applied sciences

Article

Bit Streaming Processing Algorithms for Intelligent Hardware Converters

Olga Bureneva *, Mikhail Kupriyanov and Nikolay Safyannikov

Department of Computer Science and Engineering, Faculty of Computer Technologies and Informatics, Saint-Petersburg State Electrotechnical University "LETI", ul. Professora Popova 5, 197376 St. Petersburg, Russia; mskupriyanov@etu.ru (M.K.); nmsafyannikov@etu.ru (N.S.)
* Correspondence: oibureneva@etu.ru

Abstract: The need to transfer the primary data conversions close to the sensors, to the endpoints of monitoring systems, as well as in IoT terminal devices makes the development of new approaches to computing and the design of appropriate algorithms relevant. The article shows stream processing algorithms that provide functional transformations of signals presented in bit stream form (single pulse streams, PWM signal streams) and binary codes at the same time. In such algorithms, the computational process is based on discretization, pulse frequency sweep and pulse-width sweep of codes as well as organization of parallel-serial processing. The suggested principles of algorithm organization are based on the fact that the computation is considered not as an event associated with calculation but as a continuous process of a result formation. The transition to algorithmic representations proposed by the authors makes it possible to obtain universal behavioral descriptions, independently of the specific hardware on which their implementation is performed.

Keywords: bit stream algorithms; digital converters; intelligent equipment; fault-tolerant; computational process; programmable logic device; VerilogHDL

check for updates

Citation: Bureneva, O.; Kupriyanov, M.; Safyannikov, N. Bit Streaming Processing Algorithms for Intelligent Hardware Converters. *Appl. Sci.* **2021**, *11*, 4899. https://doi.org/10.3390/app11114899

Academic Editor: Askhat Diveev

Received: 8 April 2021
Accepted: 24 May 2021
Published: 26 May 2021

1. Introduction

New approaches to the organization of computation and related hardware and software usually appear when existing solutions do not meet the growing technical requirements or when the element base changes significantly. At present, both of these factors affect the development of intelligent equipment, i.e., sensors, computing units operating near sensors, primary converters of measurement information, etc.

To reduce the amount of transferred data and the load of the communication channels as much as possible, one can perform calculations near sensing devices, including at sensors, the endpoints of monitoring systems, and terminal devices of the Internet of Things [1]. The relocation of calculations close to the sensors leads to the new problems.

First, it is necessary to perform the calculations in the same data format as generated at the sensor output. The absence of data format converters allows us to reduce the hardware cost. A frequently used option is near-sensor calculation in the analog form [2,3]. Another option is pulse form calculations, since the conversion of analog signals into pulse time parameters is not complicated. The appropriate converters were introduced in [4,5]. The data format conversion can be combined with the required calculations, as shown in [6,7].

Second, we need to develop special calculators that will be fast, energy-efficient, capable of performing real-time analysis, and provide high fault-tolerance of information processing and transmission. Examples of such special calculators are stochastic near-sensor computations [8], memory-based computations using memory for individual tasks based on multiply-accumulate operation [9], and processor elements based on pulse unary processing [10].

The change in the element base for near-sensor computing is related to active FPGA development. FPGA can be used to create both simple transducers and system-on-chip devices where software-hardware implementations are integrated [11,12]. Many sensor signal converters focused on FPGA implementation can be eventually manufactured as ASICs, providing additional opportunities in terms of power efficiency, reliability, and accuracy, as shown in [13,14].

The aim of this article is to introduce algorithms that perform functional processing of single pulse streams, PWM signal streams and binary codes simultaneously. Pulse coded signals (pulse-width modulated: PWM signals; pulse-frequency modulated: PFM signals) provide intrinsic immunity to interference. The information to be transmitted is encoded not in the signal amplitude but in the time parameters of the pulse signals. Such signals are often called "bit streams" since they use binary values of «0» or «1», and the information is transmitted continuously.

The main properties of bit stream processing algorithms are as follows:

- Pulses in the bit stream are of equivalent weight; therefore, the bit stream conversion is highly reliable. Loss of one pulse in the stream is equivalent to loss of the least significant bit of the binary code, while loss of one bit in the code can lead to loss of value equal to 2^i, where i stands for the lost bit position.
- The usage of single wire instead of multi-bit buses simplifies data transmission between the endpoints and the computational cores of the systems.
- Measurement and calculation processes can be easily parallelized.
- The presence of pauses between pulses in the stream reduces the average power consumption that implies high energy efficiency.

Implementation of transducers near sensors is one of the options to increase the endpoint devices' intelligence of sensor systems and the Internet of Things. This is in line with the general trend in the design of sensor information transducers.

In this paper we introduce new algorithms for bit stream conversion that work in the tracking mode by means of a small increment technique. Functional conversions are factorized into the increment/decrement operations performed as soon as the pulse comes. Through the example of temperature sensors signal processing, we show how our algorithms can be implemented into programmable logic devices for temperature measurement.

The algorithms we propose can be utilized to obtain universal behavioral descriptions, independent of the specific hardware on which they are implemented. Behavioral descriptions of stream processing, based on the simplest operations, can be adapted to various environments that provide the performance of the simplest logical functions and work with different pulse information carriers: electrical, biological, pneumatic, mechanical, optical and others [15–17].

2. Materials and Methods

From the structural point of view, bit stream processing algorithms correspond to processes with negative feedback, aimed at achieving an equilibrium state characterizing the final result. Step-by-step calculations in this case are absent, and mathematical transformations are obtained in a single process of result formation.

The conversion of binary code to bit streams is simple. Hence codes can be processed using functional transformation algorithms of bit stream. In this case, the computational processes are based on the following operations:

- Pulse frequency sweep (PFS);
- Pulse-width sweep (PWS).

By using PFS we convert the binary codes into PFM bit stream, i.e., bit stream characterized by the number of single pulses per unit time. By means of PWS, we transform the binary codes to the PWM signal, i.e., relative duration of the active signal value per unit of time.

2.1. Bit-Stream Multiplication Process

The main operation for bit stream processing algorithm implementation is streaming multiplication (SM). In order to perform this operation, the one for multiplied code should be converted into a pulse frequency modulated bit stream (PFM bit stream), while the other should be converted into a pulse width modulated signal (PWM signal) stream.

The conversion of a binary code to a pulse frequency modulated bit stream involves representing the code as a stream of bits, in which the number of bits in period T corresponds to the processed code N_1. The average value (frequency) of bits in period T is defined as follows:

$$F_{N_1} = \frac{N_1}{T}.$$

Conversion of the binary code to a pulse width modulated signal is achieved by generating an active signal θ, for example, equal to one. The duration of this signal for the code N_2 is determined as follows:

$$\theta_{N_2} = \frac{N_2}{N_{max}}.$$

In this equation, N_{max} is the maximum value of the code represented in the selected bit grid n, $N_{max} = 2^n - 1$. The process period T is defined as

$$T = \frac{2^n}{f},$$

where n is the digit capacity of the codes to be processed; f stands for the clock frequency of the process quantization.

During the bit stream multiplication of two codes, three operations are performed simultaneously. The first is multiplication of code N_1 by code N_2, the second is multiplication of code N_1 by inversion of code N_2, and the third is multiplication of code N_1 by one. A schematic of the parallel bit stream multiplication process is shown in Figure 1a.

The code N_1 is converted into a bit stream P_1 (blocks 1, 3). The code N_2 is converted into a PWM stream (blocks 2, 4). As a result, three bit streams are formed. The PWM stream has no effect on P1; therefore, P1 is the result of the code to the bit stream sweep (block 7): $N_{out1} = N_1$. To obtain stream P2, it is necessary to interrupt the stream based on the code N_1, with the stream PWM generated from N_2 (block 5). This interrupt provides streams multiplication and allows us to implement the following operation: $N_{out2} = N_1N_2$ (block 8). The P3 stream is the result of interrupting the stream based on the N_1 code by a stream of inverted PWM pulses. Such an interrupt implements the multiplication by inverted code $\overline{N_2}$:$N_{out3} = N_1\overline{N_2}$ (block 9).

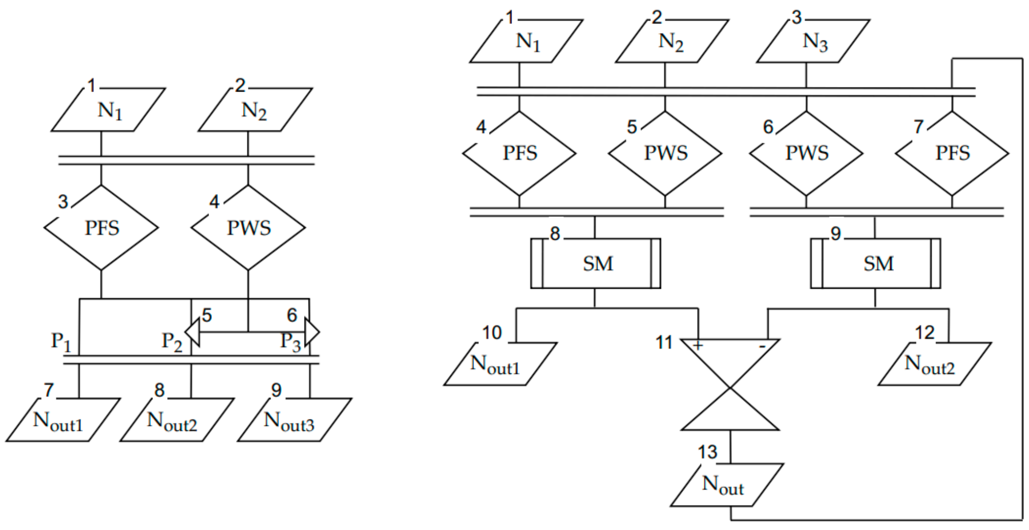
2.2. Process of Calculating of the Bit-Stream Multiplication-Division Function

The multiplication-division function is often used in the implementation of functional conversions; using this function as an example, we can show the stream calculation peculiarities. A block diagram of the multiplication-division operation process is shown in Figure 1b.

The process of calculating multiplication-division operation is periodic, and its period T depends on the digit capacity of the processed codes.

To implement the streaming mode of conversion, the input codes N_1, N_2, N_3 (blocks 1, 2, 3) and initial undefined output code N_{out} (block 13) are converted into bit streams. These threads form the positive and negative branches of the calculation process.

In each branch of the process, pulse-frequency sweep (PFS) of codes N_1, N_{out} (blocks 4, 7) and pulse-width sweep (PWS) of codes N_2, N_3 (blocks 5, 6) are executed in parallel. In the process under consideration, two streaming multiplications SM are organized (blocks 8, 9). Both of these perform multiplication of the PFM signal stream by PWM signals. That is, both SM blocks generate only the P2 streams (see Figure 1a).



(**a**) (**b**)

Figure 1. The scheme of the parallel bit-stream process: (**a**) multiplication process; (**b**) multiplication-division process.

At the output of block 8 there appears a bit stream. The number of pulses on the output of the block during the device operation period is defined as $N_{out1} = N_1N_2$ (block 10). The bit stream on the output of block 9 is the result of the following operation $N_{out2} = N_3N_{out}$ (block 12).

The resulting bit streams are combined and grouped (block 11) to get the difference $R = N_1N_2 - N_3N_{out}$, which is accumulated and converted into the output code N_{out} (block 13).

Due to the use of negative feedback, the process branches come to an equilibrium state. In this state, the intensity of streams based on codes N_{out1} and N_{out2} is equal, and hence R = 0. Using equality $R = N_1N_2 - N_3N_{out} = 0$, we obtain

$$N_{out} = \frac{N_1N_2}{N_3}. \tag{1}$$

The multiplication-division operation is implemented as a tracking process, and the result of the computation is generated during the balancing of the system. The process constantly tries to maintain an equilibrium state and restores to its state after short-term failures occur.

3. Results

The stream processes can be described algorithmically.

3.1. Bit-Stream Algorithms

The conversion of the code into a pulse-frequency stream is based on the method of small increments. According to this method, the input code is represented as a stream of single bits occurring at fixed moments of time, ti. The number of pulses per process period is equal to the normalized value of the code. Figure 2 shows the algorithm of pulse-frequency sweep.

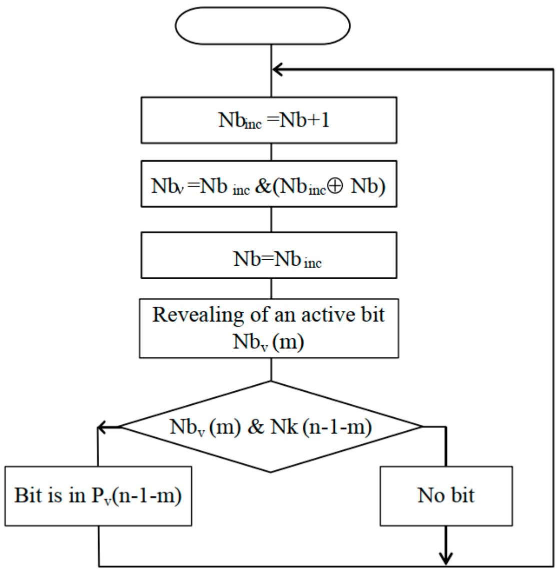


Figure 2. Algorithm of pulse-frequency sweep.

When performing the sweep of the n-bit binary code N_k, n bit streams are formed in parallel.

The algorithm works as follows: some base code n-bit binary code N_b at each moment of time t_i is incremented by 1, and N_{binc} code is formed.

The value of N_{binc} is compared with N_b in order to find the position m ($0 \leq m \leq n-1$), where N_{binc} code contains 1, and N_b code contains 0. If such a combination is found, then a single bit is generated in the stream with the number $n-1-m$ ($P_{v(n-1-m)}$).

The converted code N_k is used as a mask of the current state of the streams P_{v0}, P_{v1}, … $P_{v(n-1)}$. For masking, we use the following rule: the least significant bit 0 of the binary code N_k masks the stream $P_{v(n-1)}$, formed on the basis of the most significant bit of codes N_{binc}, and the most significant $(n-1)$ bit codes N_k masks the stream P_{v0}. The result of the masking makes it possible to determine the necessity of the bit generation to represent the N_k code at the current time t_i.

Table 1 shows an example of obtaining the bit streams P_{v0}, P_{v1}, P_{v2}, P_{v3} for n = 4.

Table 1. Example of obtaining bit streams.

N_b	N_{binc}	P_{v0}	P_{v1}	P_{v2}	P_{v3}
0000	0001	0	0	0	1
0001	0010	0	0	1	0
0010	0011	0	0	0	1
0011	0100	0	1	0	0
0100	0101	0	0	0	1
0101	0110	0	0	1	0
0110	0111	0	0	0	1
0111	1000	1	0	0	0
1000	1001	0	0	0	1
1001	1010	0	0	1	0
1010	1011	0	0	0	1
1011	1100	0	1	0	0
1100	1101	0	0	0	1
1101	1110	0	0	1	0
1110	1111	0	0	0	1
1111	0000	0	0	0	0

When considering the sweeping of the code N_k, if $N_k = 7$ (binary equivalent: 0111), the streams P_{v0}, P_{v1}, P_{v2} will be selected. The zero value of the most significant bit of the code N_k blocks the stream P_{v3}. The merging of bits of P_{v0}, P_{v1}, and P_{v2} streams allows the formation of a stream in which the number of bit pulses per period is equal to the value of the sweeping code $N_k = 7$. If $N_k = 12$ (binary equivalent: 1100), then P_{v2} and P_{v3} streams are selected. The sum of bits in P_{v2} and P_{v3} is equal $N_k = 12$. The streams P_{v0}, P_{v1} are blocked by zero values of bits number 0 and number 1 of the N_k code.

The multiplication of PFM and PWM data is realized by passing bits of the PFM stream only at times when the PWM signal is active (equal to 1).

We used the multiplication-division operation algorithm shown in Figure 3 to implement function (1). One of the input numerator codes, e.g., N_1, is converted into a bit stream, and the code N_2 is converted into a PWM signal Θ_{N_2}. As a result of bit-stream multiplication, the value $N_1\Theta_{N_2}$ is formed.

The result calculation starts when the bit corresponding to the code value N_1 appears, i.e., the output code N_{out} is incremented when $N_1\Theta_{N_2} = 1$ (blocks 3, 4). If after the pulse frequency sweep of the code N_1 the bit is not formed, the output code N_{out} does not change.

To form the compensatory actions, the denominator code N_3 is converted into a PWM signal Θ_{N_3}. The code N_{out} is converted into a bit stream, and the result of multiplying in the compensation branch of the algorithm is the value of $N_{out}\Theta_{N_3}$. When $N_{out}\Theta_{N_3} = 1$ (blocks 7, 8), it is necessary to form a compensating bit. When the compensating bit appears, the compensation mechanism starts working, and the code N_{out} is decreased by one. If $N_{out}\Theta_{N_3} = 0$, then the bit in the compensatory stream is not formed, and the output code N_{out} does not change. Operations 1–3 and 5–7 can be executed in parallel.

At the initial stage of the algorithm operation, the number of compensating actions is small, but as the output code N_{out} increases, the stream in the compensating loop becomes more intense. This continues until the process reaches equilibrium, characterized by the equality of the intensities of the streams causing the increments and decrements. If equilibrium is reached, the code N_{out} is the result of calculations. If the streams change randomly during the computation, the equilibrium is disturbed and automatically compensated, and the result is restored. Any changes in the input signal lead to the transition of the computational process to a new equilibrium state.

The period T of the algorithm operation is determined by the selected bit width of codes and is $T = 2^n t_i$, where t_i is the time interval determined frequency of the process quantization. The description of the computational process is based on the time interval t_i, chosen as the unit time, and then in relative units $T = 2^n$. The number of time samples t_i per period T is denoted as X_t.

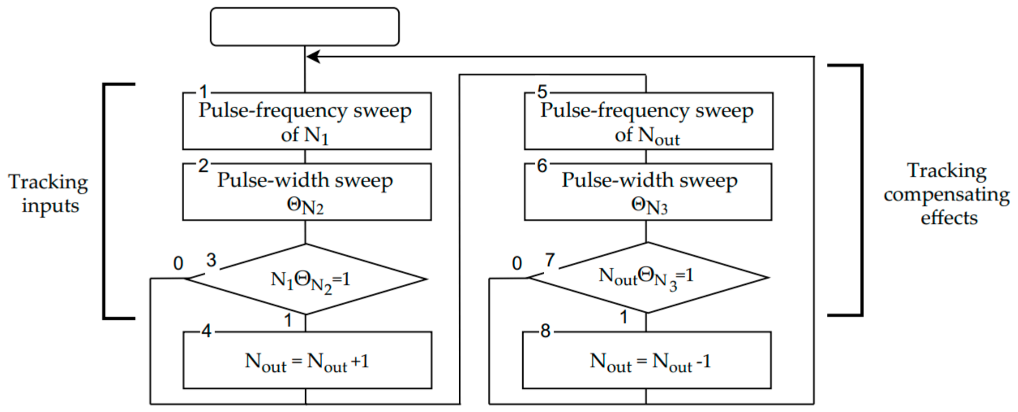

Tracking inputs
1
Pulse-frequency sweep of N_1
2
Pulse-width sweep Θ_{N2}
0
3
$N_1\Theta_{N_2}=1$
1
4
$N_{out} = N_{out}+1$
5
Pulse-frequency sweep of N_{out}
6
Pulse-width sweep Θ_{N3}
0
7
$N_{out}\Theta_{N_3}=1$
1
8
$N_{out} = N_{out}-1$
Tracking compensating effects

Figure 3. Algorithm of the multiplication-division operation.

The input and output codes are scaled by the maximum value in the used bit grid. A scaling factor equal to one is taken, so that the maximum values of N_{in} and N_{out} coincide with the maximum value of the period T in relative time units.

The number of bits in the stream generated from the input signals during the period T is determined as

$$X_{+} = \frac{N_1 \Theta_{N_2} X_{t_1}}{2^n}. \quad (2)$$

The number of time segments t_i per period can be any number. By taking $X_{t_1} = 2^n$, this simplifies the mathematical description of the process but does not change its essence. Formula (2) can be written as follows:

$$X_{+} = N_1 \Theta_{N_2}.$$

In a similar way, we can determine the number of bits generated in the compensating branch during the period T:

$$X_{-} = \frac{N_{out}\left(\Theta_{N_3}, \Theta_{N_2}, N_1\right) \Theta_{N_2} X_{t_2}}{2^n}. \quad (3)$$

The sampling frequency of the process when forming compensation streams is the same as when tracking input codes, $X_{t_2} = 2^n$. Formula (3) can be written as follows:

$$X_{-} = N_{out}\left(\Theta_{N_3}, \Theta_{N_2}, N_1\right) \Theta_{N_3}.$$

After completion of the first period of signals Θ_{N_2} and Θ_{N_3}, i.e., after 2^n cycles of the work of multiplication-division algorithm, the output code is defined as follows:

$$N_{out_1}\left(\Theta_{N_2}, \Theta_{N_3}, N_1\right) = N_{out_1} + N_1 \Theta_{N_2} - \Theta_{N_3},$$
$$N_{out}\left(\Theta_{N_2}, \Theta_{N_3}, N_1\right) = N_{out_0}\left(1 - \Theta_{N_3}\right) + N_1 \Theta_{N_2},$$

where N_{out1} is some initial value of the output code.

At the end of the algorithm second period, the following code is generated:

$$N_{out_2}\left(\Theta_{N_2}, \Theta_{N_3}, N_1\right) = N_{out_0}\left(1 - \Theta_{N_3}\right)^2 + N_1 \Theta_{N_2}\left[1 + \left(1 - \Theta_{N_3}\right)\right].$$

The result after the i-th period is defined as

$$N_{out_i}\left(\Theta_{N_2}, \Theta_{N_3}, N_1\right) = N_{out}\left(1 - \Theta_{N_3}\right)^i + N_1 \Theta_{N_2}\left[1 + \left(1 - \Theta_{N_3}\right) + \ldots + \left(1 - \Theta_{N_3}\right)^{i-1}\right].$$

The second term of this expression is a geometric progression with base q = 1 − Θ_{N_3}. It can be replaced by the amount

$$S = \frac{1 - \left(1 - \Theta_{N_3}\right)^{i-1}}{\Theta_{N_3}}.$$

Thus, the function describing the result of the algorithm at the end of period t is defined by the following expression:

$$N_{out_t} = N_{out_0}\left(1 - \Theta_{N_3}\right)^t + \frac{N_1 \Theta_{N_2}}{\Theta_{N_3}} - \frac{N_1 \Theta_{N_2}\left(1 - \Theta_{N_3}\right)^{(t-1)}}{\Theta_{N_3}}$$

As for the value of Θ_{N_3}, when it lies in the range of $0 < \Theta_{N_3} < 1$, we can use the following equations:

$$\lim_{t \to \infty}\left(1 - \Theta_{N_3}\right)^t = 0, \quad \lim_{t \to \infty}\left(1 - \Theta_{N_3}\right)^{t-1} = 0.$$

Thus, in the equilibrium state, the output code N_{out} is determined by the following dependence:

$$N_{out} = N_1 \frac{\Theta_{N_2}}{\Theta_{N_3}} \quad \text{or} \quad N_{out} = \frac{N_1 N_2}{N_3}$$

The time to reach the equilibrium state is determined by the number N_t periods, T.

If the process quantization frequency is high, we can treat the sequence of output codes as a continuous function. Therefore, the dynamics of the transient process can be determined using the following equation:

$$N_{out} = \int_0^{N_t} \left(N_1\Theta_{N_2} - N_{out}\Theta_{N_3}\right)dt.$$

The equation describes the process of transition to the tracking mode as the process of accumulating the difference of incrementing and decrementing streams during each period T. If the values N_1, Θ_{N_2}, Θ_{N_3} do not change during the transition process, the equation for the variable t can be solved as follows. Differentiating the right and left parts of the equation, we have

$$\frac{dN_{out}}{dt} = N_1\Theta_{N_2} - N_{out}\Theta_{N_3} \quad \text{whence} \quad dt = \frac{dN_{out}}{n_1\Theta_{N_2} - N_{out}\Theta_{N_3}}.$$

We integrate the expression and obtain

$$t + C = \int \frac{dN_{out}}{n_1\Theta_{N_2} - N_{out}\Theta_{N_3}}.$$

As a result of the transformation of this expression, we can get

$$t = \frac{1}{\Theta_{N_3}} \ln\left|\frac{N_{out_0} - N_1\left(\Theta_{N_2}/\Theta_{N_3}\right)}{N_{out} - N_1\left(\Theta_{N_2}/\Theta_{N_3}\right)}\right|.$$

The dependence of the transient duration is logarithmic, and its parameters are determined by the combination of input data and the initial state of the process.

3.2. *Algorithm Implementation*

We use VerilogHDL to implement and verify the considered algorithms. The VerilogHDL behavioral descriptions can be used to synthesize hardware modules.

The input signals of the pulse frequency sweep module are the reference frequency signal clk and the input code Nk. The pulse stream Fout is generated at the output of the synthesized module. Figure 4 shows the result of the Verilog module simulation with the ModelSim.

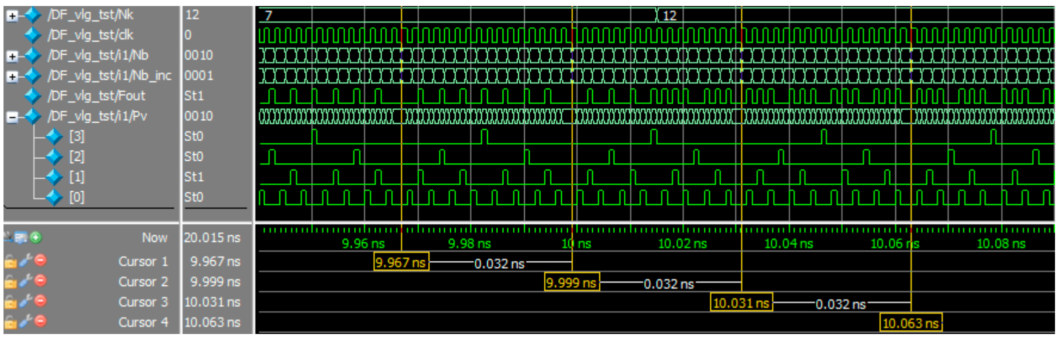


Figure 4. Result of pulse-frequency sweeping algorithm simulation.

To test the module, we used a TestBench. It is a non-synthesizable fragment of the VerilogHDL program that continuously generates the clock signal clk with frequency f_{clk} as well as the changing input code Nk. The function $urandom_range is used to generate the input code in a bit grid i; the frequency of code Nk changing during the test is $f_{clk}/(3 \times 2^i)$.

The bit width of the processed codes Nk is 4, so the pulse-frequency sweeping algorithm generates 4 pulse streams P_{v0}–P_{v3}. The lines DF_vlg_tst/i1/Pv[3...0] of the diagram show the generation of these streams. The total number of pulses in these streams during one period corresponds to the maximum possible value of the code represented in the 4-digit grid, and this value is equal to 15.

Three periods of algorithm operation are highlighted in Figure 4. The first period (time marks 9.967–9.999) shows the code Nk = 7 conversion. The number of pulses generated during the period of algorithm is 7 (line DF_vlg_tst/Fout). The second period is transitional. In this period the input code Nk is changed, so part of the period the output pulse stream is formed on the basis of the code Nk = 7, and then for Nk = 12. The third period (time marks 10.031–10.063) shows the code Nk = 12 conversion. The number of pulses generated per the period of the algorithm at the DF_vlg_tst/Fout output is 12.

The streaming multiplication-division module calculates the function using formula (1). We simulate it with the use a 10-bit implementation.

A generalized block diagram of the module is shown in Figure 5.

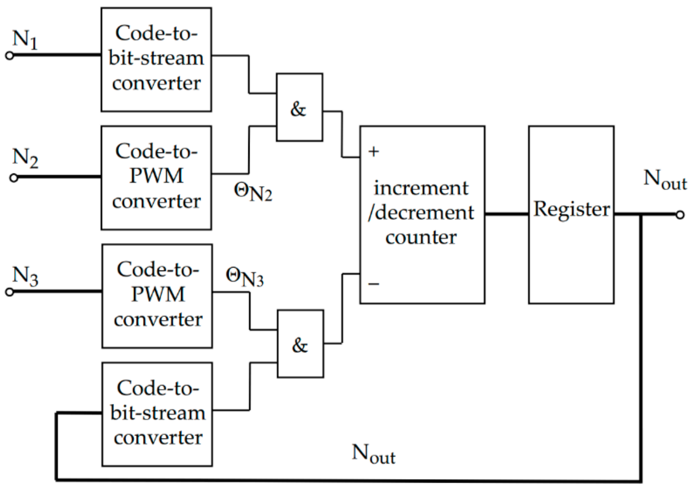
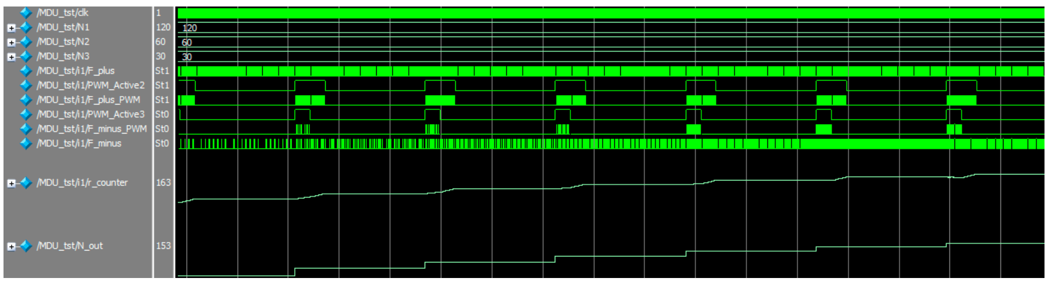


Figure 5. Result of pulse-frequency sweeping algorithm simulation.

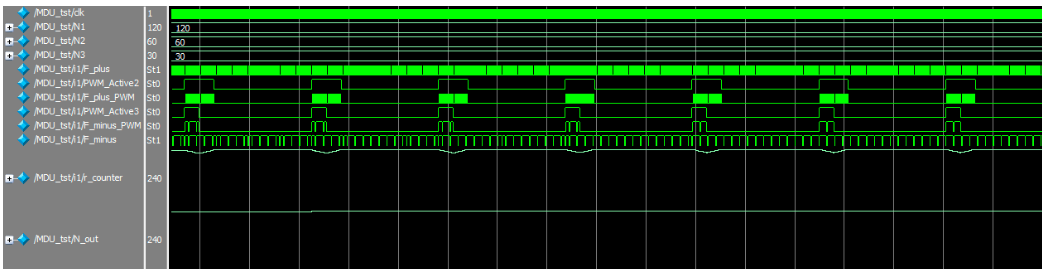
The scheme contains the following sub-modules:

- two code-to-bit-stream converters performing the conversion of codes N_1 and N_{out} into a bit stream using the Figure 2 algorithm and implementing the operations of blocks 1 and 5 of the algorithm shown in Figure 3;
- two code-to-PWM converters performing the conversion of N_2 and N_3 codes into a stream of PWM signals; they implement the operations of blocks 2 and 6 of the Figure 3 algorithm and can be implemented by known methods, for example, using counters;
- two «&» elements performing multiplication of bit stream by PWM signal (blocks 3 and 7 of the Figure 3 algorithm);
- increment/decrement counter, which counts the pulses generated in the positive and negative branches of the unit (blocks 4 and 8 of the Figure 3 algorithm).

Figure 6 shows the processes in the multiplication-division module.


/MDU_tst/clk 1
/MDU_tst/N1 120
/MDU_tst/N2 60
/MDU_tst/N3 30
/MDU_tst/i1/F_plus St1
/MDU_tst/i1/PWM_Active2 St1
/MDU_tst/i1/F_plus_PWM St1
/MDU_tst/i1/PWM_Active3 St0
/MDU_tst/i1/F_minus_PWM St0
/MDU_tst/i1/F_minus St0
/MDU_tst/i1/r_counter 163
/MDU_tst/N_out 153

(**a**)


/MDU_tst/clk 1
/MDU_tst/N1 120
/MDU_tst/N2 60
/MDU_tst/N3 30
/MDU_tst/i1/F_plus St1
/MDU_tst/i1/PWM_Active2 St0
/MDU_tst/i1/F_plus_PWM St0
/MDU_tst/i1/PWM_Active3 St0
/MDU_tst/i1/F_minus_PWM St0
/MDU_tst/i1/F_minus St1
/MDU_tst/i1/r_counter 240
/MDU_tst/N_out 240

(**b**)

Figure 6. Result of multiplication-division algorithm simulation: (**a**) transition process and (**b**) the process of tracking the result in a state of equilibrium.

To test the streaming multiplication-division module, we used TestBench, in which the clock signal clk was formed continuously and the input codes N_1, N_2 and N_3 were formed using the function $urandom_range. The frequency of change of input codes was chosen so that on the time diagram we could observe the transient process when the device tends to an equilibrium state, as well as the process of holding the stable state.

The input code N_1 is converted into a pulse stream (line MDU_tst/i1/F_plus). The codes N_2 and N_3 are converted into PWM signal streams (line MDU_tst/i1/PWM_Active2 and MDU_tst/i1/PWM_Active3). Figure 6a shows the transient where the resulting signal (MDU_tst/i1/r_count) increases on each cycle, coming closer to the result. The corresponding pulse stream becomes more intense on each cycle (MDU_tst/i1/F_minus). At the digital output (MDU_tst/i1/N_out), the data is fixed at the end of each period; this allows for a stable output code value during the period.

Figure 6b shows the process of tracking the result in the equilibrium state. The resultant signal (MDU_tst/i1/r_count) changes during each cycle, but by the end of the period it retains the value recorded at the end of the previous cycle. The corresponding pulse stream (MDU_tst/i1/F_minus) does not change. The digital output (MDU_tst/i1/N_out) stores the result in digital form. For given values N_1 = 120, N_2 = 60, and N_3 = 30, the result Nout = 240.

3.3. Application of the Developed Modules

The designed behavioral HDL modules are used in the design of the temperature regulator built into the human blood cholinesterase activity analyzer. This device provides analytical procedures with biological liquids. According to the rules of analysis, it is necessary to keep the temperature of the analytical solutions between 34 and 39 °C, since cholinesterase activity depends on temperature [18].

Figure 7 shows the cholinesterase activity analyzer. Thermal control is performed independently in two elements of the device: the measuring cuvette (A) and the reagent platform (B).

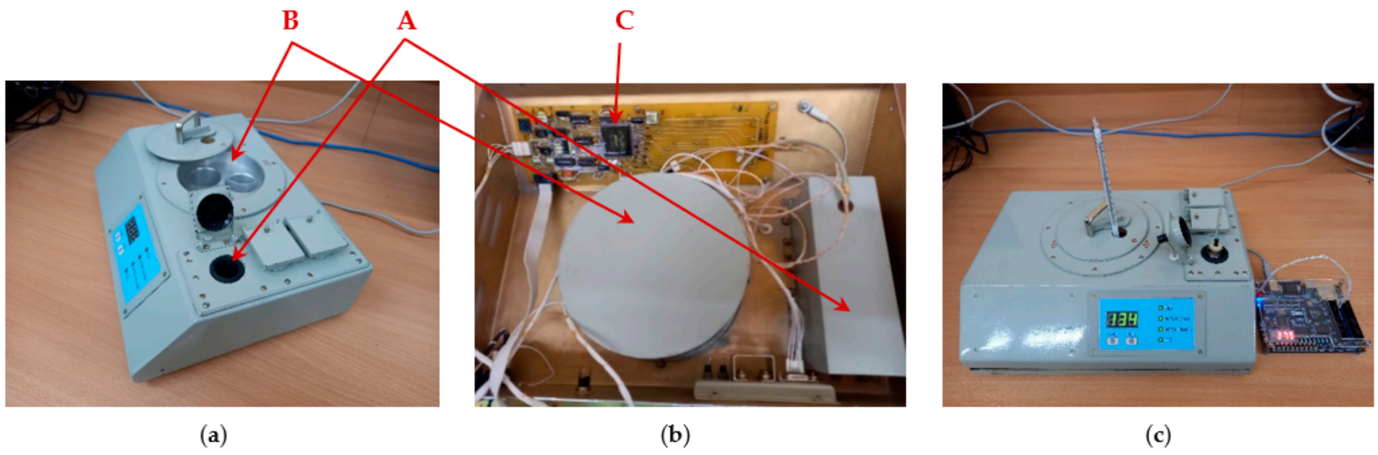


Figure 7. The cholinesterase activity analyzer: (**a**) exterior view of analyzer; (**b**) sensor installation; (**c**) temperature control using the proposed implementation and using mercury thermometers.

Measurement of cholinesterase activity is performed by measuring the time that the optical density of analytical solution changes by 10% from the initial value. The reservoir with analyte is placed in the measuring cuvette (A); the temperature of measuring cuvette is maintained within the specified range. The Analog Devices TMP03 sensor used for temperature control is inserted into the hole on the back side of the measuring cuvette using the thermally conductive paste.

The aluminum platform (B) is used to preheat the reagent containers to the specified temperature. To control the platform temperature, a second AD TMP03 sensor is installed using a thermally conductive paste in the hole on the back of the platform. Both sensors generate PWM output signals continuously.

The analyzer does not have a processing core, and its elements and modules operate under control of a digital finite state machine implemented on the CPLD chip (C). CPLD do not have special hardware elements for arithmetic calculations, but the temperature measurement requires arithmetic conversions according to the sensor characteristic:

$$\mathrm{T}(^{\circ}\mathrm{C}) = 235 - 400\frac{T_1}{T_2}. \tag{4}$$

To calculate the temperature according to Equation (4), we used designed algorithm of multiplication-division operation (Figure 3). In this case, the algorithm is not fully implemented; blocks 2 and 6 are not executed since the sensor signals are presented in the form of PWM, and operations to change the data format (pulse-width sweep, PWS) are not required. PWM signals come to processing immediately. The subtraction operation is also performed in stream form by decrementing constant 235 at the moment when the multiplication-division unit generates the next pulse of the output stream. Figure 8 shows the result of the simulation of the processes occurring during temperature measurement.

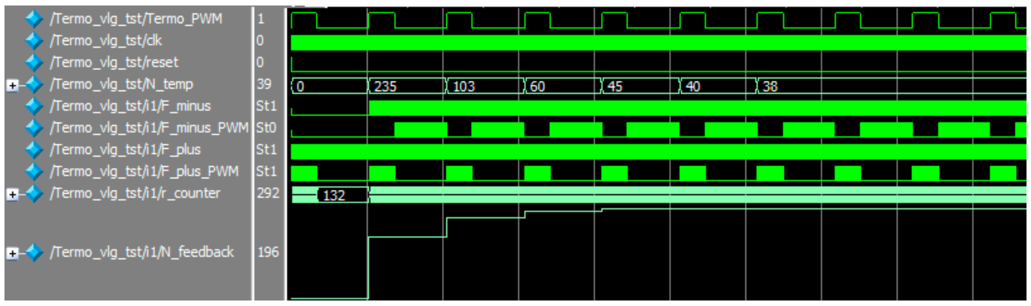

/Termo_vlg_tst/Termo_PWM 1
/Termo_vlg_tst/clk 0
/Termo_vlg_tst/reset 0
/Termo_vlg_tst/N_temp 39
/Termo_vlg_tst/i1/F_minus St1
/Termo_vlg_tst/i1/F_minus_PWM St0
/Termo_vlg_tst/i1/F_plus St1
/Termo_vlg_tst/i1/F_plus_PWM St1
/Termo_vlg_tst/i1/r_counter 292
/Termo_vlg_tst/i1/N_feedback 196
0
235
103
60
45
40
38
132

Figure 8. Result of simulation of processes in the temperature measurement system.

We tested the algorithm operation during the experiments by means of an analyzer. To indicate the temperature, we used an external board with indicators, since the alphanumeric indicator of the analyzer is intended to display the results of the analysis of cholinesterase activity. The modules based on the behavioral description of the temperature meter were implemented into the CPLD (MAX3512), and the indicator of the external board showed the measurement results. At the same time, we monitored the temperature of the measuring cuvette and platform by means of mercury thermometers; Figure 7c shows this process.

Rounding of the measurement results was performed with an accuracy of 0.5 °C, as the used sensor TMP03 for measurements in the range of 0–50 °C has such accuracy, and we conducted tests in this range. Mercury thermometers allow measurements with an accuracy of −0.1 °C. During the tests, the readings of the thermometers and the indicator of the device coincided.

The solution of this problem by the traditional method requires the use of calculators such as counters to determine the code values of the duration of PWM signals, multiplier, divider, and subtractor. These elements are available in the Quartus II library of parameterized modules. However, when selecting a chip of the CPLD class as a device, the compilation of the project finishes with error messages. To compare the traditional and bitstream approaches, we compiled a HDL description with FPGA selecting. The library elements were configured in a combinational way, and registers were not used. The results of the bit-parallel and bit-stream methods comparison are shown in Table 2.

Table 2. Bit-parallel and bit-stream method comparison.

Family/Device/Fitter Summary	Bit-Parallel Method	Bit-Stream Method
MAX3000A/EPM3512AFC256-7	Not synthesized	Total macrocells 171/512 (33%)
Cyclone II/EP2C5AF256A7/		
Total logic elements	157/4608 (3%)	70/4608 (2%)
Total combinational functions	152/4608 (3%)	49/4608 (1%)
Dedicated logic registers	32/4608 (<1%)	62/4608 (1%)
Total registers	32	62
Total pins	13/158 (8%)	13/158 (8%)
Embedded Multiplier 9-bit elements	2/26 (8%)	0/26 (0%)

A hardware cost analysis shows the cost-effectiveness of the bit-stream implementation.

The frequency characteristics were analyzed using Time Quest Timing Analyzer. The maximum frequency f_{max} for the bit-parallel method was 35.94 MHz. For the bit-stream method, the maximum frequency was 111.73 MHz, but the processing period was related to the bit rate, and for the 10-bit device version the frequency was defined as $f_{max}/2^{10}$, or 109 kHz.

The conversion accuracy is determined by the bit rate of the device and can be corrected for additional bits if necessary.

4. Discussion

In this paper, we proposed an approach for transferring the processes taking place in pulse stream devices into algorithmic form.

The suggested principles of organization of computation and designed algorithms are based on the fact that the computation is considered not as an event associated with obtaining a result, but as a continuous process of its formation. The algorithms implement processes tending to an equilibrium state, and in this state, the results of calculations are formed. The considered examples of the proposed algorithms' realization show stability to noises and hindrances in work. Automatic return to the result after failures is provided due to the negative feedback implemented in the algorithm.

The proposed approach to calculations is characterized by simple hardware implementation. It does not require the use of arithmetic blocks, even when calculations require performing multiplication and division operations. Therefore, CPLD chips can be used to implement the obtained hardware modules. The main disadvantage of the method is a significant time for equilibrium regime, which can reach 10 periods of device work. Because of this disadvantage, the proposed method can be used to build systems to control slow processes, for example, for temperature control systems.

The proposed approach can be applied to the design of primary transducers for smart sensors and sensor networks as well as in intelligent data streaming systems.

In addition, the developed algorithms can be used in the design of the pulse neural network elements. The application of algorithms makes it possible to realize computational operations in pulse streaming mode. In this case, pulses can have different physical nature: electrical, optical, mechanical, biological and others.

Author Contributions: Conceptualization, N.S., M.K. and O.B.; methodology, N.S. and O.B.; formal analysis, N.S., M.K. and O.B.; writing—original draft preparation, N.S. and O.B.; writing—review and editing, N.S. and M.K.; visualization, O.B.; supervision, N.S.; project administration, M.K.; funding acquisition, M.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Ministry of Science and Higher Education of the Russian Federation by the Agreement No. 075-15-2020-933, dated 13.11.2020, on the provision of a grant in the form of subsidies from the federal budget for the implementation of state support for the establishment and development of the world-class scientific center «Pavlov center, Integrative physiology for medicine, high-tech healthcare, and stress-resilience technologies».

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Najafi, M.H.; Faraji, S.R.; Bazargan, K.; Lilja, D. Energy-efficient near-sensor convolution using pulsed unary processing. In Proceedings of the IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP), New York, NY, USA, 15–17 July 2019.
2. Chen, Z.; Zhu, H.; Ren, E.; Liu, Z.; Jia, K.; Luo, L.; Zhang, X.; Wei, Q.; Qiao, F. Processing Near Sensor Architecture in Mixed-Signal Domain with CMOS Image Sensor of Convolutional-Kernel-Readout Method. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2019**, *67*, 389–400. [CrossRef]
3. Ma, T.; Jia, K.; Zhu, X.; Qiao, F.; Wei, Q.; Zhao, H.; Liu, X.; Yang, H. An Analog-Memoryless Near Sensor Computing Architecture for Always-On Intelligent Perception Applications. In Proceedings of the IEEE International Conference on Integrated Circuits, Technologies and Applications (ICTA), Chengdu, China, 13–15 November 2019.
4. Stout, T.; Dean, A. Voltage source based voltage-to-time converter. In Proceedings of the IEEE 58th International Midwest Symposium on Circuits and Systems (MWSCAS), Fort Collins, CO, USA, 2–5 August 2015.
5. Jia, S.; Weng, L.; Wang, W.; Wang, Y. A highly linear 5GS/s voltage-to-time converter for time-based analog-to-digital converters. In Proceedings of the IEEE Asia Pacific Microwave Conference (APMC), Kuala Lumpar, Malaysia, 13–16 November 2017.
6. Chen, K.; Chen, T.; Wei, C. Novel Pulse-Based Analog Divider with Digital Output. *IEEE Solid State Circ. Lett.* **2019**, *3*, 21–24. [CrossRef]
7. Safyannikov, N.M.; Bureneva, O.I. Time-to-Voltage Converters Based on the Time-Sharing Principle. *IEEE Access* **2020**, *8*, 17442–17453. [CrossRef]
8. Lee, V.T.; Alaghi, A.; Hayes, J.P.; Sathe, V.; Ceze, L. Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), Lausanne, Switzerland, 27–31 March 2017.
9. Liu, Z.; Ren, E.; Qiao, F.; Wei, Q.; Liu, X.; Luo, L.; Zhao, H.; Yang, H. NS-CIM: A Current-Mode Computation-in-Memory Architecture Enabling Near-Sensor Processing for Intelligent IoT Vision Nodes. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2020**, *67*, 2909–2922. [CrossRef]
10. Faraji, S.R.; Bazargan, K. Hybrid Binary-Unary Hardware Accelerator. *IEEE Trans. Comput.* **2020**, *69*, 1308–1319. [CrossRef]
11. Santos, E.J.P.; Silva, L.B.M. FPGA-based smart sensor implementation with precise frequency to digital converter for flow measurement. In Proceedings of the 2010 VI Southern Programmable Logic Conference (SPL), Ipojuca, Brazil, 24–26 March 2010.
12. Elkateeb, A. SOC-Based Sensor Mote Design. *Int. J. Mobile Netw. Commun. Telemat.* **2013**, *3*, 1–6. [CrossRef]
13. Kim, H.; Lee, B.; Mun, Y.; Kim, J.; Han, K.; Roh, Y.; Song, D.; Huh, S.; Ko, H. Reconfigurable Sensor Analog Front-End Using Low-Noise Chopper-Stabilized Delta-Sigma Capacitance-to-Digital Converter. *Micromachines* **2018**, *9*, 347. [CrossRef] [PubMed]

14. Sajjad, M.; Yusoff, M.B.Z.; Ahmed, M. Design of Double-Precision Fully-Programmable Computational Unit for FPGA and ASIC. In Proceedings of the 2020 International Conference on Computing, Electronics & Communications Engineering (iCCECE), Southend, UK, 17–18 August 2020.
15. Song, Y.; Panas, R.M.; Chizari, S.; Shaw, L.A.; Jackson, J.A.; Hopkins, J.B.; Pascall, A.J. Additively manufacturable micro-mechanical logic gates. *Nat. Commun.* **2019**, *10*, 882. [CrossRef] [PubMed]
16. Rao, D.G.S. Design of all-optical AND, OR, and XOR logic gates using photonic crystals for switching applications. *Photonic Netw. Commun.* **2021**, *41*, 109–118. [CrossRef]
17. Teo Jonathan, J.Y.; Woo, S.S.; Sarpeshkar, R. Synthetic Biology: A Unifying View and Review Using Analog Circuits. *IEEE Trans. Biomed. Circ. Syst.* **2015**, *9*, 453–474.
18. Reiner, E.; Buntić, A.; Trdak, M.; Simeon, V. Effect of temperature on the activity of human blood cholinesterases. *Arch. Toxicol.* **1974**, *32*, 347–350. [CrossRef] [PubMed]