INTEGER FAST FOURIER TRANSFORM - IMPLEMENTATION AND APPLICATION

Krzysztof Duda

Department of Measurement and Instrumentation, AGH University of Science and Technology
al. Mickiewicza 30, 30-059 Kraków (Poland) (Europe)
phone: +48 12 6173972, fax: +48 12 6338565, email: kduda@uci.agh.edu.pl

ABSTRACT

Transforms that maps integers to integers have advantages not available for floating point transforms. Integer computations are faster and free from round off errors (integer transforms are widely used for lossless coding). The paper presents an algorithm of Integer Fast Fourier Transform (intFFT) based on lifting factorization. Integer Discrete Cosine Transform (intDCT) is also computed as an example of intFFT application.

1. INTRODUCTION

Great interest in the topic of integer transforms can be observed in literature during last few years. Most of data processed in computers come from sampling of continuous systems and thus poses integer values (with range bounded by A/D converter). For those integer data integer tools seems to be natural. Two significant advances of integer transforms are: no rounding of errors during computations and faster computations. Integer transforms are widely used for lossless transform coding (especially for biomedical signals). The breakthrough in integer transforms designing was lifting introduced by I. Daubechies and W. Sweldens [1]. In the paper lifting scheme is used for computations of Integer Fast Fourier Transform (intFFT). Factorization of complex multiplications is based on the method presented in [2]. In contrary to [2] presented intFFT is based on simplified butterfly computations with division in time (one and not two complex multiplications). Presented intFFT algorithm is based on lifting '*do*' '*undo*' methodology. An example of Integer Discrete Cosine Transform is given as possible use of intFFT.

2. FORWARD AND INVERSE INTFFT

Lifting structure is presented in Fig.1. The signal in upper (lower) branch is modified by functions f_n and the signal from the opposite branch. Regardless of function f_n the lifting is always revertible. Inverse transform is simple undoing steps from the forward transform. If the input signal is integer valued and functions f_n return integers the overall transform maps integers to integers and the reconstruction error is exactly equal to zero.

Discrete Fourier Transform of a discrete signal $x(n)$, $x(n) = 0,\ for\ n < 0\ and\ n > N-1$ is defined as:

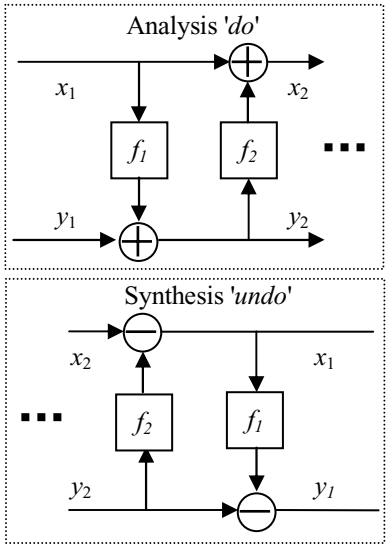

Figure 1: Lifting scheme, f_1, f_2 ... - arbitrary functions.

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn},\ k = 0, 1, ..., N-1,\ W_N^{kn} = e^{-j\frac{2\pi}{N}kn} \quad (1)$$

Equation (1) represents multiplication and addition of two complex numbers and the absolute value of W_N^{kn} equals 1. In paper [2] lifting factorization for complex multiplication is proposed. The lifting scheme for complex multiplication in case of integer valued computations is presented in Fig.2. *Z* stands for analyzed (input) complex signal, W_N^{kn} value is determined by vertical branches in the structure. In case of W_N^{kn} angle equals to $0, \pm\pi/2, \pi$ factorization is not necessary and *Z* is multiplied by +/-1 or +/-*j*. Steps from Fig.2a,b can be easily undo which results in integer division by W_N^{kn} (or multiplication by conjugate W_N^{kn}) without rounding off errors.

Implementation of intFFT is based on lifting '*do*' '*undo*' property in three levels: 1) complex multiplication by twiddle factor W_N in the way that multiplication by conjugate twiddle factor returned original value, 2) butterfly structure that allows computation in inverse (or synthesis) butterfly see Fig.1, and 3) computing intIFFT by undoing steps from intFFT.

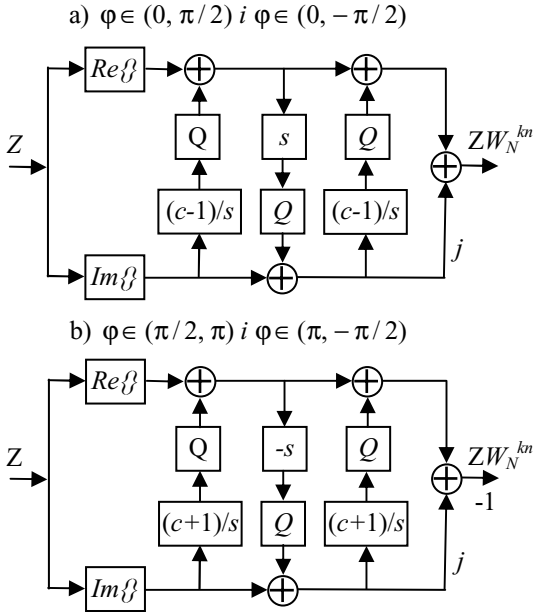

Figure 2: Lifting factorization of complex multiplication Z and W_N^{kn}[2]. Q stands for quantization (in implementation nearest integer value), $c = \cos(\varphi)$, $s = \sin(\varphi)$, $W_N^{kn} = e^{j\varphi}$.

From conditions 2 and 3 it follows that intFFT cannot be used for computing intIFFT.
The following intFFT is an algorithm with division is time. The base butterfly analysis and synthesis equations are:

Analysis:

$$\begin{aligned} T &= W_N^r X_m(q) \\ X_{m+1}(p) &= X_m(p) + T \\ X_{m+1}(q) &= X_m(p) - T \end{aligned} \tag{2}$$

Synthesis:

$$\begin{aligned} X_m(p) &= \left[X_{m+1}(p) + X_{m+1}(q)\right]/2 \\ T &= X_{m+1}(p) - X_m(p) \\ X_m(q) &= TW_N^{r*} \end{aligned} \tag{3}$$

Presented synthesis equation (3) is simply undoing steps from analysis stage (2). Complex multiplications are computed in lifting schemes presented in Fig.2. Equations (2)(3) in form of graph are depicted in Fig.3. Those butterfly computation need only one (and not two like in [2]) complex multiplication. Its worth to mention that not every butterfly can by used for intFFT in proposed algorithm. Twiddle factor cannot be associated with diagonal branch, because in that case undoing steps from analysis is impossible. Full flow chart of intFFT and intIFFT for N=4 is presented in Fig.4.
In case of intIFFT division by 2 in equation (3) cannot be done in intFFT (2), because lossless reconstruction would be lost. As a result Fourier coefficients can get high values. For example Fourier coefficients for signal with 12 bit dynamics and length N = 1024 may need 22 bits (in case of DC signal of maximum range).

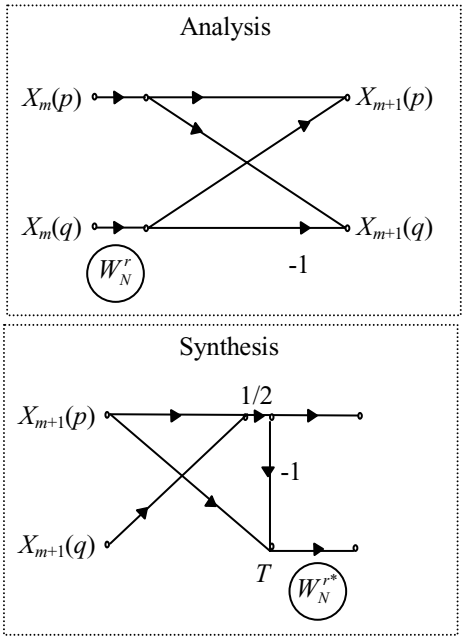

Figure 3: Flow chart of butterfly computations. Circles denote that results of $X_m(q)W_N^r$ and TW_N^{r*} are integers (see Fig.2).

Presented intFFT ensures perfect reconstruction (reconstruction error is zero) and can be computed faster then FFT (with very similar results). Those advantages were achieved by the cost of linearity and energy preservation. intFFT is not linear and condition:

$$\sum_{n=0}^{N-1}\left|x(n)\right|^2 = \frac{1}{N}\sum_{k=0}^{N-1}\left|X(k)\right|^2 \tag{4}$$

is not fulfilled.

3. COMPUTING INTEGER DISCRETE COSINE TRANSFORM BY INTFFT

Traditional FFT is probably the most popular tool in signal processing applications. As stated above intFFT gives results very similar to FFT and thus can be applied for signal analysis. In [2] intFFT was used for convolution purposes.
The following algorithm is implementation of non scaled intDCT by intFFT. Non scaled DCT is defined as:

$$X(k)\sum_{n=0}^{N-1} x(n)\cos\left(\frac{\pi(2n+1)k}{2N}\right), \quad k = 0,1,\ldots,N-1 \tag{5}$$

Presented algorithm allows computing intDCT of arbitrary length and ensures perfect reconstruction (without rounding off errors). For simplification real valued input signal is analyzed (which is the most common case, as DCT is used mainly for image compression).
Well-known algorithm of computing DCT by FFT had to be modified to ensure lossless reconstruction of intDCT.

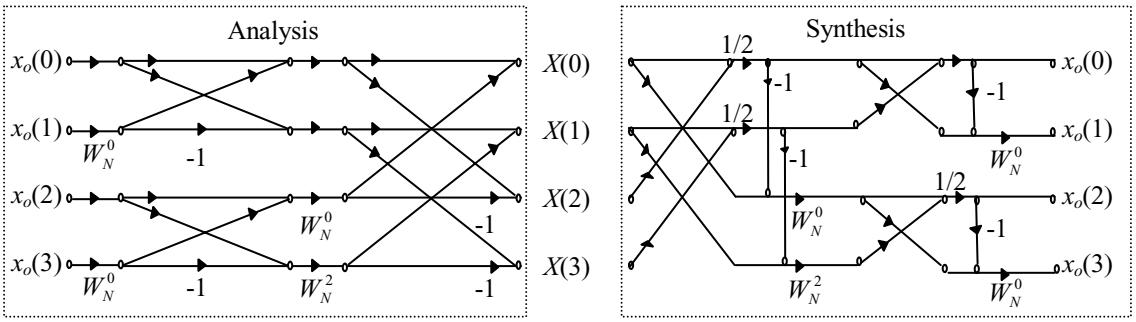

Figure 4: Full flow chart of intFFT and intIFFT for N=4, x_o - signal with bit reverse order. Symmetry between analysis and synthesis is clearly seen. During synthesis steps from analysis are undone

The modification concern reversible multiplications and '*do*' '*undo*' methodology. An overview of integer DCT methods can be found in [3].

Algorithm 1 - intDCT (based on intFFT) for real valued integer signal

$$x(n), \quad n = 0,..,N-1; \quad N = 2^l, \quad l = 1, 2, 3....$$

1. Prepare analyzed signal

$$\tilde{x}(n) = x(2n), \quad \tilde{x}(N-n-1) = x(2n+1), \quad n = 0,..., N/2-1$$

2. Compute intFFT

$$\tilde{X}(k) = \text{intFFT}\{\tilde{x}(n)\}$$

3. Prepare vector $E(k)$ and apply lifting based multiplication (see Fig. 2)

$$E(k) = \exp(-j\pi k/2N), \quad k = 0,..., N/2$$

$$\tilde{X}(k) = L\left\lfloor \tilde{X}(k)E(k) \right\rfloor \quad k = 0,..., N/2$$

4. Compose $\tilde{X}(k), k = 0,..., N-1$ vector and compute intDCT $X(k)$

$$\tilde{X}(0,..., N-1) =$$
$$= \left[\tilde{X}(0,..., N/2) \quad imag\left\{\tilde{X}(N/2-1,...,1)\right\} - j \cdot real\left\{\tilde{X}(N/2-1,...,1)\right\}\right]$$
$$X(k) = real(\tilde{X}(k))$$

Algorithm 2: - intIDCT (undo steps from forward transform)

4. Compose $\tilde{X}(k), k = 0,..., N-1$ vector

$$\tilde{X}(0,..., N-1) = X(0,..., N-1) - j \cdot \left[0 \quad X(N-1,...,1)\right]$$

3. Prepare vector $E(k)$ and apply lifting based division (see Fig. 2)

$$E(k) = \exp(-j\pi k/2N), \quad k = 0,..., N/2$$

$$\tilde{X}(k) = L\left\lfloor \tilde{X}(k)/E(k) \right\rfloor \quad k = 0,..., N/2$$

$$\tilde{X}(0,..., N-1) =$$
$$= \left[\tilde{X}(0,..., N/2) \quad real\left\{\tilde{X}(N/2-1,...,1)\right\} - j \cdot imag\left\{\tilde{X}(N/2-1,...,1)\right\}\right]$$

2. Compute intIFFT

$$\tilde{x}(k) = \text{intIFFT}\left\{\tilde{X}(n)\right\}$$

1. Compose analyzed signal $x(n)$

$$x(2n) = \tilde{x}(n), \quad x(2n+1) = \tilde{x}(N-1-n), \quad n = 0,..., N/2-1$$

4. EXAMPLE AND CONCLUSIONS

An example of computing intFFT and intDCT is presented in Fig.5. Test signal was generated in Matlab environment and consists of two sine waves with normally distributed pseudorandom noise. For analysis 1024 samples was taken with 16 bits precision. Fig.5a,b shows FFT and DCT (floating point counterparts) of this signal. Fig.5c,d presents intFFT and intDCT for the same signal and the difference between floating point and integer transform are depicted in Fig.5e,f. Similarity of integer transforms to FFT and DCT depends on quantization precision and nature of the signal. Reconstruction error for all four transforms is shown in Fig.5g,h. Integer transforms have nice feature of lossless reconstruction and are very close estimations of floating point transform. Additionally integer transforms can be implemented with integer arithmetic [3] which speed up algorithm and reduce power consumption (important feature for mobile applications).

REFERENCES

[1] Daubechies I., Sweldens W.: *Factoring wavelet transforms into lifting steps.* J. Fourier Anal. Appl., vol4, no3, pp.245-267, 1998.

[2] Soontorn Oraintara, Ying-Jui Chen, Truong Q.Nguyen: *Integer Fast Fourier Transform*. IEEE trans on SP vol. 50 no. 3 March 2002.

[3] Yonghong Zeng, Lizhi Cheng, Guoan Bi, Alex C. Kot: *Integer DCTs and Fast Algorithms*. IEEE trans on SP vol. 49 no. 11 Nov. 2001, pp. 2774–2781.

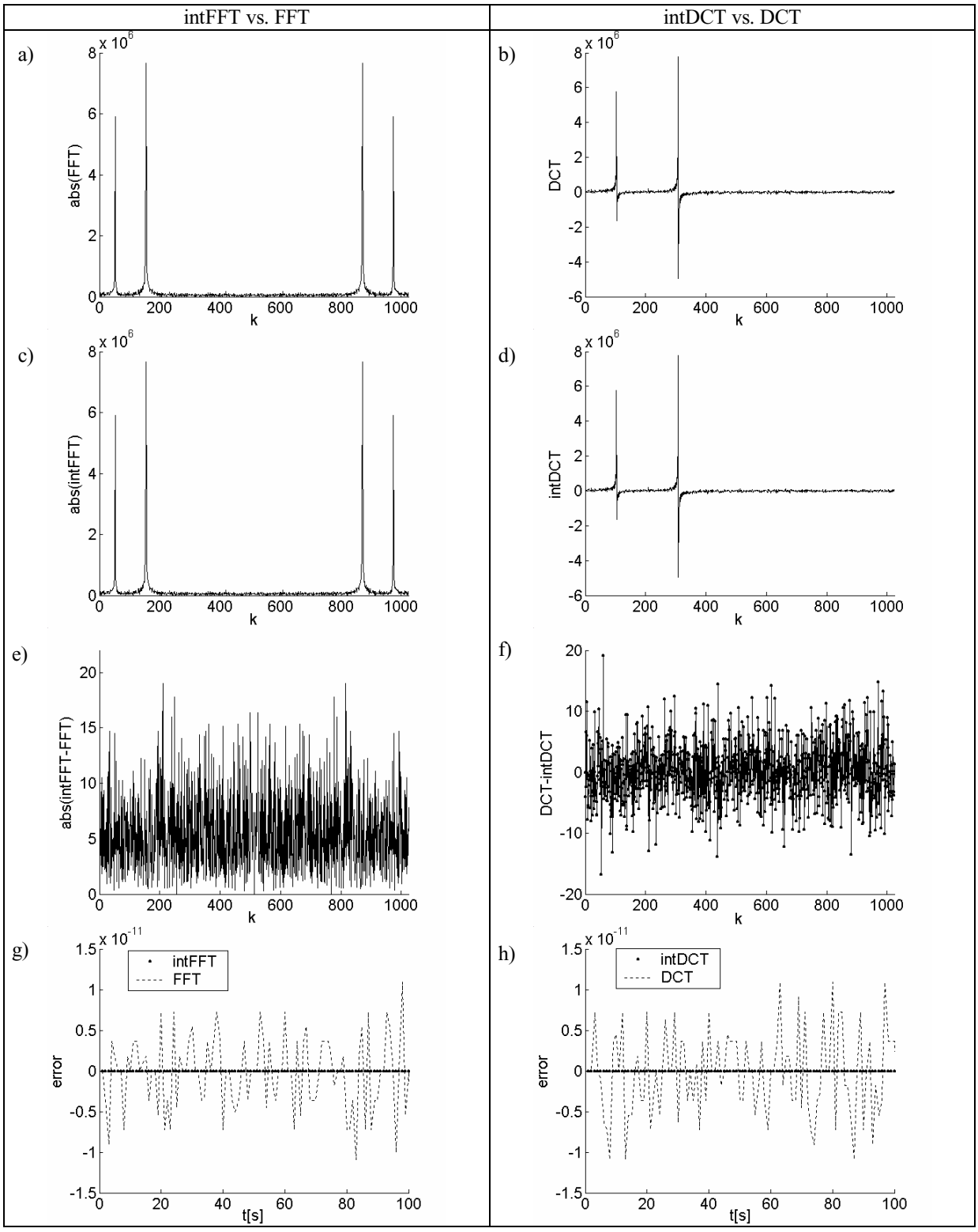

Figure 4: An example of computing intFFT and intDCT and comparison with floating point counterparts FFT and DCT. Left column shows Fourier transform and right column shows cosine transform. Analyzed signal contains 1024 samples with 16 bit precision. Plots shows: a,b - FFT, DCT; c,d - intFFT, intDCT; e,f - the difference between plots a,c and b,d; g,h reconstruction error.