structure of the adder and optimizes the circuit depth.

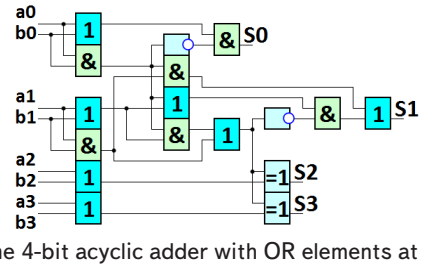

Fig. 8. The 4-bit acyclic adder with OR elements at the inputs of the last two digits to add 2-bit numbers with different signs in the modified code

Logical equations of the 4-bit acyclic adder in Fig. 8 are as follows:

$$S_0 = a_0\overline{b_0} + \overline{a_0}b_0;$$

$$S_1 = \overline{a_0}a_1\overline{b_1} + a_0b_0a_1b_1 + a_0b_0\overline{a_1}\,\overline{b_1} + \overline{a_0}\,\overline{a_1}b_1 + \overline{b_0}a_1\overline{b_1} + \overline{b_0}\,\overline{a_1}b_1;$$

$$S_2 = a_0b_0a_1\overline{a_2}\,\overline{b_2} + a_0b_0b_1\overline{a_2}\,\overline{b_2} + a_1b_1\overline{a_2}\,\overline{b_2} + \overline{a_1}\,\overline{b_1}b_2 + \overline{a_0}\,\overline{a_1}b_2 + \overline{a_1}\,\overline{b_1}a_2 + \overline{a_0}\,\overline{a_1}a_2 + \overline{b_0}\,\overline{a_1}b_2 + \overline{b_0}a_1a_2 + \overline{a_0}\,\overline{b_1}b_2 + \overline{a_0}\,\overline{b_1}a_2 + \overline{b_0}\,\overline{b_1}b_2 + \overline{b_0}\,\overline{b_1}a_2;$$

$$S_3 = +a_0b_0a_1\overline{a_3}\,\overline{b_3} + a_0b_0b_1\overline{a_3}\,\overline{b_3} + a_1b_1\overline{a_3}\,\overline{b_3} + \overline{a_1}\,\overline{b_1}b_3 + \overline{a_0}\,\overline{a_1}b_3 + \overline{a_1}\,\overline{b_1}a_3 + \overline{a_0}\,\overline{a_1}a_3 + \overline{b_0}\,\overline{a_1}b_3 + \overline{b_0}\,\overline{a_1}a_3 + \overline{a_0}\,\overline{b_1}b_3 + \overline{a_0}\,\overline{b_1}a_3 + \overline{b_0}\,\overline{b_1}b_3 + \overline{b_0}\,\overline{b_1}a_3.$$

The range of numbers of the adder circuit with logical elements OR in the last digit when processing the modified code remains within 0 to $2^n - 1$, and for the 2-bit grid of the adder shown in Fig. 8, the range of numbers in the modified code is from 0 to 3.

Table 3

Codes providing representation of a number sign for a range from 0 to 3

Codes for positive numbers				
Decimal	Direct	Inverse	Complement	MCC
0	0.00	0.00	0.00	00.00
+1	0.01	0.01	0.01	00.01
+2	0.10	0.10	0.10	00.10
+3	0.11	0.11	0.11	00.11
Codes for negative numbers				
Decimal	Direct	Inverse	Complement	MCC
–1	1.01	1.10	1.11	11.11
–2	1.10	1.01	1.10	11.10
–3	1.11	1.00	1.01	11.01

Table 4

Incomplete table of validity of the adder in Fig. 8 for adding codes with different signs in MCC

No.	Sign	a_1	a_0	Sign	b_1	b_0	Sign	s_1	s_0
1	+	0_{10}		+	0_{10}		+	0_{10}	
	0	0	0 0	0	0	0 0	0	0	0 0
2	+	0_{10}		–	1_{10}		–	1_{10}	
	0	0	0 0	1	1	0 1	1	1	0 1
3	+	0_{10}		–	2_{10}		–	2_{10}	
	0	0	0 0	1	1	1 0	1	1	1 0
4	+	0_{10}		–	3_{10}		–	3_{10}	
	0	0	0 0	1	1	1 1	1	1	1 1
5	+	1_{10}		–	1_{10}		+	0_{10}	
	0	0	0 1	1	1	1 1	0	0	0 0
6	+	1_{10}		–	2_{10}		–	1_{10}	
	0	0	0 1	1	1	1 0	1	1	1 1
7	+	1_{10}		–	3_{10}		–	2_{10}	
	0	0	0 1	1	1	0 1	1	1	1 0
8	+	2_{10}		–	1_{10}		+	1_{10}	
	0	0	1 0	1	1	1 1	0	0	0 1
9	+	2_{10}		–	2_{10}		+	0_{10}	
	0	0	1 0	1	1	1 0	0	0	0 0
10	+	2_{10}		–	3_{10}		–	1_{10}	
	0	0	0 1	1	1	0 1	1	1	1 1

5. 3. The 8-bit acyclic adder with the circuit depth of 8 elements

In order to provide same conditions of comparison, we shall present circuits of prefix (PPA) and acyclic (PAA) 8-bit adders with XOR logical elements in the last digit.

Fig. 9 presents an acyclic 8-bit PAA with XOR logical elements in the last digit and the circuit depth of 8 standard 2-input elements. Given that XOR consists of four elements (Fig. 3, *a*), the circuit complexity (Fig. 9) is 95 2-input elements.

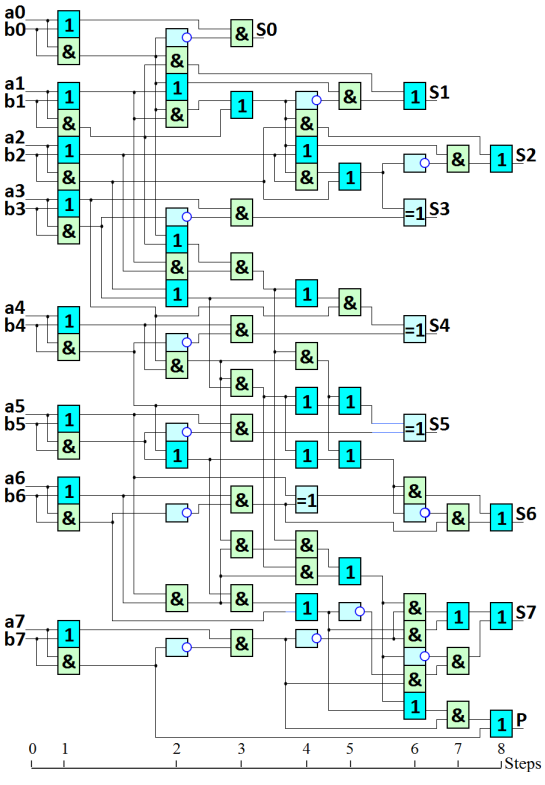

Fig. 9. Acyclic 8-bit PAA with circuit depth of 8 standard 2-input elements

Circuit of the acyclic adder in Fig. 9 confirms validity of assertion that the number of computational steps of an directed acyclic graph determines optimal number of carry operations in the circuit of an *n*-bit parallel adder [22]. The specified ratio is executed only for 4- and 8-bit adders. With the increase in bit size of the acyclic adder (16-, 32-, 64-bit ...), the number of computational steps will be determined by the logarithmic law (Fig. 10).

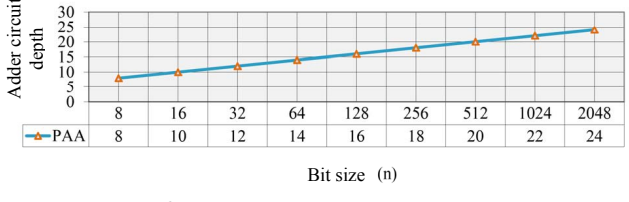

Fig. 10. Dynamics of increase in depth of the acyclic adder (PAA) circuit

Dynamics of growth of the PAA circuit depth is determined by logarithmic dependence: doubling of the adder bit size, *n*, increases the circuit depth by a constant value: two logical elements.

5. 4. The 8-bit acyclic adder with the circuit depth of 9 elements

Since Fig. 9 demonstrates the 8-bit PAA with XOR logical elements in the last digit and the circuit depth of 8 standard 2-input elements, then to construct an 8-bit PAA with the circuit depth of 9 standard 2-input elements, it is enough to use serial (for the low-order digits of the device circuit) and a not complex method for parallel computation of the sum and carry signals (for the rest of digits). The series carry method enables reduction of complexity of the hardware part of the device and does not increase the circuit depth in the acyclic model.

Fig. 11 presents an acyclic 8-bit PAA with XOR logical elements in the last digit and the circuit depth of 9 standard 2-input elements. Complexity of the circuit in Fig. 11 is 88 2-entry elements.

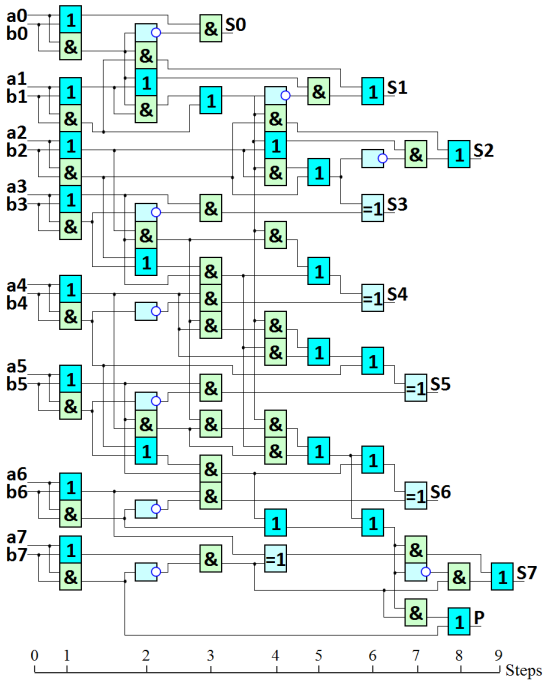

Fig. 11. Acyclic 8-bit PAA with circuit depth of 9 standard 2-input elements

The 8-bit Ling prefix adder [23–25] with XOR logical elements in the last digit and the circuit depth of 9 standard 2-input logic elements, with the adder logical structure updated to reduce complexity of the circuit is presented in Fig. 12. As XOR consists of four elements (Fig. 3, *a*), complexity of the circuit is 115 2-input elements.

Computational process of the 8-bit Ling PPA (Fig. 12) uses the following logical operations: 7 XOR, 44 AND, 28 OR, 15 Inventor. The 8-bit PAA (Fig. 11) uses 5 XOR, 34 AND, 25 OR, 9 Inventor. Given that logic of the XOR element uses four logical elements including Inventor, one can estimate indicator *S* of the 8-bit PAA adder operation quality (for example, power consumption), Fig. 11, compared to the adder shown in Fig. 12:

$$S = \frac{T_1}{T_2} = \frac{115}{88} = 1{,}3068 = 30{,}68\ \%,$$

where T_1, T_2 is the number of 2-input logic elements in the 8-bit Ling PPA (Fig. 12) and the 8-bit PAA (Fig. 11), respectively.

The prefix 8-bit Kogge-Stone PPA [3, 8] and the 8-bit Knowles PPA [26, 27] with XOR logic elements in the last digit are shown in Fig. 13. Given the depth of XOR is three elements and complexity four elements, depth of the 8-bit Kogge-Stone PPA and the 8-bit Knowles PPA (Fig. 13) will be 9 standard 2-input logical elements and the circuit complexity of 106 elements.

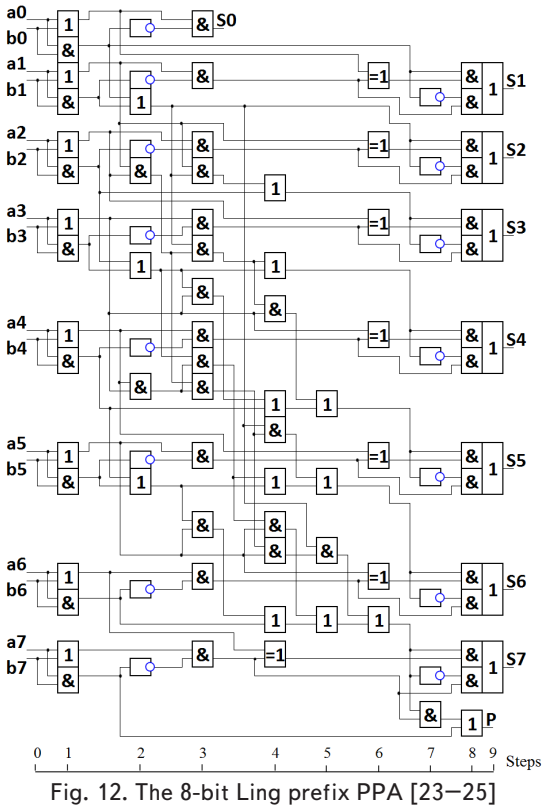

Fig. 12. The 8-bit Ling prefix PPA [23–25]

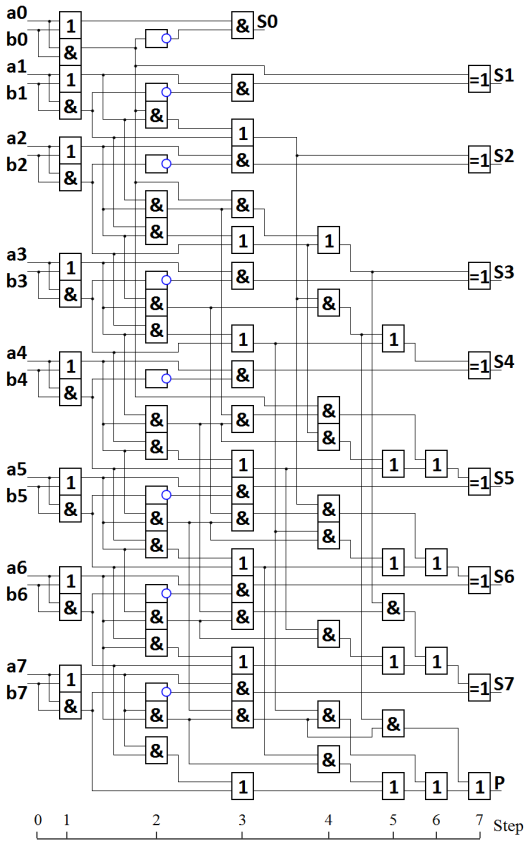

Fig. 13. The 8-bit Kogge-Stone prefix PPA [3, 8] and the 8-bit Knowles PPA [26, 27]

Computational process of the 8-bit Kogge-Stone PPA and the 8-bit Knowles PPA (Fig. 13) includes the following logical operations: 7 XOR, 44 AND, 26 OR, 8 Inventor. The 8-bit PAA adder (Fig. 11) uses 5 XOR, 34 AND, 25 OR, 9 Inventor. Given that logic of the XOR element uses four logical elements, quality indicator S (for example, in terms of power consumption) of the 8-bit PAA (Fig. 11) compared to the adder shown in Fig. 13 is as follows:

$$S=\frac{T_1}{T_2}=\frac{106}{88}=1,2045=20,45\ \%,$$

where T_1, T_2 is the number of 2-input logic elements of the 8-bit Kogge-Stone PPA and the 8-bit Knowles PPA (Fig. 13) and the 8-bit PAA (Fig. 11), respectively.

5. 5. The 8-bit acyclic adder with circuit depth of 10 elements

Fig. 14 represent the 8-bit acyclic PAA with XOR logical elements in the last digit and the circuit depth of 10 standard 2-input elements. The circuit complexity is 77 elements (Fig. 14).

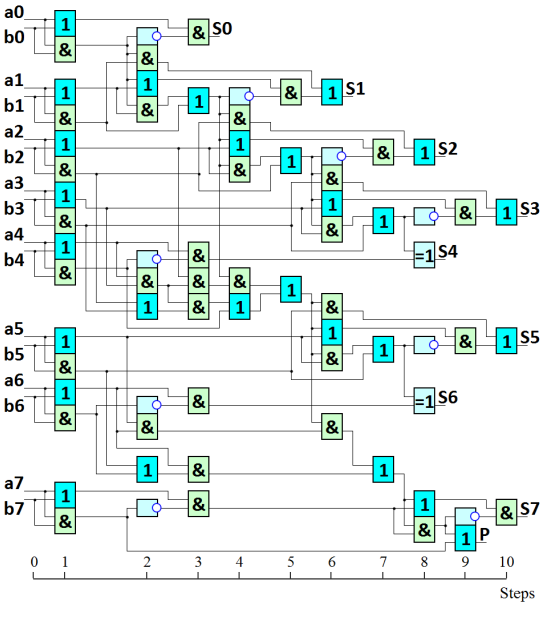

Fig. 14. The 8-bit acyclic PAA with the circuit depth of 10 standard 2-input elements

The 8-bit Sklansky prefix PPA [26, 28] with XOR logical elements in the last digit is presented in Fig. 15. Given the depth of XOR is three elements and complexity four elements, depth of the 8-bit Sklansky PPA (Fig. 15) is 10 standard 2-input logical elements, the circuit complexity of 89 elements.

Computational process of the 8-bit Sklansky PPA adder (Fig. 15) uses the following logical operations: 7 XOR, 33 AND, 20 OR, 8 Inventor. Computational process of the 8-bit PAA adder (Fig. 14) uses 2 XOR, 33 AND, 27 OR, 9 Inventor. The indicator S of the 8-bit PAA adder (Fig. 14) operation quality, for example, in terms of power consumption compared to the adder shown in Fig. 15 is as follows:

$$S=\frac{T_1}{T_2}=\frac{89}{77}=1,1558=15,58\ \%,$$

where T_1, T_2 is the number of 2-input logic elements of the 8-bit Sklansky PPA (Fig. 15) and the 8-bit PAA (Fig. 14), respectively.

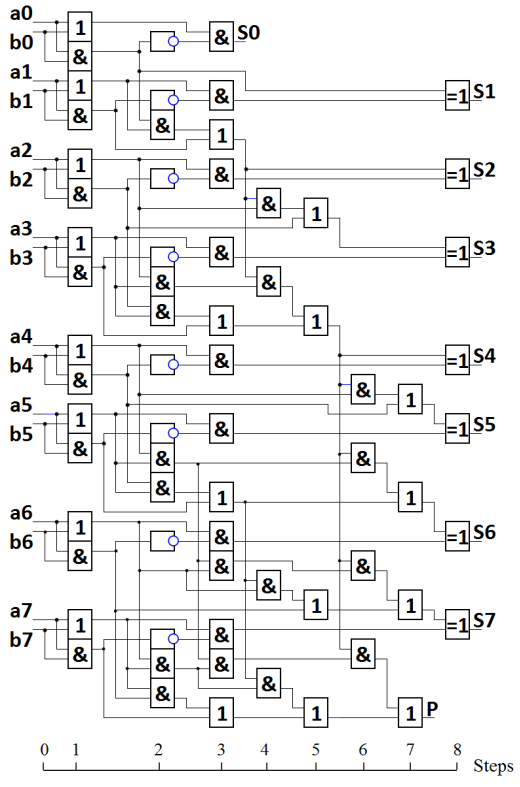

Fig. 15. The 8-bit Sklansky prefix PPA [26, 28]

The 8-bit Ladner-Fisher prefix PPA [4, 26] with XOR logical elements in the last digit is presented in Fig. 16. Given the XOR depth of three elements and complexity of four elements, depth of the 8-bit Ladner-Fisher PPA (Fig. 16) will be 10 standard 2-input logical elements, circuit complexity of 89 elements.

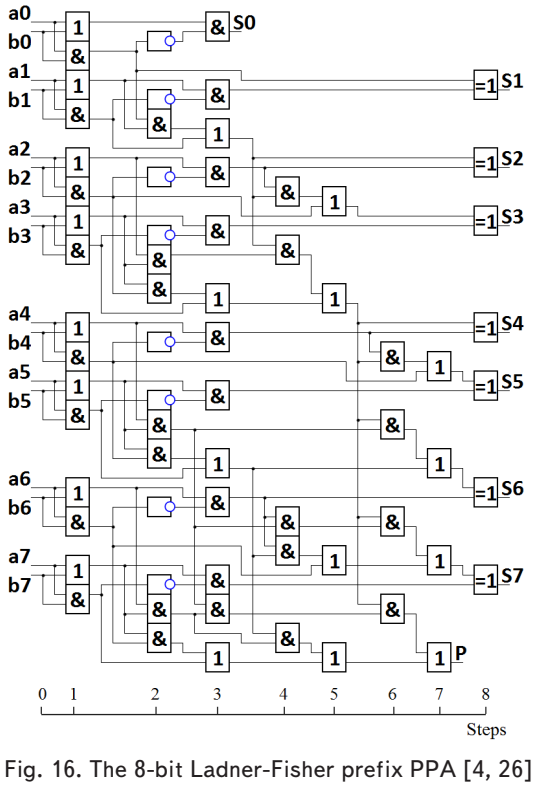

Fig. 16. The 8-bit Ladner-Fisher prefix PPA [4, 26]

Computational process of the 8-bit Ladner-Fisher PPA adder (Fig. 16) uses the following logical operations: 7 XOR, 33 AND, 20 OR, 8 Inventor, 9 Inventor. The 8-bit PAA (Fig. 14) uses 2 XOR, 33 AND, 27 OR, 9 Inventor. Indicator S of the 8-bit PAA adder operation quality (for example, in terms of power consumption), Fig. 14, compared to the adder shown in Fig. 16 is as follows:

$$S = \frac{T_1}{T_2} = \frac{89}{77} = 1{,}1558 = 15{,}58\ \%,$$

where T_1, T_2 is the number of 2-input logic elements of the 8-bit Ladner-Fisher PPA (Fig. 16) and the 8-bit PAA (Fig. 14), respectively.

5. 6. The 8-bit acyclic adder with the circuit depth of 11 elements

Fig. 17 represents the 8-bit acyclic PAA with XOR logical elements in the last digit and the circuit depth of 11 standard 2-input elements. Complexity of the circuit in Fig. 17 is 78 elements.

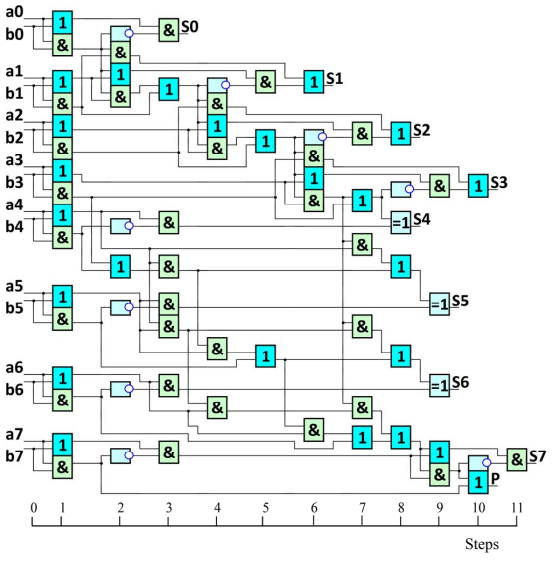

Fig. 17. The 8-bit acyclic PAA with the circuit depth of 11 standard 2-input elements

The 8-bit Han-Carlson prefix PPA [2, 26] with XOR logical elements in the last digit is presented in Fig. 18. Given the XOR depth is three elements and complexity four elements, depth of the 8-bit Han-Carlson PPA (Fig. 18) will be 11 standard 2-input logical elements, the circuit complexity is 89 elements.

Computational process of the 8-bit Han-Carlson PPA adder (Fig. 18) uses the following logical operations: 7 XOR, 33 AND, 20 OR, 8 Inventor. The 8-bit PAA adder (Fig. 17) uses 3 XOR, 32 AND, 25 OR, 9 Inventor. The indicator S of the 8-bit PAA adder operation quality (for example, in terms of power consumption), Fig. 17, compared to the adder shown in Fig. 18 is as follows:

$$S = \frac{T_1}{T_2} = \frac{89}{78} = 1{,}141 = 14{,}1\ \%,$$

where T_1, T_2 is the number of 2-input logic elements in the 8-bit Han-Carlson PPA (Fig. 18) and the 8-bit PAA (Fig. 17), respectively.

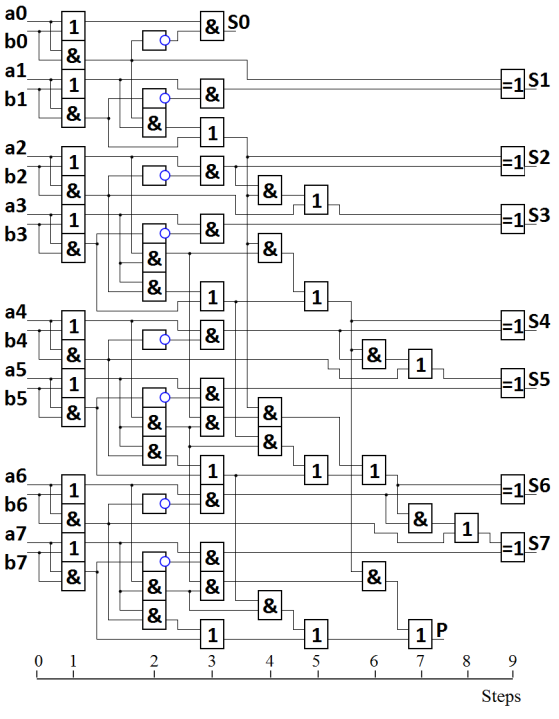

Fig. 18. The 8-bit Han-Carlson prefix PPA [2, 26]

5. 7. The 8-bit acyclic adder with the circuit depth of 12 elements

Fig. 19 shows the 8-bit acyclic PAA with XOR logical elements in the last digit and the circuit depth of 12 standard 2-input elements. Complexity of the circuit in Fig. 19 is 75 elements.

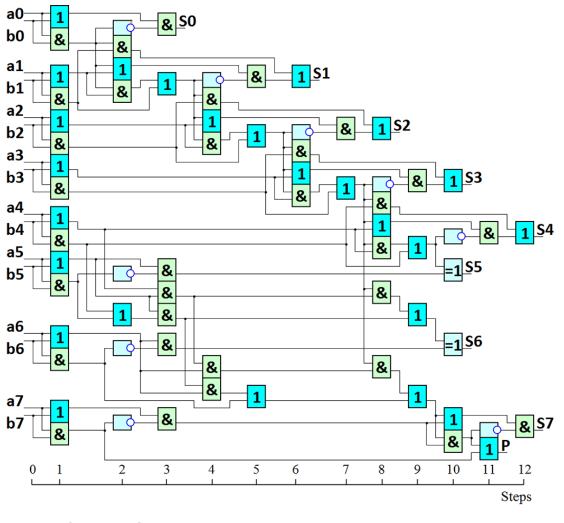

Fig. 19. The 8-bit acyclic PAA with the circuit depth of 12 standard 2-input elements

The 8-bit Brent-Kung prefix PPA [1, 8] with XOR logical elements in the last digit is presented in Fig. 20. Given the XOR depth is three elements, and complexity four elements, depth of the 8-bit Brent-Kung PPA (Fig. 20) will consist of 12 standard 2-input logical elements, complexity of the circuit is 86 elements.

Computational process of the 8-bit Brent-Kung PPA (Fig. 20) uses the following logical operations: 7 XOR, 31 AND, 19 OR, 8 Inventor. The 8-bit PAA (Fig. 19) uses 2 XOR, 32 AND, 26 OR, 9 Inventor. The indicator S (for example, in terms of power consumption) of the 8-bit PAA adder operation quality (Fig. 19) compared to the adder shown in Fig. 20 is as follows:

$$S=\frac{T_1}{T_2}=\frac{86}{75}=1{,}1467=14{,}67\ \%,$$

where T_1, T_2 is the number of 2-input logic elements of the 8-bit Brent-Kung PPA (Fig. 20) and the 8-bit PAA (Fig. 19), respectively.

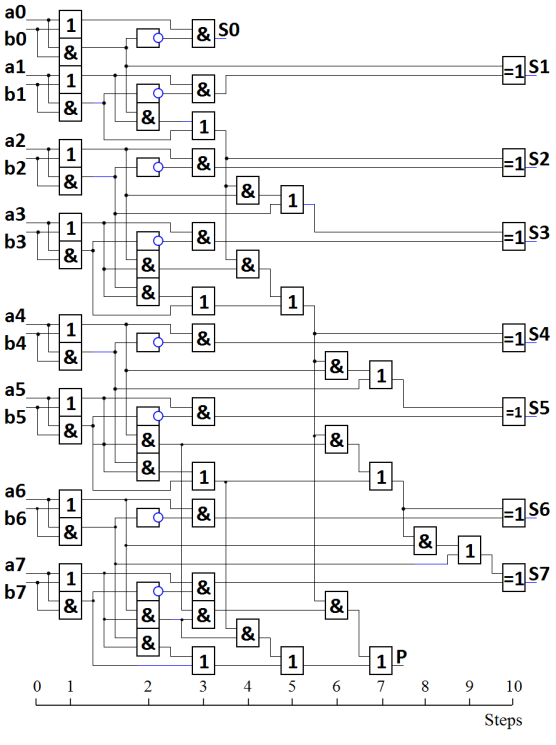

Fig. 20. The 8-bit Brent-Kung prefix PPA [1, 8]

5. 8. Comparative analysis of the 8-bit acyclic and prefix adders of binary codes

Parameters of synthesized circuits of the 8-bit acyclic and prefix adders are presented in comparative Table 5.

Table 5

Comparative table of parameters of the 8-bit acyclic and prefix adders

Parallel adder of binary codes with parallel carry		Circuit depth	Circuit complexity
PAA	Fig. 9	8	95
PAA	Fig. 11	9	88
Ling Adder	Fig. 12	9	115
Kogge-Stone and Knowles	Fig. 13	9	106
PAA	Fig. 14	10	77
Sklansky	Fig. 15	10	89
Ladner-Fisher	Fig. 16	10	89
PAA	Fig. 17	11	78
Han-Carlson	Fig. 18	11	89
PAA	Fig. 19	12	75
Brent-Kung	Fig. 20	12	86

It is seen from Table 5 that complexity of circuits of acyclic adders is smaller with the chosen value of the circuit depth.

Quality indicators of acyclic adders in comparison with prefix adders in terms of power consumption are presented in Table 6.

Table 6

Indicators of quality of acyclic adders in terms of power consumption

Parallel adder of binary codes with parallel carry		Quality indicator in terms of power consumption by the acyclic adder
PAA	Fig. 9	PPA unavailable for quality assessment PAA
PAA	Fig. 11	30.68 %
Ling Adder	Fig. 12	
PAA	Fig. 11	20.45 %
Kogge-Stone and Knowles	Fig. 13	
PAA	Fig. 14	15.58 %
Sklansky	Fig. 15	
PAA	Fig. 14	15.58 %
Ladner-Fisher	Fig. 16	
PAA	Fig. 17	14.1 %
Han-Carlson	Fig. 18	
PAA	Fig. 19	14.67 %
Brent-Kung	Fig. 20	

Fig. 21 shows dynamics of the circuit depth growth for five acyclic adders with the XOR logic elements in the last digit (Fig. 9, 11, 14, 17, 19) with an increase in the circuit bit size.

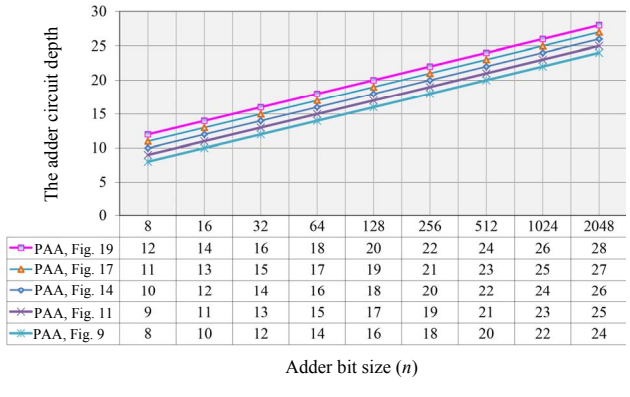

Fig. 21. Dynamics of the circuit depth growth in acyclic adders

Dynamics of growth of the PAA circuit depth is determined by the logarithmic dependence: doubling of the adder bit size, *n*, increases the circuit depth by a constant value: two logic elements.

6. Discussion of the results obtained in application of the acyclic signal processing model for synthesis of binary adders

Main drawbacks of the prefix model of computation of the sum and carry signals are as follows:

– organization of the process of parallel prefix computation involves computation beginning from the low-order digits of the adder circuit (this is the actual path (method) of the prefix) which results in excessive piling and complication of the hardware part of the device;

– the three-stage organization of computation of the prefix model signals in conditions of tireless optimization loses the prospect of continuing build-up of productivity of the digital signal processing. For example, speed of the 8-bit Ling prefix adder [23–25] (Fig. 12), Kogge-Stone prefix adder [3, 8] and Knowles prefix adder [26, 27] (Fig. 13) according to the chosen scale of the combinational circuit of the adder (as discussed in 4.1) is determined by the depth of the circuit of 9 standard 2-input logic elements. No information on further 8-bit PPA depth decrease was found.

In turn, application of the acyclic model is designed for:

– the process of series (for the low-order digits of the adder circuit) and parallel (for the rest of the digits) computation of the sum and carry signals which, in the end, reduces complexity of the hardware part of the device and does not increase the circuit depth;

– setting of an optimal number of computation steps.

The number of computation steps of an directed acyclic graph with two logical operations, AND and XOR, determines optimal carry operation number in the circuit of the *n*-bit parallel adder of binary codes. This indicates that the computational steps of the directed acyclic graph and carry of a unit to the high-order digit of the adder represent one object. Thus, the eight computational steps of an directed acyclic graph determine eight carry operations in the 8-bit PAA circuit. The mentioned ratio is executed only for 4- and 8-bit adders. With an increase in bit size of the acyclic adder (16-, 32-, 64-bit ...), the number of computational steps is determined by logarithmic law (Fig. 10). Beside the mentioned PAA, the Ling Adder, as compared to KSA, Knowles Adder, has less circuit complexity and length of the connecting wires (paths).

Thus, the use of the acyclic model, in comparison with the prefix model, for synthesizing circuits of adders of binary codes makes it possible to increase computing productivity by means of digital components. In particular, the series-parallel principle of computation of the acyclic model provides synthesis of a combinational 8-bit parallel adder with a circuit depth of 8 standard 2-input logic elements (Fig. 9), the counterpart of which is absent in the case of synthesis of the circuit using a prefix model.

Relationship between the number of computational steps of the directed acyclic graph and the number of operations of unit carry to the high-order digit causes the process of comparison of the adder structure with the respective directed acyclic graph. The purpose of this comparison is to establish the minimum sufficient number of carry operations for the operation of adding binary codes in the circuit of the parallel adder utilizing the parallel carry method. In the case when the synthesized adder received more carries compared to the number of computational steps of a corresponding directed acyclic graph, such an adder should be considered nonoptimal in terms of the number of computational operations.

Expediency of using structure of the adder with OR logical elements in the last digit consists in the fact that logic of the acyclic adder (Fig. 7) in case of non-compliance with rule (1) in the last digit provides overflow of the digital grid of the adder circuit. Thus, non-compliance with rule (1) in the last digit of the acyclic model of the adder is registered by

the signal of overflow of the digital grid, and, consequently, the sum code will not be written into computer memory. In this case, structure of the acyclic adder with OR logical elements in the last digit provides a range of adding binary codes in the range from 0 to 2^n-1.

A promising point for application of the structure of the acyclic adder with OR logical elements in the last digit is arithmetic operations with modified codes having different number signs. Modified codes are used to obtain properties of algebraic addition which is provided in the adder circuit by comparing the sign digits of the modified code with the high-order digits of the adder circuit. Since sign digits of the modified codes of the two input arguments with different signs always have structure 00 and 11 or 11 and 00, carry in the two high-order digits of the adder circuit can only be transitive. Therefore, in order to provide the logic of setting sign of the sum code at different signs of the input arguments and registering the overflow signal, it is sufficient to use one input element OR in each of the two high-order digits of the adder circuit. This measure simplifies logical structure of the adder and optimizes the circuit depth.

The range of numbers of the adder circuit with OR logical elements in the last digit remains within 0 to 2^n-1, when processing the modified code.

Use of the acyclic model is more advantageous in comparison with the counterparts due to the following factors:

– lower cost of development and implementation, since the acyclic model ensures a relatively simpler adder structure;

– presence of an optimization criterion: the number of computational steps of the acyclic graph indicates the minimum sufficient number of operations of unit carry to the high-order digit.

Since the acyclic model demonstrates the 8-bit PAA with the circuit depth of 8 standard 2-input logic elements (Fig. 9) the counterpart of which was not found for the PPA structure, the principle of improving computation of digital components moves from prefix model to acyclic one. Hence, the prospects of further studies of digital circuits may consist in reappraisal of the method of parallel expansion of the computing process in modern digital devices, reappraisal of addition algorithms in the nanometer range, reappraisal of the adder designs implemented with the use of memristors, etc.

8. Conclusions

1. Adders with OR and XOR logical elements provide the same range of addition of numbers from 0 to 2^n-1, which was experimentally proved by examples of the adders in Fig. 7, 8. However, the adder with OR logical elements in the last digit (Fig. 8) is faster (the circuit complexity is 6 elements) and has a simpler structure (the circuit complexity is 29 elements) compared with the adder circuit with XOR logical elements in the last digit (Fig. 7).

Therefore, structure of the adder with OR logical elements in the last digit gives grounds to assert expediency of its application in the processes of synthesis of arithmetic devices for processing digital data since the mentioned adder circuit is capable of:

– speed increase in comparison with counterparts;

– reduction of power consumption and heat emitted by digital devices in integrated circuits.

2. To obtain properties of an algebraic addition operation, sign digits of the modified codes are compared with high-order digits of the adder circuit. Since sign digits of the modified codes of two input arguments with different signs always have structure 00 and 11 or 11 and 00, carry in two high-order digits of the adder circuit can only be transitive. Therefore, in order to provide logic of defining sign of the sum code at different signs of the input arguments and registering the overflow signal, it is sufficient to use one input element OR in each of two high-order digits of the adder circuit. This measure simplifies logical structure of the adder and optimizes the circuit depth. The range of numbers of the adder circuit with OR logical elements in the last digit remains in the range from 0 to 2^n-1, when processing the modified code.

3. Effectiveness of the acyclic model with XOR logical elements in the last digit was demonstrated by examples of synthesis of the 8-bit parallel adders borrowed from works of other authors for comparison purposes:

– there is no PPA counterpart for the circuit of the acyclic 8-bit parallel adder with the circuit depth of 8 elements (Fig. 9);

– circuits of the Ling prefix adder (Fig. 12) [23–25], Kogge-Stone PPA [3, 8], Knowles [26, 27] PPA (Fig. 13) and the circuit of 8-bit parallel acyclic adder with the circuit depth of 9 elements (Fig. 11). Power consumption of the 8-bit PAA adder (Fig. 11) decreases by 30.68 % compared to the Ling adder (Fig. 12) and by 20.45 % compared to the Kogge-Stone PPA and Knowles PPA (Fig. 13).

– circuits of the Sklansky prefix adder (Fig. 15) [26, 28], Ladner-Fisher adder (Fig. 16) [4, 26] and the circuit of the acyclic 8-bit parallel adder with the circuit depth of 10 elements (Fig. 14). Power consumption of the 8-bit PAA adder (Fig. 14) decreases by 15.58 % compared to the Sklansky PPA (Fig. 15) and Ladner-Fisher PPA (Fig. 16);

– circuit of the Han-Carlson prefix adder (Fig. 18) [2, 26] and circuit of the acyclic 8-bit parallel adder with the circuit depth of 11 elements (Fig. 17). Power consumption of the 8-bit PAA adder (Fig. 17) decreases by 14.1 % as compared to the Han-Carlson PPA adder (Fig. 18);

– circuit of the Brent-Kung prefix adder (Fig. 20) [1, 8] and circuit of the acyclic 8-bit parallel adder with the circuit depth of 12 elements (Fig. 19). Power consumption of the 8-bit PAA adder (Fig. 19) decreases by 14.67 % as compared to the Brent-Kung PPA adder (Fig. 20).

Proceeding from the examples of the parallel adders, the acyclic model gives grounds to assert expediency of its application in the processes of synthesis of arithmetic devices for digital data processing since these circuits are capable of:

– speed increase;

– decreasing power consumption and heat emission from digital devices of integrated circuits.

References

1. Brent, Kung. A Regular Layout for Parallel Adders // IEEE Transactions on Computers. 1982. Vol. C-31, Issue 3. P. 260–264. doi: https://doi.org/10.1109/tc.1982.1675982
2. Han T., Carlson D. A. Fast area-efficient VLSI adders // 1987 IEEE 8th Symposium on Computer Arithmetic (ARITH). 1987. doi: https://doi.org/10.1109/arith.1987.6158699

3. Kogge P. M., Stone H. S. A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations // IEEE Transactions on Computers. 1973. Vol. C-22, Issue 8. P. 786–793. doi: https://doi.org/10.1109/tc.1973.5009159
4. Ladner R. E., Fischer M. J. Parallel Prefix Computation // Journal of the ACM. 1980. Vol. 27, Issue 4. P. 831–838. doi: https://doi.org/10.1145/322217.322232
5. Choi Y., Swartzlander E. E. Parallel Prefix Adder Design with Matrix Representation // 17th IEEE Symposium on Computer Arithmetic (ARITH'05). 2005. doi: https://doi.org/10.1109/arith.2005.35
6. Solomko M., Olshansky P. The Parallel Acyclic Adder // 2017 14th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM). Lviv, 2017. P. 125–129.
7. Srinivasarao B. N., Prathyusha Ch. Power Efficient Parallel Prefix Adders // International Journal of Research. 2018. Vol. 5, Issue 4. P. 472–477. URL: https://pen2print.org/index.php/ijr/article/view/12158/11483
8. Class ECE6332 Fall 12 Group-Fault-Tolerant Reconfigurable PPA. URL: http://venividiwiki.ee.virginia.edu/mediawiki/index.php/ClassECE6332Fall12Group-Fault-Tolerant_Reconfigurable_PPA
9. Ganesh Senthil R., Kalaimathi R. Design and Analysis of Kogge-Stone and Han-Carlson Adders in 130nm CMOS Technology // International Journal of Research. 2018. Vol. 05, Issue 07. P. 1063–1068. URL: https://pen2print.org/index.php/ijr/article/view/13190/
10. Ananda Kumari M., Loknadh Ch. Design an Efficient Fault Tolerant Kogge Stone Adder // International Journal of Research. 2018. Vol. 05, Issue 16. P. 1446–1449. URL: https://pen2print.org/index.php/ijr/article/view/15599/
11. Karthik K., Rajeshwar B. A New Design for Variable Latency Speculative E.C&D Han-Carlson Adder // International Journal of Research. 2017. Vol. 04, Issue 13. P. 975–980. URL: https://pen2print.org/index.php/ijr/article/view/9332/8980
12. Hima B. C., Srujana G., Rao M. V. Design of a novel BCD adder using parallel prefix technique // International Journal of Research in Electronics and Computer Engineering. 2018. Vol. 6, Issue 2. P. 2213–2219. doi: http://doi.org/10.13140/RG.2.2.26923.49443
13. Suvarna P., Murali krishna M. FPGA implementation of the carry select adder without using multiplexer // Global Journal for Research Analysis. 2017. Vol. 6, Issue 3. P. 642–643. URL: https://wwjournals.com/index.php/gjra/article/view/15467
14. Mathematical Modeling of Timing Attributes of Self-Timed Carry Select Adders / Balasubramanian P., Jacob Prathap Raj C., Anandi S., Mastorakis N., Bhavanidevi U. // Conference: 4th European Conference of Circuits Technology and Devices (in the Book, "Recent Advances in Circuits, Systems, Telecommunications and Control," Included in ISI/SCI Web of Science and Web of Knowledge. Paris, 2013. P. 228–243. URL: https://www.researchgate.net/publication/265684833_Mathematical_Modeling_of_Timing_Attributes_of_Self-Timed_Carry_Select_Adders
15. Revanna N., Swartzlander E. E. Memristor Adder Design // 2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS). Windsor, 2018. doi: https://doi.org/10.1109/MWSCAS.2018.8623864
16. Design Methodology to Explore Hybrid Approximate Adders for Energy-Efficient Image and Video Processing Accelerators / Soares L. B., da Rosa M. M. A., Diniz C. M., da Costa E. A. C., Bampi S. // IEEE Transactions on Circuits and Systems I: Regular Papers. 2019. P. 1–14. doi: https://doi.org/10.1109/tcsi.2019.2892588
17. Nagaraj S., Reddy G. M. S., Mastani S. A. Analysis of different Adders using CMOS, CPL and DPL logic // 2017 14th IEEE India Council International Conference (INDICON). 2017. doi: https://doi.org/10.1109/indicon.2017.8487636
18. Odnorozriadnyi sumator: Pat. No. 109142 UA / Nykolaichuk Ya. M., Davletova A. Ya., Krulikovskyi B. B., Vozna N. Ya. No. u201602165; declareted: 04.03.2016; published: 10.08.2016, Bul. No. 15.
19. Parhomenko P. P. Osnovy tekhnicheskoy diagnostiki. Moscow: Energiya, 1976. 464 p.
20. Logic Friday 1.02. URL: http://www.f1cd.ru/soft/base/logic_friday/logic_friday_102/
21. Orlov S. P., Martem'yanov B. V. Arifmetika EVM i logicheskie osnovy pereklyuchatel'nyh funkciy. Moscow: Mashinostroenie -1, 2005. 256 p. URL: http://vt.samgtu.ru
22. Solomko M., Krulikovskyi B. Study of carry optimization while adding binary numbers in the rademacher number-theoretic basis // Eastern-European Journal of Enterprise Technologies. 2016. Vol. 3, Issue 4 (81). P. 56–63. doi: https://doi.org/10.15587/1729-4061.2016.70355
23. Zeydel B. R., Baran D., Oklobdzija V. G. Energy-Efficient Design Methodologies: High-Performance VLSI Adders // IEEE Journal of Solid-State Circuits. 2010. Vol. 45, Issue 6. P. 1220–1233. URL: http://www.acsel-lab.com/Publications/Papers/energy_efficient_adders.pdf
24. Govindarajulu S., Vijaya Durga Royal T. Design of Energy-Efficient and High-Performance VLSI Adders // International Journal of Engineering Research. 2014. Vol. 3. P. 55–59. URL: https://pdfs.semanticscholar.org/a54c/5727cdc2be7830ea734f15eb1ba9ecfc2110.pdf
25. Pinto R., Shama K. Efficient shift-add multiplier design using parallel prefix adder // International Journal of Control Theory and Applications. 2016. Vol. 9, Issue 39. P. 45–53.
26. Two-Operand Addition. URL: https://pubweb.eng.utah.edu/~cs5830/Slides/addersx6.pdf
27. Knowles S. A family of adders // Proceedings 14th IEEE Symposium on Computer Arithmetic (Cat. No.99CB36336). 1999. doi: https://doi.org/10.1109/arith.1999.762825
28. Sklansky J. Conditional-Sum Addition Logic // IEEE Transactions on Electronic Computers. 1960. Vol. EC-9, Issue 2. P. 226–231. doi: https://doi.org/10.1109/tec.1960.5219822