Circuits, Systems, and Signal Processing (2022) 41:4928–4951
https://doi.org/10.1007/s00034-022-02012-8

Design of the Processors for Fast Cosine and Sine Fourier Transforms

Ivan Tsmots[1] · **Vasyl Rabyk**[2] · **Natalia Kryvinska**[3,4] · **Mykhaylo Yatsymirskyy**[4] · **Vasyl Teslyuk**[1]

Received: 2 November 2020 / Revised: 11 March 2022 / Accepted: 11 March 2022 /
Published online: 11 April 2022

Abstract
To solve large number of digital signal processing problems, such as on-board radar-location or hydro-acoustic systems, it is necessary to perform discrete trigonometric transforms over intensive data flows in real time with the constraints on size and power consumption. To solve this problem, the hardware implementation in the form of the VLSI has been proposed. In particular, we improve an algorithm for the fast cosine and sine Fourier transforms with a focus on the parallel-streaming hardware implementation. A flow graph of the improved algorithm has been developed on the basis of addition, subtraction and multiplication of real numbers with the relation scheme of algorithms. A linear projection of the improved algorithm for fast cosine and sine Fourier transforms on the axis parallel to the data transmission has been obtained. This makes it possible to change the type and dimensions of the transforms. Further, we develop a structure of 2-4-8-16-point processor for fast cosine and sine Fourier transforms. Such an implementation provides a reduction of the dimensions, energy consumption and performance of the transforms in real time.

Keywords Fast cosine and sine Fourier transforms · 2-4-8-16-point processor · Very large-scale integration (VLSI) · Altera Cyclone III FPGA EP3C16F484N6

1 Introduction

To solve a large number of digital signal processing problems [3, 7, 24], it is necessary to perform discrete trigonometric transforms over intensive data flows in real time on

✉ Natalia Kryvinska
natalia.kryvinska@uniba.sk

✉ Vasyl Teslyuk
vasyl.m.teslyuk@lpnu.ua

Extended author information available on the last page of the article

means that meet the constraints on size and power consumption. The development of new algorithms and parallel structures that focus on very large-scale integration is the main way to meet such requirements. Among the many parallel structures, a special place is occupied by parallel-flow structures, which provide high efficiency of hardware loading and best meet the conditions of real-time operation. In parallel-flow structures, the data are processed on a pipeline basis [25, 29].

Pipelining involves the division of structures into steps, each of which consists of two units, that is, operating unit and buffer memory. Control over parallel-flow tools is reduced to the clock pulses that move information from input to output, recording intermediate results into the buffer memory. The frequency of the clock pulses is determined by the access time to the buffer memory and the delay time of the operating unit. To ensure high efficiency of equipment, it is necessary that all pipeline steps have approximately the same time of operation implementations. Parallel-stream implementation of discrete trigonometric transforms provides high efficiency of hardware loading and most meets the real-time operating conditions.

To implement discrete trigonometric transforms, it is proposed to use discrete cosine Fourier transform (DCT) [1, 5, 11] and discrete sine Fourier transform (DST) [10, 16, 20] as the main components. These transforms provide redundant execution of discrete Fourier transforms and discrete Hartley transforms [13, 21] of complex, real, complex conjugate, even or odd sequences. The use of DCT and DST provides parallel processing of real, complex and complex conjugate sequences.

Therefore, the urgent task is to develop fast algorithms and parallel-stream structures of DCT and DST, focused on VLSI implementation.

2 Related work

Among the known [3, 5, 7, 10, 16, 20, 21, 24] fast algorithms for cosine Fourier transform and sine Fourier transform, the algorithms developed on the basis of Rader-Brenner method (CFTRB and SFTRB) are easy to implement [37]. The peculiarity of such algorithms is that their graphs are basically with the relation scheme of algorithms with basis 2, but due to the number of multiplications they correspond to the splitting algorithms and have a very simple basic operation "real butterfly" [35]. A comparative analysis of the fast CFTRB and SFTRB algorithms (FCFTRB and FSFTRB) and the well-known graph of the Cooley-Tukey complex FFT algorithm with basis 2 and frequent thinning [4, 6, 37] shows the coincidence of their structures when $m = \log_2 N$, where N is the dimension of the transform, the main stages of transition to transformations of less dimension. A significantly simplified basic operation performed on real data with valid phase factors is performed.

In [2, 30, 35], fast algorithms for implementing Fourier transform were developed, which are focused on the use of mass-parallel computing tools with a large amount of memory (graphics processing units) [30, 35]. GPUs belong to the class of SIMD processors (Single Instruction Multiple Data), the peculiarity of which is to use a single operation to simultaneously process multiple independent data. Compute unified device architecture is used to develop software for parallel image processing based on CPU and GPU [2], which reduces program development time and improves its quality

[18, 36]. The disadvantage of this implementation is that it cannot provide intensive data flows processing in real time.

Analysis of publications [2, 3, 5, 7, 10, 11, 16, 20, 21, 24, 30, 35, 37] shows that hardware implementation with extensive use of spatial and temporal parallelization of calculating discrete trigonometric transforms is the main way to increase performance. Fast algorithms for calculating cosine and sine Fourier transforms can be implemented on customized VLSI [22, 31], field-programmable gate array (FPGA) [15, 26] and digital signal processors (DSP) [14, 17, 27]. The disadvantage of implementation on customized VLSI is that such an implementation is appropriate only for large series. The disadvantage of implementing discrete trigonometric transforms on DSP is that the performance of DSP is not enough to process intensive data flows in real time. Implementation of fast algorithms for cosine and sine Fourier transforms on the basis of FPGA is currently the most appropriate.

The conducted analysis shows that the most adapted to the processing of intensive data flows are parallel-flow structures, which should be implemented on the basis of FPGA. However, in these publications little attention is paid to the development of algorithms and VLSI processors, which are programmed due to the dimension of the transform using FPGA.

Therefore, the topical tasks of today are to improve the algorithms for calculating DCT and DST, to develop and implement low-point parallel-flow processor on FPGA and on its basis synthesize the matrix pipeline processor for multi-point fast algorithms for cosine Fourier transform (FCFT) and fast algorithms for sine Fourier transform (FSFT).

The purpose of the research is to enhance calculation efficiency of DCT and DST on the basis of the improved calculation algorithms and development as well as implementation of N-point matrix pipeline processor FCFT and FSFT on FPGA. To attain the goal, the following tasks must be fulfilled:

- To improve the calculation algorithms DCT and DST focusing on implementation in the form of VLSI that will ensure parallel-flow data processing.
- To develop a structure of 2-4-8-16-point processor of fast cosine and sine Fourier transforms that will ensure a change of the dimension and type of transform (sine or cosine) in the program implementation.
- To implement the low-point processor FCFT and FSFT on FPGA as a basic component for synthesis of high-performance N-point processor.
- To synthesize the N-point matrix pipeline processor FCFT and FSFT that will ensure resynching of intensity of data coming with calculation intensity. This will make it possible to enhance efficiency of the use of the equipment.

3 Algorithms and tools for DCT and DST implementations

3.1 Improving the Algorithms for DCT and DST implementations

For N-point valid sequence $x(n)$, $n = 0, 1, \ldots, N$ formulas for DCT and DST calculations are as given in [3, 7, 37]:

$$H_C(k) = DCFT_N\{x(n)\} = \sum_{n=0}^{N-1} x(n)C_N^{kn}, \quad k = 0, 1, \ldots, N-1 \tag{1}$$

$$H_S(k) = DSFT_N\{x(n)\} = \sum_{n=0}^{N-1} x(n)S_N^{kn}, \quad C_N^r = \cos(2nr/N), \quad S_N^r = \sin(2nr/N) \tag{2}$$

Assume that N is a complex number that can be represented as the product of two other numbers L and M, that is, $N = LM$. The main idea of DCT and DST algorithms is to decompose the N-point DCT and DST into smaller-sized transforms. To do this, the input $x(n)$ and output $H_C(k)$, $H_S(k)$ sequences are numbered as follows:

$$n = Ml + m, l = 0, 1, \ldots, L-1, \quad m = 0, 1, \ldots, M-1 \tag{3}$$

$$k = Lr + s, \quad s = 0, 1, \ldots, L-1, \quad r = 0, 1, \ldots, M-1 \tag{4}$$

Substituting (3) and (4) into (5), (6), we receive:

$$H_C(sr) = \sum_{m=0}^{M-1}\sum_{l=0}^{l-1} x(lm)C_N^{(Ml+m)(Lr+s)} \tag{5}$$

$$H_S(sr) = \sum_{m=0}^{M-1}\sum_{l=0}^{l-1} x(lm)S_N^{(Ml+m)(Lr+s)} \tag{6}$$

Functions $C_N^{(Ml+m)(Lr+s)}$ i $S_N^{(Ml+m)(Lr+s)}$ can be written as follows:

$$C_N^{(Ml+m)(Lr+s)} = C_N^{MLr} \times C_N^{Mls} \times C_N^{Lrm} \times C_N^{ms} = C_L^{ls} \times C_M^{rm} \times C_N^{ms} \tag{7}$$

$$S_N^{(Ml+m)(Lr+s)} = S_N^{MLr} \times S_N^{Mls} \times S_N^{Lrm} \times S_N^{ms} = S_L^{ls} \times S_M^{rm} \times S_N^{ms} \tag{8}$$

Given the appearance of formulas (7) and (8), formulas (5) and (6) can be written in the following way:

$$H_C(sr) = \sum_{m=0}^{M-1} C_M^{mr}[C_N^{mr}\{\sum_{l=0}^{l-1} x(lm)C_L^{sl)}\}] \tag{9}$$

$$H_S(sr) = \sum_{m=0}^{M-1} S_M^{mr}[S_N^{mr}\{\sum_{l=0}^{l-1} x(lm)S_L^{sl)}\}] \tag{10}$$

Calculation algorithms $H_C(k)$ and $H_S(k)$ based on formulas (9) and (10) require for the following steps:

Step1. We determine M of L-point transforms of subsequences $x(l, m)$ in accordance with the values of formulas (9) and (10) in parentheses, i.e.,

$$g_c(s, m) = \sum_{l=0}^{L-1} x(l, m) C_l^{sl} \tag{11}$$

Step 2. We find the intermediate arrays $h_c(s, m)$ and $h_s(s, m)$, $h_c(s, m)$ (see square brackets, respectively, (9) and (10))

$$h_c(s, m) = q(s, m) C_N^{ms}, h_s(s, m) = q(s, m) S_N^{ms} \tag{12}$$

Step 3. We determine L of M-point transforms of arrays $h_c(s, m)$ and $h_s(s, m)$

$$H_C(sr) = \sum_{m=0}^{M-1} h(sm) C_M^{mr}, H_C(sr) = \sum_{m=0}^{M-1} h(sm) C_M^{mr} \tag{13}$$

To develop a small-point parallel-stream VLSI processor for FCFT and FSFT with the ability to change the dimensions of the transforms, the FCFTRB and FSFTRB algorithm was improved. The flow graph of the improved algorithm of 2-4-8-16-point FCFTRB and FSFTRB [16, 28, 39] is given in Fig. 1, where q is a phase factor [32].

The peculiarity of the developed flow graph of the algorithm for 2-4-8-16-point FCFTRB and FSFTRB is that it is focused on parallel-stream VLSI implementation.

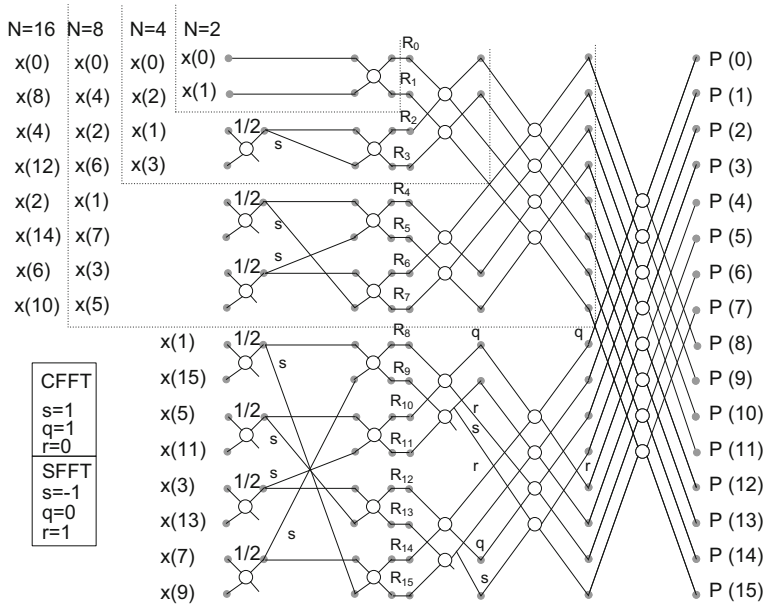

Fig. 1 Flow graph of the improved algorithm for 2-4-8-16-point FCFTRB and FSFTRB

In this flow graph of the algorithm, the choice of FCFT or FSFT is made using the appropriate special multipliers.

Special multipliers while implementing 16-point FCFT are the following ones: $R_0 = R_1 = R_2 = R_4 = R_6 = 1, R_3 = R_5 = 0, R_7 = R_{12} = R_{14} = -\sqrt{2}, R_8 = R_{10} = -2,, R_9 = -R_{13} = 2C_{16}^2 S_{16}^1, R_{11} = R_{15} = -2C_{16}^2 S_{16}^3$ and during implementation of FCFT they are presented by: $R_0 = R_1 = R_7 = 0, R_2 = R_3 = R_4 = R_6 = 1, R_5 = R_8 = R_{10} = \sqrt{2}, R_{12} = R_{14} = -2, R_9 = R_{13} = 2C_{16}^2 S_{16}^3, R_{11} = -R_{15} = 2C_{16}^2 S_{16}^1$.

In addition to performing 16-point FCFT or FSFT, the above algorithm provides the ability to implement 2-point, 4-point and 8-point transforms correspondingly. The flow graph of the algorithm for 2-4-8-16-point FCFTRB and FSFTRB has been improved by reducing in the ratio 2 the number of stages with the use of multiplication operation. This ensured the use of only one multiplication device in its parallel-stream hardware implementation.

The basic operations, which are used to implement the improved algorithm for FCFTRB or FSFTRB, are given in Fig. 2.

The improved algorithm for 2-4-8-16-point FCFTRB or FSFTRB is implemented on basic operations of the first and second types, which are performed on real numbers. The basic operation of the first type is reduced to the addition and subtraction, and the basic operation of the second type is reduced to the addition, subtraction and multiplication operations. 16-point algorithm for FCFTRB or FSFTRB is focused on parallel-stream hardware implementation by performing the multiplication operation only at one stage.

For parallel-stream hardware implementation of the improved 16-point FCFTRB or FSFTRB algorithm, it is necessary to obtain its linear projection on the axis parallel to data transmission. The linear projection of the improved algorithm of 2-4-8-16-point FCFTRB or FSFTRB on an axis parallel to data transmission is given in Fig. 3, where F_{EOCS} is operator for even and odd components selection, F_{SSP}—switching, storage

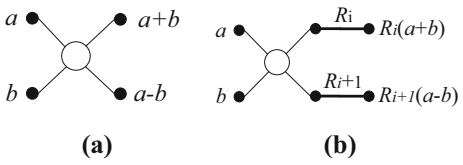

Fig. 2 The basic operations of the improved algorithm for 2-4-8-16-point FCFTRB or FSFTRB: **a** first type; **b** second type

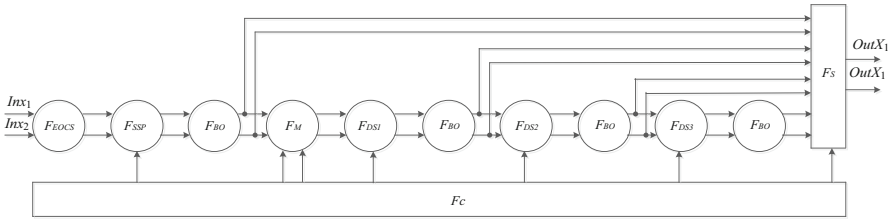

Fig. 3 A linear projection of the improved 2-4-8-16-point algorithm for FCFTRB or FSFTRB

and data permutation operator, F_{BO}—functional operator for performing the basic operation, F_M—functional multiplication operator, F_{DS_1}, F_{DS_2} and F_{DS_3}—the first, second and third delay and switching operators, F_S—switching operator, F_C—control operator.

A peculiarity of the linear projection of the improved 16-point algorithm for FCFTRB or FSFTRB is the ability to change the type (FCFT or FSFT) and the dimensions of the transforms (2-4-8-16 point) by introducing control F_C and switching F_S operators. Control operator F_C provides the spacious and temporal deployment of the FCFT or FSFT calculation process, and the switching operator F_S provides a change in the dimension of the transform. The main functional operators of the linear projection of the improved 16-point algorithm for FCFTRB or FSFTRB are switching, storage and data permutation F_{SSP} operators and the functional operator for performing basic operation F_{BO} "real butterfly."

3.2 Developing 2-4-8-16-Point VLSI processor for FCFT and FSFT

For parallel-stream implementation of the improved 2-4-8-16-point algorithm for FCFTRB or FSFTRB as a VLSI-processor [38], it has to operate on the pipeline principle. Pipelining of the VLSI processor involves its division into steps by introducing the buffer memory. In this case, each step of the pipeline consists of two components, i.e., operating units and switching units, data storage and permutation [9, 12, 29]. To make the most of the benefits of VLSI technology, it is proposed to develop a parallel-stream processor for FCFT or FSFT based on the following principles:

- modularity of structure;
- use of a system of elementary arithmetic operations;
- localization and simplification of relations between the steps of the pipeline;
- ensuring balance between input/output and calculations;
- use of pipelining and spatial parallelization of the calculation process;
- minimization of the external communication interface.

The parallel-stream VLSI processor for FCFT or FSFT is implemented by hardware display of a linear projection of the graph of the improved 2-4-8-16-point algorithm for FCFTRB or FSFTRB. The block diagram of the parallel-stream processor for 2-4-8-16-point FCFT or FSFT is shown in Fig. 4, where CP is the input of the clock pulses; TD—input for selecting the transform dimension; cos/sin—input to select the type of transform; $EOCSU$—even and odd components selection unit, Rg—register, Sw—switch, CU—control unit, PE—processor element [34].

The processor for 2-4-8-16-point FCFT-FSFT is implemented on the basis of $EOCSU$ and four series-connected PE. The processor for FCFT-FSFT is controlled by the CU, which consists of non-volatile memory, address counter and registers. The values of phase multipliers and control signals for $EOCSU$, four PE_1-PE_4 and Sw are stored in non-volatile memory at certain addresses. The main components of the structure of the parallel-stream processor (Fig. 4) are $EOCSU$, four PE, four input Sw, destination registers Rg_1, Rg_2 and CU, which reflect in hardware a linear projection of the improved 2-4-8-16-point algorithm for FCFTRB and FSFTRB.

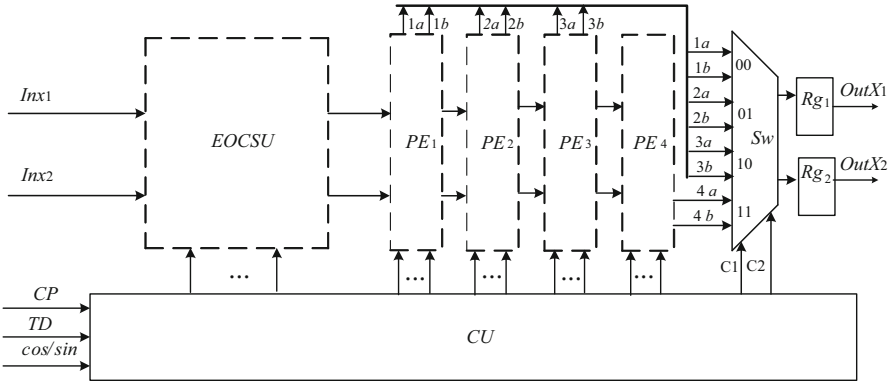

Fig. 4 The block diagram of 2-4-8-16-point processor for FCFT or FSFT

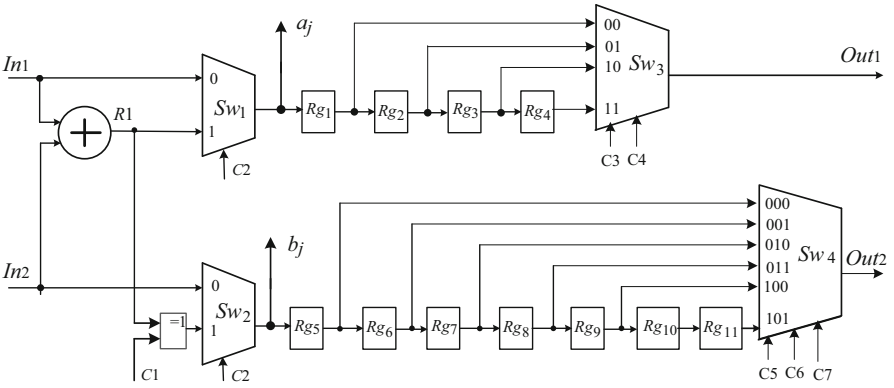

Fig. 5 Block diagram of $EOCSU$

The given components and relations between them fully reflect the structure of the parallel-stream processor for 2-4-8-16-point FCFTRB and FSFTRB.

Block diagram of $EOCSU$ is shown in Fig. 5, where $C1$-$C6$ are control signals, In_1, In_2—the data inputs, Out_1, Out_2—the outputs of even and odd components.

The $EOCSU$ includes the following main components: adder, switches Sw_1-Sw_4 and delay registers Rg_1-Rg_{11}. Input data are fed to the inputs of the adder and the first inputs of the switches Sw_1 and Sw_2. The value $a + b$ is obtained at the outputs of the adder, which is shifted one digit right and fed to the second input of the switch Sw_1 and the first input of the group of logic elements EXCLUSIVE OR. The value of the control signal $C1$ provides multiplication of data by coefficients s ($C1 = 0$ multiplication by 1, $C1 = 1$ multiplication by minus 1). Switches Sw_1 and Sw_2 are controlled by the signal $C2$ ($C2 = 0$ data transfer from the first inputs, $C2 = 1$ data transfer from the second inputs). Using the signals $C3$-$C4$ and $C5$-$C6$, the number of

Table 1 The time diagram of the $EOCSU$ performance with the 16-point FCFT-FSFT

Time	$In_1 \ldots In_2$	C1	C2	Rg_1 Rg_5	Rg_2 Rg_6	Rg_3 Rg_7	Rg_4 Rg_8
0	x(0) x(8)	0	0				
1	x(4) x(12)	1	1	a1 b1			
2	x(2) x(14)	1	1	a2 b2	a1 b1		
3	x(6) x(10)	1	1	a3 b3	a2 b2	a1 b1	
4	x(1) x(15)	1	1	a4 b4	a3 b3	a2 b2	a1 b1
5	x(5) x(11)	1	1	a5 b5	a4 b4	a3 b3	a2 b2
6	x(3) x(13)	1	1	a6 b6	a5 b5	a4 b4	a3 b3
7	x(7) x(9)	1	1	a7 b7	a6 b6	a5 b5	a4 b4
8	x(0) x(8)	1	1	a8 b8	a7 b7	a6 b6	a5 b5
9	x(4) x(12)	0	0	a1 b1	a8 b8	a7 b7	a6 b6
10	x(2) x(14)	1	1	a2 b2	a1 b1	a8 b8	a7 b7
11	x(6) x(10)	1	1	a3 b3	a2 b2	a1 b1	a8 b8

Time	Rg_9	Rg_{10}	Rg_{11}	C3 C4	C5 C6 C7	Out_1 Out_2
0				11		
1				11		
2				11		
3				11		
4				11	011	a1 b1
5	b1			11	011	a2 b2
6	b2	b1		11	010	a3 b4
7	b3	b2	b1	11	100	a4 b3
8	b4	b3	b2	11	000	a5 b8
9	b5	b4	b3	11	010	a6 b7
10	b6	b5	b4	11	100	a7 b6
11	b7	b6	b5	11	101	a8 b5

data delay cycles is selected in accordance with the flow graph of the algorithm for FCFT-FSFT (Fig. 1).

The time diagram of the $EOCSU$ performance with the 16-point FCFT-FSFT is given in Table 1.

The time diagram (Table 1) shows how the data move in $EOCSU$ EOCSU from inputs In_1 and In_2 to outputs Out_1 and Out_2 while performing from 16-point FCFT-FSFT. In addition to that, the given diagram shows how the state of control signals changes at inputs C1,…, C7 while performing from 16-point FCFT-FSFT.

In Table 1 the values a1,…,a8 and b1,…,b8 are determined the following way:

$$a1 = x(0);\ b1 = x(8);\ a2 = 1/2[x(4) + x(12)];\ b2 = -1/2[x(4) + x(12)];$$
$$a3 = 1/2[x(2) + x(14)];\ b3 = -1/2[x(2) + x(14)];\ a4 = 1/2[x(6) + x(10)];$$
$$b4 = -1/2[x(6) + x(10)];$$
$$a5 = 1/2[x(1) + x(15)];\ b5 = -1/2[x(1) + x(15)];\ a6 = 1/2[x(5) + x(11)];$$
$$b6 = -1/2[x(5) + x(11)];$$
$$a7 = 1/2[x(3) + x(13)];\ b7 = -1/2[x(3) + x(13)];\ a8 = 1/2[x(7) + x(9)];$$
$$b8 = -1/2[x(7) + x(9)].$$

The first PE_1 performs calculations of the basic operation of the second type $a* = (a + b)R_i$, $b* = (a - b)R_{i+1}$, and others (PE_2, PE_3, PE_4) are responsible for data transfer delay and the performance of the basic operation of the first type $a* = a + b$, $b* = a - b$. The structure of PE_1 is shown in Fig. 6.

To increase the performance of PE_1, the pipeline registers Rg_1–Rg_4 are added to it, which provide a reduction of the pipelining cycle. Multiplication in PE_1 is performed with ordinary accuracy.

The generalized structure of PE_2, PE_3 and PE_4 is shown in Fig. 7. These processor elements include control signals $C1$-$C5$, adder and subtractor, groups of logic elements AND and EXCLUSIVE OR, switches Sw_1 and Sw_2 and delay devices, which consist of a set of series-connected registers (Rg_1–$Rg_{2^{j-2}}$).

A peculiarity of the processor elements PE_2, PE_3 and PE_4 is a different number of data transmission delay cycles, the value of which is determined by its sequence number j ($j = 2, 3, 4$) and is equal to $Z_j = 2^{j-2}$. The control signals $C1$-$C5$ are designed for switching and data transmission control. Signal $C1$ provides switching

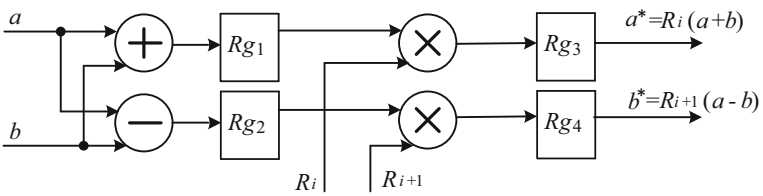

Fig. 6 PE_1 structure

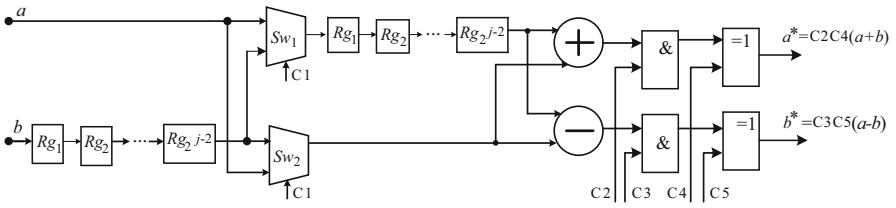

Fig. 7 Generalized structure of PE_2, PE_3 and PE_4

of switches Sw_1 and Sw_2 ($C1 = 0$—data transfer from the first inputs, $C1 = 1$—data transfer from the second inputs), signals $C2$, $C3$, $C4$ and $C5$ provide multiplication of data by coefficients s, q, r ($C2 = 1$, $C3 = 1$—multiplication by 1, $C2 = 0$, $C3 = 0$—data multiplied by 0, $C4 = 0$, $C5 = 0$—data multiplied by 1, $C4 = 1$, $C5 = 1$—data multiplied by minus 1).

The processor provides 2-4-8- or 16-point FCFT-FSFT, while the clock frequency of data input and data output does not depend on the dimension of the transform. Before starting to work the size of the transform is selected by control signals, which are fed to the input of the TD and the type of the transform (log .1—FCFT, log .0—FSFT) is selected by the signal at the input cos/sin.

The processor is controlled by the CU, which generates control signals for switches Sw, switching and delay circuits $SDS1$ and $SDS2$ and supplies the phase multipliers R_i, R_{i+1} to the inputs of the multiplication devices of PE_1. The dimension of transfer is chosen by the switch Sw (Fig. 4) and the switches Sw_4 i Sw_5 (Fig. 5). For the choice of 2-, 4-, 8- and 16-тpoint FCFT-FSFT at the inputs C1 and C2 of the switch Sw and at the inputs C3, C4 of the switch Sw_4 the next values are given: 00, 01, 10 i 11. The switch Sw_5 is controlled by the signals C5, C6 and C7, at which for performance of 2- and 4-point FCFT-FSFT the next values come: 000 i 001. While performing 8-point FCFT-FSFT by the signals C5, C6 and C7, the next values are got: 010, 010, 001 and 011. The values of signals C5, C6 i C7 with 16-point FCFT-FSFT are given in Table 1.

Developed 2-4-8-16-point processor for FCFT-FSFT works on the pipeline principle with the clock cycle $T_k = t_{Rg} + t_{MD}$, where t_{Rg} and t_{MD} are the operating time of the register and the multiplication device, respectively. The maximal initial delay determined by the number of embedded pipelined stages while performing from 16-point FCFT-FSFT is equal to 13 clock cycles.

The hardware costs for the implementation of such a processor are determined as follows:

$$W_{16-\mathrm{Pr}} = W_{\mathrm{EOCSU}} + W_{\mathrm{PE1}} + W_{\mathrm{PE2}} + W_{\mathrm{PE3}} + W_{\mathrm{PE4}} + 2W_{\mathrm{Sw4\,-\,1}} + 2W_{Rg} + W_{CU} \tag{14}$$

where W_{EOCSU} is hardware costs for even and odd components selection unit, $W_{\mathrm{PE_1}}$, $W_{\mathrm{PE_2}}$, $W_{\mathrm{PE_3}}$, $W_{\mathrm{PE_4}}$—hardware costs for the 1st, 2nd, 3rd and 4th processor elements, respectively, $W_{\mathrm{Sw_{4-1}}}$—hardware costs for the switch 4 in 1, W_{Rg}—hardware costs for the register, W_{CU}—hardware costs for the control unit. To estimate the hardware costs for the implementation of the 16-point processor for FCFT-FSFT, it is necessary to calculate the costs for the implementation of the main components ($EOCSU$, PE_1, PE_2, PE_3, PE_4 and CU). Hardware costs for the implementation of the main components based on functional units are determined as follows:

$$W_{EOCSU} = W_{Ad} + 2(W_{Sw_{2-1}} + W_{Sw_{4-1}}) + 8W_{Rg} + W_{ExcOR}, \tag{15}$$

$$W_{PE1} = W_{Ad} + W_{Sb} + 4W_{Rg} + 2W_{MD}, \tag{16}$$

$$W_{PE2} = W_{Ad} + W_{Sd} + 2W_{Sw_{2-1}} + 2W_I + 2W_{ExcOR} + 2W_{Rg}, \tag{17}$$

$$W_{PE3} = W_{Ad} + W_{Sb} + 2W_{Sw_{2-1}} + 2W_I + 2W_{ExcOR} + 4W_{Rg}, \tag{18}$$

$$W_{PE4} = W_{Ad} + W_{Sd} + 2W_{Sw_{2-1}} + 2W_I + 2W_{ExcOR} + 8W_{Rg}, \tag{19}$$

$$W_{CU} = 4W_{Rg} + W_{Cl} + W_M, \tag{20}$$

where $W_{Ad}, W_{Sb}, W_{Rg}, W_{Sw_{2-1}}, W_{Sw_{4-1}}, W_{MD}, W_{Cl}, W_M, W_I, W_{ExcOR}$, are the hardware costs for the adder, subtractor, register, switch 2 in 1, switch 4 in 1, multiplication device, counter, memory, logic elements AND and EXCLUSIVE OR, respectively. Substituting the values from formulas (15)–(20) into formula 14 we obtain:

$$\begin{aligned} W_{16-\mathrm{Pr}} = {} & 4W_{Ad} + 3W_{Sd} + 2W_{MD} + 32W_{Rg} + W_{Cl} + W_M \\ & + 8W_{Sw_{2-1}} + 4W_{Sw_{4-1}} + 6nW_I + 7nW_{ExcOR} \end{aligned} \tag{21}$$

To estimate the costs for processor implementation, a logic valve, which is an element of the inverter, AND, OR type, is taken as a measurement unit. The equipment costs for implementation of functional nodes in the valves are given in Table 2 [35].

In Table 2, the analytical expressions are given for estimating the costs in valves for implementation of functional nodes depending on their bitness n. In order to estimate the costs in valves for processor implementation, it is necessary to determine the number and bitness of each type of functional node needed for processor implementation. The estimate of the costs in valves for processor implementation is done by adding the costs for implementation of logical elements of inverter type and, or the costs for the determined number and bitness of functional nodes.

Substituting the values of the implementation costs for functional units given in Table 1 into formula (21), we receive:

$$W_{16-\mathrm{Pr}} = 484n + 20n^2 \tag{22}$$

Table 2 The equipment costs for implementation of functional nodes

N п/п	Functional nodes	Equipment (valves) costs
1	EXCLUSIVE OR	3
2	Register	$7n$
3	n-bit adder	$18n$
4	n-bit subtractor	$18n$
5	Multiplication device	$10n^2$
6	Counter	$11n$
7	m-input, n- bit switch	2 mn
8	m-input, n-bit memory	$2^m n$

Formula (22) allows us to estimate the equipment costs in the valves for the implementation of 16-point processor FCFT-FSFT.

3.3 Implementing 2-4-8-16-point processor components of FCFT-FSFT on FPGA

To implement 2-4-8-16-point processor for FCFT-FSFT, Altera Cyclone III FPGA EP3C16F484N6 [8, 19, 33] with the Quartus II development environment was chosen. The main components of the 2-4-8-16-point processor for FCFT-FSFT are processor elements PE1–PE4, which are implemented as separate modules. The input data processed by FCFT-FSFT processor are 16-bit. Data processing is carried out in a fixed-point format, which is fixed before the highest digit. A necessary condition for the use of this arithmetic is input data and special factor transforms so that $|x(n)| < 1$ та $|R_i| < 1$. The values of the special factors R_i for the 16-point transform are given in Table 3.

Using FPGA, a circuit of the first processor element PE_1 was developed, which implements the basic operation of the second type $a* = (a+b)R_i$, $b* = (a-b)R_{i+1}$. The circuit of the processor element PE_1 is shown in Fig. 8, where Clk is clock pulses, InA, InB—16-bit data inputs, R_Wj—inputs of phase multipliers, A_Out, B_Out—outputs of the basic operation results of the second type.

The main components on the basis of which the first processor element PE_1 is implemented are the following ones: Shift_R_1—one-digit right shift device,

Table 3 The values of the special factors R_i

i	FCFT		FSFT	
	R_i(DEC)	R_i(HEX)	R_i(DEC)	R_i(HEX)
0	0.5000000	0 × 4000	0	0
1	0.5000000	0 × 4000	0	0
2	0.5000000	0 × 4000	0.5000000	0 × 4000
3	0	0	0.5000000	0 × 4000
4	0.5000000	0 × 4000	0.5000000	0 × 4000
5	0	0	0.7071068	0 × 5A82
6	0.5000000	0 × 4000	0.5000000	0 × 4000
7	− 0.7071068	0 × A57E	0	0
8	− 1.0000000	0 × 8001	0.7071068	0 × 5A82
9	0.2705981	0 × 22A2	0.6532815	0 × 539E
10	− 1.0000000	0 × 8001	0.7071068	0 × 5A82
11	− 0.6532815	0 × AC62	0.2705981	0 × 22A2
12	− 0.7071068	0 × A57E	− 1.0000000	0 × 8001
13	− 0.2705981	0 × DD5E	0.6532815	0 × 539E
14	− 0.7071068	0 × A57E	− 1.0000000	0 × 8001
15	− 0.6532815	0 × AC62	− 0.2705981	0 × DD5E

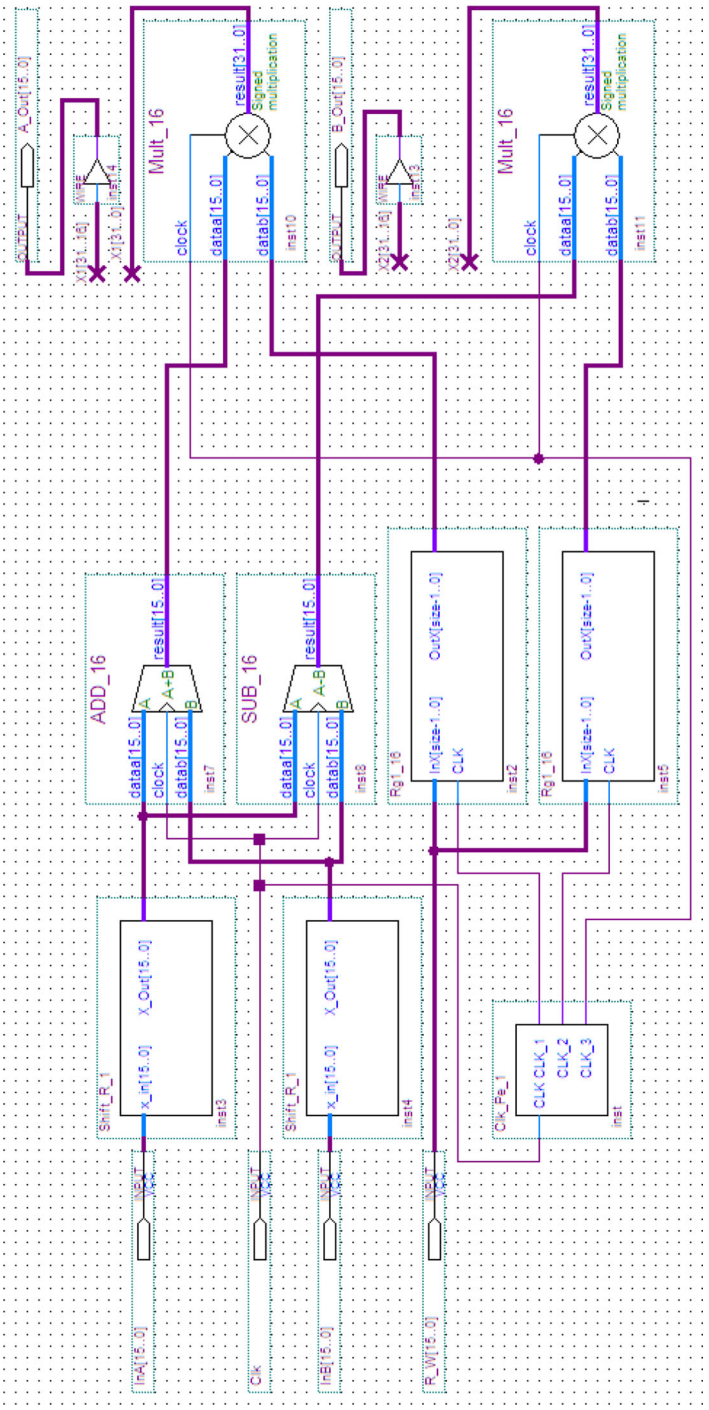

Fig. 8 A circuit of the first processor element PE_1

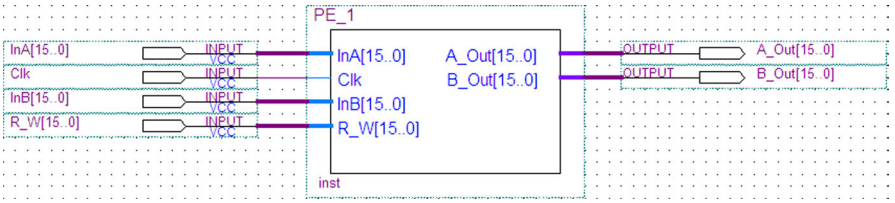

Fig. 9 Graphical representation and interface of the processor element PE_1

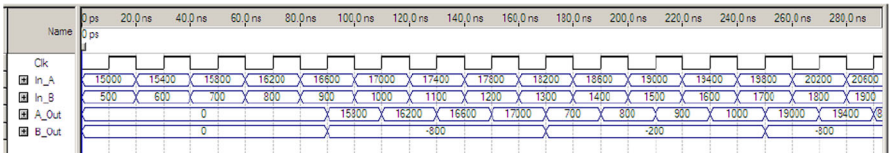

Fig. 10 Time diagrams of the processor element PE_1 operation

ADD_16—16-bit adder, SUB_16—16-bit subtraction device, Mult_16—16-bit multiplication device, Rg1_16—16-bit register, Clk_Pe_1—synchronization unit. Data from inputs In_A, In_B are fed to the shift device Shift_R_1, which perform right one-digit shift, i.e., division by two. From the outputs of the shift devices Shift_R_1 data are fed to the inputs of the adder ADD_16 and the subtraction device SUB_16, at the outputs of which we obtain the sum of $a+b$ and the difference $a-b$, respectively. The operation of the adder ADD_16 and the subtraction device SUB_16 is synchronized with the clock pulses Clk. Special multipliers R_j from the inputs R_Wj are written to the registers Rg1_16 by the clock pulses Clk_1 and Clk_2. With the help of hardware multipliers Mult_16 the multiplication of the sum $a+b$ and the difference a-b by the phase factors from the outputs of the registers Rg1_16 is performed. At the outputs A_Out and B_Out we receive the products $(a+b)R_j$ and $(a-b)R_{j+1}$, which come from the outputs of the multiplication devices Mult_16. Graphical representation and interface of the first processor element PE_1 are shown in Fig. 9.

Time diagrams of the processor element PE_1 are shown in Fig. 10. The time diagram (Fig. 10) determines the length of clock cycle time of the processor element PE_1 and shows step-by-step data movement in the given processor element while performing the basic operation of the second type $a^* = (a+b)R_i$, $b^* = (a-b)R_{i+1}$.

It is seen in the time diagrams that the results of the calculations $(a+b)R_j$ and $(a-b)R_{j+1}$ at the outputs A_Out and B_Out $(a-b)R_{j+1}$ are obtained during two clock pulses Clk. The time diagram shows an example of operation of the processor element PE_1 for values InA = 5000, InB = 1000, R_Wj = 16,384, R_Wj + 1 = 16,384.

The structures of the processor elements PE_2, PE_3 and PE_4 are developed, which implement the basic operation of the first type $a* = a+b$, $b* = a-b$. The circuit of the processor element PE_3, which implements the basic operation of the first type, is shown in Fig. 11.

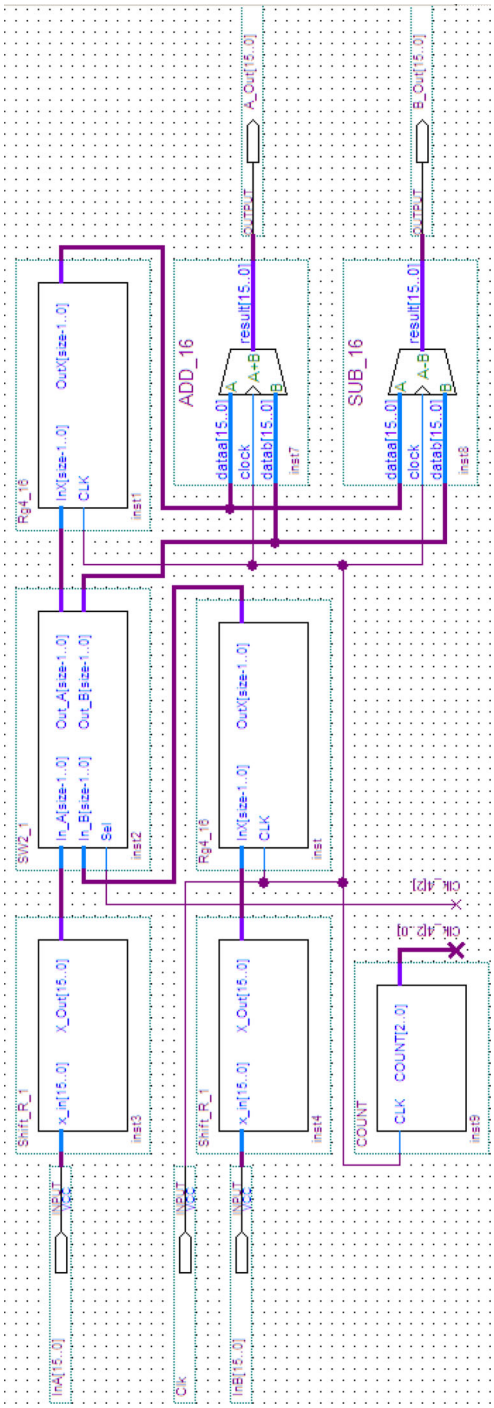

Fig. 11 The circuit of the processor element PE_3

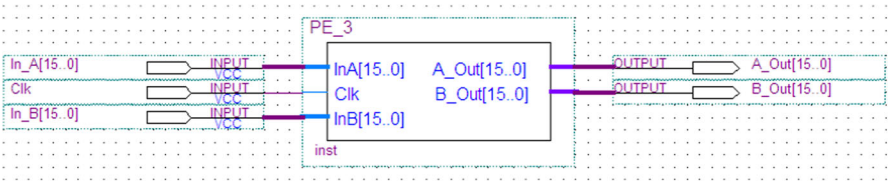

Fig. 12 Graphical representation of the processor element (PE_3)

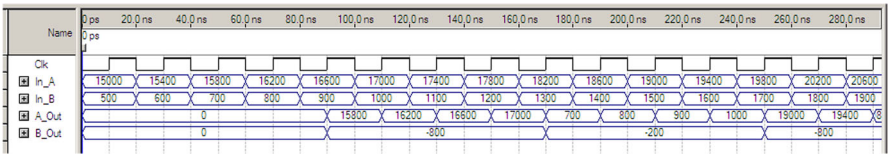

Fig. 13 Time diagrams of the processor element PE_3 operation

A peculiarity of the processor elements PE_2, PE_3 and PE_4 is the uniformity of the structure. These processor elements differ from each other only in a different number of data transfer delay cycles, the value of which is determined by the sequence number j of the processor element $Z_j = 2^{j-2}$. Graphical representation and interface of one of the processor elements (PE_3) is shown in Fig. 12.

The third processor element PE_3 performs a delay of 4 clock pulses. Data from the inputs In_A, In_B are fed to the inputs of the first and second shift devices Shift_R_1, which perform one-digit right shift. From the outputs of the first shift device, the data are fed to the first input of the switch Sw2_1, and the data from the second shift device are fed to the second input of the switch through the first register Rg4_16. The first register Rg4_16 delays the data to 4 clock pulses Clk. The operation of the switch Sw2_1 is synchronized by the speed Clk / 4. Speed division is performed by the counter Count. The switch Sw2_1 has two 16-bit input buses (In_A, In_B), an input bus selection signal (Sel) and two 16-bit output buses Out_A, Out_B.

If at the input Sel the level of logic is "1," then the input signals are switched to the output crosswise (Out_A = In_B; Out_B = In_A). Otherwise, the signals are switched directly to the output (Out_A = In_A; Out_B = In_B). Data from the switch Sw2_1 output Out_A go to the input of the second register Rg4_16 and from the output Out_B data are fed to the inputs of the adder ADD_16 (addend) and SUB_16 (subtrahend). From the output of the second register, the delayed signals are fed to the other inputs of the elements ADD_16 (first term) and SUB_16 (minuend). At the outputs of the elements ADD_16 and SUB_16 the sum $a + b$ and the difference $a - b$ of the input data are obtained.

Time diagrams of the processor element PE_3 are shown in Fig. 13. During the first 4 clock pulses Clk data are not transmitted to the outputs A_Out and B_Out. During the next 4 clock pulses the results of the calculation are obtained at the outputs A_Out, B_Out. Time diagram (Fig. 13) shows an example of the processor element PE_3 operation at the values InA = 15,000, InB = 16,600.

Table 4 The number of components of the processor FCFT-FSFT and the expenses for their implementation

The name of components	The number of components	The number of gates
Adder	9	2592
Multiplication device	2	5120
Memory 16 × 45 bit	1	720
Register	42	4704
4-digit counter	1	44
2-input switch	8	512
4- input switch	3	384
8- input switch	1	256
Elements of logic	240	600

Implementation of the processor element PE_1 requires the use of 68 logic elements, 65 registers, 4 built-in hardware multipliers and 81 outputs. The hardware resources required for the implementation of each of the processor elements PE_2, PE_3, PE_4 are consisted of 66 logic elements, 36 registers, 66 outputs.

The number of the used components for implementation of the 16-digit 2-4-8-16-point processor FCFT-FSFT and the number of gates for their implementation are given in Table 4.

The estimated number of the gates necessary for implementation of the 16-digit 2-4-8-16-point processor FCFT-FSFT equals about 14,932 gates.

On the basis of the developed processor elements, a model of processor of 2-4-8-16-point FCFT or FSFT has been synthesized. The model is used for its work in all performances.

3.4 Implementation of N-point processor for FCFT-FSFT based on VLSI processors for FCFT-FSFT

The main requirement for the N-point processor for FCFT-FSFT is to ensure the processing of data flows in real time with minimal hardware costs. To do this, the following condition must be met:

$$P_d \leq D_{N-\mathrm{Pr}}, \tag{23}$$

where P_d is the intensity of the data flow, $D_{N-\mathrm{Pr}}$ is the calculation intensity of the N-point processor for FCFT-FSFT. The intensity of the data flow to the N-point processor is determined as follows:

$$P_d = hn_d F_d, \tag{24}$$

where h is the number of data channels; n_d is a bit size of data channels; F_d is a data frequency. The calculation intensity of the N-point processor for FCFT-FSFT is

determined as follows:

$$D_{N-\mathrm{Pr}} = 2pn_s/C_{pc}, \tag{25}$$

where C_{pc} is the pipeline clock cycle of the N-point processor, n_s is the bit set of data processing channels in the low-point VLSI processor, p is the number of low-point VLSI processors in the tier.

We synthesize the N-point processor for FCFT-FSFT on the basis of the developed 16-point VLSI processor for FCFT-FSFT, which has two processing paths, the ability to adjust the size (2-4-8- or 16-point) and the type of transfer FCFT or FSFT. To implement the N-point processor for FCFT-FSFT in real time we choose a matrix structure that works on the pipeline principle. For the synthesis of such a processor, the required number of the developed 16-point VLSI processors for FCFT-FSFT is determined by the formula:

$$R = p \times m = \left\lceil \frac{P_d}{D_{N-\mathrm{Pr}}} \right\rceil \times \left\lceil \frac{\log_2 N}{\log_2 L} \right\rceil, \tag{26}$$

where $L = 16$, $\lceil\rceil$—is the sign of rounding to a larger whole. Lower-point VLSI processors form a two-dimensional array of size $p \times m$, where m is the number of series-connected low-point VLSI processors, i.e., tiers of the matrix processor; p is the number of low-point VLSI processors in the tier. In each tier, the last m may be the exception and 16-point FCFT-FSFT is performed. In the last m tier, 2^k-point transform is performed, where $k = \log_2 N - l(m-1)$. The structure of the pipeline matrix processor for N-point FCFT-FSFT is shown in Fig. 14, where SPM is specialized parallel memory, CU—control unit, PS—pipeline step, MD—multiplication device, VLSI low-point processor for FCFT-FSFT, CP—input of clock pulses, CC—input of conversion code.

A peculiarity of the SPM structure is its adaptation to the data structure and to the algorithms for calculating FCFT-FSFT. This memory provides simultaneous access to a set of data, ordering, delay and switching of data flows. The main components of the SPM include the storage, the switching network, the address generators and the control unit. The capacity of each unit of SPM is equal to $Q_{SPM} = 2U_i$, where U is the size of the transform; i is the data width.

Matrix N-point processor for FCFT-FSFT works on the pipeline principle, providing the division of the array of low-point VLSI processors into the pipeline steps by introducing SPM. In this case, each step of the pipeline consists of p low-point VLSI processors, $2p$ multiplication devices and SPM. Different arrays m are processed by N-point matrix pipeline processor. Frequency of change of arrays is determined by the macrocycle of the pipeline, which is equal to the intensity of the calculation of $D_{N-\mathrm{Pr}}$.

Matrix pipeline implementation of the N-point processor for FCFT-FSFT provides high efficiency of loading the hardware and most corresponds to working conditions in real time. The use of SPM for the interaction between the steps of the pipeline minimizes the problems associated with the synchronization of low-point VLSI processors and the N-point matrix processor in general.

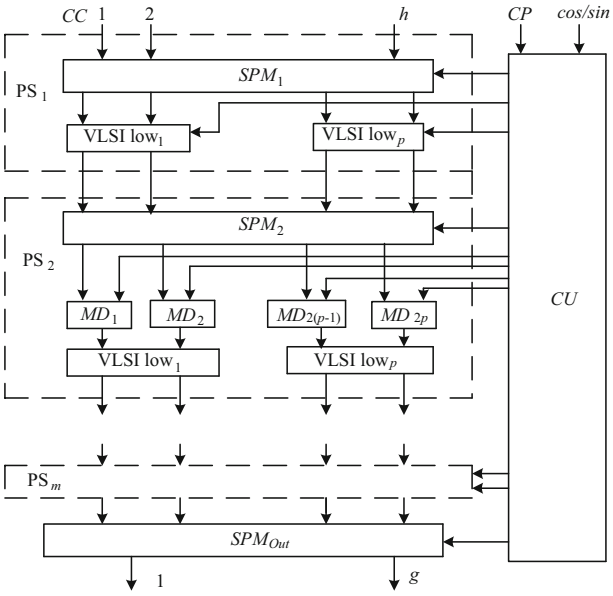

Fig. 14 The structure of the pipeline matrix processor for N-point FCFT-FSFT

A peculiarity of the developed matrix pipeline processor is the ability to adapt to the intensity of data flow. The interface of the developed processor provides simultaneous input of h data and output of g processing results in each clock cycle. Control over the matrix pipeline processor for FCFT-FSFT is reduced to generation of clock pulses that move data from input to output.

Hardware costs for the implementation of matrix pipeline N-point processor for FCFT-FSFT (Fig. 14) are determined by the following formula:

$$W_{N-\mathrm{Pr}} = mpW_{16-\mathrm{Pr}} + (m+1)pW_{SPM} + 2p(m-1)W_{MD} + W_{CU}, \qquad (27)$$

where $W_{16-\mathrm{Pr}}, W_{SPM}, W_{MD}, W_{CU}$ are hardware costs for the 16-point processor for FCFT-FSFT, specialized parallel memory, multiplication device and control unit.

To estimate the hardware costs for matrix pipeline N-point processors for FCFT-FSFT we take $p = \frac{N}{L} = \frac{N}{16}, m = \frac{\log_2 N}{\log_2 L} = \frac{\log_2 N}{4}$ and $n = 16$. Such N-point processors for FCFT-FSFT provide the maximum intensity of calculation $D_{N-\mathrm{Pr}}$. Hardware costs for the implementation of matrix pipeline N-point processors for FCFT-FSFT with the maximum calculation intensity are shown in Fig. 15.

It is possible to reduce hardware costs for the implementation of matrix pipeline N-point processors for FCFT-FSFT by reducing the number of VLSI processors in the pipeline stages. This reduction in the number of VLSI processors will decrease the computational intensity and increase the computation time of N-point FCFT-FSFT.

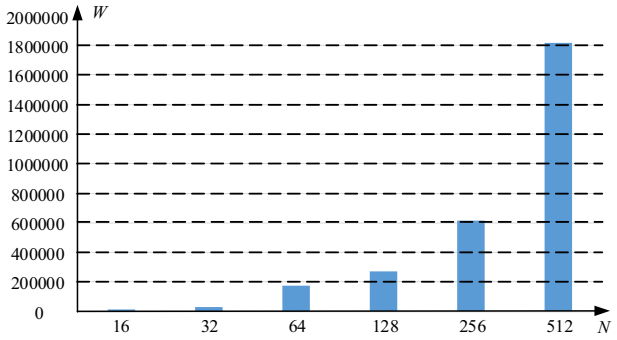

Fig. 15 Hardware costs for the implementation of matrix pipeline N-point processors for FCFT-FSFT (N is a transform dimension, W is a number of gates)

3.5 Discussion of the results of hardware implementation

The developed parallel-stream VLSI processor FCFT-FSFT is oriented toward processing input data in real time. In such processing it is necessary to meet the condition $F_{id} < F_{cc}$, where F_{id} is frequency of input data coming, and F_{cc}—clock cycle frequency of performance of processor pipeline. There are two possible approaches to meet this condition.

The first approach is an increase of F_{cc} by pipeline implementation of multiplication device. In this implementation, the processor can work with pipeline clock cycle equal to $T_k = t_{rRg} + t_{add} + t_{dle}$ $T_k = t_{\Pr} + t_{\mathrm{CM}} + 3t_I$, where t_{rRg} is the time of reading information from register, t_{add}—the time of addition, t_{dle}—the time of information delay by logical element.

The second approach is parallel inclusion two and more VLSI processors FCFT-FSFT. The number of simultaneously inclusioned processors is determined by frequency of input data coming Fd.

4 Conclusions

We improved in this paper an algorithm for the FCFTRB-FSFTRB. We focused on hardware implementation, which by preserving the relation scheme of algorithms with basis 2 and focusing the multiplication operation on one stage of the calculation provides a reduction in hardware costs and implementation on the same type of processor elements.

Besides, a parallel-stream VLSI processor for FCFT-FSFT has been developed, which, due to the possibility of changing phase factors and switching data transmission from the outputs of processor elements to the output of the processor, allows choosing the type of the transform and its dimension.

Even and odd components selection unit and processor elements that perform basic operations of the first and second type have been implemented on Altera Cyclone III FPGA EP3C16F484N6 with the Quartus II development environment.

N-point matrix pipeline processor for fast cosine and sine Fourier transforms, which is highly efficient for the equipment use, was synthesized on the basis of the developed 2-4-8-16-point parallel stream processor.

The developed 2-4-8-16-point processor of fast cosine and sine Fourier transforms, processors of discrete Fourier and Hartley transforms of complex, real, complex-linked, even and odd sequences ensure paralleling of the processing process and their simple performance.

Data availability Our manuscript has no associated data.

Declarations

Conflict of interest The authors declare that they have no conflict of interest/competing interests.

References

1. N. Ahmed, T. Natarajan, K.R. Rao, Discrete cosine transform. IEEE Trans. Comput. **23**, 90–93 (1974)
2. A. Batyuk, E. Struk, I. Tsmots, Development principles and criteria for the selection of VLSI-structures for coordinated parallel calculation of basic operations of real-time digital signal processing algorithms, in *Proceedings of the 9th International Conference on The Experience of Designing and Application of CAD Systems in Microelectronics, CADSM 2007, Lviv-Polyana, Ukraine, 19–24 Feb*. 2007, pp. 179–180.
3. R.E. Blahut, *Fast algorithms for signal processing*. 1ed. Cambridge University Press: New York, USA, 2010
4. N. Brisebarre, M. Joldeş, J.-M. Muller, A.-M. Naneş, J. Picot, Error analysis of some operations involved in the cooley-Tukey fast fourier transform. ACM Trans. Math. Soft. **46**(2), 1–27 (2020)
5. W. Chen, C. Smith, S. Fralick, A fast computational algorithm for the discrete cosine transform. IEEE Trans. Commun. **25**(9), 1004–1009 (1977)
6. J.W. Cooley, J.W. Tukey, An algorithm for machine calculation of complex Fourier series. Math. Comput. **19**, 297–301 (1965)
7. P. Duhamel, M. Vitterli, Fast Fourier transform: a tutorial review and a state of the art. Signal Process. **19**, 259–299 (1990)
8. Electronic components database. Available online: https://www.digchip.com/datasheets/parts/datasheet/033/EP3C16F484C6.php. Accessed 5 Aug 2020.
9. M. Garrido, J. Grajal, M.A. Sanchez, O. Gustafsson, Pipelined radix-2k feedforward FFT architectures. IEEE Trans Very Large Scale Integr. Syst. **21**, 23–32 (2011)
10. M. Garrido, S. Huang, S. Chen, O. Gustafsson, The serial commutator FFT. IEEE Trans. Circuits Syst. II Express Briefs **63**(10), 974–978 (2016)
11. L.O. Hnativ, Integer cosine transforms for high-efficiency image and video coding. Cybern. Syst. Anal. **52**(5), 802–816 (2016)
12. C. Ingemarsson, P. Källström, F. Qureshi, O. Gustafsson, Efficient FPGA Mapping of Pipeline. *IEEE Trans*. Very Large Scale Integr. Syst. **25**, 2486–2497 (2017)
13. K.J. Jones, R. Coster, Area-efficient and scalable solution to real-data fast fourier transform via regularised fast Hartley transform. IET Signal Proc. **1**(3), 128–138 (2007)
14. M. Kasyanchuk, I. Yakymenko, S. Ivas'ev, Ya. Nykolaychuk, Fundamental theoretical and algorithmic principles of the applied tasks decision of theory of numbers and construction of the high-performance special processors on their basis, in *Proceedings of the XI International Conference on The Experience of Designing and Application of CAD Systems in Microelectronics, CADSM 2011, 23–25 February, 2011, Polyana-Svalyava (Zakarpattya), Ukraine*, pp.168–169 (2011).
15. V. Kumar, K.K. Mahapatra, et al. An efficient distributed arithmetic based VLSI architecture for DCT. *IEEE Trans. Circ. Syst. I Regular Papers*, pp. 978–983 (2011).
16. A.C. Mert, E. Kalali, I. Hamzaoglu, High performance 2D transform hardware for future video coding. IEEE Trans. Consum. Electron. **63**(2), 117–125 (2017)

17. J.G. Nash, Distributed-memory-based FFT architecture and FPGA implementations. Electronics **7**, 116 (2018)
18. A. Nukada, Y. Maruyama, S. Matsuoka, High performance 3-D FFT using multiple CUDA GPUs, in *Proceedings of the 5th Annual Workshop on General Purpose Processing with Graphics Processing Units, GPGPU-5, New York, NY, USA, ACM*, , pp. 57–63 (2012).
19. OBrien Labs. Altera Cyclone II, III, IV Development Kits. Available online: https://obrienlabs.blogspot.com/2010/12/altera-cyclone-iii-development-kits.html. (accessed on 05.08.2020).
20. I. Prots'ko, V. Teslyuk, Algorithm of efficient computation DSTI-IV using cyclic convolutions. WSEAS Trans. Signal Process. **10**, 278–288 (2014)
21. I. Prots'ko, Algorithm of efficient computation of generalised discrete Hartley transform based on cyclic convolutions. IET Signal Proc. **4**(8), 301–308 (2014)
22. D. Puchala, K. Stokfiszewski, B. Szczepaniak, M. Yatsymirskyy, Effectiveness of Fast Fourier Transform implementations on GPU and CPU. Przeglad Elektrotechniczny **92**(7), 69–71 (2016)
23. M. Raguraman, D. Saravanan, FPGA implementation of approximate 2d discrete cosine transforms. Circuits Syst. **7**, 434–445 (2016)
24. K.R. Rao, D.N. Kim, J.J. Hwang, *Fast Fourier Transform—Algorithms and Applications* (Springer, Berlin, 2011)
25. B.R. Rau, J.A. Fisher, Instruction-level parallel processing: history, overview and perspective. J. Supercomput. **7**(1), 9–50 (1993)
26. J.J. Rodriguez-Andina, M.J. Moure, M.D. Valdes-Pena, Advanced features and industrial applications of FPGAs—a review. IEEE Trans. Ind. Inform. **11**, 853–864 (2015)
27. P. Saha, A. Banerjee, A. Dandapat, P. Bhattacharyya, ASIC implementation of high speed processor for calculating discrete Fourier transformation using circular convolution technique. WSEAS Trans. Circuits Syst. **10**, 278–288 (2011)
28. S. Shen, W. Shen, Y. Fan, X. Zeng, A Unified 4/8/16/32-Point Integer IDCT architecture for multiple video coding standards, in *IEEE International Conference on Multimedia and Expo, ICME, Melbourne, VIC Australia, 9-13*, pp. 788–793 (2012)
29. G. Sohi, Instruction issue logic for high-performance interruptible, multiple functional unit, pipelined computers. IEEE Trans. Comput. **39**(3), 349–359 (1990)
30. K. Stokfiszewski, K. Wieloch, M. Yatsymirskyy, The fast fourier transform partitioning scheme for GPU's computation effectiveness improvement, in *Advances in Intelligent Systems and Computing,* Springer: Lviv, Ukraine, 2018, Volume 689, pp. 511–522 (2018).
31. T.-Y. Sung, Y.-S. Shieh, An efficient VLSI linear array for DCT/IDCT using subband decomposition algorithm. *Hindawi Publishing Corporation Mathematical Problems in Engineering*, 2010, 87–93.
32. T. Tao, S. Liu, H. Ma, M. Li, X. Zhou, X. Wang, J. Weng, Twiddle factor neutralization method for heterodyne velocimetry. Rev. Sci. Instrum. **85**(1), 013101 (2014). https://doi.org/10.1063/1.4859598
33. Terasic Technologies FPGA Dev Kits for Altera Cyclone® II, III, & IV. Available online: https://ru.mouser.com/new/terasic-technologies/terasic-fpga-dev-cyclone-kits/ (accessed on 05.08.2020).
34. R.L. Tokheim, *Digital Electronics: Principles and Application*. 8th edition. McGraw Hill Higher Education: New York, USA, January 16, 576 (2013).
35. I.G. Tsmots, *Informatsijni tehnologii ta spetsializovani zasoby obrobky sygnaliv i zobrazhen u realnomu chasi*. UAD: Lviv, Ukraine, (2005). (**in Ukrainian**).
36. J. Wu, Jaja, j. High Performance FFT Based Poisson Solver on a CPU-GPU Heterogeneous Platform. Processing (IPDPS) 2013 IEEE 27th International Symposium on Parallel & Distributed, Boston, MA, USA, 20–24 May 2013, pp. 115–125.
37. M.M. Yatsymirskyy, *Shvydki algorytmy ortogonalnyh trygonometrychnyh peretvoren*. Akademichnyj Expres: Lviv, Ukraine, (1997). (**in Ukrainian**).
38. Md. ZainulAbidin, M.O. Sharrif, Tan shao theong design of a VLSI digit slicing fast Fourier transform processor. Microelectron. J. **22**(5–6), 15–26 (1991)
39. X. Zhao, J. Chen, M. Karczewicz, L. Zhang, X. Li, W. Chien, Enhanced multiple transform for video coding, in *Proceedings on data compression conference, Snowbird, UT, USA*, pp. 73–82 (2016).

Authors and Affiliations

Ivan Tsmots[1] · Vasyl Rabyk[2] · Natalia Kryvinska[3,4] · Mykhaylo Yatsymirskyy[4] · Vasyl Teslyuk[1]

Ivan Tsmots
ivan.tsmots@gmail.com

Vasyl Rabyk
rabykv@ukr.net

Mykhaylo Yatsymirskyy
mykhaylo.yatsymirskyy@p.lodz.pl

1 Department of Automated Control Systems, Lviv Polytechnic National University, Lviv 79013, Ukraine

2 Department of RadioPhysics and Computer Technologies, Ivan Franko National University of Lviv, 1, Universytetska Str., Lviv 79000, Ukraine

3 Department of Information Systems, Faculty of Management, Comenius University, Bratislava, Bratislava 25 82005, Slovakia

4 Institute of Information Technology, Lodz University of Technology, Wolczanska 215 Street, Lodz, Poland