Adv. Radio Sci., 6, 89–94, 2008
www.adv-radio-sci.net/6/89/2008/

Advances in
Radio Science

Matrix-Vector Based Fast Fourier Transformations on SDR Architectures

Y. He[2], K. Hueske[1], J. Götze[1], and E. Coersmeier[2]

[1]University of Dortmund, Information Processing Lab, Otto-Hahn-Str. 4, 44221 Dortmund, Germany
[2]Nokia Research Center, Bochum, Meesmannstr. 103, 44807 Bochum, Germany

Abstract. Today Discrete Fourier Transforms (DFTs) are applied in various radio standards based on OFDM (Orthogonal Frequency Division Multiplex). It is important to gain a fast computational speed for the DFT, which is usually achieved by using specialized Fast Fourier Transform (FFT) engines. However, in face of the Software Defined Radio (SDR) development, more general (parallel) processor architectures are often desirable, which are not tailored to FFT computations. Therefore, alternative approaches are required to reduce the complexity of the DFT. Starting from a matrix-vector based description of the FFT idea, we will present different factorizations of the DFT matrix, which allow a reduction of the complexity that lies between the original DFT and the minimum FFT complexity. The computational complexities of these factorizations and their suitability for implementation on different processor architectures are investigated.

1 Introduction

Recently multi-carrier modulation techniques, such as OFDM, have received great attention in high-speed data communication systems and have been selected for several communication standards (e.g. WLAN, DVB and WiMax). A central baseband signal processing task required in OFDM transceivers is the Fourier transform of the received data, which is generally realized as FFT (Cooley, 1965). To meet the real-time processing requirements, various specialized FFT processors have been proposed (Son, 2002). On the other hand Software Defined Radio (SDR) approaches for mobile terminals are gaining momentum (Dillinger, 2003). To meet the demand for high computing power and flexibility, specific reconfigurable accelerators (Heyne, 2004) or parallel processor architectures (Berthold, 2004) are used. The DFT can be implemented as FFT, which defines the optimum in terms of mathematical complexity, but as the used architectures are not necessarily optimized for butterfly structures, the question arises: Are there any efficient procedures, which reduce the computational complexity of the DFT significantly (as far as possible to the FFT complexity), but at the same time are more tailor-made to specific hardware architectures than the FFT?. In this paper a matrix-vector based formulation of the FFT algorithm (Van Loan, 1992) is derived from the respective matrix factorizations of the DFT matrix. Due to the structure of the butterfly matrices and the matrix factorizations, the matrix-vector based FFT allows several separations of the resulting matrix product. These separations have an increased computational complexity compared to the FFT, but because of their mathematical structure they allow more flexible implementations on different architectures (e.g. multi processors, vector processors, accelerated systems). The paper is organized as follows: In Sect. 2 we will clarify the definition and notation of the DFT. A short repetition of the FFT idea and the corresponding factorization is described in Sect. 3. Then, in Sect. 4 different separations of the butterfly matrices are presented, which lead to different algorithmic structures and different computational complexities. Implementation issues of these separations and their suitability for different hardware architectures are discussed. Conclusions are given in Sect. 5.

Correspondence to: K. Hueske
(klaus.hueske@uni-dortmund.de)

2 Discrete Fourier Transform (DFT)

The sequence of N (with $N{=}2^m, m{\in}\mathbb{N}$) complex numbers $x_1, \cdots, x_N$ is transformed into the sequence of N complex numbers $y_1, \cdots, y_N$ by the DFT according to

$$y_t = \sum_{n=1}^{N} x_n w_N^{(t-1)(n-1)}, \quad t = 1, \cdots, N, \tag{1}$$

where $w_N{=}e^{-j\frac{2\pi}{N}}$ is a primitive N-th root of the unity. The DFT can also be formulated as a matrix vector product (Van Loan, 1992). With $\mathbf{x}_N{=}[x_1 \cdots x_N]^T \in \mathbb{C}^N$ and $\mathbf{y}_N{=}[y_1 \cdots y_N]^T \in \mathbb{C}^N$, the DFT can be described as matrix vector product $\mathbf{y}_N{=}\mathbf{F}_N{\cdot}\mathbf{x}_N$, where $\mathbf{F}_N \in \mathbb{C}^{N\times N}$ is the DFT-Matrix with elements $f_{nt}{=}w_N^{(t-1)\cdot(n-1)}$ and $n, t{=}1, \cdots, N$.

3 Matrix-vector based FFT

To give a repetition of the known radix-2 FFT algorithms (Cooley, 1965), this section will show the connection between the matrix $\mathbf{F}_N$ and the matrix $\mathbf{F}_{N/2}$, which enables us to compute an N-point DFT from a pair of $(N/2)$-point DFTs. To transfer the FFT idea to the DFT matrix, we introduce a so-called permutation matrix $\mathbf{P}_N$ of size $(N\times N)$, which performs an odd-even ordering of the rows (multiplied from the left with $\mathbf{P}_N$) or columns (multiplied from the right with $\mathbf{P}_N^T$) of a matrix. Furthermore $\mathbf{P}_N^T{\cdot}\mathbf{P}_N{=}\mathbf{I}_N$ holds. The following factorizations and separations are shown for the decimation in time (DIT) FFT. They can be easily transferred to the decimation in frequency (DIF) FFT.

3.1 FFT-Matrix decomposition

Applying $\mathbf{P}_N^T$ to the right of the DFT-matrix $\mathbf{F}_N$ yields

$$\mathbf{F}'_N = \mathbf{F}_N \cdot \mathbf{P}_N^T = \begin{bmatrix} \mathbf{F}_{\frac{N}{2}} & \mathbf{D}_{\frac{N}{2}} \cdot \mathbf{F}_{\frac{N}{2}} \\ \mathbf{F}_{\frac{N}{2}} & -\mathbf{D}_{\frac{N}{2}} \cdot \mathbf{F}_{\frac{N}{2}} \end{bmatrix}. \tag{2}$$

The resulting product $\mathbf{F}'_N$ has the following structure: The odd indexed columns, forming the left half of $\mathbf{F}'_N$, are composed of $\mathbf{F}_{N/2}$, while the even indexed columns, forming the right half of $\mathbf{F}'_N$, are composed of $\mathbf{D}_{N/2}{\cdot}\mathbf{F}_{N/2}$ and $-\mathbf{D}_{N/2}{\cdot}\mathbf{F}_{N/2}$. This matrix can be decomposed into the product of two matrices (as shown in Eq. 3),

$$\mathbf{F}'_N = \begin{bmatrix} \mathbf{I}_{\frac{N}{2}} & \mathbf{D}_{\frac{N}{2}} \\ \mathbf{I}_{\frac{N}{2}} & -\mathbf{D}_{\frac{N}{2}} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{F}_{\frac{N}{2}} & \mathbf{0} \\ \mathbf{0} & \mathbf{F}_{\frac{N}{2}} \end{bmatrix} \tag{3}$$

where $\mathbf{D}_{N/2}$ is a diagonal matrix with elements $d_{nn}{=}w_N^{(n-1)}$ and $n{=}1, \cdots, N/2$. The first matrix represents FFT butterfly operations, while the second matrix contains two independent DFTs of length $N/2$. Note that the second matrix is a block diagonal matrix.

3.2 Matrix-Vector based computation of the FFT

To apply the modified DFT $\mathbf{F}'_N$ to an input vector $\mathbf{x}_N$, a permutation of $\mathbf{x}_N$ is required, i.e.

$$\mathbf{P}_N \cdot \mathbf{x}_N = [\mathbf{x}_o \mathbf{x}_e]^T\,, \tag{4}$$

with $\mathbf{x}_o \in \mathbb{C}^{\frac{N}{2}}$ the odd and $\mathbf{x}_e \in \mathbb{C}^{\frac{N}{2}}$ the even indexed part of $\mathbf{x}_N$. Introducing $\mathbf{y}_1, \mathbf{y}_2 \in \mathbb{C}^{\frac{N}{2}}$ we can formulate the DFT as follows:

$$\mathbf{y}_N = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix} = \mathbf{F}_N \cdot \mathbf{x}_N = \mathbf{F}_N \cdot \underbrace{\mathbf{P}_N^T \cdot \mathbf{P}_N}_{\mathbf{I}_N} \cdot \mathbf{x}_N \tag{5}$$

$$= \begin{bmatrix} \mathbf{I}_{\frac{N}{2}} & \mathbf{D}_{\frac{N}{2}} \\ \mathbf{I}_{\frac{N}{2}} & -\mathbf{D}_{\frac{N}{2}} \end{bmatrix} \begin{bmatrix} \mathbf{F}_{\frac{N}{2}} & \mathbf{0} \\ \mathbf{0} & \mathbf{F}_{\frac{N}{2}} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{x}_o \\ \mathbf{x}_e \end{bmatrix}$$

Using $\mathbf{y}_o{=}\mathbf{F}_{\frac{N}{2}}{\cdot}\mathbf{x}_o$ and $\mathbf{y}_e{=}\mathbf{F}_{\frac{N}{2}}{\cdot}\mathbf{x}_e$ we obtain:

$$\mathbf{y}_1 = \mathbf{y}_o + \mathbf{D}_{\frac{N}{2}} \cdot \mathbf{y}_e = \mathbf{y}_o + \mathbf{d}_{\frac{N}{2}} \cdot *\mathbf{y}_e \tag{6}$$

$$\mathbf{y}_2 = \mathbf{y}_o - \mathbf{D}_{\frac{N}{2}} \cdot \mathbf{y}_e = \mathbf{y}_o - \mathbf{d}_{\frac{N}{2}} \cdot *\mathbf{y}_e. \tag{7}$$

With $\mathbf{d}_{\frac{N}{2}}$ the vectorized diagonal of $\mathbf{D}_{\frac{N}{2}}$, with $(\cdot *)$ the component-wise vector multiplication and taking into account that the two products are identical, Eq. (6) and Eq. (7) need altogether $\frac{N}{2}$ multiplications and $2 \cdot \frac{N}{2}$ additions. It is easy to see that a DFT of size N can be executed by two DFTs of size $\frac{N}{2}$. For $N{=}2^m$ this process can be recursively repeated $m{=}\log_2 N$ times. The required number of operations then will be $\frac{N}{2}\log_2 N$ multiplications and $N \log_2 N$ additions, which represents the complexity of the FFT.

3.3 Factorization of the DFT matrix

To obtain alternative factorizations of the DFT matrix, the number of recursions described in the previous section can be modified. Introducing a new parameter k we can define the number of performed recursions as $m{-}k$ with $N{=}2^m$ giving the size of the initial DFT matrix. For arbitrary m and k the resulting block submatrices in $\mathbf{F}_B$ are of size $(2^k{\times}2^k)$ and can be executed as independent DFTs (or FFTs). Because the length of the input vector $\mathbf{x}_N$ is given by $N{=}2^m$ the number of block submatrices equals $2^m/2^k{=}2^{m-k}$. After calculating 2^{m-k} independent DFTs, $m{-}k$ butterfly matrices $\mathbf{B}_i$ have to be applied to finish the calculation. The entire DFT is then given as

$$\mathbf{y}_N = \mathbf{B}_{m-k} \cdot \mathbf{B}_{m-k-1} \cdots \mathbf{B}_1 \cdot \mathbf{F}_B \cdot \mathbf{x}_P\,, \tag{8}$$

with $\mathbf{x}_p$ the $(m{-}k)$-times permuted input vector $\mathbf{x}_N$. An example for $m{=}4$ and $k{=}2$ is given below.

Example 1:

$$\boldsymbol{y}_N = \underbrace{\begin{bmatrix} \boldsymbol{I}_8 & \boldsymbol{D}_8 \\ \boldsymbol{I}_8 & -\boldsymbol{D}_8 \end{bmatrix}}_{\boldsymbol{B}_2} \cdot \underbrace{\begin{bmatrix} \boldsymbol{I}_4 & \boldsymbol{D}_4 & 0 & 0 \\ \boldsymbol{I}_4 & -\boldsymbol{D}_4 & 0 & 0 \\ 0 & 0 & \boldsymbol{I}_4 & \boldsymbol{D}_4 \\ 0 & 0 & \boldsymbol{I}_4 & -\boldsymbol{D}_4 \end{bmatrix}}_{\boldsymbol{B}_1} \cdot \underbrace{\begin{bmatrix} \boldsymbol{F}_{\frac{N}{4}} & 0 & 0 & 0 \\ 0 & \boldsymbol{F}_{\frac{N}{4}} & 0 & 0 \\ 0 & 0 & \boldsymbol{F}_{\frac{N}{4}} & 0 \\ 0 & 0 & 0 & \boldsymbol{F}_{\frac{N}{4}} \end{bmatrix}}_{\boldsymbol{F}_B} \cdot \boldsymbol{x}_p$$

3.4 Construction of butterfly matrices

Increasing the order of recursion $m-k$ will also increase the number of butterfly matrices $\mathbf{B}_i$. While the independent DFT submatrices in $\mathbf{F}_B$ can be processed independently, the butterfly matrices can be treated in different ways:

·As sequence of matrix multiplications

$$\prod_{i=0}^{m-k-1} \mathbf{B}_{m-k-i}$$

·As one matrix multiplication with the butterfly product matrix

$$\mathbf{B} = \prod_{i=0}^{m-k-1} \mathbf{B}_{m-k-i}$$

·As partly combined matrix multiplication with the products

$$\prod_{i=0}^{p_1} \mathbf{B}_{m-k-i} \prod_{i=p_1-1}^{p_2} \mathbf{B}_{m-k-i} \prod_{i=p_r-1}^{m-k-1} \mathbf{B}_{m-k-i}$$

with arbitrarily chosen integer numbers p fulfilling $(1<p_1<p_2<\ldots<p_r<m-k)$.

Example 2: The following example illustrates the construction of the butterfly matrices. For the given parameters m=8 and k=4 already $2^{m-k}=2^4=16$ small DFTs are executed by $\mathbf{F}_B$. The structure of the remaining butterfly matrices $\mathbf{B}_i$ is depicted in Fig. 1 on the left hand side (non-zero elements are depicted as lines). The product of the butterfly matrices for increasing i is shown on the right hand side. The matrices are all of dimension $2^8 \times 2^8$, the variable nz in the figure shows the number of non-zero elements. Focusing on the butterfly product matrix in the bottom right, we can find that only nz=4096 of all $2^8 \cdot 2^8$=65536 elements are non-zero. This means, every 16-th diagonal of the butterfly product matrix is non-zero, since the other multiplications have already been performed by the block diagonal matrix $\mathbf{F}_B$. In general, every 2^k-th diagonal is non-zero, which means, that the interspace will be enlarged with increasing k.

4 Implementation

In this section we will compare different separations of the butterfly matrix in terms of complexity and implementation issues.

4.1 Complexity

The application of the DFT matrix requires $(N-1)\cdot(N-1)$ multiplications, i.e. the product of all rows and columns, except the first row/column, which is all ones:

$$M_{\mathrm{DFT}} = N^2 - 2N + 1 = 2^m \cdot 2^m - 2 \cdot 2^m + 1.$$

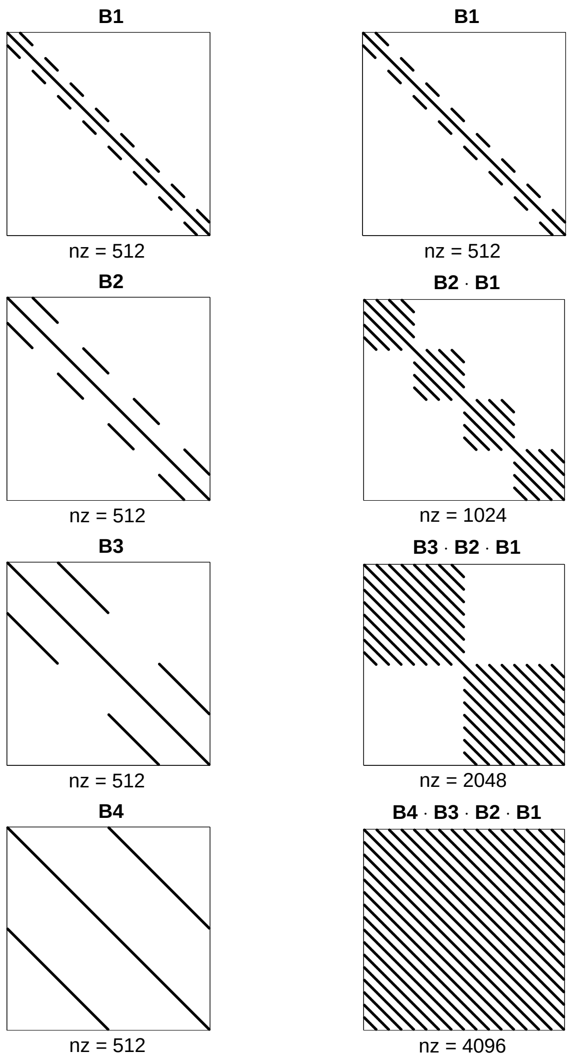

Fig. 1. The single butterfly matrices $\mathbf{B}_i$ on the left hand side; their products on the right hand side.

The FFT is based on butterfly operations, which require $N/2=2^{m-1}$ multiplications each (see Sect. 3.2). Since we have m butterfly matrices, we obtain:

$$M_{\mathrm{FFT}} = m \cdot 2^{m-1} = \frac{N}{2} \log_2 N.$$

In the following we will examine the complexity of four different separations, which we expect to lie between M_{DFT} and M_{FFT}.

Separation A. We use $(\mathbf{B}_{m-k}\cdots\mathbf{B}_1\cdot\mathbf{F}_B)$, where the submatrices of $\mathbf{F}_B$ are executed as independent DFTs and the butterfly matrices are separately executed one by one. The number of the DFT blocks in $\mathbf{F}_B$ is 2^{m-k} and each DFT needs $2^k\cdot 2^k - 2\cdot 2^k + 1$ multiplications. The number of butterfly matrices is $m-k$ and each butterfly needs 2^{m-1} multiplications.

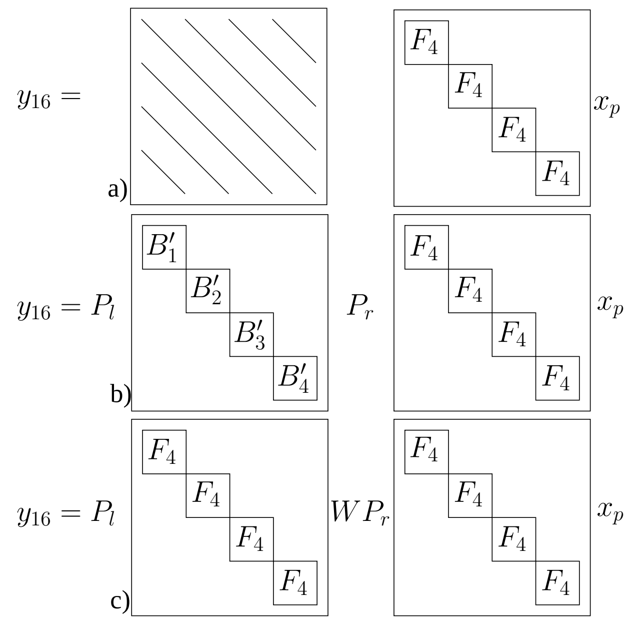

Fig. 2. Parallel FFT following Separation D.

This leads to

$$M_A = (m-k)\cdot 2^{m-1} + (2^k\cdot 2^k - 2\cdot 2^k + 1)\cdot 2^{m-k}.$$

Separation B. We use $\mathbf{B}\cdot\mathbf{F}_B$, where the submatrices of $\mathbf{F}_B$ are still executed as independent DFTs, and $\mathbf{B}$ is the butterfly product matrix. Since only every 2^k-th diagonal is occupied, the number of required multiplications for $\mathbf{B}$ is $(2^m-1)\cdot(\frac{2^m}{2^k}-1)$. The number of multiplications for $\mathbf{F}_B$ is identical to separation A , such that

$$M_B = (2^m-1)\cdot(\frac{2^m}{2^k}-1) + (2^k\cdot 2^k - 2\cdot 2^k + 1)\cdot 2^{m-k}.$$

Separation C. We again use $\mathbf{B}\cdot\mathbf{F}_B$, but now the submatrices of $\mathbf{F}_B$ are executed as independent FFTs. There are $k\cdot 2^{k-1}$ multiplications required for each FFT, and totally there are 2^{m-k} FFT blocks, so the number of multiplications is given as

$$M_C = (2^m-1)\cdot(\frac{2^m}{2^k}-1) + k\cdot 2^{k-1}\cdot 2^{m-k}.$$

Separation D. A special case of separation C is separation D, which describes a partly parallel implementation of the original FFT for the case $k{=}m/2$. The factorization is given as

$$\mathbf{y}_N = \mathbf{P}_l\cdot\mathbf{F}_B\cdot\mathbf{W}\cdot\mathbf{P}_r\cdot\mathbf{F}_B\cdot\mathbf{x}_p, \qquad (9)$$

with $\mathbf{W}$ a twiddle factor diagonal matrix and $\mathbf{P}_l$ and $\mathbf{P}_r$ permutation matrices that sort the rows and columns of a matrix, such that $\mathbf{P}_l\cdot\mathbf{F}_B\cdot\mathbf{W}\cdot\mathbf{P}_r{=}\mathbf{B}$ is the butterfly product matrix. To

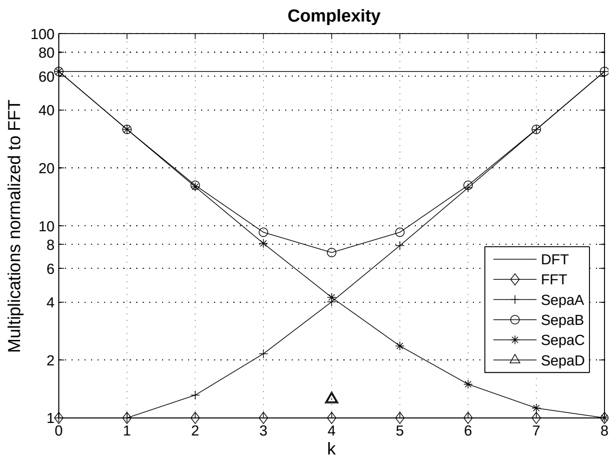

Fig. 3. Complexities for DFT, FFT and various separations.

clarify this, an example is given in Fig. 2 for $m{=}4$ and $k{=}2$. **Example 3**: In a) we can see the graphical representation of example 2, with the butterfly product matrix $\mathbf{B}$ on the left and the block FFT matrix $\mathbf{F}_B$ on the right hand side. By sorting rows and columns of $\mathbf{B}$ we can transform the sparse diagonal matrix $\mathbf{B}$ to a block diagonal matrix, as shown in b). This leads to a matrix based notation of the Jeon-Reeves FFT algorithms (Jeon (1986)). As the blocks $(\mathbf{B}'_1,\cdots,\mathbf{B}'_4)$ are not identical, constant twiddle factors are extracted from the matrices, which transfers the block submatrices to DFT matrices, like shown in c). The extracted twiddle factors are collected in $\mathbf{W}$. The computational complexity of this separation is determined by the number of FFTs $(2\cdot 2^k)$ and the additional twiddle factor multiplications ($N{=}2^m$):

$$M_D = 2\cdot 2^k\cdot k\cdot 2^{k-1} + 2^m = 2^m(1+m/2) \qquad (10)$$

The number of required multiplications for separations A–D are given in Fig. 3 for $m{=}8$ and varying k as multiples of M_{FFT}. The upper bound is defined by the DFT complexity, the lower bound shows the FFT complexity. Note that separation D is only defined for $k{=}m/2{=}4$.

4.2 Implementation issues

Here we will discuss possible implementations of the presented separations on different hardware architectures. The discussion is of a qualitative character, i.e. implementation details like memory access, interconnection complexity or inter processor communication time are not considered here.

Separation A. Matrix-vector processors as special kind of SIMD (Single Instruction Multiple Data) architectures are discussed as possible platforms for SDR (Schoenes (2003)). As an example we will show an implementation of separation A using a general systolic array structure for matrix multiplications, like depicted in Fig. 4. With $m{=}5$ and $k{=}2$ we have to perform 8 DFTs of length 4 in $\mathbf{F}_B$. To obtain the matrix-matrix structure we separate the input vector $\mathbf{x}_p$

into 8 parts ($\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_8$) of length 4, which means that all $\mathbf{F}_4 \cdot \mathbf{x}_j$ with j=1, ..., 8 can be calculated independently. The multiple matrix-vector products can then be described as matrix-matrix product $\mathbf{F}_4 \cdot \mathbf{X}_p$ with $\mathbf{X}_p$=[$\mathbf{x}_1$ $\mathbf{x}_2 \cdot \mathbf{x}_8$]. The remaining butterfly operations are performed according to Eq. (6) and Eq. (7). The even numbered columns of the product $\mathbf{F}_4 \cdot \mathbf{X}_p$ are multiplied component-wise with $\mathbf{d}_4$ and then added/subtracted to the corresponding odd numbered columns. For example, the second column is multiplied by $\mathbf{d}_4$ and then added/subtracted to the first column, resulting in the intermediate vectors $\mathbf{y}_{11}$ and $\mathbf{y}_{21}$ that are stored in the registers of the first and second column of the processor array. In the next step these intermediate results are multiplied by $\mathbf{d}_8$ and added/subtracted to the other intermediate results. After $m-k$ steps the registers contain the Fourier transform of $\mathbf{x}$.
The computational complexity for this approach is only slightly increased for small values of k (compare Fig. 3), but a high speed partly parallel Fourier transformation can be realized.

Separation B. For generic matrix-vector architectures the DFT can be described as simple matrix-vector product using separation B. In this case the number of required multiplications is increased considerably compared to the FFT, but nevertheless, it is interesting to see the significant decrease of the number of multiplications (e.g. for k=3, 4, 5 in Fig. 3) compared to the DFT, just by separating the DFT matrix into a product of the two matrices $\mathbf{B}$ and $\mathbf{F}_B$.

Separation C. To match the real time requirements in SDR systems, general purpose processors are often combined with hardware accelerators, like e.g. for the Fourier transform. As the flexibility of these accelerators regarding to the DFT size might be limited, separation C can be used to map DFTs of any size onto these engines. For example in a systems with FFT accelerators of size 64 and a required FFT length of 256 (this corresponds to m=8 and k=6), the block DFTs in $\mathbf{F}_B$ can be realized on the accelerators while the remaining 2 butterfly operations are executed as a simple matrix vector multiplication on a generic host processor. This leads to a major flexibility regarding the DFT size of the used accelerators in context of the required DFT size of the specific application. Furthermore existing architectures with fixed length FFT accelerators can be easily reused.
Depending on the difference of m and k, the mathematical complexity is increased (compare Fig. 3). For the example the number of required multiplications is increased by a factor 1.5, but the 4 FFTs of size 64 can be executed in parallel, if more than one accelerator is available in the system.

Separation D. The FFT proposed by Jeon and Reeves is especially suitable for multi processor architectures (Jeon (1986)). However the extraction of the twiddle factors in separation D also enables the use of homogeneous parallel FFT cores in an SDR system, as the resulting block submatrices are all identical. This means an N-point DFT can be separated into $2\sqrt{N}$ FFTs of size $\sqrt{N}$. The first $\sqrt{N}$ FFTs are

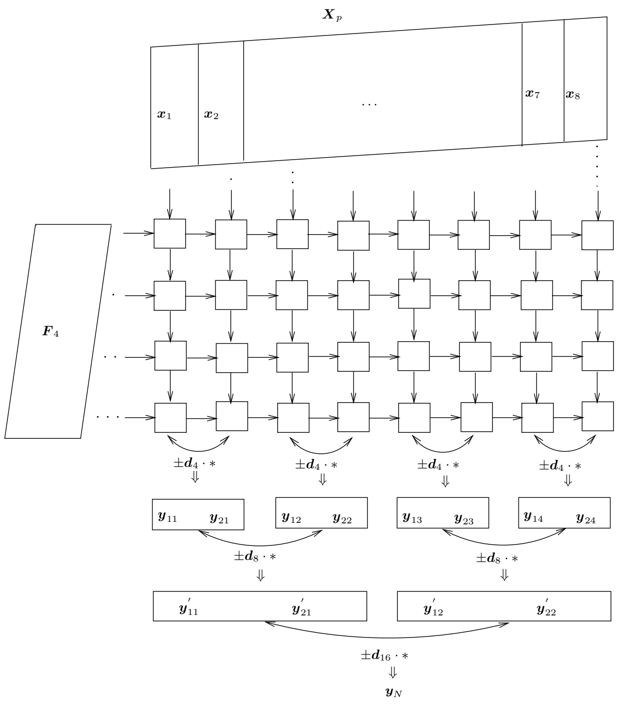
$\boldsymbol{x}_p$
$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$\cdots$
$\boldsymbol{x}_7$
$\boldsymbol{x}_8$
$\boldsymbol{F}_4$
$\pm \boldsymbol{d}_4 \cdot *$
$\boldsymbol{y}_{11}$ $\boldsymbol{y}_{21}$
$\boldsymbol{y}_{12}$ $\boldsymbol{y}_{22}$
$\boldsymbol{y}_{13}$ $\boldsymbol{y}_{23}$
$\boldsymbol{y}_{14}$ $\boldsymbol{y}_{24}$
$\pm \boldsymbol{d}_8 \cdot *$
$\boldsymbol{y}'_{11}$ $\boldsymbol{y}'_{21}$
$\boldsymbol{y}'_{12}$ $\boldsymbol{y}'_{22}$
$\pm \boldsymbol{d}_{16} \cdot *$
$\boldsymbol{y}_N$

Fig. 4. Matrix-vector processor based implementation structure.

computed using the specialized processor cores, the results are then permuted and multiplied by $\mathbf{W}$. The remaining $\sqrt{N}$ blocks are again computed using the FFT cores.

5 Conclusions

The presented separations of the DFT matrix provide different algorithmic structures, which allow a flexibility in terms of the used hardware architectures for the implementation of the DFT. Depending on the architecture, the most suitable separation can be chosen, e.g. separation A for matrix processors or separation C for architectures using FFT hardware accelerators. Depending on the chosen parameters the computational complexity in terms of multiplications is only slightly increased, but some architectures may benefit from the modified algorithmic structure (e.g. addressing, bus communication). The behaviour of the presented separations on specific existing hardware architectures has yet to be examined in future work.

References

Berthold, U., Riehmeier, A.-R., and Jondral, F. K.: Spectral partitioning for modular software defined radio, 59th Vehicular Technology Conference (VTC2004), 2, 1218–1222, May 2004.

Cooley, J. W. and Tukey, J. W.: An algorithm for the machine calculation of complex Fourier series, Math. Comput., 19, 297–301, 1965.

Dillinger, M., Madani, K., and Alonistioti, N.: Software Defined Radio-Architectures, Systems and Functions, Wiley Series in Software Radio, John Wiley & Sons Ltd., 2003.

Heyne, B. and Götze, J.: A Pure Cordic Based FFT for Reconfigurable Digital Signal Processing, 12th European Signal Processing Conference (Eusipco2004), Vienna, September 2004.

Jeon, C. H. and Reeves, A. P.: The fast Fourier transform on a multicluster computer architecture, Proceedings of the 19th Annual Hawaii International Conference on System Sciences, 103–110, 1986.

Schoenes, M., Eberli, S., Burg, A., et al.: A novel SIMD DSP architecture for software defined radio, Proc. of the 46th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS2003), 3, 1443–1446, December 2003.

Son, B. S., Jo, B. G., Sunwoo, M. H., and Yong, S. K.: A high-speed FFT processor for OFDM systems, Proc. of the IEEE International Symposium on Circuits and Systems (ISCAS2002), 3, 281–284, May 2002.

Van Loan, C.: Computational Frameworks for the Fast Fourier Transform, Frontiers in applied mathematics, Society for Industrial and Applied Mathematics, Philadelphia, 1992.