Available online at www.sciencedirect.com
ScienceDirect
Signal Processing 87 (2007) 1402–1407
SIGNAL PROCESSING
www.elsevier.com/locate/sigpro

Novel design of multiplier-less FFT processors

Yuan Zhou*, J.M. Noras, S.J. Shepherd

School of EDT, University of Bradford, Bradford, West Yorkshire BD7 1DP, UK

Received 30 June 2006; received in revised form 23 November 2006; accepted 5 December 2006
Available online 23 January 2007

Abstract

This paper presents a novel and hardware-efficient architecture for power-of-two FFT processors. The proposed design is based on the phase-amplitude splitting technique which converts a DFT to cyclic convolutions and additions. The cyclic convolutions are implemented with a filter-like structure and the additions are computed with several stages of butterfly processing units. The proposed architecture requires no multiplier, and comparisons with other designs show it can save up to 39% total equivalent gates for an 8-bit 16-point FPGA-based FFT processor.

Keywords: FFT; Systolic array; CSD; Phase-amplitude splitting; Multiplier-less architecture

1. Introduction

The fast Fourier transform (FFT) is a vital DSP tool, and is widely used in engineering and scientific research. Researchers have devoted much effort to developing FFT processors based on various FFT algorithms for real-time applications implemented using very large-scale integrated (VLSI) circuits [1–11]. One-dimensional FFT processors in the literature divide into two categories: systolic and non-systolic processors. Systolic processors take in input data continuously, while non-systolic processors operate on one frame of data at a time, and accept another batch of data only when the previous transform finishes. An N-point non-systolic processor usually requires much less area but more clock cycles than an N-point systolic processor. Thus, non-systolic processor architectures are suitable for applications which demand small devices, while systolic processor architectures are preferred for high-speed applications [12]. In this paper, novel designs of systolic FFT processors are presented.

2. Systolic processor architectures

A systolic FFT processor usually consists of a chain of processing units (PEs) which pass data through the system continuously. High speed is achieved at the cost of extensive silicon area. This is due to the numbers of large commutator blocks, which permute the I/O data and intermediate results, and also the multiplier blocks. Many approaches have been proposed to perform the permutations and multiplications. In Despain's designs [1–3], internal shift registers are used to schedule the data entering the butterfly and store intermediate results. Other designs [4–6] use delay commutators to switch data among multiple data paths so that data enter PEs in a proper order.

*Corresponding author.
E-mail address: zhouynan613@hotmail.com (Y. Zhou).

0165-1684/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.sigpro.2006.12.004

In the above designs, multiplications are implemented with adder trees. As the multipliers consume much silicon area, various implementation proposals have been made to save area, including the CORDIC technique [2], the distributed arithmetic-based design [7], the memory-based design [8] and adder-based designs [9–11]. In comparison with ROM-based designs [7,8], adder-based designs can further reduce area requirements. In filter-like adder-based designs [9], a prime length DFT is first converted to a cyclic convolution, which is performed using a filter-like architecture. In parallel adder-based designs [10], an N-point DFT is first converted to a cyclic convolution by the chirp-Z transform. This convolution is then implemented with parallel adders. A parallel adder-based processor normally consists of three parts: the pre-processing unit, the post-processing unit and the parallel adder unit.

For power-of-two FFTs, existing adder-based designs still require at least two multipliers: one in the pre-processing unit and one in the post-processing unit. However, the multipliers can be removed with a novel factorization technique which converts a power-of-two DFT into short cyclic convolutions and additions. This paper proposes a novel design of power-of-two FFT processors which requires no common multiplier at all. The proposed design has much lower hardware complexity compared with other existing adder-based power-of-two FFT processors. The new factorization technique is described in Section 3, and then the design of a 16-point FFT processor is given as an example in Section 4. The hardware cost of various designs is compared in Section 5, and conclusions are presented in Section 6.

3. The phase-amplitude splitting technique

As a discrete Fourier transform (DFT) matrix can be seen as a redundant representation of N-points located on the unit circle and can be generated by rotating a vector of unit amplitude to different phase angles, the DFT matrix contains both the amplitude and phase information of the unit vector. Treating the phase and amplitude as orthogonal subspaces results in a new decomposition, proposed by Shepherd et al. [13], the phase-amplitude (PA) decomposition of the DFT matrix.

The PA splitting method decomposes a DFT matrix FN into the product of a matrix G and a matrix H using a special mapping: each element $F_{j,k}$ of the DFT matrix FN is compared with a threshold, and the G matrix is generated according to the comparison, so that the resulting matrix G has only entries ± 1, $\pm \mathrm{i}$ and $\pm 1 \pm \mathrm{i}$. The H matrix is computed by multiplying the inverse of G with the DFT matrix. H is either a dense or a sparse matrix, with real floating-point entries. For a prime DFT of size N, the matrix H is dense, and the main $(N-1)\times(N-1)$ block in H can be converted to a cyclic matrix using row and column permutations. For a power-of-two DFT, the matrix H is sparse, and can be converted to a block diagonal matrix using row and column permutations. For example, the factorization of the 5-point DFT is

$$F_5 = G_5 \times H_5, \tag{1}$$

where F_5 denotes the 5-point DFT matrix,

$$G_5 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & -\mathrm{i} & -1-\mathrm{i} & -1+\mathrm{i} & \mathrm{i} \\ 1 & -1-\mathrm{i} & \mathrm{i} & -\mathrm{i} & -1+\mathrm{i} \\ 1 & -1+\mathrm{i} & -\mathrm{i} & \mathrm{i} & -1-\mathrm{i} \\ 1 & \mathrm{i} & -1+\mathrm{i} & -1-\mathrm{i} & -\mathrm{i} \end{bmatrix}$$

and

$$H_5 = \begin{bmatrix} 1 & 0.1667 & 0.1667 & 0.1667 & 0.1667 \\ 0 & 0.8726 & -0.1620 & 0.0196 & 0.1031 \\ 0 & 0.0196 & 0.8726 & 0.1031 & -0.1620 \\ 0 & -0.1620 & 0.1031 & 0.8726 & 0.0196 \\ 0 & 0.1031 & 0.0196 & -0.1620 & 0.8726 \end{bmatrix}.$$

The factorization of the 8-point DFT is

$$F_8 = G_8 \times H_8, \tag{2}$$

where F_8 denotes the 8-point DFT matrix,

$$G_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1-\mathrm{i} & -\mathrm{i} & -1-\mathrm{i} & -1 & -1+\mathrm{i} & \mathrm{i} & 1+\mathrm{i} \\ 1 & -\mathrm{i} & -1 & \mathrm{i} & 1 & -\mathrm{i} & -1 & \mathrm{i} \\ 1 & -1-\mathrm{i} & \mathrm{i} & 1-\mathrm{i} & -1 & 1+\mathrm{i} & -\mathrm{i} & -1+\mathrm{i} \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -1+\mathrm{i} & -\mathrm{i} & 1+\mathrm{i} & -1 & 1-\mathrm{i} & \mathrm{i} & -1-\mathrm{i} \\ 1 & \mathrm{i} & -1 & -\mathrm{i} & 1 & \mathrm{i} & -1 & -\mathrm{i} \\ 1 & 1+\mathrm{i} & \mathrm{i} & -1+\mathrm{i} & -1 & -1-\mathrm{i} & -\mathrm{i} & 1-\mathrm{i} \end{bmatrix}$$

and

$$H_8 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.8536 & 0 & 0 & 0 & 0.1464 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.8536 & 0 & 0 & 0 & 0.1464 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0.1464 & 0 & 0 & 0 & 0.8536 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0.1464 & 0 & 0 & 0 & 0.8536 \end{bmatrix}.$$

This PA splitting method can be used for factoring not only short DFTs as above, but applies directly to large DFTs such as the 541, 7919, 2048 and 8192-point DFTs.

4. Design of the 16-point FFT processor

In this section, the design of a 16-point FFT processor is given as an example to show one potential application of the PA splitting technique.

Using the PA technique, the 16-point DFT matrix splits into G and H. H, which is sparse, is converted into a block diagonal matrix using row and column permutations. G factors into two matrices with 4-point DFTs and a diagonal matrix with integer elements, using column permutations. So the 16-point FFT can be performed as follows:

$$Y = P_{16} \times (I_4 \otimes F_4) \times P_{16} \times GD \times (I_4 \otimes F_4) \times HD \times P_{16} \times X, \quad (3)$$

where

$X = [x_0, x_1, x_2, x_3, \ldots, x_{14}, x_{15}]^{\mathrm{T}}$,
$Y = [y_0, y_1, y_2, y_3, \ldots, y_{14}, y_{15}]^{\mathrm{T}}$, $\otimes$ denotes the tensor product, $x_0, x_1, x_2, \ldots, x_{15}$ are discrete inputs and $y_0, y_1, y_2, \ldots, y_{15}$ are discrete outputs. The factors of the 16-point DFT matrix can be generated using the following MATLAB routine:

I_4 = eye(4);
I_{16} = eye(16);
F_4 = dftmtx(4);
P_{16} = [I_{16}(1,:); I_{16}(5,:); I_{16}(9,:); I_{16}(13,:); I_{16}(2,:); I_{16}(6,:); I_{16}(10,:); I_{16}(14,:); I_{16}(3,:); I_{16}(7,:); I_{16}(11,:); I_{16}(15,:); I_{16}(4,:); I_{16}(8,:); I_{16}(12,:); I_{16}(16,:)];
HB_{41} = circulant([0.8536, 0,0.1464, 0]′);
HB_{42} = circulant([0.8887, 0.2646, −0.0352, −0.1181]′);
HD = blkdiag(I_4, HB_{42}, HB_{41}, HB'_{42});
GD = diag([1, 1, 1, 1, 1, 1, 1−i, −i, 1, 1−i, −i, −1−i, 1, −i, −1−i, −1]);

The function eye(n) generates the $n \times n$ identity matrix, dftmtx(n) returns the $n \times n$ DFT matrix, and circulant( ) creates a circulant matrix with its first column equal to the input vector. The function blkdiag( ) produces a block diagonal matrix using the input matrices, and the diag( ) generates a diagonal matrix using the input vector.

As shown in (3), the 16-point DFT is converted to short cyclic convolutions, additions, 4-point DFTs and permutations. The 4-point DFT involved in the transform is given below in (4). As can be seen, the input data to the 4-point DFT should be permuted so that the output of the butterfly computation can stay in place:

$$\begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & -i & i \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} T_0 \\ T_1 \\ T_2 \\ T_3 \end{bmatrix}, \quad (4)$$

where T_0, T_1, T_2, T_3 are discrete inputs and S_0, S_1, S_2, S_3 are discrete outputs.

Fig. 1 shows the architecture of the 16-point FFT processor. The data are first sent to a 16 word RAM operating in the read-before-write mode. The address signals of the RAM are 0, 1, 2 … 15 in the first 16 clock cycles, then 0, 4, 8, 12, 1, 5, 9, 13, 2, 6,10, 14, 3, 7, 11, 15 in the next 16 clock cycles. The address signals repeat every 32 cycles and the 16 words of data $\{x_0, x_1, x_2 \ldots x_{15}\}$ are partitioned into length-4 sequences. The data are then sent into the cyclic convolution block. After cyclic convolution, 4-point DFTs are performed on each of the length-4 sequences. The first delay commutator reorders the data in each sequence so that the stay-in-place butterfly operations can be performed. The output of the DFTs are multiplied with constants from the set $\{1, 1-i, -i, -1-i, -1\}$. 4-point DFTs are performed again. The output of the processor is obtained in a natural order.

The architecture of the cyclic convolution block is shown in Fig. 2. It consists of an adder network to perform multiplications by constant twiddle factors, a block B1 to add up the partial results of the cyclic convolutions, and a block B2 to schedule the results.

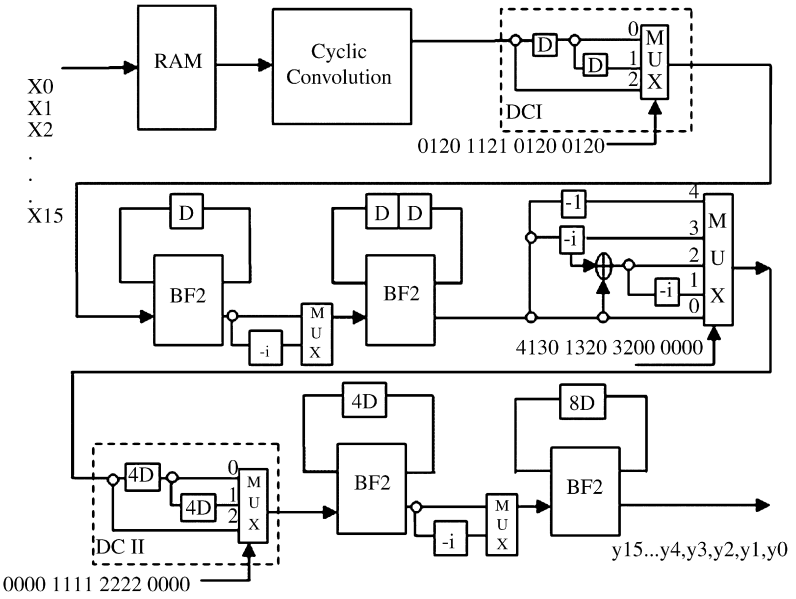

Fig. 1. Block diagram of the 16-point FFT processor.

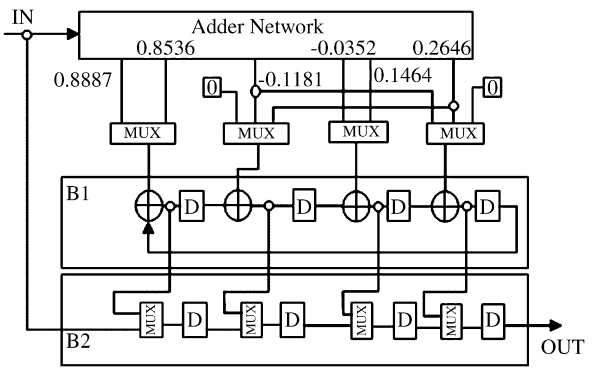

Fig. 2. Architecture of the cyclic convolution block.

Table 1
CSD coding of the twiddle factors

TW	CSD coding								
0.8536	1	0	0	−1	0	−1	0	1	0
0.1464	0	0	0	1	0	0	1	0	1
0.8887	1	0	0	−1	0	0	1	0	−1
0.2646	0	0	1	0	0	0	1	0	−1
0.0352	0	0	0	0	0	1	0	0	1
0.1181	0	0	0	1	0	0	0	−1	0

The adder network is designed by using canonic signed digit (CSD) coding and sub-expression sharing techniques. The 9-bit CSD coding of the twiddle factors, 1-bit of the integer part and 8-bit of the fraction part, is given in table in Table 1. As can

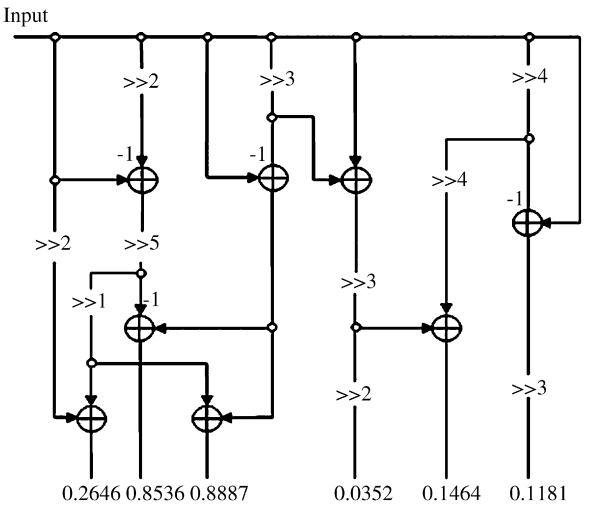

Fig. 3. Adder network for constant multiplications.

be seen, the terms $x - x \gg 2$, $x + x \gg 3$ and $x - x \gg 3$, where x denotes input data, can be shared by two or more constant multiplications. Fig. 3 shows the structure of the adder network.

As shown above, the input data of the cyclic convolution block $\{x_0, x_4, x_8, x_{12}, x_1, x_5, x_9, x_{13}, x_2, x_{10}, x_6, x_{14}, x_3, x_7, x_{11}, x_{15}\}$ are partitioned into four length-4 sequences. The first length-4 sequence $\{x_0, x_4, x_8, x_{12}\}$ is sent into B2 directly, and the data are routed to the output port through the registers and multiplexers in B2. The second sequence $\{x_1, x_5, x_9, x_{13}\}$ is sent to the adder network and

the block B1 for the cyclic convolution with the vector $[0.8887, -0.1181, -0.0352, 0.2646]$. When x_{13} comes in, the convolution results are loaded into the registers in B2. Similarly, the third sequence $\{x_2, x_6, x_{10}, x_{14}\}$ is sent to the adder network and the block B1 for the 4-point cyclic convolutions with the vector $[0.8536, 0, 0.1464, 0]$, and the forth sequence $\{x_3, x_7, x_{11}, x_{15}\}$ for the cyclic convolution with the vector $[0.8887, 0.2646, -0.0352, -0.1181]$.

5. Hardware cost and comparison

Assuming the input data of the processor are complex, the system word length is 8 bits and the twiddle factors are 9 bits. Each butterfly unit requires two adders and two 2-to-1 multiplexers. The hardware cost of the design is summarized in Table 2, which also gives the hardware cost of the radix-4 design based on [3] and the design based on [11] which is based on the chirp-z transformation. The size of ROM for twiddle factors is included. For the radix-4 design based on [3], a 32 word RAM is required for re-ordering the output and 12 words of RAM for twiddle factors. The design in [11] needs 10 words of RAM for holding twiddle factors.

As shown in the table, the proposed design requires much less area than the design in [11] in terms of the number of multipliers and adders.

To ensure 1-bit accuracy in a shift-and-add multiplier for multiplying integers with decimal fractions, the internal word length of the multiplier should be increased by $\log_2 W_L$ bits, where W_L is the system word length. So a complex multiplier requires 16 11-bit adders and an 8-bit adder to add up the products when W_L equals to 8, that is approximately 23 8-bit adders. If the multiplier is pipelined by 4 delays to achieve the same logic depth as the proposed design, 16 11-bit registers are required, that is 8 registers for partial results and 8 for shift operations on the multiplicand. Thus, some 22 8-bit registers are required. So the design based on [3] requires 31 adders and 37 registers. Compared to the proposed design, it requires 10 more adders, 32 more storage units and ten fewer multiplexers. In other words, the proposed design saves roughly 10 complex adders and 27 storage units, which are 32% of the total adders and 46% of the total storage units required in the design based on [3], as a multiplexer requires less than half of the area of a register in an 8-bit system [9].

If the system word length was doubled, the number of adders required by the complex multiplier would also double. The hardware cost of the design based on [3] would be 49 16-bit adders, 35 16-bit registers, twenty words of RAM and eleven multiplexers with the multiplier pipelined by four delays. On the other hand, the number of adders required in the proposed design would be not greater than 29 and there would be no change in the number of other hardware units. So the proposed design can save more area when the system word length increases.

Moreover, the proposed design is suitable for high-speed applications as the combinational logic depth in each stage is kept low. As shown in Figs. 1 and 3, the average combinational logic depth is only three layers of additions.

The design is implemented as a drop-in module for the Xilinx Virtex-IIxc2v-2000 FPGA with speed grade-4, and can reach a frequency of 65 MHz. It utilizes 851 out of 10,752 slices, 1564 out of 21,504 look up tables and 1 out of 56 Block RAMs. The total equivalent gate count is 90,928. In contrast to the proposed design, the design based on [3] and the shift-and-add multiplier uses 735 out of 10,752 slices, 1,236 out of 21,504 look up tables and 2 out of 56 Block RAMs. The total equivalent gate count is 149,745. In both designs, unscaled fixed-point arithmetic is used, the internal word length expands from 8 to 13 bits, and the results are obtained in a natural order. Compared to the design based on [3], the proposed multiplier-less systolic design saves 58,817 gates, which is 39% of the total gates required, at the same speed level.

The PA splitting-based method can also be used for building 32 and 64-point FFT processors. Table 3 gives hardware costs of the 16, 32 and 64-point FFT processors.

Table 2
Comparison of the hardware cost of various designs

Method	Multiplier	Adder	Register	ROM/RAM (words)	Multiplexer
Radix-4 based on [3]	1	8	15	44	11
Radix-4 in [11]	2	24	8	42	9
Proposed	0	21	33	16	21

Table 3
Hardware costs of the 16, 32 and 64-point FFT processors

	Multiplier	Adder	Register	RAM/ ROM (words)	MUX
16-point	0	21	33	16	21
32-point	0	32	64	80	30
64-point	0	44	144	112	50

6. Conclusion

This paper presents a novel and hardware-efficient systolic architecture for power-of-two DFTs. The proposed design is based on the phase-amplitude splitting technique which converts a DFT to cyclic convolutions and additions. The cyclic convolutions can be implemented with a filter-like structure and the additions can be computed with several stages of butterfly processing units. The proposed design requires no multiplier. The design of the 16-point FFT processor is given as an example to show the advantages of the new approach. Comparisons with other designs show the proposed architecture can save up to 39% gate cost for an 8-bit 16-point FPGA-based FFT processor.

References

[1] A.M. Despain, Fourier transform computer using CORDIC iterations, IEEE Trans. Comput. C-23 (10) (Oct. 1974) 993–1001.

[2] A.M. Despain, Very fast Fourier transform algorithms hardware for implementation, IEEE Trans. Comput. C-28 (5) (May 1979) 333–341.

[3] E.H. Wold, A.M. Despain, Pipeline and parallel-pipeline FFT processors for VLSI implementation, IEEE Trans. Comput. C-33 (5) (1984) 414–426.

[4] L.R. Rabiner, B. Gold, Theory and Application of Digital Signal Processing, Prentice-Hall, Inc., 1975.

[5] E.E. Swartzlander, W.K.W. Young, S.J. Joseph, A radix 4 delay commutator for fast Fourier transform processor implementation, IEEE J. Solid-State Circuits SC-19 (5) (Oct. 1984) 702–709.

[6] G. Bi, E.V. Jones, A pipelined FFT processor fore word sequential data, IEEE Trans. Acoust. Speech Signal Process. 37 (12) (Dec. 1989) 1982–1985.

[7] S.A. White, Applications of distributed arithmetic to digital signal processing: a tutorial review, IEEE ASSP Mag. (June 1989) 4–19.

[8] J.I. Guo, C.M. Liu, C.W. Jen, The efficient memory based VLSI array designs for DFT and DCT, IEEE Trans. CAS II 39 (10) (October 1992) 723–733.

[9] T.-S. Chang, C.-W. Jen, Hardware efficient transform designs with cyclic formulation and subexpression sharing, in: Proceedings of the 1998 IEEE ISCAS, vol. 2, 1998, pp. 398–401.

[10] J.I. Guo, An efficient parallel adder based design for one dimensional discrete Fourier transform, Proc. Natl. Sci. Counc. ROC, 24 (3) (2000) 195–204.

[11] C.D. Chien, C.C. Lin, C.H. Yang, J.I. Guo, Design and realization of a new hardware efficient IP core for the 1-D discrete Fourier transform, IEE Proc. Circuits, Dev. Syst. 152 (3) (2005) 247–258.

[12] K.K. Parhi, VLSI Digital Signal Processing Systems: Design and Implementation, Wiley, 1999.

[13] S.J. Shepherd, P.W.J. Van Eetvelt, S.M.R. Jones, J.M. Noras, H.S. Rajamani, A lower complexity discrete fourier transform, Math. Today 39 (5) (October 2003) 150–156.