Aleksandr CARIOW, Galina CARIOWA

West Pomeranian University of Technology, Szczecin

doi:10.15199/48.2017.01.21

A Hardware-oriented Algorithm for Complex-valued Constant Matrix-vector Multiplication

***Abstract**. In this communication we present a hardware-oriented algorithm for constant matrix-vector product calculating, when the all elements of vector and matrix are complex numbers. The main idea behind our algorithm is to combine the advantages of Winograd's inner product formula with Gauss's trick for complex number multiplication. The proposed algorithm versus the naïve method of analogous calculations drastically reduces the number of multipliers required for FPGA implementation of complex-valued constant matrix-vector multiplication. If the fully parallel hardware implementation of naïve (schoolbook) method for complex-valued matrix-vector multiplication requires 4MN multipliers, 2M N-inputs adders and 2MN two-input adders, the proposed algorithm requires only 3N(M+1)/2 multipliers and [3M(N+2)+1,5N+2] two-input adders and 3(M+1) N/2-input adders.*

***Streszczenie.** W komunikacie został zaprezentowany sprzętowo-zorientowany algorytm mnożenia macierzy stałych przez wektor zmiennych w założeniu, gdy zarówno elementy macierzy jak i elementy wektora są liczbami zespolonymi. Główna idea proponowanego algorytmu polega na łącznym zastosowaniu wzoru Winograda do wyznaczania iloczynu skalarnego oraz formuły Gaussa mnożenia liczb zespolonych. W porównaniu z tradycyjnym sposobem realizacji obliczeń proponowany algorytm pozwala zredukować liczbę układów mnożących niezbędnych do całkowicie równoległej realizacji na platformie FPGA układu wyznaczania iloczynu wektorowo-macierzowego. Jeśli całkowicie równoległa implementacja tradycyjnej metody wyznaczania omawianych iloczynów wymaga 4MN bloków mnożących, 2M N-wejściowych sumatorów oraz 2MN sumatorów dwuwejściowych, to proponowany algorytm wymaga tylko 3N(M+1)/2 błoków mnożenia, [3M(N+2)+1,5N+2] sumatorów dwuwejściowych i 3(M+1) sumatorów N/2-wejściowych. (**Sprzętowo-zorientowany algorytm wyznaczania iloczynu macierzy stałych przez wektor zmiennych dla danych zespolonych**).*

Keywords: algorithm design and analysis, FPGA, VLSI, high performance computing
Słowa kluczowe: konstruowanie i analiza algorytmów, FPGA, VLSI, wysokowydajne obliczenia

Introduction

Most of the computation algorithms which are used in digital signal, image and video processing, computer graphics and vision and high performance supercomputing applications have matrix-vector multiplication as the kernel operation [1, 2]. For this reason, the rationalization of these operations is devoted to numerous publications [3-18]. In some cases, elements of the multiplied matrices and vectors are complex numbers [5-9]. In the general case a fully parallel hardware implementation of a rectangular complex-valued matrix-vector multiplication requires MN multipliers of complex numbers. In the case where the matrix elements are constants, we can use encoders instead of multipliers. This solution greatly simplifies implementation, reduces the power dissipation and lowers the price of the device. On the other hand, when we are dealing with FPGA chips that contain several tens or even hundreds of embedded multipliers, the building and using of additional encoders instead of multipliers is irrational. Examples could be that of the Xilinx Spartan-3 family of FPGA's which includes between 4 and 104 18x18 on-chip multipliers and the Altera Cyclone-III family of FPGA's which include between 23 and 396 18×8 on-chip multipliers. Another Altera's Stratix-V GS family of FPGA's has between 600 and 1963 variable precision on-chip blocks optimized for 27×27 bit multiplication. In this case, it would be unreasonable to refuse the possibility of using embedded multipliers. Nevertheless, the number of on-chip multipliers is always limited, and this number may sometimes not be enough to implement a high-speed fully parallel matrix-vector multiplier. Therefore, finding ways to reduce the number of multipliers in the implementation of matrix-vector multiplier is an extremely urgent task. Some interesting solutions related to the rationalization of the complex-valued matrix-matrix and matrix-vector multiplications have already been obtained [10-13]. There are also original and effective algorithms for constant matrix-vector multiplication. However, the rationalized algorithm for complex-valued constant matrix-vector multiplications has not yet been published. For this reason, in this paper, we propose such algorithm.

Preliminary remarks

The complex-valued vector-matrix product may be defined as:

(1) $$\mathbf{Y}_{M\times 1} = \mathbf{A}_{M\times N}\mathbf{X}_{N\times 1}$$

where $\mathbf{X}_{N\times 1} = [x_0, x_1, ..., x_{N-1}]^{\mathrm{T}}$ - is N-dimensional complex-valued input vector, $\mathbf{Y}_{M\times 1} = [y_0, y_1, ..., y_{M-1}]^{\mathrm{T}}$ - is M-dimensional complex-valued output vector, and

$$\mathbf{A}_{M\times N} = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,N-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M-1,0} & a_{M-1,1} & \cdots & a_{M-1,N-1} \end{bmatrix},$$

where $n = 0,1,...,N-1$, $m = 0,1,...,M-1$, and $x_n = x_n^{(r)} + jx_n^{(i)}$, $a_{m,n} = a_{m,n}^{(r)} + ja_{m,n}^{(i)}$, $y_m = y_m^{(r)} + jy_m^{(i)}$.

In this expression $x_n^{(r)}$, $x_n^{(i)}$, $y_m^{(r)}$, $y_m^{(i)}$ are real variables, $a_{m,n}^{(r)}$, $a_{m,n}^{(i)}$ are real constants, and j is the imaginary unit, satisfying $j^2 = -1$. Superscript r means the real part of complex number, and the superscript i means the imaginary part of complex number. The task is to calculate the product defined by the expression (1) with the minimal multiplicative complexity.

Brief background

It is well known, that complex multiplication requires four real multiplications and two real additions, because:

(2) $$(a + jb)(c + jd) = ac - bd + j(ad + bc).$$

So, we can observe that the direct computation of (1) requires NM complex multiplications ($4NM$ real multiplications) and $2M(2N-1)$ real additions.

According to Winograd's formula for inner product calculation each element of vector $\mathbf{Y}_{M\times 1}$ can be calculated as follows [15]:

(3) $$y_m = \sum_{k=0}^{\frac{N}{2}-1}[(a_{m,2k} + x_{2k+1})(a_{m,2k+1} + x_{2k})] - c_m - \xi_N ,$$

where

$$c_m = \sum_{k=0}^{\frac{N}{2}-1} a_{m,2k} \cdot a_{m,2k+1} \text{ and } \xi_N = \sum_{k=0}^{\frac{N}{2}-1} x_{2k} \cdot x_{2k+1}$$

if N is even. (The case of odd N, will not be considered here, as it can easily be reduced to the even length N). It is clear that if we are dealing with complex-valued data, then $c_m = c_m^{(r)} + jc_m^{(i)}$ and $\xi_N = \xi_N^{(r)} + j\xi_N^{(i)}$, where $\xi_N^{(r)}$ and $\xi_N^{(i)}$ are real and imaginary parts of calculated real variable ξ_N respectively, $c_m^{(r)}$ and $c_m^{(i)}$ are real and imaginary parts of calculated in advance constants c_m. Here it should be emphasized that because $a_{m,n}$ are constants, the c_m can be precomputed and stored in a lookup table in advance. Thus, the calculation of c_m does not require the execution of arithmetic operations during realization of the algorithm. The calculation of ξ_N requires the implementation of the $N/2$ complex multiplications. Therefore, we can observe that the computation of (3) for all m requires only $N(M+1)/2$ complex multiplications ($2N(M+1)$ real multiplications). However, the number of real additions in this case is significantly increased.

It is well known too, that the complex multiplication can be carried out using only three real multiplications and five real additions, because [13]:

(4) $$(a+jb)(c+jd) = ac - bd + j[(a+b)(c+d) - ac - bd] .$$

Expression (4) is well known as Gauss's trick for multiplication of complex numbers [16]. Taking into account this trick the expression (3) can be calculated using the only $3N(M+1)/2$ multiplications of real numbers at the expense of further increase in the number of real additions.

The algorithm

First, we present the vector $\mathbf{X}_{N\times 1} = [x_0, x_1, ..., x_{N-1}]^{\mathrm{T}}$ in the following form: $\mathbf{X}_{2N\times 1} = [x_0^{(r)}, x_0^{(i)}, x_1^{(r)}, x_1^{(i)}, ..., x_{N-1}^{(r)}, x_{N-1}^{(i)}]^{\mathrm{T}}$, and vector $\mathbf{Y}_{M\times 1} = [y_0, y_1, ..., y_{M-1}]$ - in the following form: $\mathbf{Y}_{2M\times 1} = [y_0^{(r)}, y_0^{(i)}, y_1^{(r)}, y_1^{(i)}, ..., y_{M-1}^{(r)}, y_{M-1}^{(i)}]^{\mathrm{T}}$.

Next, we split vector $\mathbf{X}_{2N\times 1}$ into two separate vectors $\mathbf{X}_{N\times 1}^{(1)}$ and $\mathbf{X}_{N\times 1}^{(2)}$ containing only even-numbered and only odd-numbered elements respectively:

$$\mathbf{X}_{N\times 1}^{(1)} = [x_0^{(r)}, x_0^{(i)}, x_2^{(r)}, x_2^{(i)}, ..., x_{N-2}^{(r)}, x_{N-2}^{(i)}]^{\mathrm{T}} ,$$
$$\mathbf{X}_{N\times 1}^{(2)} = [x_1^{(r)}, x_1^{(i)}, x_3^{(r)}, x_3^{(i)}, ..., x_{N-1}^{(r)}, x_{N-1}^{(i)}]^{\mathrm{T}} .$$

Then from the elements of the matrix $\mathbf{A}_{M\times N}$ we form two super-vectors of data:

$$\mathbf{A}_{MN\times 1}^{(1)} = [\hat{\mathbf{A}}_{2M\times 1}^{(0)}, \hat{\mathbf{A}}_{2M\times 1}^{(1)}, ..., \hat{\mathbf{A}}_{2M\times 1}^{(\frac{N}{2}-1)}]^{\mathrm{T}} ,$$
$$\mathbf{A}_{MN\times 1}^{(2)} = [\breve{\mathbf{A}}_{2M\times 1}^{(0)}, \breve{\mathbf{A}}_{2M\times 1}^{(1)}, ..., \breve{\mathbf{A}}_{2M\times 1}^{(\frac{N}{2}-1)}]^{\mathrm{T}} ,$$

where

$$\hat{\mathbf{A}}_{2M\times 1}^{(k)} = [a_{0,2k+1}^{(r)}, a_{0,2k+1}^{(i)}, a_{1,2k+1}^{(r)}, a_{1,2k+1}^{(i)}, ..., a_{M-1,2k+1}^{(r)}, a_{M-1,2k+1}^{(i)}]^{\mathrm{T}} ,$$
$$\breve{\mathbf{A}}_{2M\times 1}^{(k)} = [a_{0,2k}^{(r)}, a_{0,2k}^{(i)}, a_{1,2k}^{(r)}, a_{1,2k}^{(i)}, ..., a_{M-1,2k}^{(r)}, a_{M-1,2k}^{(i)}]^{\mathrm{T}} ,$$

And now we introduce the vectors

$$\mathbf{C}_{2M\times 1} = [-c_0^{(r)}, -c_0^{(i)}, -c_1^{(r)}, -c_1^{(i)}, ..., -c_{M-1}^{(r)}, -c_{M-1}^{(i)}]^{\mathrm{T}} ,$$
$$\mathbf{\Xi}_{2M\times 1} = [-\xi_N^{(r)}, -\xi_N^{(i)}, -\xi_N^{(r)}, -\xi_N^{(i)}, ..., -\xi_N^{(r)}, -\xi_N^{(i)}]^{\mathrm{T}} .$$

Next, we introduce some auxiliary matrices:

$$\mathbf{P}_{MN\times N} = \mathbf{I}_{\frac{N}{2}} \otimes (\mathbf{1}_{M\times 1} \otimes \mathbf{I}_2) , \ \breve{\mathbf{T}}_{\frac{3}{2}MN\times MN} = \mathbf{I}_{\frac{MN}{2}} \otimes \mathbf{T}_{3\times 2} ,$$
$$\mathbf{\Sigma}_{3M\times\frac{3MN}{2}} = \mathbf{1}_{1\times\frac{N}{2}} \otimes (\mathbf{I}_M \otimes \mathbf{I}_3) , \ \hat{\mathbf{T}}_{2M\times 3M} = \mathbf{I}_M \otimes \mathbf{T}_{2\times 3} ,$$
$$\mathbf{T}_{3\times 2} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & -1 \end{bmatrix}, \ \mathbf{T}_{2\times 3} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}.$$

where $\mathbf{1}_{M\times N}$ - is an $M\times N$ matrix of ones (a matrix where every element is equal to one), $\mathbf{I}_N$ - is an identity $N\times N$ matrix and sign „$\otimes$" denotes tensor product of two matrices [17].

Using the above matrices the rationalized computational procedure for calculating the constant matrix-vector product can be written as follows:

(5) $$\mathbf{Y}_{2M\times 1} = \mathbf{\Xi}_{2M\times 1} + \{\mathbf{C}_{2M\times 1} + \hat{\mathbf{T}}_{2M\times 3M}[\mathbf{\Sigma}_{3M\times\frac{3}{2}MN} \times$$
$$\times \mathbf{D}_{\frac{3}{2}MN} \breve{\mathbf{T}}_{\frac{3}{2}MN\times MN} (\mathbf{A}_{MN\times 1}^{(1)} + \mathbf{P}_{MN\times N}\mathbf{X}_{N\times 1}^{(1)})]\},$$
$$\mathbf{D}_{\frac{3}{2}MN} = \bigoplus_{l=0}^{\frac{MN}{2}-1} \mathbf{D}_3^{(l)} , \ \mathbf{D}_3^{(l)} = diag(s_0^{(l)}, s_1^{(l)}, s_2^{(l)}) ,$$

where sign „$\oplus$" denotes direct sum of the matrices which are numbered in accordance with the increase of the superscript value [17].

If the elements of $\mathbf{D}_{\frac{3}{2}MN}$ placed vertically without disturbing the order and written in the form of the vector $\mathbf{S}_{\frac{3}{2}MN\times 1} = \mathbf{D}_{\frac{3}{2}MN}\mathbf{1}_{\frac{3}{2}MN\times 1}$, then they can be calculated using the following vector-matrix procedure:

(6) $$\mathbf{S}_{\frac{3}{2}MN\times 1} = \widetilde{\mathbf{T}}_{\frac{3}{2}MN\times MN}(\mathbf{A}_{MN\times 1}^{(2)} + \mathbf{P}_{MN\times N}\mathbf{X}_{N\times 1}^{(2)}) ,$$
$$\widetilde{\mathbf{T}}_{\frac{3}{2}MN\times MN} = \mathbf{I}_{\frac{MN}{2}} \otimes \widetilde{\mathbf{T}}_{3\times 2} , \ \widetilde{\mathbf{T}}_{3\times 2} = \begin{bmatrix} 1 & -1 \\ 1 & 1 \\ 0 & 1 \end{bmatrix}.$$

As already noted, the elements of the vector $\mathbf{C}_{2M\times 1}$ can be calculated in advance. However, the elements of vector $\mathbf{\Xi}_{2M\times 1}$ must be calculated during the realization of the algorithm. The procedure describes the implementation of computing elements of this vector can be represented in the following form:

(7) $$\mathbf{\Xi}_{2M\times 1} = \mathbf{P}_{2M\times 2}\mathbf{T}_{2\times 3}\mathbf{\Sigma}_{3\times\frac{3N}{2}}\mathbf{\Psi}_{\frac{3N}{2}}\breve{\mathbf{T}}_{\frac{3N}{2}\times N}\mathbf{X}_{N\times 1}^{(1)} ,$$

where

$$\breve{\mathbf{T}}_{\frac{3N}{2}\times N} = \mathbf{I}_{\frac{N}{2}} \otimes \mathbf{T}_{3\times 2} , \ \mathbf{P}_{2M\times 2} = \mathbf{1}_{M\times 1} \otimes \mathbf{I}_2 , \ \mathbf{\Sigma}_{3\times\frac{3N}{2}} = \mathbf{1}_{1\times\frac{N}{2}} \otimes \mathbf{I}_3$$

and

$$\mathbf{\Psi}_{\frac{3}{2}N} = \bigoplus_{k=0}^{\frac{N}{2}-1} \mathbf{\Psi}_3^{(k)},\ \mathbf{\Psi}_3^{(k)} = diag(\varepsilon_0^{(2k+1)}, \varepsilon_1^{(2k+1)}, \varepsilon_2^{(2k+1)}).$$

If the elements of $\mathbf{\Psi}_{\frac{3}{2}N}$ placed vertically without disturbing the order and written in the form of the vector $\overline{\mathbf{\Psi}}_{\frac{3}{2}N\times 1} = \mathbf{\Psi}_{\frac{3}{2}N}\mathbf{1}_{\frac{3}{2}N\times 1}$, then they can be calculated using the following vector-matrix procedure:

$$\overline{\mathbf{\Psi}}_{\frac{3}{2}N\times 1} = \widetilde{\mathbf{T}}_{\frac{3}{2}N\times N}\mathbf{X}_{N\times 1}^{(2)},\ \widetilde{\mathbf{T}}_{\frac{3}{2}N\times N} = \mathbf{I}_{\frac{N}{2}} \otimes \widetilde{\mathbf{T}}_{3\times 2}.$$

Consider, for example, the case of $N=4$ and $M=3$. Then the procedure (5) takes the following form:

$$\mathbf{Y}_{6\times 1} = \mathbf{\Xi}_{6\times 1} + \{\mathbf{C}_{6\times 1} + \hat{\mathbf{T}}_{6\times 9}[\mathbf{\Sigma}_{9\times 18}\times \mathbf{D}_{18}\breve{\mathbf{T}}_{18\times 12}(\mathbf{A}_{12\times 1}^{(1)} + \mathbf{P}_{12\times 4}\mathbf{X}_{4\times 1}^{(1)})]\},$$

where

$$\mathbf{Y}_{6\times 1} = [y_0^{(r)}, y_0^{(i)}, y_1^{(r)}, y_1^{(i)}, y_3^{(r)}, y_3^{(i)}]^{\mathrm{T}},$$

$$\mathbf{X}_{4\times 1}^{(1)} = [x_0^{(r)}, x_0^{(i)}, x_2^{(r)}, x_2^{(i)}]^{\mathrm{T}},\ \mathbf{X}_{4\times 1}^{(2)} = [x_1^{(r)}, x_1^{(i)}, x_3^{(r)}, x_1^{(i)}]^{\mathrm{T}},$$

$$\mathbf{D}_{18} = \bigoplus_{l=0}^{5} \mathbf{D}_3^{(l)},\ \mathbf{D}_3^{(l)} = diag(s_0^{(l)}, s_1^{(l)}, s_2^{(l)}),$$

$$\mathbf{S}_{18\times 1} = \mathbf{D}_{18}\mathbf{1}_{18\times 1} = \widetilde{\mathbf{T}}_{18\times 12}(\mathbf{A}_{12\times 1}^{(2)} + \mathbf{P}_{12\times 4}\mathbf{X}_{4\times 1}^{(2)}),$$

$$\mathbf{A}_{12\times 1}^{(1)} = [\hat{\mathbf{A}}_{6\times 1}^{(0)}, \hat{\mathbf{A}}_{6\times 1}^{(1)}]^{\mathrm{T}},\ \mathbf{A}_{12\times 1}^{(2)} = [\breve{\mathbf{A}}_{6\times 1}^{(0)}, \breve{\mathbf{A}}_{6\times 1}^{(1)}]^{\mathrm{T}},$$

$$\mathbf{\Xi}_{6\times 1} = [-\xi_4^{(r)}, -\xi_4^{(i)}, -\xi_4^{(r)}, -\xi_4^{(i)}, -\xi_4^{(r)}, -\xi_4^{(i)}]^{\mathrm{T}},$$

$$\mathbf{C}_{6\times 1} = [-c_0^{(r)}, -c_0^{(i)}, -c_1^{(r)}, -c_1^{(i)}, -c_2^{(r)}, -c_2^{(i)}]^{\mathrm{T}},$$

$$\hat{\mathbf{A}}_{6\times 1}^{(0)} = [a_{0,1}^{(r)}, a_{0,1}^{(i)}, a_{1,1}^{(r)}, a_{1,1}^{(i)}, a_{2,1}^{(r)}, a_{2,1}^{(i)}]^{\mathrm{T}},\ \widetilde{\mathbf{T}}_{18\times 12} = \mathbf{I}_6 \otimes \widetilde{\mathbf{T}}_{3\times 2}$$

$$\hat{\mathbf{A}}_{6\times 1}^{(1)} = [a_{0,3}^{(r)}, a_{0,3}^{(i)}, a_{1,3}^{(r)}, a_{1,3}^{(i)}, a_{2,3}^{(r)}, a_{2,3}^{(i)}]^{\mathrm{T}},\ \breve{\mathbf{T}}_{18\times 12} = \mathbf{I}_6 \otimes \mathbf{T}_{3\times 2},$$

$$\breve{\mathbf{A}}_{6\times 1}^{(0)} = [a_{0,0}^{(r)}, a_{0,0}^{(i)}, a_{1,0}^{(r)}, a_{1,0}^{(i)}, a_{2,0}^{(r)}, a_{2,0}^{(i)}]^{\mathrm{T}},\ \hat{\mathbf{T}}_{6\times 9} = \mathbf{I}_3 \otimes \mathbf{T}_{2\times 3},$$

$$\breve{\mathbf{A}}_{6\times 1}^{(1)} = [a_{0,2}^{(r)}, a_{0,2}^{(i)}, a_{1,2}^{(r)}, a_{1,2}^{(i)}, a_{2,2}^{(r)}, a_{2,2}^{(i)}]^{\mathrm{T}},\ \widetilde{\mathbf{T}}_{6\times 4} = \mathbf{I}_2 \otimes \widetilde{\mathbf{T}}_{3\times 2},$$

$$\mathbf{P}_{12\times 4} = \mathbf{I}_2 \otimes (\mathbf{1}_{3\times 1} \otimes \mathbf{I}_2),\ \mathbf{\Sigma}_{9\times 18} = \mathbf{1}_{1\times 2} \otimes \mathbf{I}_9,$$

$$\mathbf{\Xi}_{6\times 1} = (-\mathbf{I}_N)\mathbf{P}_{6\times 2}\mathbf{T}_{2\times 3}\mathbf{\Sigma}_{3\times 6}\mathbf{\Psi}_6\breve{\mathbf{T}}_{6\times 4}\mathbf{X}_{4\times 1}^{(1)},\ \breve{\mathbf{T}}_{6\times 4} = \mathbf{I}_2 \otimes \mathbf{T}_{3\times 2},$$

$$\overline{\mathbf{\Psi}}_{6\times 1} = \mathbf{\Psi}_6\mathbf{1}_{6\times 1} = \widetilde{\mathbf{T}}_{6\times 4}\mathbf{X}_{4\times 1}^{(2)},\ \mathbf{P}_{6\times 2} = \mathbf{1}_{3\times 1} \otimes \mathbf{I}_2,\ \mathbf{\Sigma}_{3\times 6} = \mathbf{1}_{1\times 2} \otimes \mathbf{I}_3,$$

and $\mathbf{\Psi}_6 = \bigoplus_{k=0}^{1} \mathbf{\Psi}_3^{(k)},\ \mathbf{\Psi}_3^{(k)} = diag(\varepsilon_0^{(2k+1)}, \varepsilon_1^{(2k+1)}, \varepsilon_2^{(2k+1)})$,

The data flow diagram for realization of proposed algorithm is illustrated in Figure 1. In turn, Figure 2 shows a data flow diagram for computing elements of the matrix $\mathbf{D}_{3MN/2}$ in accordance with the procedure (6). In this paper, the data flow diagrams are oriented from left to right. Note [13-14] that the circles in these figures show the operation of multiplication by a real number (variable) inscribed inside a circle. Rectangles denote the real additions with values inscribed inside a rectangle. Straight lines in the figures denote the operation of data transfer. At points where lines converge, the data are summarized. (The dashed lines indicate the subtraction operation). We use the usual lines without arrows specifically so as not to clutter the picture. Figure 3*a* shows a data flow diagram for computing elements of the vector $\mathbf{\Xi}_{2M\times 1}$ in accordance with the procedure (7). In turn, Figure 3*b* shows a data flow diagram for computing elements of the diagonal matrix $\mathbf{\Psi}_{3N/2}$.

Discussion of hardware complexity

We calculate how many multipliers and adders are required, and compare this with the number required for a fully parallel naïve implementation of complex-valued matrix–vector product in Eq. (1). The number of conventional two-input multipliers required using the proposed algorithm is $3N(M+1)/2$. Thus using the proposed algorithm the number of multipliers to implement the complex-valued constant matrix-vector product is drastically reduced. Additionally our algorithm requires $2M(N+1)$ one-input adders with constant numbers (ordinary encoders), $M(N+4)+1{,}5N+2$ conventional two-input adders, and $3(M+1)$ $(N/2)$ -input adders. Instead of encoders we can apply the ordinary two-input adders. Then the implementation of the algorithm will require $3N(M+1)/2$ multipliers $3M(N+2)+1{,}5N+2$ two-input signed adders and $3(M+1)$ $(N/2)$ -input adders.

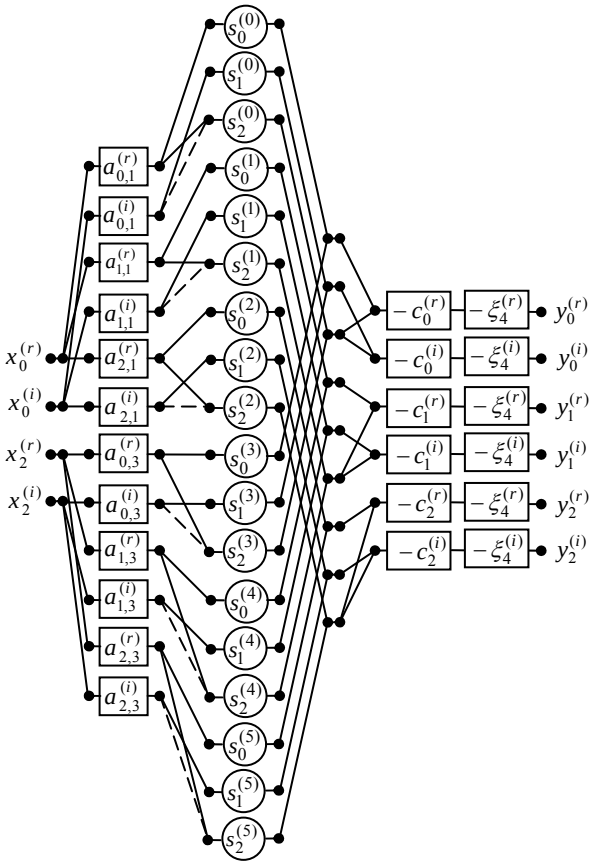

Fig.1. Data flow diagram for rationalized complex-valued constant matrix-vector multiplication algorithm for *N*=4, *M*=3.

In turn, the number of conventional two-input multipliers required using fully parallel implementation of “schoolbook” method for complex-valued matrix-vector multiplication is $4MN$. This implementation also requires $2M$ N -inputs adders and $2MN$ two-input adders. Thus, our proposed algorithm saves 50 and even more percent of two-input embedded multipliers but it significantly increases number of adders compared with direct method of fully-parallel implementation. For applications where the "cost" of a multiplication is greater than that of an addition, the new algorithm is always more computationally efficient than direct evaluation of the matrix-vector product. This allows concluding that the suggested solution may be useful in a number of cases and have practical application allowing to minimize complex-valued constant matrix-vector multiplier’s hardware implementation costs.

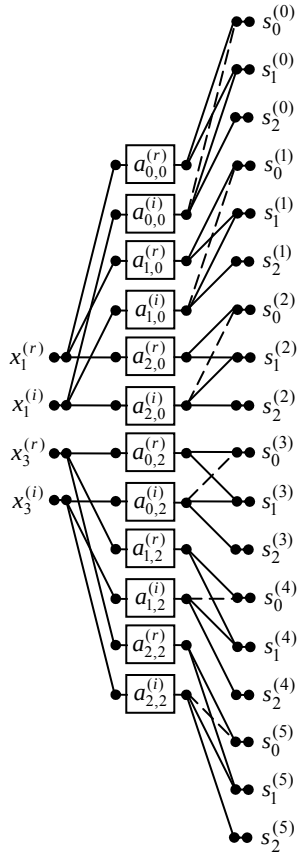

Fig.2. The data flow diagram for calculating elements of diagonal matrix $\mathbf{D}_{3MN/2}$ for N=4, M=3.

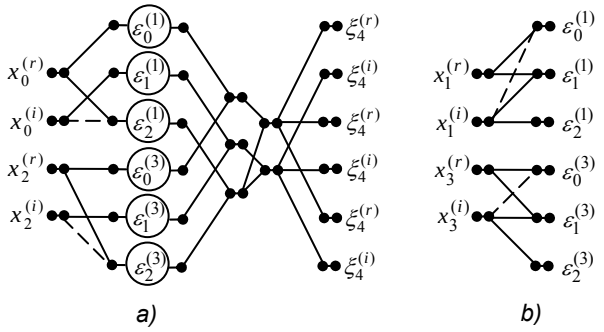

Fig.3. The data flow diagrams for calculating elements of vector $\mathbf{\Xi}_{6\times1}$ (*a*), and for calculating elements of diagonal matrix $\mathbf{\Psi}_6$ (*b*).

Concluding remarks

The article presents a new hardware-oriented algorithm for computing the complex-valued constant matrix-vector multiplication. To reduce the hardware complexity (number of two-operand multipliers), we exploit the Winograd's inner product formula and Gauss trick for complex number multiplication. This allows the effective use of parallelization of computations on the one hand and results in a reduction in hardware implementation cost of complex-valued constant matrix-vector multiplier on the other hand.

If the FPGA-chip already contains embedded multipliers, their number is always limited. This means that if the implemented algorithm contains a large number of multiplications, the developed processor may not always fit into the chip. So, the implementation of proposed in this paper algorithm on the base of FPGA chips, that have built-fin binary multipliers, also allows saving the number of these blocks or realizing the whole complex-valued constant matrix-vector multiplying unit with the use of a smaller number of simpler and cheaper FGPA chips. It will enable to design of data processing units using a chips which contain a minimum required number of embedded multipliers and thereby consume and dissipate least power. How to implement a fully parallel complex-valued constant matrix-vector multiplier on the base of concrete FPGA platform is beyond the scope of this article, but it's a subject for follow-up articles.

***Authors**: prof. PhD, D.Sc. Aleksandr Cariow, PhD Galina Cariowa, Department of Computer Architectures and Telecommunications, Faculty of Computer Sciences, West Pomeranian University of Technology, Szczecin, ul. Żołnierska 51, 71-210 Szczecin, E-mail: atariov@wi.zut.edu.pl, gtariova@wi.zut.edu.pl*

REFERENCES

[1] Blahut, R. E. Fast algorithms for digital signal processing, Addison-Wesley Publishing company, Inc. 1985

[2] Pratt, W. K. Digital Image Processing (Second Edition), John Wiley & Sons, New York, 1991.

[3] Fujimoto, N. Dense matrix-vector multiplication on the CUDA architecture, Parallel Processing Letters, vol. 18, no. 4, 511-530, 2008.

[4] Qasim, S. M., Telba, A. A. and Al Mazroo, A. Y. FPGA Design and Implementation of Matrix Multiplier Architectures for Image and Signal Processing, IJCSNS International Journal of Computer Science and Network Security, vol.10, no.2, 168-176, 2010.

[5] Fam, A. T. Efficient complex matrix multiplication, IEEE Transactions on Computers, vol. 37, no. 7, 877-879, 1988.

[6] Connolly, F. T. Yagle, A. E. Fast algorithms for complex matrix multiplication using surrogates, IEEE Transactions on Acoustics, Speech and Signal Processing, vol. 37, no. 6, 938 – 939, 1989.

[7] Ollila, E. Koivunen, V. Poor, H. V. Complex-valued signal processing — essential models, tools and statistics, Information Theory and Applications Workshop, 6-11 Feb. 2011, 1 – 10.

[8] Guoqiang Li, Liren Liu Complex-valued matrix-vector multiplication using twos complement representation". Optics Communications, vol. 105, no. 3-4, 161-166.

[9] Barazesh B., Michalina J. and Picco A. A VLSI signal processor with complex arithmetic capability. IEEE Transactions on Circuits and Systems, 35(5), 495–505, 1988.

[10] Gustafsson O., Ohlsson H. and Wanhammar L., Low-complexity constant coefficient matrix multiplication using a minimum spanning tree approach, Proceedings of the 6th Nordic Signal Processing Symposium (NORSIG 2004), June 9 - 11, Espoo, Finland, 141-144, 2004.

[11] Boullis N. Tisserand A. Some optimizations of hardware multiplication by constant matrices, IEEE Transactions on Computers, 2005, vol. 54, no 10, 1271 – 1282.

[12] Kinane A. Muresan V. Towards an optimized VLSI design algorithm for the constant matrix multiplication problem. In: Proc. IEEE International Symposium on Circuits and Systems (ISCAS-2006), 5111 – 5114, 2006.

[13] Cariow A., Cariowa G., An algorithm for complex-valued vector-matrix multiplication. Electrical Review, R 88, no 10b, 213-216, 2012.

[14] Cariow A., Cariowa G., A rationalized algorithm for complex-valued inner product calculation, Measurement Automation and Monitoring, no 7, pp. 674-676, 2012.

[15] Winograd S. A new algorithm for inner Product, IEEE Transactions on Computers, vol. C-17, no 7, 693 – 694, 1968.

[16] Knuth D. E., The Art Of Computing Programming, vol. 2, Semi-numerical Algorithms, Addison-Wesley, Reading, MA, USA, Second Ed., 1981.

[17] Regalia P. A. and Mitra K. S., Kronecker Products, Unitary Matrices and Signal Processing Applications, SIAM Review., vol. 31, no. 4, pp. 586-613, 1989