DISCRETE COSINE TRANSFORM-only AND DISCRETE SINE TRANSFORM-only WINDOWED UPDATE ALGORITHMS FOR SHIFTING DATA WITH HARDWARE IMPLEMENTATION

by

Vikram Karwal

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Electrical Engineering

Charlotte

2009

Approved by:

_____
Dr. Yogendra P. Kakad

_____
Dr. Barry G. Sherlock

_____
Dr. Farid M. Tranjan

_____
Dr. Stephen M. Bobbio

_____
Dr. Bharat Joshi

_____
Dr. Isaac M. Sonin

ABSTRACT

VIKRAM KARWAL. Discrete cosine transform-only and discrete sine transform-only windowed update algorithms for shifting data with hardware implementation. (Under the direction of DR. YOGENDRA P. KAKAD and DR. BARRY G. SHERLOCK)

Discrete Cosine Transform (DCT) and Discrete Sine Transform (DST) are widely used in image and data compression applications. To process the DCT or DST of a signal a portion of length N is extracted by windowing. By shifting the window point by point the entire signal can be processed. The algorithms are developed that are capable of updating the DCT and DST independently to reflect the modified window contents i.e. for calculating the DCT of the shifted sequence no DST coefficients are used and similarly for calculating the DST of the shifted sequence no DCT coefficients are used. These algorithms constitute an improvement over previous DCT/DST update algorithms as it establishes independence between the DCT and the DST. The update algorithms used to calculate the transform of the shifted sequence uses less computation as compared to directly evaluating the modified transform via standard fast transform algorithms. Firstly, the r-point, $1 \leq r \leq N$-1, update algorithms are derived in the presence of the rectangular window. Thereafter, one point independent windowed update in the presence of split-triangular, Hanning, Hamming and Blackman windows are developed. The algorithms were implemented in C language to test their correctness. Thereafter the hardware circuits capable of computing the independent update of DCT-II for the rectangular window of size N=8 and step size of 1 and 4 are developed. The windowed update algorithms are derived for DCT and DST type-I through IV, however the hardware implementation of type-II is given as it is the most frequently used transform.

ACKNOWLEDGMENT

I would first like to thank my advisor Dr. Yogendra P. Kakad for his constant support and advice as I worked towards the completion of this dissertation. He was a source of knowledge and constant support throughout my academic career at UNC Charlotte. He has been a source of great inspiration for me.

Special thanks to my co-advisor Dr. Barry Sherlock for his constant support throughout the process of completion of this dissertation.

I would also like to thank Dr. Farid Tranjan, Dr. Bharat Joshi, Dr. Stephen Bobbio and Dr. Isaac Sonin for being the members of my committee.

Lastly, I would like to acknowledge my parents and fiancée Preeti for their good wishes, love and support as I worked towards the completion of this work.

TABLE OF CONTENTS

LIST OF FIGURES ix

LIST OF ABBREVIATIONS xi

CHAPTER 1: INTRODUCTION 1

1.1 Background 1

1.1.1 Evolution of Discrete cosine transform (DCT) and discrete sine transform (DST) 2

1.2 DCT and DST Definitions 6

1.3 The Karhunen-Loéve Transform 8

1.4 Relation between the Karhunen-Loéve and cosine transform 11

1.5 Relation between the Karhunen-Loéve and sine transform 13

1.6 Properties of DCT's and DST's 14

1.7 Organization of the research 16

CHAPTER 2: DCT/DST RECTANGULAR WINDOWED INDEPENDENT UPDATE ALGORITHMS 22

2.1 Evolution of DCT/DST update algorithms 22

2.2 DCT/DST type-I rectangular update derivations 25

2.2.1 Derivation of the *r*-point equations for the DCT and DST 26

2.2.2 Computation of *r*-point independent update equations for DCT and DST 29

2.3 DCT/DST type-II rectangular update derivations 37

2.3.1 Derivation of the *r*-point equations 38

2.3.2 Computation of *r*-point independent update equations 41

2.4 DCT/DST type-III rectangular update derivations 46

2.4.1 Derivation of the *r*-point equations 47

2.4.2 Computation of *r*-point independent update equations DCT/DST-III 52

2.5 DCT/DST type-IV rectangular update derivations 60

2.5.1 Derivation of the *r*-point equations 61

2.5.2 Computation of *r*-point independent update equations DCT/DST-IV 65

2.6 Derivation of DCT-II independent equation in time domain 71

CHAPTER 3: DCT/DST TRAPEZOIDAL WINDOWED INDEPENDENT UPDATE ALGORITHMS 76

3.1 Introduction to independent DCT and DST update algorithms for trapezoidal window 76

3.2 DCT/DST split-triangular simultaneous update algorithms 77

3.3 Independent DCT/DST type-I split-triangular update algorithms for moving data 81

3.3.1 Computation of $C_{w(new)}$ and $S_{w(new)}$ 81

3.3.2 Derivation of correction values for oldest time-step 85

3.4 Independent DCT/DST type-II split-triangular update algorithms for moving data 88

3.4.1 Computation of $C_{w(new)}$ and $S_{w(new)}$ 88

3.4.2 Derivation of correction values for oldest time-step 92

3.5 Independent DCT/DST type-III split-triangular update algorithms for moving data 93

3.5.1 Computation of $C_{w(new)}$ and $S_{w(new)}$ 94

3.5.2 Derivation of correction values for oldest time-step 97

3.6 Independent DCT/DST type-IV split-triangular update algorithms for moving data 99

3.6.1 Computation of $C_{w(new)}$ and $S_{w(new)}$ 99

3.6.2 Derivation of correction values for oldest time-step 103

CHAPTER 4: DCT/DST HAMMING, HANNING, BLACKMAN WINDOWED INDEPENDENT UPDATE ALGORITHMS 106

4.1 Introduction to independent DCT and DST update algorithms for hanning, hamming, and Blackman windows 106

4.2 DCT/DST type-I windowed independent update algorithms in presence of hanning, hamming and Blackman windows 107

4.3 DCT/DST type-II windowed independent update algorithms in presence of hanning, hamming and Blackman windows 111

4.4 DCT/DST type-III windowed independent update algorithms in presence of hanning, hamming and Blackman windows 118

4.5 DCT/DST type-IV windowed independent update algorithms in presence of hanning, hamming and Blackman windows 122

CHAPTER 5: HARDWARE IMPLEMENTATION OF INDEPENDENT DCT-II RECTANGULAR WINDOWED UPDATE ALGORITHM 130

5.1 Introduction to hardware implementation 130

5.2 Explanation of the Dataflow 131

5.2.1 The 16-bit subtractor (SUB_2_16) 132

5.2.2 Adder module (ADD_4_16) 133

5.2.3 Adder module (ADD_8_16) 133

5.3 Hardware implementation for DCT-II independent one point update algorithm 134

5.4 Hardware implementation for DCT-II independent four point update algorithm 143

CHAPTER 6: CONCLUSION AND FUTURE WORK 151

REFERENCES 156

APPENDIX A: C FUNCTIONS FOR INDEPENDENT DCT/DST UPDATE ALGORITHMS 159

APPENDIX B: VHDL CODE FOR HARDWARE IMPLEMENTATION OF INDEPENDENT DCT-II ALGORITHM 190

APPENDIX C: JAVA CODES TO CONVERT DECIMAL TO BINARY AND VICE VERSA 227

LIST OF FIGURES

FIGURE 2.1: Simultaneous DCT/DST update algorithm. 25

FIGURE 2.2: Independent DCT update algorithm. 25

FIGURE 3.1: Split-triangular window *w(x)* with tail length n_o. 78

FIGURE 5.1: SUB_2_16 Module schematic. 133

FIGURE 5.2: ADD_4_16 Module schematic. 133

FIGURE 5.3: ADD_8_16 Module schematic. 134

FIGURE 5.4: Dataflow graph for independent DCT-II one point update. 138

FIGURE 5.5: The simulation results: C0M through C7M. 139

FIGURE 5.6: The simulation results: C0 through C7. 140

FIGURE 5.7: The simulation results from update module. 141

FIGURE 5.8: The simulation results from second iteration in update module. 142

FIGURE 5.9: Dataflow graph for independent DCT-II four point update. 146

FIGURE 5.10: ModelSim simulation for Step 1 four point update. 147

FIGURE 5.11: Simulation results for Step 2 four point update. 148

FIGURE 5.12: The simulation results from update module. 149

FIGURE 5.13: The simulation results from update module second iteration. 150

FIGURE A.1: C function to calculate *r*-point rectangular windowed update for DCT-I. 160

FIGURE A.2: C function implementing DCT-I split-triangular windowed update. 162

FIGURE A.3: C function to calculate *r*-point rectangular windowed update for DST-I. 163

FIGURE A.4: C function implementing DST-I split-triangular windowed update. 165

FIGURE A.5: C program to calculate *r*-point rectangular windowed update DCT-II 170

FIGURE A.6: C program to calculate split-triangular windowed update for DCT-II. 173

FIGURE A.7: C code snippet to calculate independent windowed update for DCT-II in presence of Hanning, Hamming and Blackman windows. 174

FIGURE A.8: C program to calculate *r*-point rectangular windowed update for DST-II. 175

FIGURE A.9: C function to calculate split-triangular windowed update for DST-II. 177

FIGURE A.10: C code snippet to calculate independent windowed update for DST-II in presence of Hanning, Hamming and Blackman windows. 177

FIGURE A.11: C function to calculate *r*-point rectangular windowed update for DCT-III. 178

FIGURE A.12: C function to calculate split-triangular windowed update for for DCT-III. 180

FIGURE A.13: C function to calculate *r*-point rectangular windowed update for DST-III. 181

FIGURE A.14: C function to calculate split-triangular windowed update for DST-III. 183

FIGURE A.15: C function to calculate *r*-point rectangular windowed update for DCT-IV. 184

FIGURE A.16: C function to calculate split-triangular windowed update for for DCT-IV. 186

FIGURE A.17: C code snippet to calculate independent windowed update for DCT-IV in presence of Hanning, Hamming and Blackman windows. 186

FIGURE A.18: C function to calculate *r*-point rectangular windowed update for DST-IV. 187

FIGURE A.19: C function to calculate split-triangular windowed update for DST-IV. 189

FIGURE A.20: C code snippet to calculate independent windowed update for DST-IV in presence of Hanning, Hamming and Blackman windows. 189

LIST OF ABBREVIATIONS

DCT	Discrete Cosine Transform
DST	Discrete Sine Transform
VHDL	Very High Speed Integrated Circuits Hardware Description Language
FCT	Fourier Cosine Transform
JPEG	Joint Photographic Experts Group
MPEG	Motion Picture Experts Group
ODCT's	Odd Discrete Cosine Transforms
ODST's	Odd Discrete Sine Transforms
LUT	Look-Up Table
KLT	Karhunen- Loéve Transform
DTT's	Discrete Trigonometric Transforms
DFT	Discrete Fourier Transform
1-D	One Dimensional
FRDCT's	Fractional DCT's (FRDCT's)
FRDST's	Fractional DST's
MSE	Mean Square Error
RMS	Root mean square error
EDCT's	Even Discrete Cosine Transforms
EDST's	Even Discrete Sine Transforms
MSB	Most Significant Bit

CHAPTER 1: INTRODUCTION

1.1 Background

Transform coding is extensively used in the area of signal processing that provides an efficient way for transmitting and storing data. The input data sequence is divided into suitably sized blocks and thereafter reversible linear transforms are performed. The transformed sequence has much lower degree of redundancy than in the original signal. One of the transforms suitable for Markov-1 type signals that has emerged as a benchmark is the Karhunen-Loéve Transform (KLT) [1, 2, 3]. Most of the typical image data is of Markov-1 type. First-order Markov signals are signals for which the probability of an observation at time *n* depends only on the observation at time $n-1$ i.e. they satisfies the relationship $x_n = \rho x_{n-1} + \xi_n$, where ξ_n represents the white noise and ρ is the inter data-element correlation coefficient [1]. It is usually used for data having very large adjacent-element correlation. KLT is referred to as an optimal transform because it completely decorrelates the signal in the transform domain, minimizes the MSE in bandwidth reduction or data compression, contains the most variance (energy) in the fewest number of transform coefficients and minimizes the total entropy of the sequence. KLT minimizes the distortion between the original and the reconstructed data. However, KLT is a signal dependent transform in the sense that its basis functions depend upon the eigenvectors of the corresponding signal autocorrelation matrix. Since for every different random signal the basis function will be different, therefore it is

sometimes inefficient or impractical to have an efficient practical implementation of the KLT algorithm. The Discrete Cosine Transform (DCT) and the Discrete Sine Transform (DST) perform quite closely to the ideal KLT and have emerged as the practical alternatives to the ideal KLT [4, 5]. Therefore, in this research we choose to study the DCT and the DST transform pair which emerged as one of the most widely used transforms pairs in the field of signal processing. In this research we develop new fast independent update algorithms for the DCT and DST that are capable of performing the update of the real time shifting sequence in the presence of windows.

1.1.1 Evolution of Discrete Cosine Transform (DCT) and Discrete Sine Transform (DST)

Discrete cosine transform (DCT) and discrete sine transform (DST), together known as the Discrete Trigonometric Transforms (DTT's), are the class of sinusoidal unitary transforms [6]. DCT and DST are an invertible linear transform with discrete cosine and sine as their basis functions. DCT is a widely used transform and was initially introduced in 1974 [7]. The great deal of popularity of the DCT and the DST is due to the existence of fast algorithms that allow their efficient computation. Like the Discrete Fourier Transform (DFT), a lot of algorithmic research has been undertaken for the fast implementation of the DCT and DST [8-14]. Many fast algorithms for the efficient computation of DCT and DST have been developed and are found in the literature [1]. These algorithms developed can be classified mainly into two broad categories, one which does the indirect computation of the transform using other well known orthogonal transforms, and other set of algorithms that involve direct computation using recursive sparse matrix factorization of the transform matrix like the radix-2, split-radix and mixed-radix algorithms [15, 16].

The DCT algorithm is widely used for data compression because of its powerful bandwidth reduction capability. Discrete Cosine Transforms are orthogonal, information of a signal is preserved under transformation, and separable i.e. a multidimensional DCT can be implemented by a series of one-dimensional transforms. The benefit of this property is that the fast algorithms developed for the one-dimensional DCT can be directly extended to multidimensional transforms; this is advantageous from a simulation viewpoint as well as for hardware realization [17, 18]. DCT like the Discrete Fourier Transform (DFT), transforms a signal or image from the spatial domain to the frequency domain, where much of the energy lies in the lower frequencies coefficients. The main advantage of the DCT over the DFT is that DCT involves only real multiplications [1]. The DCT does a better job of concentrating energy into lower order coefficients than the DFT for image data. This property of the DCT, the energy compaction, resulted in adoption of the DCT as a standard for image compression in JPEG and MPEG standards. DCT (DST) algorithm uses cosine (sine) as the basis function thus eliminating the use of complex numbers, unlike the Fourier transform, which uses complex function $e^{-j\omega t}$ as its basis [4, 5].

For the real-time processing of the infinite input sequence of data, a portion of the sequence is sampled and the transform is performed. New input data points are shifted in as and when they are available and the oldest data points are shifted out. To calculate the transform of the new sequence the technique of using the update algorithm [8, 9, 12, 22] is used. Algorithms have been developed to update the DCT and DST simultaneously to reflect the modified window contents [22, 23]. These update algorithms are computationally less expensive than directly re-evaluating the transform via standard fast

transform algorithms. Initially analogous algorithms for the update of the Discrete Fourier Transform (DFT) were developed in [10-12]. Areas of application where these "update" algorithms can be really beneficial include the real-time data communication, real-time analysis of financial market data, target detection/recognition, adaptive system identification, filtering and real-time analysis of financial market data.

A great deal of research for the development of fast algorithms to efficiently compute the DCT and DST has been achieved since its introduction in 1974 [1]. The remaining part of this section lists some of the efficient ways of computing the transforms.

DCT's and DST's are real-valued transforms that transforms the integer-valued signals to the floating-point coefficients [24, 25]. The floating-point implementation in hardware is slow, requires too much memory and also consumes more power. However for faster realization, floating-point multipliers can be approximated by integers. Implementation is realized in fixed-point arithmetic where each floating-point multiplier is approximated as $\pm m \cdot 2^b$, where b is an integer exponent and m is an integer mantissa [1]. Therefore the DCT's and DST's with integer coefficients are of great interest since it's easier to design and can be implemented efficiently consuming low-power. Since this implementation consumes less power, the major applications include mobile computing and hand-held devices that run on batteries. The resulting integer transforms accuracy is comparable with the original real-valued transforms and preserve all their basic mathematical properties such as linearity, orthogonality, symmetry of the basis vectors and recursivity. The transform coding gain and transform efficiency of the integer approximated transform are also comparable to the original transform. The integer

approximated algorithms enable efficient hardware implementation because the design uses only binary additions and shifts [18].

A number of fast 1-D rotation-based algorithms for the computation of DCT's and DST's based on the recursive sparse matrix factorizations of the corresponding DCT and DST have been developed [17]. Fast direct 2-D DCT/DST algorithms also exist in theory [19]. Since 2-D DCT/DST are separable, the 2-D DCT/DST computation can be realized by sequentially using any fast 1-D DCT/DST algorithms on rows and columns of the input data matrix. Therefore the multi-dimensional DCT/DST algorithms can be decomposed into 1-D DCT/DST algorithms.

Most of the fast DCT/DST algorithms developed earlier were radix-2 algorithms, however, in practice various sequence lengths other than a power of 2 may occur. To extend this idea to include any length, new fast even/odd-length, composite-length, radix-q ($N = q^n$) and mixed-radix, where $N = 2^n q,$ algorithms have been proposed [15, 16]. These algorithms developed for sequence lengths other than $N = 2^n$ however generally have higher computational complexities.

Due to the correlation property of the DCT and DST, a great deal of the energy of the signal lies in the low-frequency coefficients. Therefore we can select only these high information coefficients and neglect the other low energy coefficients. Such a method where only a subset of the output coefficients is used to enhance the computational efficiency is known as "pruning". In other words, instead of assuming that the lengths of input and output data sequences to be equal as in the case of standard DCT algorithms we choose only high information coefficients. This is really useful for the data compression applications. Yet another class of algorithms result known as pruning algorithms [26, 27].

A class of algorithms with recursive filter structures is developed in reference [19]. In this class of recursive algorithms DCT/DST kernels are converted to regular regressive structures based on sinusoidal recursive formulae, or recurrence formulae for Chebyshev polynomials, or Clenshaw's recurrence formula. There recursive structures are specifically useful for the parallel VLSI implementation of the variable length DCT's and DST's.

Fractional DCT's (FRDCT's) and fractional DST's (FRDST's) are developed in [28, 29] and are based on eigen decomposition of the corresponding DCT and DST matrices i.e. FRDCTs and FRDSTs are defined through the "fractional" real powers of DCT and DST matrices. This class of algorithms led to efficient implementation of the DCT and DST algorithms.

A classic computer does not allow to calculate *N*-point DCT's or DST's, $N = 2^n$, in less than linear time. Fast quantum algorithms for DCT's and DST's [1] is a class of algorithms capable of realizing *N*-point DCT's and DST's using a quantum computer with much less computational complexity as compared to the classical computer. Hence, extremely fast quantum DCT/DST algorithms can be derived and implemented on the quantum machines.

1.2 DCT and DST Definitions

DCT's and DST's are a class of discrete sinusoidal unitary transforms also known as the Discrete Trigonometric Transforms (DTT's). The definitions of even DCT's type-I through type-IV are as follows [7]:

$$C_I(k) = \sqrt{\frac{2}{N}} P_k \sum_{x=0}^{N} P_x f(x) \cos\frac{xk\pi}{N} \tag{1.1}$$

for $k = 0,1,...,N$

$$C_{II}(k) = \sqrt{\frac{2}{N}} P_k \sum_{x=0}^{N-1} f(x) \cos \frac{(2x+1)k\pi}{2N} \tag{1.2}$$

for $k = 0,1,...,N-1$

$$C_{III}(k) = \sqrt{\frac{2}{N}} \sum_{x=0}^{N-1} P_x f(x) \cos \frac{x(2k+1)\pi}{2N} \tag{1.3}$$

for $k = 0,1,...,N-1$

$$C_{IV}(k) = \sqrt{\frac{2}{N}} \sum_{x=0}^{N-1} f(x) \cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right] \tag{1.4}$$

for $k = 0,1,...,N-1$

and DST type-I through type-IV are:

$$S_I(k) = \sqrt{\frac{2}{N}} \sum_{x=1}^{N-1} f(x) \sin \frac{xk\pi}{N} \tag{1.5}$$

for $k = 1,2,...,N-1,$

$$S_{II}(k) = \sqrt{\frac{2}{N}} P_k \sum_{x=0}^{N-1} f(x) \sin \frac{(2x+1)k\pi}{2N} \tag{1.6}$$

for $k = 1,...,N,$

$$S_{III}(k) = \sqrt{\frac{2}{N}} \sum_{x=0}^{N} P_x f(x) \sin \frac{x(2k+1)\pi}{2N} \tag{1.7}$$

for $k = 1,2,...,N,$

$$S_{IV}(k) = \sqrt{\frac{2}{N}} \sum_{x=0}^{N-1} f(x) \sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right] \tag{1.8}$$

for $k = 0,1,...,N-1,$

where, $P_j = \frac{1}{\sqrt{2}}$ for $j = 0$ or N, and 1 otherwise. C represents the DCT transform, S represents the DST transform on the input signal $f(x)$.

1.3 The Karhunen-Loéve transform

The Karhunen-Loéve Transform (KLT), also known as Hotelling Transform and the Principal Component Transform, was first introduced by Karhunen and Loéve [1]. This section discusses the eigen-solutions of the optimal KLT and sections 1.4 and 1.5 list its relation to the DCT and DST respectively.

Let us consider the case of Markov-1 type signal x of length N with a zero mean value i.e.

$$x = \{x_0, x_1, ..., x_{N-1}\}^T \tag{1.9}$$

If $\{\Phi_i\} = [\Phi_0, \Phi_1, ..., \Phi_{N-1}]$ is the orthogonal basis vectors, representing a set of linearly independent vectors that span the N-dimensional vector space. Then x can be expressed as a linear combination of these orthogonal basis vectors i.e.

$$x = \sum_{i=0}^{N-1} X_i \Phi_i \tag{1.10}$$

The coefficients of expansion X_i, can be calculated using the inner product principle

$$X_i = \langle x, \Phi_i \rangle / \langle \Phi_i, \Phi_i \rangle \qquad \text{for } i=0,1,\ldots,N\text{-}1 \tag{1.11}$$

Thus the vector can be represented either by actual samples $\{x_n\}$ as in equation (1.9) or the N numbers in equation (1.11). To achieve data compression and reduction in the bandwidth for data transmission, signal x in equation (1.10) can be represented by only the first D coefficients, where ($D<N$), if other coefficients can be neglected, then vector x can be reconstructed by the first D coefficients. The truncated version $\tilde{x}$ (i.e. the version with the first D coefficients) can be written as:

$$\tilde{x} = \sum_{i=0}^{D-1} X_i \Phi_i \tag{1.12}$$

The Mean Square Error (MSE), ε, resulting from the truncation is given by:

$$\varepsilon = E[(x - \tilde{x})^2] \tag{1.13}$$

We want the parameter ε to be as small as possible. The problem is to find the set of basis vectors $\{\Phi_i\}$ that will minimize the above error.

$$= E\left[\langle \sum_{i=D}^{N-1} X_i \Phi_i , \sum_{i=D}^{N-1} X_i \Phi_i \rangle \right] \tag{1.14}$$

E being the expectation operator. If in addition the basis vectors are to be normalized, then the additional condition to be satisfied is:

$$\langle \Phi_i , \Phi_k \rangle = \delta_{ik} \qquad \text{for } i,k = 0,1,2,\ldots,N\text{-}1, \tag{1.15}$$

Solving equation (1.13) yields:

$$\varepsilon = E\left[\sum_{i=D}^{N-1} |X_i|^2 \right]$$

i.e.

$$\varepsilon = E\left[\sum_{i=D}^{N-1} |\langle x, \Phi_i \rangle|^2 \right]$$

Therefore,

$$\varepsilon = E\left[\sum_{i=D}^{N-1} \Phi_i{}^T x x^T \Phi_i \right] = \sum_{i=D}^{N-1} \Phi_i{}^T E[x x^T] \Phi_i \tag{1.16}$$

the auto-covariance matrix *A* of the signal *x* can be written as:

$$A = E[x x^T] \tag{1.17}$$

Substituting equation (1.17) in equation (1.16) results in:

$$\varepsilon = \sum_{i=D}^{N-1} \Phi_i{}^T A \Phi_i \tag{1.18}$$

The variational equation needed to find the basis vectors is:

$$\left(\frac{\delta}{\delta\Phi_i}\right)\left(\varepsilon-\mu_i\langle\Phi_i,\Phi_i\rangle\right)=0 \tag{1.19}$$

When the variational derivative is taken, we get the following equation,

$$\left([A]-\mu_i[I_N]\right)\Phi_i=0 \qquad \text{for } i=0,1,\ldots,N\text{-}1 \tag{1.20}$$

where, μ_i represents the Lagrange multiplier, used for the constraint, and I_N the identity matrix of size $N\times N$. The resultant set of eigenvectors should diagonalize the auto-covariance matrix i.e. $[\Phi]^{-1}[A][\Phi]=diag[\mu_0,\mu_1,...,\mu_{N-1}]$. The eigenvectors given by $\{\Phi_i\}$ form the Karhunen-Loéve transform matrix. For the first-order Markov signal, auto-covariance matrix is given by [1]:

$$[A]_{ik}=\rho^{|i-k|} \qquad \text{where } i,k=0,1,\ldots,N\text{-}1. \tag{1.21}$$

ρ is the positive adjacent correlation coefficient with a magnitude less than unity.

i.e.
$$[A]_{ik}=\begin{bmatrix} 1 & \rho & \ldots & \rho^{N-1} \\ \rho & 1 & \ldots & \rho^{N-2} \\ \ldots & \ldots & \ldots & \ldots \\ \rho^{N-1} & \rho^{N-2} & \ldots & 1 \end{bmatrix} \tag{1.22}$$

The off-diagonal entries correspond to nonzero inter-element correlation. In discrete domain, the n^{th} component of the m^{th} eigenvector for the solution of equation (1.20) is given by:

$$\Phi_m(n)=\left[\frac{2}{(N+\mu_m)}\right]^{1/2}\sin\left\{\omega_m\left[n-\frac{(N-1)}{2}\right]+(m+1)\frac{\pi}{2}\right\} \tag{1.23}$$

where, μ_m the eigenvalues are defined by equation (1.24), and ω_m's are the real positive roots of the transcendental equation.

$$\mu_m=-\frac{1-\rho^2}{1-2\rho\cos(\omega_m)+\rho^2} \tag{1.24}$$

$$\tan(N\omega_m) = \frac{-(1-\rho^2)\sin(\omega_m)}{(1+\rho^2)\cos(\omega_m) - 2\rho]} \tag{1.25}$$

ω_m are the N eigensolutions.

KLT acts as a benchmark and is an optimal transform for Markov-1 type signals. KLT acts as the benchmark and the degree to which other transforms exhibit behavior close to KLT is a measure of their efficiency. The processing complexity and closeness to the optimal KLT is the measure of importance of the transform.

1.4 Relation between the Karhunen-Loéve and cosine transform

The Discrete Cosine Transform (DCT) performs quiet close to the optimal KLT and it was demonstrated theoretically by [4] that it can be derived from the optimum Karhunen-Loéve transform in the limiting case when the adjacent data-element correlation ρ tends to unity. The transform basis function for the discrete signal is given by:

$$[\Phi]_{mn} = \Phi_m(n)$$

$$= \left[\frac{2}{(N+\mu_m)}\right]^{1/2} \sin\left\{\omega_m\left[n - \frac{(N-1)}{2}\right] + (m+1)\frac{\pi}{2}\right\} \quad \text{where, } m,n=0,1,\ldots,N\text{-}1. \tag{1.26}$$

As ρ tends to one, ω_m lies in the range $0 \le \omega_m \le \pi$ and the denominator of equation (1.25) is nonzero. Therefore it implies that $\tan(N\omega_m) = 0$ i.e.

$$\omega_m = \frac{k\pi}{N} \text{ where } k=0,1,\ldots,N\text{-}1$$

except ω_0 which results in denominator of equation (1.25) to be zero. To find ω_0 using small-angle substitution i.e. $\tan\alpha \to \alpha$, $\sin\alpha \to \alpha$, $\cos\alpha \to 1-\alpha^2/2$, as $\alpha \to 0$.

Equation (1.25) reduces to:

$$N\omega_0 = \frac{-(1-\rho^2)\omega_0}{(1+\rho^2)\left(1-{\omega_0}^2/2\right)-2\rho]}$$

i.e.
$$= \frac{-(1-\rho^2)\omega_0}{(1-\rho)^2-(1+\rho^2){\omega_0}^2/2} \tag{1.27}$$

As ρ tends to one, ${\omega_0}^2 = (1-\rho^2)/N$.

Substituting the above value of ω_m in equation (1.26) yields:

$$= \left[\frac{2}{(N+\mu_m)}\right]^{1/2} \sin\left\{\frac{k\pi}{N}\left[n-\frac{(N-1)}{2}\right]+(m+1)\frac{\pi}{2}\right\} \tag{1.28}$$

Analyzing equation (1.24) the eigenvalue equation for all *m*=0 yields:

$$\mu_0 = \frac{1-\rho^2}{1-2\rho\cos(\omega_0)+\rho^2} \tag{1.29}$$

Small-angle substitutions i.e. $\cos\alpha \to 1-\alpha^2/2$ above equation reduces to

$$\mu_0 = \frac{1-\rho^2}{1-2\rho\left(1-{\omega_0}^2/2\right)+\rho^2}$$

Substituting ${\omega_0}^2 = (1-\rho^2)/N$ as ρ tends to one results in,

$$\mu_0 = \frac{1-\rho^2}{1-2\rho\left(1-(1-\rho^2)/2N\right)+\rho^2}$$

$$= \frac{1-\rho^2}{1-2\rho+\rho(1-\rho^2)/N+\rho^2}$$

$$= \frac{1+\rho}{1-\rho+\rho(1+\rho)/N} \tag{1.30}$$

Equation (1.30) tends to *N* as ρ tends to one and

$$\sum_{m=0}^{N-1}[A]_{mn} = \sum_{m=0}^{N-1}\mu_m$$

Since all the diagonal elements of the auto-covariance matrix [A] for Markov-1 type signal are equal to 1 leads to (μ_0=N). The equations (1.28) can be rewritten as:

$$[\Phi]_{n0} = \frac{1}{\sqrt{N}}$$

$$[\Phi]_{nm} = \sqrt{\frac{2}{N}} \sin\left[m\left(n+\frac{1}{2} \right)\frac{\pi}{N} + \frac{\pi}{2} \right]$$

$$= \sqrt{\frac{2}{N}} \cos\left[m\left(n+\frac{1}{2} \right)\frac{\pi}{N} \right] \quad \text{for } m \neq 0 \tag{1.31}$$

Therefore,

$$[\Phi]_{nm} = \sqrt{\frac{2}{N}} k_m \cos\left[m\left(n+\frac{1}{2} \right)\frac{\pi}{N} \right] \quad \text{for } m,n=0,1,\ldots,N\text{-}1 \tag{1.32}$$

where, $k_m = 1/\sqrt{2}$ if m=0 and 1 otherwise. It is obvious that equation (1.32) matches equation (1.2) which represents the DCT-II. It can be inferred from equation (1.32) that for stationary Markov-I type signals with adjacent correlation coefficient ρ approaching 1, DCT-II is equivalent to KLT independent of the value of N. In the case of perfectly correlated signal, all the variance, i.e. energy, of the signal is in the first eigenvector, which is the DC component. As ρ decreases, the variance of the signal is spread over the other eigenvectors as well.

1.5 Relation between the Karhunen-Loéve and sine transform

Section 1.4 shows theoretically how cosine transform can be derived from the optimum Karhunen-Loéve transform in the limiting case when the adjacent data-element correlation tends to unity. In an alternative limiting case when the correlation tends to

zero, the sine transform results [5]. The derivation of how sine transform can be derived from the KLT is presented here:

From equation (1.25) for a limiting case wherein ρ tends to zero yields,

$$\tan(N\omega_m) = -\tan\omega_m$$

i.e.
$$\tan(N\omega_m) = \tan(\pi - \omega_m)$$

$$= \tan(\pi - \omega_m + m\pi) \tag{1.33}$$

Therefore,

$$\omega_m = \frac{m+1}{N+1}\pi \tag{1.34}$$

As ρ tends to zero, $\mu_m \to 1$ for all m, substituting the value of ω_m and μ_m in equation (1.23) results in:

$$\Phi_m(n) = \left[\frac{2}{N+1}\right]^{1/2} \sin\left\{\frac{\pi(m+1)}{2(N+1)} \times \{2n - N + 1 + (N+1)\}\right\}$$

$$= \left(\frac{2}{N+1}\right)^{1/2} \sin\left(\frac{\pi(m+1)(n+1)}{N+1}\right) \tag{1.35}$$

Analyzing equation (1.35) shows that the transform basis functions are sine, demonstrating that sine transform can be derived as a limiting case of the optimal KLT for the case where the adjacent element correlation coefficient ρ tends to zero. It is obvious that equation (1.35) matches equation (1.6) which represents the DST-II.

1.6 Properties of DCT's and DST's

Some of the important properties of the DCT's and DST's [1] are:

1. *The unitarity property*: DCT is a unitary transform i.e. orthonormal, hence the energy of the coefficients in the transform domain is equal to the energy of the original input

signal. The columns of these transform matrices are eigenvectors of symmetric real matrices.

2. *The linearity property*: The DCT and DST transforms are linear that is for two sequences $x(t)$ and $y(t)$:

$$D[\alpha x(t) + \beta y(t)] = \alpha D[x(t)] + \beta D[y(t)] \tag{1.36}$$

where α and β are constants.

3. *The scaling in time property*: The discrete transforms deal with discrete time samples and the results of the transforms are discrete frequency samples. A scaling in time will result in an inverse scaling in the frequency domain with no impact on the overall transform. Based on the time-frequency uncertainty principle, if Δt and Δf are respectively the time and frequency units then

$$\Delta t \text{ and } \Delta f \geq (1/2\pi)\Delta f \tag{1.37}$$

Therefore, when Δt is scaled by a factor a to become $a\,\Delta t$, the frequency unit Δf must be scaled by the inverse of a to become $(1/a)\Delta f$ so that equation (1.37) remains unchanged. The overall magnitude of the transform remains unchanged.

4. *The shift in time property*: Let $x = \{x_0, x_1, ..., x_{N-1}\}^T$ then as one new data point is shifted in, the shifted sequence is given by $x_+ = \{x_1, x_2, ..., x_N\}^T$, where x_+ is a one point shifted version of sequence x. While these two sequence vectors are completely different for the purpose of transform analysis, it has been shown that the transforms are related to each other [9, 22]. The shift properties of the DCT's and DST's are particularly useful in applications where time constraints may not permit the immediate processing of every incoming sample point. Earlier algorithms developed by [22, 23] are modified to achieve

independence between DCT and DST update algorithms. These update algorithms can be used to process the modified DCT/DST of the shifted sequence with much less computational burden as compared to using the transform definition. Chapter 2, 3 and 4 mainly deals with developing and modifying the existing algorithms related to shift in time properties of DTT's.

5. *The difference property*: There are some applications such as differential pulse code modulation (DPCM), where the differencing of adjacent samples in a signal is required. Considering a sequence consisting of differences of adjacent samples in a signal

$$d(n) = x(n+1) - x(n) \qquad \text{for } n=0,1,\ldots,N\text{-}1, \tag{1.38}$$

Which in vector form is

$$\mathbf{d} = \mathbf{x}_{+} - \mathbf{x} \tag{1.39}$$

Taking DCT of the above equation yields:

$$C[D] \equiv C[X_{+}] - C[X] \tag{1.40}$$

The shift properties of DCT, i.e. property 4, can be applied to solve equation (1.40).

6. *Convolution properties*: The concept of convolution used in DFT can be extended for the discrete cosine transform and discrete sine transform. Consider two sequences $x(n)$ and $y(n)$ for $n=0,1,\ldots,N\text{-}1$. The convolution-multiplication relation for DCT-I is:

$$C[x(n) \otimes_c y(n)] = C[x(n)] \times C[y(n)] \tag{1.41}$$

Where $\otimes$ denotes the convolution operation.

1.7 Organization of the research

This chapter involves the background study and discusses the importance of transform coding techniques for processing of digital signals. The evolution of the sinusoidal unitary transforms, namely the DCT's and DST's is introduced. The

definitions of the four even DCT's (EDCT's), also known as type-I through type-IV DCT's, and four even DST's (EDST's), also known as type-I through type-IV DST's, are listed and their properties of the DTT's are discussed. Thereafter the development of the various fast algorithms for the efficient implementation of the DCT's and DST's are studied. The optimal KLT and its near ideal behavior for signal processing applications is discussed and the derivation of DCT and the DST as a special constraint of KLT is listed.

In chapter 2 the DCT-only *r*-point rectangular windowed update algorithms independent of the DST coefficients and correspondingly DST-only rectangular window update algorithms independent of the DCT coefficients are derived. DCT-II and DST-II are the transforms, as discussed earlier, closest to the optimal KLT in performance. Although DCT/DST-II are the most often used transforms, the algorithms are developed for DST/DCT type-I through IV. The algorithms derived here are capable of updating *r*, $1 \le r \le N-1$, new input data points at a time. The data sequence is sampled with the rectangular window of length *N* and DCT/DST is performed on this sampled window. The window is then shifted along the entire signal and the transform of the entire window is performed. Instead of using the definition to perform the transform of the shifted sequence, "update" algorithm can be used which are computationally much more efficient [22]. The independent update algorithms developed, use the new data point which are shifted in, the old data points shifted out, the current time unit DCT coefficients and one older time unit DCT coefficient values to calculate the transform of the new data sequence with shifted data. The algorithms developed can be used recursively to calculate the update whenever the new input data points become available. Analogous DST-only windowed update algorithms independent of the DCT coefficients

are developed. Earlier independent algorithms developed by Chicharo and Xi [30] were capable of finding the transform of the updated sequence but they were limited to one new data point at a time. In chapter 2, this concept was extended to calculate the update in the presence of windows. Also, the *r*-point update algorithms in the presence of rectangular windows are derived. Thereafter algorithms in the presence of more appropriate windows [32] such as split-triangular window, Hamming, Hanning and Blackman windows are developed. Sherlock and Kakad [22, 23, 33] developed the updated algorithms in the presence of rectangular, split-triangular, Hanning, Hamming and Blackman windows. However, these algorithms required simultaneous update of the DCT and DST coefficients i.e. whenever the transform of the updated sequence is desired we need the new points shifted in, the old data points shifted out and the DCT and DST coefficients of the older time sequence. In other words, the process of updating a DCT (or DST) requires updating the corresponding DST (or DCT) as well. However, because of this interdependency there is excessive computational burden. The algorithms developed in Chapter 2 are an improvement over the algorithms developed earlier because it enables an *r*-point independent update of the real time input sequence in presence of the rectangular window which results in more efficiency.

This concept of windowed update can be extended to sample the input infinite sequence with other well known windows such as split-triangular, Hamming, Hanning and Blackman windows. Windows other than rectangular are useful because they reduce the edge effects such as ringing introduced by the use of the rectangular window. The rectangular window treats the *N*-point data sequence to have symmetries and periodicities which usually do not occur in practical signals. The use of these windows modifies the

sample to make it continuous at the edges, smoother, when regarded as periodically repeated. Chapter 3 develops independent update algorithms for split-triangular window and Chapter 4 focuses on windowed update for Hamming, Hanning and Blackman windows.

The concept of using the split-triangular window to sample the infinite sequence and moving the split-triangular window gradually one point at a time to move over the entire sequence was initially developed by [22, 23], however these algorithms involved the simultaneous update of the DCT and DST coefficients. This idea was extended in Chapter 3 and algorithms are derived that are capable of independently updating the windowed sequence in the presence of split-triangular window. There algorithms are derived for DCT and DST type-I through IV. The split-triangular window is shifted one point at a time over the entire data sequence and the updated windowed transform is calculated. C language code is written to test the correctness of the derived algorithms and tested for different values of input data points and varying sequence lengths.

Chapter 4 utilizes Hamming, Hanning and Blackman windows to sample the input data sequence. The independent windowed update algorithms are developed that are capable of updating the sequence whenever the new data points become available in the presence of Hanning, Hamming and Blackman window. C language code is written for the analytically derived update algorithms and the tests were carried out for different values of input data points. As discussed earlier, the shift algorithms developed in [22, 23] are capable of updating the signals in the presence of Hanning, Hamming and Blackman windows but require simultaneous update of DCT and DST coefficients. The algorithms developed in Chapter 4 however, are capable of independently updating the

DCT (or DST) using only DCT (or DST) coefficients. This is particularly useful in applications were only the DCT coefficients or only the DST coefficients are available.

Chapter 5 focuses on the hardware implementation of the DCT-II independent update algorithms developed in Chapter 2 for the case of rectangular window of length N=8. Hardware implementation is carried out for one point update at a time i.e. r=1 and four point update at a time i.e. for the case, r=4. State machines are written in VHDL and simulated on ModelSim. The hardware is capable of computing the DCT update coefficients using the current value of DCT coefficients, one older time-step DCT coefficients, the new data points shifted in and the old data point(s) shifted out without using the DST coefficients. The chapter lists the simulation results for test vectors to validate the hardware design. The state machine implements equation (2.25), the r-point independent rectangular windowed update equation developed in Chapter 2. Initial processing of the data, to calculate the current time sequence DCT and one older time period DCT is carried out using the conventional mode module developed in [31]. Thereafter, the independent update algorithm developed in this research is used to calculate the transform of the updated sequence as the new data points become available. The test bench is used to test the implemented design through the RMS error measure between the values obtained through the hardware design and the values calculated using the transform definition (using matlab). The dataflow diagrams for the two architectures implemented are also discussed.

Chapter 6 culminates the research work in terms of concluding remarks and suggestions for future work. It also includes the performance analysis of the algorithms

developed in Chapter 2, 3 and 4 discussing the complexity of the algorithms developed in this research.

Appendix A provides the C language codes used to implement the algorithms for independent windowed update for DCT/DST I through IV for rectangular, split-triangular window, Hamming, Hanning and Blackman windows.

Appendix B includes the VHDL codes used to implement the hardware architectures for one point update and four point update for DCT-only rectangular windowed update. Test benches used to test the correctness of the implemented hardware are also given.

Appendix C provides the Java functions used to convert the decimal number to binary number and binary number to decimal number.

CHAPTER 2: DCT/DST RECTANGULAR WINDOWED INDEPENDENT UPDATE ALGORITHMS

2.1 Evolution of DCT/DST update algorithms

Applications where the DCT/DST transform of the real-time data is to be processed, the input data is windowed and thereafter the window is shifted point by point to form the DCT/DST of the entire signal. Initially algorithms were developed by Rao and Yip [9], known as shift properties of DCT/DST, to "update" the DCT and DST transform as the new input data points become available. These "update" algorithms used to calculate the updated DCT/DST is far more efficient than directly calculating the DCT/DST of the shifted sequence. As these update algorithms are computationally much less complex than directly calculating the transforms, these algorithms are used in several different applications such as adaptive system identification and filtering, real-time analysis of financial market data etc. These update algorithms developed by Rao and Yip [9] are limited to one-point update at a time and can be used only for the case of rectangular window. Also, these algorithms perform simultaneous update of DCT and DST coefficients i.e. to form the updated DCT coefficients, the DST coefficients need to be calculated simultaneously and vice versa. This concept was further enhanced by Kakad and Sherlock [22, 23] to include r-point update where $1 \leq r \leq N$-1 and algorithms were developed for update in the presence of rectangular window, split-triangular window, Hamming, Hanning and Blackman windows. Since rectangular window treats data to be continuous at the edges it introduces ringing effects. Other windows exist in

theory that can be used to reduce this ringing effect [32].

The DCT and DST type-II update algorithms as developed by Kakad and Sherlock [22] for rectangular window are given by equations (2.1) and (2.2) respectively. These algorithms require the simultaneous update of DCT and DST coefficients.

For DCT type-II the update equation is given as follows:

$$C_{+}(k) = \cos\frac{rk\pi}{N} C(k) + \sin\frac{rk\pi}{N} S(k) + P_k \sqrt{\frac{2}{N}} \sum_{x=0}^{r-1} [(-1)^k f(N+r-1-x) - f(r-1-x)] \cos\frac{(2x+1)k\pi}{2N} \Bigg]. \qquad (2.1)$$

$$\text{for } k = 0,1,\ldots,N-1,$$

where $C(k)$ and $S(k)$ are the definitions of DCT and DST respectively as derived by [7] and given in equations (2.15) and (2.16). $C_{+}(k)$ represents the updated DCT coefficients.

The analogous equation for updating the DST type-II is:

$$S_{+}(k) = \cos\frac{rk\pi}{N} S(k) - \sin\frac{rk\pi}{N} C(k) - P_k \sqrt{\frac{2}{N}} \sum_{x=0}^{r-1} [(-1)^k f(N+r-1-x) - f(r-1-x)] \sin\frac{(2x+1)k\pi}{2N} \Bigg]. \qquad (2.2)$$

$$\text{for } k = 1,2,\ldots,N.$$

where, $S_{+}(k)$ represents the updated DST coefficients.

Chicharo and Xi [30] enhanced this technique and developed one-point independent update algorithms for DCT and DST. However these algorithms were only capable of updating one point at a time and worked for non-windowed data. The results for DCT/DST type-II one-point independent update are listed in equation (2.3) and equation (2.4) for convenience.

$$C(n+1,k) = 2\cos\frac{k\pi}{N}C(n,k) - C(n-1,k)$$

$$+\sqrt{\frac{2}{N}}P_k\cos\frac{k\pi}{2N}\left[(-1)^k f(n) - (-1)^k f(n-1) - f(n-N) + f(n-N-1)\right] \qquad (2.3)$$

$$\text{for } k = 0,1,\ldots,N-1,$$

$$S(n+1,k) = 2\cos\frac{k\pi}{N}S(n,k) - S(n-1,k)$$

$$+\sqrt{\frac{2}{N}}P_k\sin\frac{k\pi}{N}\left[-(-1)^k f(n) - (-1)^k f(n-1) + f(n-N) + f(n-N-1)\right] \qquad (2.4)$$

$$\text{for } k = 1,\ldots,N.$$

where, $C(n,k)$ and $S(n,k)$ are the DCT and DST coefficients of the original data sequence.

In this research *r*-point independent update algorithms for DCT and DST type-I through IV are developed. The DCT independent update algorithms do not require the DST coefficients and the DST independent update algorithms do not require the DCT coefficients. These algorithms will result in easier implementation for applications where only the DCT or only the DST coefficients are required. Firstly, the independent update algorithms for rectangular windows are developed (Chapter 2) and thereafter algorithms for use with split-triangular window (Chapter 3) and Hamming and Hanning windows (Chapter 4) are developed. C language program codes were written, see Appendix A, and tested for different sequence length *N* and input data points to test the accuracy of the derived algorithms.

Figure (2.1) shows the basic idea behind calculating the DCT or the DST update as defined by [22, 23, 33] and figure (2.2) shows the idea behind the *r*-point independent DCT update algorithms developed in this chapter.

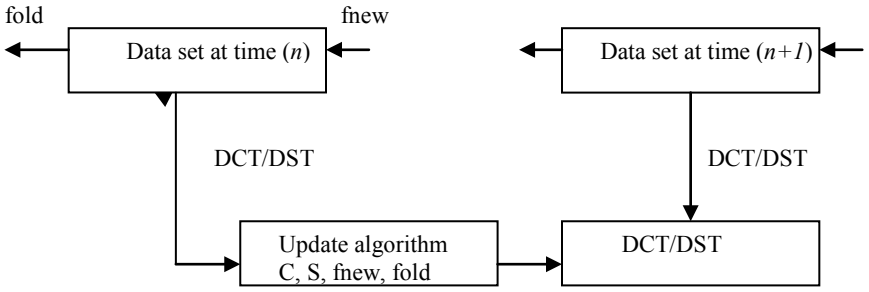

Figure 2.1: Simultaneous DCT/DST update algorithm

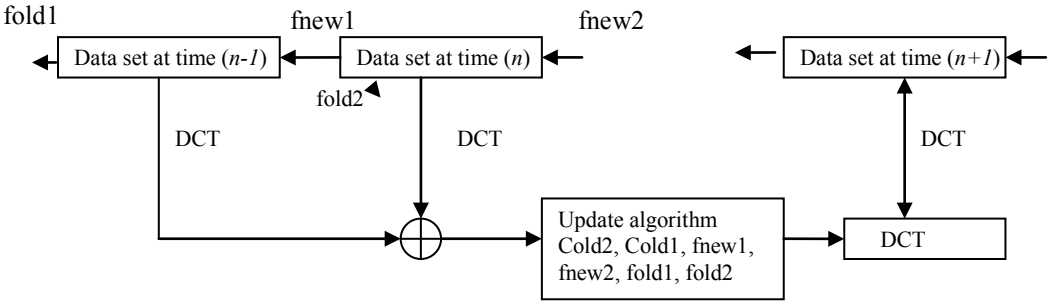

Figure 2.2: Independent DCT update algorithm

2.2 DCT/DST type-I rectangular update derivations

In this section *r*-point independent rectangular windowed update equations for DCT and DST type-I are derived. As developed in [7] the DCT and DST of type-I are defined as follows:

$$C(k) = \sqrt{\frac{2}{N}} P_k \sum_{x=0}^{N} P_x f(x) \cos\frac{xk\pi}{N} \qquad \text{for } k = 0,1,...,N \qquad (2.5)$$

and

$$S(k) = \sqrt{\frac{2}{N}} \sum_{x=1}^{N-1} f(x) \sin\frac{xk\pi}{N} \qquad \text{for } k = 1,2,...,N-1 \qquad (2.6)$$

where, $P_j = \frac{1}{\sqrt{2}}$ for $j = 0$ or N, and 1 otherwise. Xi and Chicharo [30] modified the

notation to include the sequence pointer *n*, equations (2.7) and (2.8), to keep track of shifting sequences in time domain.

$$C(n,k) = \sqrt{\frac{2}{N}} P_k \sum_{x=0}^{N} P_x f(n-N+x) \cos\frac{xk\pi}{N} \qquad (2.7)$$

$$\text{for } k = 0,1,...,N$$

and

$$S(n,k) = \sqrt{\frac{2}{N}} \sum_{x=1}^{N-1} f(n-N+x) \sin\frac{xk\pi}{N} \qquad (2.8)$$

$$\text{for } k = 1,2,...,N-1.$$

2.2.1 Derivation of the *r*-point equations for the DCT and DST

In this subsection we modify the derivation originally given in [22] for the *r*-point update of DCT and DST to include the sequence pointer of equations (2.7) and (2.8). The resulting formulae depend on both DCT and DST coefficients, in Section 2.2.2 we will derive independent DCT-only and DST-only formulae.

Let $C(n+r,k)$ for k=0,…..,N, represent the updated DCT coefficients including the effect of *r*-points of new input data. From equation (2.7):

$$\sqrt{\frac{N}{2}} C(n+r,k) = P_k \sum_{x=1}^{N-1} f(n-N+x+r) \cos\frac{xk\pi}{N}$$

$$+ P_k \frac{1}{\sqrt{2}} f(n-N+r) + (-1)^k P_k \frac{1}{\sqrt{2}} f(n+r)$$

Solving term 1 we get:

$$= P_k \sum_{x=1}^{N-1-r} f(n-N+x+r) \cos\frac{xk\pi}{N} + P_k \sum_{x=N-r}^{N-1} f(n-N+x+r) \cos\frac{xk\pi}{N}$$

Substituting y=x+r in the first term and i=x+r-N in the second term we get:

$$= P_k \sum_{y=r+1}^{N-1} f(n-N+y) \cos\frac{(y-r)k\pi}{N} + P_k \sum_{i=0}^{r-1} f(n+i) \cos\frac{(i-r+N)k\pi}{N}$$

Therefore,

$$\sqrt{\frac{N}{2}}C(n+r,k) = P_k\sum_{y=0}^{N} f(n-N+y)\cos\frac{(y-r)k\pi}{N} - P_k\sum_{y=0}^{r} f(n-N+y)\cos\frac{(y-r)k\pi}{N}$$
$$- P_k f(n)(-1)^k\cos\frac{rk\pi}{N} + P_k\sum_{i=0}^{r-1} f(n+i)\cos\left[\frac{(i-r)k\pi}{N} + k\pi\right]$$

$$= P_k\sum_{y=0}^{N} f(n-N+y)\cos\frac{yk\pi}{N}\cos\frac{rk\pi}{N} + P_k\sum_{y=0}^{N} f(n-N+y)\sin\frac{yk\pi}{N}\sin\frac{rk\pi}{N}$$

$$- P_k\sum_{y=0}^{r-1} f(n-N+y)\cos\frac{(y-r)k\pi}{N} - P_k f(n-N+r)$$

$$- P_k f(n)(-1)^k\cos\frac{rk\pi}{N} + (-1)^k P_k\sum_{y=0}^{r-1} f(n+y)\cos\frac{(y-r)k\pi}{N}$$

$$+ P_k\frac{1}{\sqrt{2}} f(n-N+r) + (-1)^k P_k\frac{1}{\sqrt{2}} f(n+r)$$

Simplifying the first term we get:

$$= P_k\sum_{y=0}^{N} f(n-N+y)\cos\frac{yk\pi}{N}\cos\frac{rk\pi}{N}$$

$$= P_k\sum_{y=1}^{N-1} P_y f(n-N+y)\cos\frac{yk\pi}{N}\cos\frac{rk\pi}{N} + P_k f(n-N)\cos\frac{rk\pi}{N}$$

$$+ P_k f(n)(-1)^k\cos\frac{rk\pi}{N}$$

Therefore,

$$= P_k\sum_{y=0}^{N} P_y f(n-N+y)\cos\frac{yk\pi}{N}\cos\frac{rk\pi}{N} + P_k f(n-N)\cos\frac{rk\pi}{N} + P_k f(n)(-1)^k\cos\frac{rk\pi}{N}$$

$$- P_k\frac{1}{\sqrt{2}} f(n-N)\cos\frac{rk\pi}{N} - (-1)^k P_k\frac{1}{\sqrt{2}} f(n)\cos\frac{rk\pi}{N}$$

Using the definition of the DCT and the DST as defined in equations (2.7) and (2.8) and substituting y=*r*-1-*x* results in:

$$C(n+r,k)=\cos\frac{rk\pi}{N}C(n,k)+P_k\sin\frac{rk\pi}{N}S(n,k)$$
$$+\sqrt{\frac{2}{N}}P_k\left[\left(1-\frac{1}{\sqrt{2}}\right)f(n-N)\cos\frac{rk\pi}{N}-(-1)^k\frac{1}{\sqrt{2}}f(n)\cos\frac{rk\pi}{N}\right.$$
$$+\left(\frac{1}{\sqrt{2}}-1\right)f(n-N+r)+\frac{1}{\sqrt{2}}f(n+r)(-1)^k$$
$$\left.+\sum_{x=0}^{r-1}[(-1)^k f(n+r-1-x)-f(n-N+r-1-x)]\cos\frac{(x+1)k\pi}{N}\right]. \qquad (2.9)$$

Equation (2.9) is the *r*-point equation for DCT type-I but it involves DST coefficient too. Let $S(n+r,k)$ for k=1,…..,N-1, represent the updated DST coefficients including the effect of *r*-points of new input data. From equation (2.8):

$$\sqrt{\frac{N}{2}}S(n+r,k)=\sum_{x=1}^{N-1}f(n-N+x+r)\sin\frac{xk\pi}{N}$$

$$\sqrt{\frac{N}{2}}S(n+r,k)=\sum_{x=1}^{N-1-r}f(n-N+x+r)\sin\frac{xk\pi}{N}+\sum_{x=N-r}^{N-1}f(n-N+x+r)\sin\frac{xk\pi}{N}$$

Substituting $y=x+r$ in the first term and $i=x+r-N$ in the second term we get:

$$=\sum_{y=r+1}^{N-1}f(n-N+y)\sin\frac{(y-r)k\pi}{N}+\sum_{i=0}^{r-1}f(n+i)\sin\frac{(i-r+N)k\pi}{N}$$

$$=\sum_{y=1}^{N-1}f(n-N+y)\sin\frac{(y-r)k\pi}{N}-\sum_{y=1}^{r}f(n-N+y)\sin\frac{(y-r)k\pi}{N}$$
$$+\sum_{i=0}^{r-1}f(n+i)\sin\frac{(i-r+N)k\pi}{N}$$
$$=\cos\frac{rk\pi}{N}\sum_{y=1}^{N-1}f(n-N+y)\sin\frac{yk\pi}{N}-\sin\frac{rk\pi}{N}\sum_{y=1}^{N-1}f(n-N+y)\cos\frac{yk\pi}{N}$$
$$-\sum_{y=1}^{r}f(n-N+y)\sin\frac{(y-r)k\pi}{N}+(-1)^k\sum_{y=0}^{r-1}f(n+y)\sin\frac{(y-r)k\pi}{N}$$

Therefore,

$$= \cos\frac{rk\pi}{N}\sum_{y=0}^{N} f(n-N+y)\sin\frac{yk\pi}{N} - \sin\frac{rk\pi}{N}\sum_{y=0}^{N} P_y f(n-N+y)\cos\frac{yk\pi}{N}$$

$$+\frac{1}{\sqrt{2}} f(n-N)\sin\frac{rk\pi}{N} + (-1)^k \frac{1}{\sqrt{2}} f(n)\sin\frac{rk\pi}{N} - f(n-N)\sin\frac{rk\pi}{N}$$

$$-\sum_{y=0}^{r-1} f(n-N+y)\sin\frac{(y-r)k\pi}{N} + (-1)^k \sum_{y=0}^{r-1} f(n+y)\sin\frac{(y-r)k\pi}{N}$$

Using the definition of the DCT and the DST as defined in equations (2.7) and (2.8):

$$S(n+r,k) = \cos\frac{rk\pi}{N} S(n,k) - \sin\frac{rk\pi}{N} C(n,k)$$

$$+\sqrt{\frac{2}{N}}\sin\frac{rk\pi}{N}\left[\left(\frac{1}{\sqrt{2}}-1\right)f(n-N) + \frac{1}{\sqrt{2}} f(n)(-1)^k\right]$$

$$+\sqrt{\frac{2}{N}}\sum_{y=0}^{r-1}[(-1)^k f(n+y) - f(n-N+y)]\sin\frac{(y-r)k\pi}{N}$$

Substituting $y=r-1-x$ results in:

$$S(n+r,k) = \cos\frac{rk\pi}{N} S(n,k) - \sin\frac{rk\pi}{N} C(n,k)$$

$$+\sqrt{\frac{2}{N}}\sin\frac{rk\pi}{N}\left[\left(\frac{1}{\sqrt{2}}-1\right)f(n-N) + \frac{1}{\sqrt{2}} f(n)(-1)^k\right]$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}[f(n-N+r-1-x) - (-1)^k f(n+r-1-x)]\sin\frac{(x+1)k\pi}{N} \qquad (2.10)$$

Equation (2.10) is the *r*-point update equation for DST-I.

2.2.2 Computation of *r*-point independent update equations for DCT and DST

Equation (2.9) provides an *r*-point update for the DCT but it includes both $C(n,k)$ and $S(n,k)$ coefficients. Therefore it would be necessary to update both the DCT and the

DST even if only the DCT coefficients are required. We therefore seek an equation that provides an update of the DCT in terms of DCT coefficients only.

Taking the two sided *z*-transform of equation (2.9) yields:

$$z^r C(z,k) = A_r C(z,k) + P_k B_r S(z,k)$$

$$+\left[\sqrt{\frac{2}{N}}P_k\left(1-\frac{1}{\sqrt{2}}\right)z^{-N}A_r - (-1)^k\sqrt{\frac{2}{N}}P_k\frac{1}{\sqrt{2}}A_r + \sqrt{\frac{2}{N}}P_k\left(\frac{1}{\sqrt{2}}-1\right)z^{-N+r} + (-1)^k\sqrt{\frac{2}{N}}P_k\frac{1}{\sqrt{2}}z^r\right.$$

$$\left.+\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{r-1}[(-1)^k z^{r-1-x} - z^{-N+r-1-x}]\cos\frac{(x+1)k\pi}{N}\right]F(z).$$

where, $A_r = \cos\frac{rk\pi}{N}$, $B_r = \sin\frac{rk\pi}{N}$ and $F(z)$ represents the z-transform of the input sequence $f(n)$.

Therefore,

$$(z^r - A_r)C(z,k) = P_k B_r S(z,k)$$

$$+\left[\sqrt{\frac{2}{N}}P_k\left(1-\frac{1}{\sqrt{2}}\right)z^{-N}A_r - (-1)^k\sqrt{\frac{2}{N}}P_k\frac{1}{\sqrt{2}}A_r + \sqrt{\frac{2}{N}}P_k\left(\frac{1}{\sqrt{2}}-1\right)z^{-N+r} + (-1)^k\sqrt{\frac{2}{N}}P_k\frac{1}{\sqrt{2}}z^r\right.$$

$$\left.+\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{r-1}[(-1)^k z^{r-1-x} - z^{-N+r-1-x}]\cos\frac{(x+1)k\pi}{N}\right]F(z)$$

Solving for $C(z,k)$ yields:

$$C(z,k) = \frac{P_k B_r}{(z^r - A_r)}S(z,k) + \frac{1}{(z^r - A_r)}\left[\sqrt{\frac{2}{N}}P_k\left(1-\frac{1}{\sqrt{2}}\right)z^{-N}A_r\right.$$

$$-(-1)^k\sqrt{\frac{2}{N}}P_k\frac{1}{\sqrt{2}}A_r + \sqrt{\frac{2}{N}}P_k\left(\frac{1}{\sqrt{2}}-1\right)z^{-N+r} + (-1)^k\sqrt{\frac{2}{N}}P_k\frac{1}{\sqrt{2}}z^r$$

$$\left.+\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{r-1}[(-1)^k z^{r-1-x} - z^{-N+r-1-x}]\cos\frac{(x+1)k\pi}{N}\right]F(z)$$

$$C(z,k)=\frac{P_k B_r z^{-r}}{(1-z^{-r}A_r)}S(z,k)+\frac{1}{(1-z^{-r}A_r)}\left[\sqrt{\frac{2}{N}}P_k\left(1-\frac{1}{\sqrt{2}}\right)z^{-N-r}A_r\right.$$

$$-(-1)^k\sqrt{\frac{2}{N}}P_k\frac{1}{\sqrt{2}}A_r z^{-r}+\sqrt{\frac{2}{N}}P_k\left(\frac{1}{\sqrt{2}}-1\right)z^{-N}+(-1)^k\sqrt{\frac{2}{N}}P_k\frac{1}{\sqrt{2}}$$

$$\left.+\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{r-1}[(-1)^k z^{-1-x}-z^{-N-1-x}]\cos\frac{(x+1)k\pi}{N}\right]F(z) \qquad (2.11)$$

Taking z-transform of equation (2.10) yields:

$$z^r S(z,k)=A_r S(z,k)-B_r C(z,k)+\left[\sqrt{\frac{2}{N}}\left(\frac{1}{\sqrt{2}}-1\right)B_r z^{-N}+\sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}B_r(-1)^k\right.$$

$$\left.+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}[z^{-N+r-1-x}-(-1)^k z^{r-1-x}]\sin\frac{(x+1)k\pi}{N}\right]F(z)$$

$$(z^r-A_r)S(z,k)=-B_r C(z,k)+\left[\sqrt{\frac{2}{N}}\left(\frac{1}{\sqrt{2}}-1\right)B_r z^{-N}+\sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}B_r(-1)^k\right.$$

$$\left.+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}[z^{-N+r-1-x}-(-1)^k z^{r-1-x}]\sin\frac{(x+1)k\pi}{N}\right]F(z)$$

Solving for $S(z,k)$:

$$S(z,k)=-\frac{B_r}{(z^r-A_r)}C(z,k)+\frac{1}{(z^r-A_r)}\left[\sqrt{\frac{2}{N}}\left(\frac{1}{\sqrt{2}}-1\right)B_r z^{-N}+\sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}B_r(-1)^k\right.$$

$$\left.+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}[z^{-N+r-1-x}-(-1)^k z^{r-1-x}]\sin\frac{(x+1)k\pi}{N}\right]F(z)$$

Therefore,

$$S(z,k)=-\frac{B_r z^{-r}}{(1-A_r z^{-r})}C(z,k)+\frac{1}{(1-A_r z^{-r})}\left[\sqrt{\frac{2}{N}}\left(\frac{1}{\sqrt{2}}-1\right)B_r z^{-N-r}+\sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}B_r(-1)^k z^{-r}\right.$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}[z^{-N-1-x}-(-1)^k z^{-1-x}]\sin\frac{(x+1)k\pi}{N}\Bigg]F(z) \qquad (2.12)$$

Eliminating $S(z,k)$ by substituting equation (2.12) in equation (2.11) yields:

$$C(z,k)=-\frac{P_k {B_r}^2 z^{-2r}}{(1-z^{-r}A_r)^2}C(z,k)+\frac{P_k B_r z^{-r}}{(1-z^{-r}A_r)^2}\Bigg[\sqrt{\frac{2}{N}}\left(\frac{1}{\sqrt{2}}-1\right)B_r z^{-N-r}$$

$$+(-1)^k\sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}B_r z^{-r}+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}[z^{-N-1-x}-(-1)^k z^{-1-x}]\sin\frac{(x+1)k\pi}{N}\Bigg]F(z)$$

$$+\frac{1}{(1-A_r z^{-r})}\Bigg[\sqrt{\frac{2}{N}}\left(1-\frac{1}{\sqrt{2}}\right)P_k A_r z^{-N-r}-(-1)^k\sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}P_k A_r z^{-r}$$

$$+\sqrt{\frac{2}{N}}\left(\frac{1}{\sqrt{2}}-1\right)P_k z^{-N}+(-1)^k\sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}P_k$$

$$+\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{r-1}[(-1)^k z^{-1-x}-z^{-N-1-x}]\cos\frac{(x+1)k\pi}{N}\Bigg]F(z)$$

Gathering terms in $C(z,k)$ onto the left hand side,

$$\left\{(1-z^{-r}A_r)^2+P_k {B_r}^2 z^{-2r}\right\}C(z,k)=$$

$$\sqrt{\frac{2}{N}}\left(\frac{1}{\sqrt{2}}-1\right)P_k {B_r}^2 z^{-N-2r}+(-1)^k\sqrt{\frac{2}{N}}P_k\frac{1}{\sqrt{2}}{B_r}^2 z^{-2r}$$

$$+\sqrt{\frac{2}{N}}P_k B_r\sum_{x=0}^{r-1}[z^{-N-1-x-r}-(-1)^k z^{-1-x-r}]\sin\frac{(x+1)k\pi}{N}$$

$$+\sqrt{\frac{2}{N}}\left(1-\frac{1}{\sqrt{2}}\right)P_k A_r z^{-N-r}-(-1)^k\sqrt{\frac{2}{N}}P_k\frac{1}{\sqrt{2}}A_r z^{-r}$$

$$+\sqrt{\frac{2}{N}}\left(\frac{1}{\sqrt{2}}-1\right)P_k z^{-N}+(-1)^k\sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}P_k$$

$$+\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{r-1}[(-1)^k z^{-1-x}-z^{-N-1-x}]\cos\frac{(x+1)k\pi}{N}$$

$$+\sqrt{\frac{2}{N}}\left(\frac{1}{\sqrt{2}}-1\right)P_k {A_r}^2 z^{-N-2r}+(-1)^k\sqrt{\frac{2}{N}}P_k\frac{1}{\sqrt{2}}{A_r}^2 z^{-2r}$$

$$+\sqrt{\frac{2}{N}}\left(1-\frac{1}{\sqrt{2}}\right)P_k A_r z^{-N-r}-(-1)^k\sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}P_k A_r z^{-r}$$

$$+\sqrt{\frac{2}{N}}P_k A_r\sum_{x=0}^{r-1}[z^{-N-1-x-r}-(-1)^k z^{-1-x-r}]\cos\frac{(x+1)k\pi}{N}$$

Multiplying throughout by z^r and then taking the inverse z-transform results in:

$$C(n+1,k)+({A_r}^2+P_k{B_r}^2)C(n-2r+1,k)-2A_r C(n-r+1,k)=$$

$$\sqrt{\frac{2}{N}}\left(\frac{1}{\sqrt{2}}-1\right)P_k{B_r}^2 f(n-N-2r+1)+(-1)^k\sqrt{\frac{2}{N}}P_k\frac{1}{\sqrt{2}}{B_r}^2 f(n-2r+1)$$

$$+\sqrt{\frac{2}{N}}P_k B_r\sum_{x=0}^{r-1}[f(n-N-x-r)-(-1)^k f(n-x-r)]\sin\frac{(x+1)k\pi}{N}$$

$$+\sqrt{\frac{2}{N}}\left(1-\frac{1}{\sqrt{2}}\right)P_k A_r f(n-N-r+1)-(-1)^k\sqrt{\frac{2}{N}}P_k\frac{1}{\sqrt{2}}A_r f(n-r+1)$$

$$+\sqrt{\frac{2}{N}}\left(\frac{1}{\sqrt{2}}-1\right)P_k f(n-N+1)+(-1)^k\sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}P_k f(n+1)$$

$$+\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{r-1}[(-1)^k f(n-x)-f(n-N-x)]\cos\frac{(x+1)k\pi}{N}$$

$$+\sqrt{\frac{2}{N}}\left(\frac{1}{\sqrt{2}}-1\right)P_k{A_r}^2 f(n-N-2r+1)+(-1)^k\sqrt{\frac{2}{N}}P_k\frac{1}{\sqrt{2}}{A_r}^2 f(n-2r+1)$$

$$+\sqrt{\frac{2}{N}}\left(1-\frac{1}{\sqrt{2}}\right)P_k A_r f(n-N-r+1)-(-1)^k\sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}P_k A_r f(n-r+1)$$

$$+\sqrt{\frac{2}{N}}P_k A_r \sum_{x=0}^{r-1}[f(n-N-x-r)-(-1)^k f(n-x-r)]\cos\frac{(x+1)k\pi}{N}$$

Solving further yields:

$$C(n+r,k) = 2A_r C(n,k) - (A_r{}^2 + P_k B_r{}^2)C(n-r,k)$$

$$+\sqrt{\frac{2}{N}}P_k B_r \sum_{x=0}^{r-1}[f(n-N-x-1)-(-1)^k f(n-x-1)]\sin\frac{(x+1)k\pi}{N}$$

$$+P_k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}[(-1)^k f(n+r-1-x)-f(n-N+r-1-x)+A_r f(n-N-x-1)$$

$$-A_r(-1)^k f(n-x-1)]\cos\frac{(x+1)k\pi}{N}$$

$$+\sqrt{\frac{2}{N}}\left(\frac{1}{\sqrt{2}}-1\right)P_k f(n-N-r)+(-1)^k\sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}P_k f(n-r)$$

$$+2\sqrt{\frac{2}{N}}\left(1-\frac{1}{\sqrt{2}}\right)P_k A_r f(n-N)-(-1)^k\sqrt{\frac{2}{N}}\sqrt{2}P_k A_r f(n)$$

$$+\sqrt{\frac{2}{N}}\left(\frac{1}{\sqrt{2}}-1\right)P_k f(n-N+r)+(-1)^k\sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}P_k f(n+r) \qquad (2.13)$$

$$\text{for } k = 0,1,...,N.$$

Equation (2.13) is the *r*-point update equation for DCT-I. It can be used to update the DCT using the DCT coefficients of the current and one previous time sequences, new input data points and the old data points shifted out, independent of the discrete sine coefficients. The update equation involves real time update of *N* point data taking *r*-points at a time. The C language implementation of the equation to implement *r*-point update where $1 \le r \le N$-1 is included in figure (A.1) of Appendix A. The code accepts the new data points as they become available and uses equation (2.13) to calculate the

updated DCT. The values obtained from the update algorithm are compared with the values obtained through the conventional formula and RMS error values are calculated.

To find the *r*-point independent update equation for DST-I we substitute equation (2.11) in equation (2.12):

$$S(z,k) = -\frac{P_k {B_r}^2 z^{-2r}}{(1-z^{-r}A_r)^2} S(z,k) - \frac{B_r}{(1-z^{-r}A_r)^2}\left[\sqrt{\frac{2}{N}}P_k\left(1-\frac{1}{\sqrt{2}}\right)z^{-N-2r}A_r\right.$$

$$-(-1)^k\sqrt{\frac{2}{N}}P_k\frac{1}{\sqrt{2}}A_r z^{-2r} + \sqrt{\frac{2}{N}}P_k\left(\frac{1}{\sqrt{2}}-1\right)z^{-N-r} + (-1)^k\sqrt{\frac{2}{N}}P_k\frac{1}{\sqrt{2}}z^{-r}$$

$$\left.+\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{r-1}[(-1)^k z^{-1-x-r} - z^{-N-1-x-r}]\cos\frac{(x+1)k\pi}{N}\right]F(z)$$

$$+\frac{1}{(1-A_r z^{-r})}\left[\sqrt{\frac{2}{N}}\left(\frac{1}{\sqrt{2}}-1\right)B_r z^{-N-r} + \sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}B_r(-1)^k z^{-r}\right.$$

$$\left.+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}[z^{-N-1-x} - (-1)^k z^{-1-x}]\sin\frac{(x+1)k\pi}{N}\right]F(z)$$

Gathering terms in $S(z,k)$ onto the left hand side,

$$\left\{(1-z^{-r}A_r)^2 + P_k {B_r}^2 z^{-2r}\right\}S(z,k) =$$

$$-\sqrt{\frac{2}{N}}\left(1-\frac{1}{\sqrt{2}}\right)P_k A_r B_r z^{-N-2r} + (-1)^k\sqrt{\frac{2}{N}}P_k\frac{1}{\sqrt{2}}A_r B_r z^{-2r}$$

$$+\sqrt{\frac{2}{N}}\left(1-\frac{1}{\sqrt{2}}\right)P_k B_r z^{-N-r} - (-1)^k\sqrt{\frac{2}{N}}P_k\frac{1}{\sqrt{2}}B_r z^{-r}$$

$$+\sqrt{\frac{2}{N}}P_k B_r\sum_{x=0}^{r-1}[z^{-N-1-x-r} - (-1)^k z^{-1-x-r}]\cos\frac{(x+1)k\pi}{N}$$

$$+\sqrt{\frac{2}{N}}\left(\frac{1}{\sqrt{2}}-1\right)B_r z^{-N-r} + (-1)^k\sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}B_r z^{-r}$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}[z^{-N-1-x}-(-1)^k z^{-1-x}]\sin\frac{(x+1)k\pi}{N}$$

$$+\sqrt{\frac{2}{N}}\left(1-\frac{1}{\sqrt{2}}\right)A_r B_r z^{-N-2r}-(-1)^k\sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}A_r B_r z^{-2r}$$

$$+\sqrt{\frac{2}{N}}A_r\sum_{x=0}^{r-1}[(-1)^k z^{-1-x-r}-z^{-N-1-x-r}]\sin\frac{(x+1)k\pi}{N}$$

Multiplying throughout by z^r and then taking the inverse z-transform yields:

$$S(n+1,k)+(A_r{}^2+P_k B_r{}^2)S(n-2r+1,k)-2A_k S(n-r+1,k)=$$

$$-\sqrt{\frac{2}{N}}\left(1-\frac{1}{\sqrt{2}}\right)P_k A_r B_r f(n-N-2r+1)+(-1)^k\sqrt{\frac{2}{N}}P_k\frac{1}{\sqrt{2}}A_r B_r f(n-2r+1)$$

$$+\sqrt{\frac{2}{N}}\left(1-\frac{1}{\sqrt{2}}\right)P_k B_r f(n-N-r+1)-(-1)^k\sqrt{\frac{2}{N}}P_k\frac{1}{\sqrt{2}}B_r f(n-r+1)$$

$$+\sqrt{\frac{2}{N}}P_k B_r\sum_{x=0}^{r-1}[f(n-N-x-r)-(-1)^k f(n-x-r)]\cos\frac{(x+1)k\pi}{N}$$

$$+\sqrt{\frac{2}{N}}\left(\frac{1}{\sqrt{2}}-1\right)B_r f(n-N-r+1)+(-1)^k\sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}B_r f(n-r+1)$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}[f(n-N-x)-(-1)^k f(n-x)]\sin\frac{(x+1)k\pi}{N}$$

$$+\sqrt{\frac{2}{N}}\left(1-\frac{1}{\sqrt{2}}\right)A_r B_r f(n-N-2r+1)-(-1)^k\sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}A_r B_r f(n-2r+1)$$

$$+\sqrt{\frac{2}{N}}A_r\sum_{x=0}^{r-1}[(-1)^k f(n-x-r)-f(n-N-x-r)]\sin\frac{(x+1)k\pi}{N}$$

Therefore,

$$S(n+r,k)=2A_k S(n,k)-(A_r{}^2+P_k B_r{}^2)S(n-r,k)$$

$$+\sqrt{\frac{2}{N}}P_k B_r \sum_{x=0}^{r-1}[f(n-1-N-x)-(-1)^k f(n-1-x)]\cos\frac{(x+1)k\pi}{N}$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}[f(n+r-1-N-x)-(-1)^k f(n+r-1-x)+A_r(-1)^k f(n-1-x)$$

$$-A_r f(n-1-N-x)]\sin\frac{(x+1)k\pi}{N}$$

$$+\sqrt{\frac{2}{N}}\left(1-\frac{1}{\sqrt{2}}\right)A_r B_r(1-P_k)f(n-N-r)+(-1)^k\sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}A_r B_r(P_k-1)f(n-r)$$

$$+\sqrt{\frac{2}{N}}\left(1-\frac{1}{\sqrt{2}}\right)(P_k-1)B_r f(n-N)-(-1)^k\sqrt{\frac{2}{N}}(P_k-1)\frac{1}{\sqrt{2}}B_r f(n) \qquad (2.14)$$

$$\text{for } k=1,...,N-1.$$

The C language implementation of equation (2.14) is included in figure (A.3) of Appendix A.

2.3 DCT/DST type-II rectangular update derivations

As presented in reference [7] the DCT and DST of type-II are defined as follows:

$$C(k)=\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{N-1}f(x)\cos\frac{(2x+1)k\pi}{2N} \qquad (2.15)$$

$$\text{for } k=0,1,...,N-1,$$

and

$$S(k)=\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{N-1}f(x)\sin\frac{(2x+1)k\pi}{2N} \qquad (2.16)$$

$$\text{for } k=1,...,N,$$

where, $P_j=\frac{1}{\sqrt{2}}$ for $j=0$ or N, and 1 otherwise. Equation (2.16) uses the argument $(2x+1)$ instead of $(2x-1)$ in the conventional definition of DST to represent the same

set of signal data points. Both the definitions are fully identical and result in numerically identical transform coefficients.

Xi and Chicharo [30] modified the notation to include the sequence pointer *n*, to keep track of points in the presence of shifting data,

$$C(n,k) = \sqrt{\frac{2}{N}} P_k \sum_{x=0}^{N-1} f(n-N+x) \cos\frac{(2x+1)k\pi}{2N} \qquad (2.17)$$

$$\text{for } k = 0,1,\ldots,N-1,$$

and

$$S(n,k) = \sqrt{\frac{2}{N}} P_k \sum_{x=0}^{N-1} f(n-N+x) \sin\frac{(2x+1)k\pi}{2N} \qquad (2.18)$$

$$\text{for } k = 1,\ldots,N.$$

2.3.1 Derivation of the *r*-point equations

In this subsection we modify the derivation originally given in [22] for the *r*-point update of DCT and DST to include the sequence pointer of equations (2.17) and (2.18). The resulting formulae depend on both DCT and DST coefficients, in Section 2.3.2 we will derive independent DCT-only and DST-only formulae.

Let $C(n+r,k)$ for k=0,1,…,N-1, represent the updated DCT coefficients including the effect of *r*-points of new input data. From equation (2.17):

$$\sqrt{\frac{N}{2}} C(n+r,k) = P_k \sum_{x=0}^{N-1-r} f(n-N+x+r) \cos\frac{(2x+1)k\pi}{2N}$$

$$+ P_k \sum_{x=N-r}^{N-1} f(n-N+x+r) \cos\frac{(2x+1)k\pi}{2N}.$$

Substituting $y=x+r$ in the first term and $i=x+r-N$ in the second term results in:

$$\sqrt{\frac{N}{2}} C(n+r,k) = P_k \sum_{y=r}^{N-1} f(n-N+y) \cos\frac{(2(y-r)+1)k\pi}{2N}$$

$$+ P_k \sum_{i=0}^{r-1} f(n+i) \cos \frac{(2(N-r+i)+1)k\pi}{2N}.$$

$$= P_k \sum_{y=0}^{N-1} f(n-N+y) \cos\left[\frac{(2y+1)k\pi}{2N} - \frac{rk\pi}{N}\right]$$

$$- P_k \sum_{y=0}^{r-1} f(n-N+y) \cos \frac{(2(y-r)+1)k\pi}{2N}$$

$$+ P_k \sum_{i=0}^{r-1} f(n+i) \cos\left[\frac{(2(i-r)+1)k\pi}{2N} + k\pi\right]$$

$$= P_k \sum_{y=0}^{N-1} f(n-N+y) \cos \frac{(2y+1)k\pi}{2N} \cos \frac{rk\pi}{N}$$

$$+ P_k \sum_{y=0}^{N-1} f(n-N+y) \sin \frac{(2y+1)k\pi}{2N} \sin \frac{rk\pi}{N}$$

$$- P_k \sum_{y=0}^{r-1} f(n-N+y) \cos \frac{(2(y-r)+1)k\pi}{2N}$$

$$+ (-1)^k P_k \sum_{y=0}^{r-1} f(n+y) \cos \frac{(2(y-r)+1)k\pi}{2N}.$$

Using the definition of the DCT and the DST as defined in equations (2.17) and (2.18):

$$C(n+r,k) = \cos \frac{rk\pi}{N} C(n,k) + \sin \frac{rk\pi}{N} S(n,k)$$

$$+ \sqrt{\frac{2}{N}} P_k \sum_{y=0}^{r-1} [(-1)^k f(n+y) - f(n-N+y)] \cos \frac{(2(y-r)+1)k\pi}{2N},$$

Substituting x=r-1-y results in:

$$C(n+r,k) = \cos \frac{rk\pi}{N} C(n,k) + \sin \frac{rk\pi}{N} S(n,k)$$

$$+ \sqrt{\frac{2}{N}} P_k \sum_{x=0}^{r-1} [(-1)^k f(n+r-1-x) - f(n-N+r-1-x)] \cos \frac{(2x+1)k\pi}{2N}. \qquad (2.19)$$

Let $S(n+r,k)$ for k=1,…,N, represent the updated DST coefficients including the effect of r-points of new input data. From equation (2.18):

$$\sqrt{\frac{N}{2}}S(n+r,k) = P_k \sum_{x=0}^{N-1-r} f(n-N+x+r)\sin\frac{(2x+1)k\pi}{2N} + P_k \sum_{x=N-r}^{N-1} f(n-N+x+r)\sin\frac{(2x+1)k\pi}{2N},$$

Substituting $y=x+r$ in the first term and $i=x+r-N$ in the second term results in:

$$\begin{aligned}\sqrt{\frac{N}{2}}S(n+r,k) &= P_k\sum_{y=r}^{N-1} f(n-N+y)\sin\frac{(2(y-r)+1)k\pi}{2N} + P_k\sum_{i=0}^{r-1} f(n+i)\sin\frac{(2(N-r+i)+1)k\pi}{2N},\\ &= P_k\sum_{y=0}^{N-1} f(n-N+y)\sin\left[\frac{(2y+1)k\pi}{2N}-\frac{rk\pi}{N}\right] - P_k\sum_{y=0}^{r-1} f(n-N+y)\sin\frac{(2(y-r)+1)k\pi}{2N} + P_k\sum_{i=0}^{r-1} f(n+i)\sin\left[\frac{(2(i-r)+1)k\pi}{2N}+k\pi\right]\\ &= P_k\sum_{y=0}^{N-1} f(n-N+y)\sin\frac{(2y+1)k\pi}{2N}\cos\frac{rk\pi}{N} - P_k\sum_{y=0}^{N-1} f(n-N+y)\cos\frac{(2y+1)k\pi}{2N}\sin\frac{rk\pi}{N} - P_k\sum_{y=0}^{r-1} f(n-N+y)\sin\frac{(2(y-r)+1)k\pi}{2N} + (-1)^k P_k\sum_{y=0}^{r-1} f(n+y)\sin\frac{(2(y-r)+1)k\pi}{2N}.\end{aligned}$$

Using the definition of the DCT and the DST as defined in equations (2.17) and (2.18) yields:

$$S(n+r,k) = \cos\frac{rk\pi}{N} S(n,k) - \sin\frac{rk\pi}{N} C(n,k)$$

$$+ \sqrt{\frac{2}{N}} P_k \sum_{y=0}^{r-1} [f(n-N+y) - (-1)^k f(n+y)] \sin\frac{(2(y-r)+1)k\pi}{2N},$$

Substituting x=r-1-y results in:

$$S(n+r,k) = \cos\frac{rk\pi}{N} S(n,k) - \sin\frac{rk\pi}{N} C(n,k)$$

$$+ \sqrt{\frac{2}{N}} P_k \sum_{x=0}^{r-1} [f(n-N+r-1-x) - (-1)^k f(n+r-1-x)] \sin\frac{(2x+1)k\pi}{2N}. \qquad (2.20)$$

Equation (2.20) is the r-point equation for the DST type-II.

2.3.2 Computation of r-point independent update equations

Equation (2.19) provides an r-point update for the DCT but it includes both $C(n,k)$ and $S(n,k)$ coefficients. Therefore it would be necessary to update both the DCT and the DST even if only the DCT coefficients are required. We however need an independent update of DCT i.e. an equation that provides an update of the DCT in terms of DCT coefficients only.

Taking the z-transform of equation (2.19) yields:

$$z^r C(z,k) = \cos\frac{rk\pi}{N} C(z,k) + \sin\frac{rk\pi}{N} S(z,k)$$

$$+ \sqrt{\frac{2}{N}} P_k \sum_{x=0}^{r-1} [(-1)^k z^{r-1-x} - z^{-N+r-1-x}] F(z) \cos\frac{(2x+1)k\pi}{2N}, \qquad (2.21)$$

where, $F(z)$ represents the z-transform of the input sequence $f(n)$.

Taking the z-transform of equation (2.20) yields:

$$z^r S(z,k) = \cos\frac{rk\pi}{N} S(z,k) - \sin\frac{rk\pi}{N} C(z,k)$$

$$+ \sqrt{\frac{2}{N}} P_k \sum_{x=0}^{r-1} [z^{-N+r-1-x} - (-1)^k z^{r-1-x}] F(z) \sin\frac{(2x+1)k\pi}{2N}. \qquad (2.22)$$

Solving equation (2.21) for $C(z,k)$ yields:

$$C(z,k) = \frac{\sin\frac{rk\pi}{N}}{z^r - \cos\frac{rk\pi}{N}} S(z,k)$$

$$+ \frac{\sqrt{\frac{2}{N}} P_k \sum_{x=0}^{r-1} [(-1)^k z^{r-1-x} - z^{-N+r-1-x}] \cos\frac{(2x+1)k\pi}{2N}}{z^r - \cos\frac{rk\pi}{N}} F(z).$$

Therefore,

$$C(z,k) = \frac{z^{-r} \sin\frac{rk\pi}{N}}{1 - z^{-r} \cos\frac{rk\pi}{N}} S(z,k)$$

$$+ \frac{\sqrt{\frac{2}{N}} P_k \sum_{x=0}^{r-1} [(-1)^k z^{-1-x} - z^{-N-1-x}] \cos\frac{(2x+1)k\pi}{2N}}{1 - z^{-r} \cos\frac{rk\pi}{N}} F(z). \qquad (2.23)$$

Solving equation (2.22) for $S(z,k)$ yields:

$$S(z,k) = -\frac{\sin\frac{rk\pi}{N}}{z^r - \cos\frac{rk\pi}{N}} C(z,k)$$

$$+ \frac{\sqrt{\frac{2}{N}} P_k \sum_{x=0}^{r-1} [z^{-N+r-1-x} - (-1)^k z^{r-1-x}] \sin\frac{(2x+1)k\pi}{2N}}{z^r - \cos\frac{rk\pi}{N}} F(z).$$

Therefore,

$$S(z,k) = -\frac{z^{-r}\sin\frac{rk\pi}{N}}{1-z^{-r}\cos\frac{rk\pi}{N}}C(z,k)$$

$$+\frac{\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{r-1}[z^{-N-1-x}-(-1)^k z^{-1-x}]\sin\frac{(2x+1)k\pi}{2N}}{1-z^{-r}\cos\frac{rk\pi}{N}}F(z). \qquad (2.24)$$

Eliminating $S(z,k)$ by substituting equation (2.24) in equation (2.23) results in:

$$C(z,k) = -\frac{z^{-2r}\sin^2\frac{rk\pi}{N}}{\left[1-z^{-r}\cos\frac{rk\pi}{N}\right]^2}C(z,k)$$

$$+\frac{\sqrt{\frac{2}{N}}P_k z^{-r}\sin\frac{rk\pi}{N}\sum_{x=0}^{r-1}[z^{-N-1-x}-(-1)^k z^{-1-x}]\sin\frac{(2x+1)k\pi}{2N}}{\left[1-z^{-r}\cos\frac{rk\pi}{N}\right]^2}F(z)$$

$$+\frac{\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{r-1}[(-1)^k z^{-1-x}-z^{-N-1-x}]\cos\frac{(2x+1)k\pi}{2N}}{1-z^{-r}\cos\frac{rk\pi}{N}}F(z).$$

Gathering terms in $C(z,k)$ onto the left hand side,

$$\left[\left[1-z^{-r}\cos\frac{rk\pi}{N}\right]^2+z^{-2r}\sin^2\frac{rk\pi}{N}\right]C(z,k) =$$

$$\sqrt{\frac{2}{N}}P_k z^{-r}\sin\frac{rk\pi}{N}\sum_{x=0}^{r-1}[z^{-N-1-x}-(-1)^k z^{-1-x}]F(z)\sin\frac{(2x+1)k\pi}{2N}$$

$$+\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{r-1}[(-1)^k z^{-1-x}-z^{-N-1-x}]F(z)\cos\frac{(2x+1)k\pi}{2N}$$

$$-\sqrt{\frac{2}{N}}P_k z^{-r}\cos\frac{rk\pi}{N}\sum_{x=0}^{r-1}[(-1)^k z^{-1-x}-z^{-N-1-x}]F(z)\cos\frac{(2x+1)k\pi}{2N}.$$

Therefore,

$$\left[1+z^{-2r}-2z^{-r}\cos\frac{rk\pi}{N}\right]C(z,k)=$$

$$\sqrt{\frac{2}{N}}P_k\sin\frac{rk\pi}{N}\sum_{x=0}^{r-1}[z^{-r-N-1-x}-(-1)^k z^{-r-1-x}]F(z)\sin\frac{(2x+1)k\pi}{2N}$$

$$+\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{r-1}[(-1)^k z^{-1-x}-z^{-N-1-x}]F(z)\cos\frac{(2x+1)k\pi}{2N}$$

$$-\sqrt{\frac{2}{N}}P_k\cos\frac{rk\pi}{N}\sum_{x=0}^{r-1}[(-1)^k z^{-r-1-x}-z^{-r-N-1-x}]F(z)\cos\frac{(2x+1)k\pi}{2N}.$$

Multiplying throughout by z^r and then taking the inverse z-transform yields:

$$C(n+r,k)=2\cos\frac{rk\pi}{N}C(n,k)-C(n-r,k)$$

$$+\sqrt{\frac{2}{N}}P_k\sin\frac{rk\pi}{N}\sum_{x=0}^{r-1}[f(n-N-x-1)-(-1)^k f(n-x-1)]\sin\frac{(2x+1)k\pi}{2N}$$

$$+\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{r-1}[(-1)^k f(n+r-x-1)-f(n+r-N-x-1)]\cos\frac{(2x+1)k\pi}{2N}$$

$$-\sqrt{\frac{2}{N}}P_k\cos\frac{rk\pi}{N}\sum_{x=0}^{r-1}[(-1)^k f(n-x-1)-f(n-N-x-1)]\cos\frac{(2x+1)k\pi}{2N}. \quad (2.25)$$

for $k=0,1,...,N-1$.

Equation (2.25) can be used to update the DCT using DCT coefficients of the current and one previous time sequences, new input data points and the old data points shifted out, independent of the discrete sine coefficients. The update equation involves real time update of N point data taking r-points at a time. The C language implementation of

equation (2.25) to implement *r*-point update where $1 \le r \le N$-1 is listed in figure (A.5) of Appendix A.

Similarly the *r*-point update equation for the DST may be derived by substituting equation (2.23) in equation (2.24).

$$S(z,k) = -\frac{z^{-2r}\sin^2\frac{rk\pi}{N}}{\left[1-z^{-r}\cos\frac{rk\pi}{N}\right]^2}S(z,k)$$

$$+\frac{\sqrt{\frac{2}{N}}P_k z^{-r}\sin\frac{rk\pi}{N}\sum_{x=0}^{r-1}[z^{-N-1-x}-(-1)^k z^{-1-x}]\cos\frac{(2x+1)k\pi}{2N}}{\left[1-z^{-r}\cos\frac{rk\pi}{N}\right]^2}F(z)$$

$$+\frac{\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{r-1}[z^{-N-1-x}-(-1)^k z^{-1-x}]\sin\frac{(2x+1)k\pi}{2N}}{1-z^{-r}\cos\frac{rk\pi}{N}}F(z).$$

Gathering terms in $S(z,k)$ onto the left hand side,

$$\left[\left[1-z^{-r}\cos\frac{rk\pi}{N}\right]^2+z^{-2r}\sin^2\frac{rk\pi}{N}\right]S(z,k) =$$

$$\sqrt{\frac{2}{N}}P_k z^{-r}\sin\frac{rk\pi}{N}\sum_{x=0}^{r-1}[z^{-N-1-x}-(-1)^k z^{-1-x}]F(z)\cos\frac{(2x+1)k\pi}{2N}$$

$$+\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{r-1}[z^{-N-1-x}-(-1)^k z^{-1-x}]F(z)\cos\frac{(2x+1)k\pi}{2N}$$

$$-\sqrt{\frac{2}{N}}P_k z^{-r}\cos\frac{rk\pi}{N}\sum_{x=0}^{r-1}[z^{-N-1-x}-(-1)^k z^{-1-x}]F(z)\sin\frac{(2x+1)k\pi}{2N}.$$

Therefore,

$$\left[1+z^{-2r}-2z^{-r}\cos\frac{rk\pi}{N}\right]S(z,k)=$$

$$\sqrt{\frac{2}{N}}P_k\sin\frac{rk\pi}{N}\sum_{x=0}^{r-1}[z^{-r-N-1-x}-(-1)^k z^{-r-1-x}]F(z)\cos\frac{(2x+1)k\pi}{2N}$$

$$+\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{r-1}[z^{-N-1-x}-(-1)^k z^{-1-x}]F(z)\sin\frac{(2x+1)k\pi}{2N}$$

$$-\sqrt{\frac{2}{N}}P_k\cos\frac{rk\pi}{N}\sum_{x=0}^{r-1}[z^{-r-N-1-x}-(-1)^k z^{-r-1-x}]F(z)\sin\frac{(2x+1)k\pi}{2N}.$$

Multiplying throughout by z^r and then taking the inverse z-transform yields:

$$S(n+r,k)=2\cos\frac{rk\pi}{N}S(n,k)-S(n-r,k)$$

$$+\sqrt{\frac{2}{N}}P_k\sin\frac{rk\pi}{N}\sum_{x=0}^{r-1}[f(n-N-x-1)-(-1)^k f(n-x-1)]\cos\frac{(2x+1)k\pi}{2N}$$

$$+\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{r-1}[f(n+r-N-x-1)-(-1)^k f(n+r-x-1)]\sin\frac{(2x+1)k\pi}{2N}$$

$$-\sqrt{\frac{2}{N}}P_k\cos\frac{rk\pi}{N}\sum_{x=0}^{r-1}[f(n-N-x-1)-(-1)^k f(n-x-1)]\sin\frac{(2x+1)k\pi}{2N}. \qquad (2.26)$$

$$\text{for } k=1,\ldots,N.$$

The C language implementation of equation (2.26) to implement r-point update where $1 \le r \le N$-1 is given in figure (A.8) of Appendix A. The code accepts the new data points as inputs, and uses equation (2.26) to calculate the independent DST update of the shifted sequence. The values obtained from the update algorithm are compared to the DST of the shifted sequence calculated using definition, and RMS values are calculated.

2.4 DCT/DST type-III rectangular update derivations

As presented in reference [7] the DCT and DST of type-III are defined as follows:

$$C(k) = \sqrt{\frac{2}{N}} \sum_{x=0}^{N-1} P_x f(x) \cos \frac{x(2k+1)\pi}{2N} \tag{2.27}$$

for $k = 0,1,..., N-1$,

and

$$S(k) = \sqrt{\frac{2}{N}} \sum_{x=0}^{N} P_x f(x) \sin \frac{x(2k+1)\pi}{2N} \tag{2.28}$$

for $k = 1,2,..., N$,

where, $P_j = \frac{1}{\sqrt{2}}$ for $j = 0$ or N, and 1 otherwise. Xi and Chicharo [30] modified the notation to include the sequence pointer n, to keep track of points in the presence of shifting data,

$$C(n,k) = \sqrt{\frac{2}{N}} \sum_{x=0}^{N-1} P_x f(n-N+x) \cos \frac{x(2k+1)\pi}{2N} \tag{2.29}$$

for $k = 0,1,..., N-1$,

and

$$S(n,k) = \sqrt{\frac{2}{N}} \sum_{x=0}^{N} P_x f(n-N+x) \sin \frac{x(2k+1)\pi}{2N} \tag{2.30}$$

for $k = 1,2,..., N$.

2.4.1 Derivation of *r*-point equations

In this subsection we extend the derivation originally given in [22] for the *r*-point update of DCT and DST to include the sequence pointer of equations (2.29) and (2.30). The resulting formulae depend on both DCT and DST coefficients. In Section 2.4.2 independent DCT-only and DST-only formulae are derived.

Let $C(n+r,k)$ for k=0,1,…,N-1, represent the updated DCT coefficients including the effect of *r*-points of new input data. From equation (2.29):

$$\sqrt{\frac{N}{2}}C(n+r,k)=\sum_{x=0}^{N-1}f(n-N+x+r)\cos\frac{x(2k+1)\pi}{2N}$$

$$+\frac{1}{\sqrt{2}}f(n-N+r)-f(n-N+r)$$

$$=\sum_{x=0}^{N-1-r}f(n-N+x+r)\cos\frac{x(2k+1)\pi}{2N}+\sum_{x=N-r}^{N-1}f(n-N+x+r)\cos\frac{x(2k+1)\pi}{2N}$$

$$+\frac{1}{\sqrt{2}}f(n-N+r)-f(n-N+r)$$

Substituting $y=x+r$ in the first term and $i=x+r-N$ in the second term yields:

$$=\sum_{y=r}^{N-1}f(n-N+y)\cos\frac{(y-r)(2k+1)\pi}{2N}+\sum_{i=0}^{r-1}f(n+i)\cos\frac{(i-r+N)(2k+1)\pi}{2N}$$

$$+\frac{1}{\sqrt{2}}f(n-N+r)-f(n-N+r)$$

$$=\sum_{y=0}^{N-1}f(n-N+y)\cos\frac{(y-r)(2k+1)\pi}{2N}-\sum_{y=0}^{r-1}f(n-N+y)\cos\frac{(y-r)(2k+1)\pi}{2N}$$

$$-\sum_{i=0}^{r-1}f(n+i)\sin\frac{(i-r)(2k+1)\pi}{2N}\sin\frac{(2k+1)\pi}{2}+\frac{1}{\sqrt{2}}f(n-N+r)-f(n-N+r)$$

Simplifying above equation yields:

$$=\sum_{y=0}^{N-1}f(n-N+y)\cos\frac{y(2k+1)\pi}{2N}\cos\frac{(2k+1)r\pi}{2N}$$

$$+\sum_{y=0}^{N-1}f(n-N+y)\sin\frac{y(2k+1)\pi}{2N}\sin\frac{(2k+1)r\pi}{2N}$$

$$-\sum_{y=0}^{r-1}f(n-N+y)\cos\frac{(y-r)(2k+1)\pi}{2N}$$

$$-\sum_{y=0}^{r-1}f(n+y)\sin\frac{(y-r)(2k+1)\pi}{2N}\sin\frac{(2k+1)\pi}{2}$$

$$+\left(\frac{1}{\sqrt{2}}-1\right)f(n-N+r)$$

$$=E_r\sum_{y=0}^{N-1}P_y f(n-N+y)\cos\frac{y(2k+1)\pi}{2N}+E_r f(n-N)-E_r\frac{1}{\sqrt{2}}f(n-N)$$

$$+F_r\sum_{y=0}^{N}P_y f(n-N+y)\sin\frac{y(2k+1)\pi}{2N}-(-1)^k F_r\frac{1}{\sqrt{2}}x(n)$$

$$-\sum_{y=0}^{r-1}f(n-N+y)\cos\frac{(y-r)(2k+1)\pi}{2N}$$

$$-(-1)^k\sum_{y=0}^{r-1}f(n+y)\sin\frac{(y-r)(2k+1)\pi}{2N}$$

$$+\left(\frac{1}{\sqrt{2}}-1\right)f(n-N+r)$$

where,

$$E_r=\cos\frac{(2k+1)r\pi}{2N}\quad\text{and}\quad F_r=\sin\frac{(2k+1)r\pi}{2N}$$

Using the definition of the DCT-III and the DST-III as defined by equations (2.29) and (2.30) results in:

$$C(n+r,k)=E_r C(n,k)+F_r S(n,k)$$

$$-\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n-N+r-1-x)\cos\frac{(x+1)(2k+1)\pi}{2N}$$

$$+(-1)^k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n+r-1-x)\sin\frac{(x+1)(2k+1)\pi}{2N}$$

$$-\sqrt{\frac{2}{N}}\left[(-1)^k F_r\frac{1}{\sqrt{2}}f(n)-E_r\left(1-\frac{1}{\sqrt{2}}\right)f(n-N)+\left(1-\frac{1}{\sqrt{2}}\right)f(n-N+r)\right]$$

(2.31)

Equation (2.31) is the *r*-point update equation for DCT-III but it also contains the DST coefficients.

Similarly, let $S(n+r,k)$ for k=1,…,N-1, represent the updated DST coefficients including the effect of *r*-points of new input data. From equation (2.30).

$$\sqrt{\frac{N}{2}}S(n+r,k)=\sum_{x=0}^{N}f(n-N+x+r)\sin\frac{x(2k+1)\pi}{2N}$$

$$+(-1)^k\frac{1}{\sqrt{2}}f(n+r)-(-1)^k f(n+r)$$

$$=\sum_{x=0}^{N-1-r}f(n-N+x+r)\sin\frac{x(2k+1)\pi}{2N}+\sum_{x=N-r}^{N-1}f(n-N+x+r)\sin\frac{x(2k+1)\pi}{2N}$$

$$+(-1)^k\frac{1}{\sqrt{2}}f(n+r)-(-1)^k f(n+r)$$

Substituting $y=x+r$ in the first term and $i=x+r-N$ in the second term results in:

$$=\sum_{y=r}^{N-1}f(n-N+y)\sin\frac{(y-r)(2k+1)\pi}{2N}+\sum_{i=0}^{r}f(n+i)\sin\frac{(i-r+N)(2k+1)\pi}{2N}$$

$$+(-1)^k\frac{1}{\sqrt{2}}f(n+r)-(-1)^k f(n+r)$$

$$=\sum_{y=0}^{N-1}f(n-N+y)\sin\frac{(y-r)(2k+1)\pi}{2N}-\sum_{y=0}^{r-1}f(n-N+y)\sin\frac{(y-r)(2k+1)\pi}{2N}$$

$$+\sum_{i=0}^{r}f(n+i)\cos\frac{(i-r)(2k+1)\pi}{2N}\sin\frac{(2k+1)\pi}{2}+(-1)^k\frac{1}{\sqrt{2}}f(n+r)-(-1)^k f(n+r)$$

Simplifying the above equation yields:

$$=\sum_{y=0}^{N-1}f(n-N+y)\sin\frac{y(2k+1)\pi}{2N}\cos\frac{(2k+1)r\pi}{2N}$$

$$-\sum_{y=0}^{N-1}f(n-N+y)\cos\frac{y(2k+1)\pi}{2N}\sin\frac{(2k+1)r\pi}{2N}$$

$$-\sum_{y=0}^{r-1} f(n-N+y)\sin\frac{(y-r)(2k+1)\pi}{2N}$$

$$+(-1)^k\sum_{y=0}^{r} f(n+y)\cos\frac{(y-r)(2k+1)\pi}{2N}$$

$$+(-1)^k\frac{1}{\sqrt{2}}f(n+r)-(-1)^k f(n+r)$$

$$=E_r\sum_{y=0}^{N} P_y f(n-N+y)\sin\frac{y(2k+1)\pi}{2N}-(-1)^k E_r\frac{1}{\sqrt{2}}f(n)$$

$$-F_r\sum_{y=0}^{N-1} P_y f(n-N+y)\cos\frac{y(2k+1)\pi}{2N}-F_r x(n-N)+F_r\frac{1}{\sqrt{2}}x(n-N)$$

$$-\sum_{y=0}^{r-1} f(n-N+y)\sin\frac{(y-r)(2k+1)\pi}{2N}+(-1)^k\sum_{y=0}^{r-1} f(n+y)\cos\frac{(y-r)(2k+1)\pi}{2N}$$

$$+(-1)^k\frac{1}{\sqrt{2}}f(n+r)$$

Therefore,

$$S(n+r,k)=E_r S(n,k)-F_r C(n,k)-\sqrt{\frac{2}{N}}\sum_{y=0}^{r-1} f(n-N+y)\sin\frac{(y-r)(2k+1)\pi}{2N}$$

$$+(-1)^k\sqrt{\frac{2}{N}}\sum_{y=0}^{r-1} f(n+y)\cos\frac{(y-r)(2k+1)\pi}{2N}$$

$$+\sqrt{\frac{2}{N}}\left[F_r\left(\frac{1}{\sqrt{2}}-1\right)f(n-N)+(-1)^k\frac{1}{\sqrt{2}}f(n+r)-(-1)^k E_r\frac{1}{\sqrt{2}}f(n)\right]$$

Substituting x=r-1-y yields:

$$S(n+r,k)=E_r S(n,k)-F_r C(n,k)$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} f(n-N+r-1-x)\sin\frac{(x+1)(2k+1)\pi}{2N}$$

$$+(-1)^k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n+r-1-x)\cos\frac{(x+1)(2k+1)\pi}{2N}$$

$$+\sqrt{\frac{2}{N}}\left[F_r\left(\frac{1}{\sqrt{2}}-1\right)f(n-N)+(-1)^k\frac{1}{\sqrt{2}}f(n+r)-(-1)^k E_r\frac{1}{\sqrt{2}}f(n)\right] \quad (2.32)$$

Equation (2.32) is the *r*-point update equation for DST-III.

2.4.2 Computation of *r*-point independent update equations for DCT and DST type-III

Equation (2.31) provides an *r*-point update for the DCT but it includes both $C(n,k)$ and $S(n,k)$ coefficients. Therefore it would be necessary to update both the DCT and the DST even if only the DCT coefficients are required. We therefore seek an equation that provides an update of the DCT in terms of DCT coefficients only.

Taking the *z*-transform of equation (2.31) yields:

$$z^r C(z,k)=E_r C(z,k)+F_r S(z,k)-\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}z^{-N+r-1-x}\cos\frac{(x+1)(2k+1)\pi}{2N}$$

$$+(-1)^k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}z^{r-1-x}\sin\frac{(x+1)(2k+1)\pi}{2N}$$

$$-\sqrt{\frac{2}{N}}\left[(-1)^k F_r\frac{1}{\sqrt{2}}-E_r\left(1-\frac{1}{\sqrt{2}}\right)z^{-N}+\left(1-\frac{1}{\sqrt{2}}\right)z^{-N+r}\right]$$

Therefore,

$$(z^r-E_r)C(z,k)=F_r S(z,k)-\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}z^{-N+r-1-x}\cos\frac{(x+1)(2k+1)\pi}{2N}$$

$$+(-1)^k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}z^{r-1-x}\sin\frac{(x+1)(2k+1)\pi}{2N}$$

$$-\sqrt{\frac{2}{N}}\left[(-1)^k F_r\frac{1}{\sqrt{2}}-E_r\left(1-\frac{1}{\sqrt{2}}\right)z^{-N}+\left(1-\frac{1}{\sqrt{2}}\right)z^{-N+r}\right]$$

Solving for $C(z,k)$ yields:

$$C(z,k)=\frac{F_r}{(z^r-E_r)}S(z,k)-\frac{\sqrt{\frac{2}{N}}}{(z^r-E_r)}\sum_{x=0}^{r-1}z^{-N+r-1-x}\cos\frac{(x+1)(2k+1)\pi}{2N}$$

$$+\frac{(-1)^k\sqrt{\frac{2}{N}}}{(z^r-E_r)}\sum_{x=0}^{r-1}z^{r-1-x}\sin\frac{(x+1)(2k+1)\pi}{2N}$$

$$-\frac{\sqrt{\frac{2}{N}}}{(z^r-E_r)}\left[(-1)^k F_r\frac{1}{\sqrt{2}}-E_r\left(1-\frac{1}{\sqrt{2}}\right)z^{-N}+\left(1-\frac{1}{\sqrt{2}}\right)z^{-N+r}\right]$$

Solving further results in:

$$C(z,k)=\frac{F_r z^{-r}}{(1-z^{-r}E_r)}S(z,k)-\frac{\sqrt{\frac{2}{N}}}{(1-z^{-r}E_r)}\sum_{x=0}^{r-1}z^{-N-1-x}\cos\frac{(x+1)(2k+1)\pi}{2N}$$

$$+\frac{(-1)^k\sqrt{\frac{2}{N}}}{(1-z^{-r}E_r)}\sum_{x=0}^{r-1}z^{-1-x}\sin\frac{(x+1)(2k+1)\pi}{2N}$$

$$-\frac{\sqrt{\frac{2}{N}}}{(1-z^{-r}E_r)}\left[(-1)^k F_r\frac{1}{\sqrt{2}}z^{-r}-E_r\left(1-\frac{1}{\sqrt{2}}\right)z^{-N-r}+\left(1-\frac{1}{\sqrt{2}}\right)z^{-N}\right] \quad (2.33)$$

Taking z-transform of equation (2.32) results in:

$$z^r S(z,k)=E_r S(z,k)-F_r C(z,k)$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}z^{-N+r-1-x}\sin\frac{(x+1)(2k+1)\pi}{2N}+(-1)^k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}z^{r-1-x}\cos\frac{(x+1)(2k+1)\pi}{2N}$$

$$+\sqrt{\frac{2}{N}}\left[F_r\left(\frac{1}{\sqrt{2}}-1\right)z^{-N}+(-1)^k\frac{1}{\sqrt{2}}z^r-(-1)^k E_r\frac{1}{\sqrt{2}}\right]$$

Therefore,

$$(z^r-E_r)S(z,k)=-F_r C(z,k)$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} z^{-N+r-1-x}\sin\frac{(x+1)(2k+1)\pi}{2N}+(-1)^k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} z^{r-1-x}\cos\frac{(x+1)(2k+1)\pi}{2N}$$

$$+\sqrt{\frac{2}{N}}\left[F_r\left(\frac{1}{\sqrt{2}}-1\right)z^{-N}+(-1)^k\frac{1}{\sqrt{2}}z^r-(-1)^k E_r\frac{1}{\sqrt{2}}\right]$$

Solving for $S(z,k)$ results in:

$$S(z,k)=-\frac{F_r}{(z^r-E_r)}C(z,k)$$

$$+\frac{\sqrt{\frac{2}{N}}}{(z^r-E_r)}\sum_{x=0}^{r-1} z^{-N+r-1-x}\sin\frac{(x+1)(2k+1)\pi}{2N}+\frac{(-1)^k\sqrt{\frac{2}{N}}}{(z^r-E_r)}\sum_{x=0}^{r-1} z^{r-1-x}\cos\frac{(x+1)(2k+1)\pi}{2N}$$

$$+\frac{\sqrt{\frac{2}{N}}}{(z^r-E_r)}\left[F_r\left(\frac{1}{\sqrt{2}}-1\right)z^{-N}+(-1)^k\frac{1}{\sqrt{2}}z^r-(-1)^k E_r\frac{1}{\sqrt{2}}\right]$$

Therefore,

$$S(z,k)=-\frac{F_r z^{-r}}{(1-E_r z^{-r})}C(z,k)$$

$$+\frac{\sqrt{\frac{2}{N}}}{(1-E_r z^{-r})}\sum_{x=0}^{r-1} z^{-N-1-x}\sin\frac{(x+1)(2k+1)\pi}{2N}+\frac{(-1)^k\sqrt{\frac{2}{N}}}{(1-E_r z^{-r})}\sum_{x=0}^{r-1} z^{-1-x}\cos\frac{(x+1)(2k+1)\pi}{2N}$$

$$+\frac{\sqrt{\frac{2}{N}}}{(1-E_r z^{-r})}\left[F_r\left(\frac{1}{\sqrt{2}}-1\right)z^{-N-r}+(-1)^k\frac{1}{\sqrt{2}}-(-1)^k E_r\frac{1}{\sqrt{2}}z^{-r}\right] \qquad (2.34)$$

Eliminating $S(z,k)$ by substituting equation (2.34) in equation (2.33) yields:

$$C(z,k)=-\frac{{F_r}^2 z^{-2r}}{(1-E_r z^{-r})^2}C(z,k)$$

$$+\frac{F_r\sqrt{\frac{2}{N}}}{(1-E_r z^{-r})^2}\sum_{x=0}^{r-1} z^{-N-1-x-r}\sin\frac{(x+1)(2k+1)\pi}{2N}+\frac{(-1)^k F_r\sqrt{\frac{2}{N}}}{(1-E_r z^{-r})^2}\sum_{x=0}^{r-1} z^{-1-x-r}\cos\frac{(x+1)(2k+1)\pi}{2N}$$

$$+\frac{F_r\sqrt{\frac{2}{N}}}{(1-E_r z^{-r})^2}\left[F_r\left(\frac{1}{\sqrt{2}}-1\right)z^{-N-2r}+(-1)^k\frac{1}{\sqrt{2}}z^{-r}-(-1)^k E_r\frac{1}{\sqrt{2}}z^{-2r}\right]$$

$$-\frac{\sqrt{\frac{2}{N}}}{(1-z^{-r}E_r)}\sum_{x=0}^{r-1} z^{-N-1-x}\cos\frac{(x+1)(2k+1)\pi}{2N}+\frac{(-1)^k\sqrt{\frac{2}{N}}}{(1-z^{-r}E_r)}\sum_{x=0}^{r-1} z^{-1-x}\sin\frac{(x+1)(2k+1)\pi}{2N}$$

$$-\frac{\sqrt{\frac{2}{N}}}{(1-z^{-r}E_r)}\left[(-1)^k F_r\frac{1}{\sqrt{2}}z^{-r}-E_r\left(1-\frac{1}{\sqrt{2}}\right)z^{-N-r}+\left(1-\frac{1}{\sqrt{2}}\right)z^{-N}\right]$$

Gathering terms in $C(z,k)$ onto the left hand side,

$$\left\{\left(1-z^{-r}E_r\right)^2+{F_r}^2 z^{-2r}\right\}C(z,k)=$$

$$F_r\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} z^{-N-1-x-r}\sin\frac{(x+1)(2k+1)\pi}{2N}+(-1)^k F_r\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} z^{-1-x-r}\cos\frac{(x+1)(2k+1)\pi}{2N}$$

$$+F_r\sqrt{\frac{2}{N}}\left[F_r\left(\frac{1}{\sqrt{2}}-1\right)z^{-N-2r}+(-1)^k\frac{1}{\sqrt{2}}z^{-r}-(-1)^k E_r\frac{1}{\sqrt{2}}z^{-2r}\right]$$

$$-\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} z^{-N-1-x}\cos\frac{(x+1)(2k+1)\pi}{2N}+E_r\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} z^{-N-1-x-r}\cos\frac{(x+1)(2k+1)\pi}{2N}$$

$$+(-1)^k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} z^{-1-x}\sin\frac{(x+1)(2k+1)\pi}{2N}-E_r(-1)^k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} z^{-1-x-r}\sin\frac{(x+1)(2k+1)\pi}{2N}$$

$$-\sqrt{\frac{2}{N}}\left[(-1)^k F_r\frac{1}{\sqrt{2}}z^{-r}-E_r\left(1-\frac{1}{\sqrt{2}}\right)z^{-N-r}+\left(1-\frac{1}{\sqrt{2}}\right)z^{-N}\right]$$

$$+E_r\sqrt{\frac{2}{N}}\left[(-1)^k F_r\frac{1}{\sqrt{2}}z^{-2r}-E_r\left(1-\frac{1}{\sqrt{2}}\right)z^{-N-2r}+\left(1-\frac{1}{\sqrt{2}}\right)z^{-N-r}\right]$$

Therefore,

$$C(n+1,k)+C(n-2r+1,k)-2E_rC(n-r+1,k)=$$

$$F_r\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n-N-x-r)\sin\frac{(x+1)(2k+1)\pi}{2N}$$

$$+(-1)^kF_r\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n-x-r)\cos\frac{(x+1)(2k+1)\pi}{2N}$$

$$-\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n-N-x)\cos\frac{(x+1)(2k+1)\pi}{2N}$$

$$+E_r\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n-N-x-r)\cos\frac{(x+1)(2k+1)\pi}{2N}$$

$$+(-1)^k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n-x)\sin\frac{(x+1)(2k+1)\pi}{2N}$$

$$-E_r(-1)^k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n-x-r)\sin\frac{(x+1)(2k+1)\pi}{2N}$$

$$-\sqrt{\frac{2}{N}}\left[(-1)^kF_r\frac{1}{\sqrt{2}}f(n-r+1)-E_r\left(1-\frac{1}{\sqrt{2}}\right)f(n-N-r+1)+\left(1-\frac{1}{\sqrt{2}}\right)f(n-N+1)\right]$$

$$+F_r\sqrt{\frac{2}{N}}\left[F_r\left(\frac{1}{\sqrt{2}}-1\right)f(n-N-2r+1)+(-1)^k\frac{1}{\sqrt{2}}f(n-r+1)\right.$$

$$\left.-(-1)^kE_r\frac{1}{\sqrt{2}}f(n-2r+1)\right]$$

$$+E_r\sqrt{\frac{2}{N}}\left[(-1)^kF_r\frac{1}{\sqrt{2}}f(n-2r+1)-E_r\left(1-\frac{1}{\sqrt{2}}\right)f(n-N-2r+1)\right.$$

$$\left.+\left(1-\frac{1}{\sqrt{2}}\right)f(n-N-r+1)\right]$$

Multiplying throughout by z^r and then taking the inverse z-transform yields:

$$C(n+r,k)=2E_rC(n,k)-C(n-r,k)$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}\left[(-1)^k f(n+r-1-x)+F_r f(n-N-x-1)-(-1)^k E_r f(n-1-x)\right]\sin\frac{(x+1)(2k+1)\pi}{2N}$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}\left[(-1)^k F_r f(n-1-x)-f(n-N-x+r-1)+E_r f(n-N-x-1)\right]\cos\frac{(x+1)(2k+1)\pi}{2N}$$

$$-\sqrt{\frac{2}{N}}\left(1-\frac{1}{\sqrt{2}}\right)f(n-N-r)+2\sqrt{\frac{2}{N}}\left(1-\frac{1}{\sqrt{2}}\right)E_r f(n-N)$$

$$-\sqrt{\frac{2}{N}}\left(1-\frac{1}{\sqrt{2}}\right)f(n-N+r) \qquad (2.35)$$

$$\text{for } k=0,1,\ldots,N-1.$$

Equation (2.35) is the general *r*-point independent update equation for DCT-III. The C language implementation of the *r*-point DCT type-III update is included in figure (A.11) of Appendix A.

To find the *r*-point update equation for DST-III we substitute equation (2.33) in equation (2.34) that results in:

$$S(z,k)=-\frac{F_r{}^2 z^{-2r}}{(1-E_r z^{-r})^2}S(z,k)$$

$$+\frac{F_r\sqrt{\frac{2}{N}}}{(1-E_r z^{-r})^2}\sum_{x=0}^{r-1}z^{-N-1-x-r}\cos\frac{(x+1)(2k+1)\pi}{2N}-\frac{(-1)^k F_r\sqrt{\frac{2}{N}}}{(1-E_r z^{-r})^2}\sum_{x=0}^{r-1}z^{-1-x-r}\sin\frac{(x+1)(2k+1)\pi}{2N}$$

$$+\frac{F_r\sqrt{\frac{2}{N}}}{(1-E_r z^{-r})^2}\left[(-1)^k F_r\frac{1}{\sqrt{2}}z^{-2r}-E_r\left(1-\frac{1}{\sqrt{2}}\right)z^{-N-2r}+\left(1-\frac{1}{\sqrt{2}}\right)z^{-N-r}\right]$$

$$+\frac{\sqrt{\frac{2}{N}}}{(1-z^{-r}E_r)}\sum_{x=0}^{r-1}z^{-N-1-x}\sin\frac{(x+1)(2k+1)\pi}{2N}+\frac{(-1)^k\sqrt{\frac{2}{N}}}{(1-z^{-r}E_r)}\sum_{x=0}^{r-1}z^{-1-x}\cos\frac{(x+1)(2k+1)\pi}{2N}$$

$$+\frac{\sqrt{\frac{2}{N}}}{(1-z^{-r}E_r)}\left[F_r\left(\frac{1}{\sqrt{2}}-1\right)z^{-N-r}+(-1)^k\frac{1}{\sqrt{2}}-(-1)^k\frac{1}{\sqrt{2}}E_r z^{-r}\right]$$

Gathering terms in $S(z,k)$ onto the left hand side,

$$\left\{\left(1-z^{-r}E_r\right)^2+{F_r}^2 z^{-2r}\right\}S(z,k)=$$

$$F_r\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}z^{-N-1-x-r}\cos\frac{(x+1)(2k+1)\pi}{2N}-(-1)^k F_r\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}z^{-1-x-r}\sin\frac{(x+1)(2k+1)\pi}{2N}$$

$$+F_r\sqrt{\frac{2}{N}}\left[(-1)^k F_r\frac{1}{\sqrt{2}}z^{-2r}-E_r\left(1-\frac{1}{\sqrt{2}}\right)z^{-N-2r}+\left(1-\frac{1}{\sqrt{2}}\right)z^{-N-r}\right]$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}z^{-N-1-x}\sin\frac{(x+1)(2k+1)\pi}{2N}-E_r\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}z^{-N-1-x-r}\sin\frac{(x+1)(2k+1)\pi}{2N}$$

$$+(-1)^k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}z^{-1-x}\cos\frac{(x+1)(2k+1)\pi}{2N}-E_r(-1)^k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}z^{-1-x-r}\cos\frac{(x+1)(2k+1)\pi}{2N}$$

$$+\sqrt{\frac{2}{N}}\left[F_r\left(\frac{1}{\sqrt{2}}-1\right)z^{-N-r}+(-1)^k\frac{1}{\sqrt{2}}-(-1)^k\frac{1}{\sqrt{2}}E_r z^{-r}\right]$$

$$-E_r\sqrt{\frac{2}{N}}\left[F_r\left(\frac{1}{\sqrt{2}}-1\right)z^{-N-2r}+(-1)^k\frac{1}{\sqrt{2}}z^{-r}-(-1)^k\frac{1}{\sqrt{2}}E_r z^{-2r}\right]$$

Multiplying throughout by z^r and then taking the inverse z-transform yields:

$$S(n+1,k)+S(n-2r+1,k)-2E_r S(n-r+1,k)=$$

$$F_r\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n-N-x-r)\cos\frac{(x+1)(2k+1)\pi}{2N}$$

$$-(-1)^k F_r\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n-x-r)\sin\frac{(x+1)(2k+1)\pi}{2N}$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n-N-x)\sin\frac{(x+1)(2k+1)\pi}{2N}$$

$$- E_r \sqrt{\frac{2}{N}} \sum_{x=0}^{r-1} f(n-N-x-r) \sin \frac{(x+1)(2k+1)\pi}{2N}$$

$$+ (-1)^k \sqrt{\frac{2}{N}} \sum_{x=0}^{r-1} f(n-x) \cos \frac{(x+1)(2k+1)\pi}{2N}$$

$$- E_r (-1)^k \sqrt{\frac{2}{N}} \sum_{x=0}^{r-1} f(n-x-r) \cos \frac{(x+1)(2k+1)\pi}{2N}$$

$$+ \sqrt{\frac{2}{N}} \left[F_r \left(\frac{1}{\sqrt{2}} - 1 \right) f(n-N-r+1) + (-1)^k \frac{1}{\sqrt{2}} f(n+1) - (-1)^k \frac{1}{\sqrt{2}} E_r f(n-r+1) \right]$$

$$+ F_r \sqrt{\frac{2}{N}} \left[(-1)^k F_r \frac{1}{\sqrt{2}} f(n-2r+1) - E_r \left(1 - \frac{1}{\sqrt{2}} \right) f(n-N-2r+1) \right.$$

$$\left. + \left(1 - \frac{1}{\sqrt{2}} \right) f(n-N-r+1) \right]$$

$$- E_r \sqrt{\frac{2}{N}} \left[F_r \left(\frac{1}{\sqrt{2}} - 1 \right) f(n-N-2r+1) + (-1)^k \frac{1}{\sqrt{2}} f(n-r+1) \right.$$

$$\left. - (-1)^k \frac{1}{\sqrt{2}} E_r f(n-2r+1) \right]$$

Therefore,

$$S(n+r,k) = 2E_r S(n,k) - S(n-r,k)$$

$$+ \sqrt{\frac{2}{N}} \sum_{x=0}^{r-1} \left[(-1)^k f(n+r-1-x) + F_r f(n-N-x-1) - (-1)^k E_r f(n-x-1) \right]$$

$$\cos \frac{(x+1)(2k+1)\pi}{2N}$$

$$+ \sqrt{\frac{2}{N}} \sum_{x=0}^{r-1} \left[f(n-N-x+r-1) - (-1)^k F_r f(n-x-1) - E_r f(n-N-x-1) \right]$$

$$\sin \frac{(x+1)(2k+1)\pi}{2N}$$

$$+(-1)^k\sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}f(n-r)+(-1)^k\sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}f(n+r)-(-1)^k\sqrt{2}\sqrt{\frac{2}{N}}E_r f(n) \tag{2.36}$$

$$\text{for } k=1,...,N.$$

Equation (2.36) is the general *r*-point independent update equation for DST type-III. The C language implementation of equation (2.36) to implement *r*-point update is listed in figure (A.13) of Appendix A.

2.5 DCT/DST type-IV rectangular update derivations

As developed by [7] the DCT and DST of type-IV are defined as follows:

$$C(k)=\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}f(x)\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right] \tag{2.37}$$

$$\text{for } k=0,1,...,N-1,$$

and

$$S(k)=\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}f(x)\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right] \tag{2.38}$$

$$\text{for } k=0,1,...,N-1,$$

where, $P_j=\frac{1}{\sqrt{2}}$ for $j=0$ or N, and 1 otherwise. Xi and Chicharo [30] modified the notation to include the sequence pointer *n*, to keep track of points in the presence of shifting data,

$$C(n,k)=\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}f(n-N+x)\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right] \tag{2.39}$$

$$\text{for } k=0,1,...,N-1,$$

and

$$S(n,k)=\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}f(n-N+x)\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right] \tag{2.40}$$

$$\text{for } k=0,1,...,N-1.$$

2.5.1 Derivation of *r*-point equations

In this subsection we extend the derivation originally given in [22] for the *r*-point update of DCT and DST to include the sequence pointer of equations (2.39) and (2.40). The resulting formulae depend on both DCT and DST coefficients, in Section 2.5.2 we will derive independent DCT-only and DST-only formulae.

Let $C(n+r,k)$ where k=0,…,N-1, represent the updated DCT coefficients including the effect of *r*-points of new input data. From equation (2.39):

$$C(n+r,k) = \sqrt{\frac{2}{N}} \sum_{x=0}^{N-1} f(n-N+x+r) \cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$\sqrt{\frac{N}{2}} C(n+r,k) = \sum_{x=0}^{N-1-r} f(n-N+x+r) \cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+ \sum_{x=N-r}^{N-1} f(n-N+x+r) \cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

Substituting $y=x+r$ in the first term and $i=x+r-N$ in the second term yields:

$$\sqrt{\frac{N}{2}} C(n+r,k) = \sum_{y=r}^{N-1} f(n-N+y) \cos\left[\left(\frac{2(y-r)+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+ \sum_{i=0}^{r-1} f(n+i) \cos\left[\left(\frac{2(i-r+N)+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$= \sum_{y=0}^{N-1} f(n-N+y) \cos\left[\left(\frac{2(y-r)+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$- \sum_{y=0}^{r-1} f(n-N+y) \cos\left[\left(\frac{2(y-r)+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+ \sum_{i=0}^{r-1} f(n+i) \cos\left[\left(\frac{2(i-r+N)+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$= \sum_{y=0}^{N-1} f(n-N+y)\cos\left[\left(\frac{2y+1}{2}-r\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$-\sum_{y=0}^{r-1} f(n-N+y)\cos\left[\left(\frac{2(y-r)+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+\sum_{i=0}^{r-1} f(n+i)\cos\left[\left(\frac{2i-2r+1}{2}+N\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$= \sum_{y=0}^{N-1} f(n-N+y)\cos\left[\left(\frac{2y+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]\cos\left[\left(\frac{r(2k+1)}{2}\right)\frac{\pi}{N}\right]$$

$$+\sum_{y=0}^{N-1} f(n-N+y)\sin\left[\left(\frac{2y+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]\sin\left[\left(\frac{r(2k+1)}{2}\right)\frac{\pi}{N}\right]$$

$$-\sum_{y=0}^{r-1} f(n-N+y)\cos\left[\left(\frac{2(y-r)+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$-\sum_{i=0}^{r-1} f(n+i)\sin\left[\left(\frac{2i-2r+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]\sin\left[\left(\frac{2k+1}{2}\right)\pi\right]$$

Using the definition of DCT-IV and DST-IV as defined in equation (2.39) and (2.40) results in:

$$C(n+r,k) = C(n,k)\cos\frac{r(2k+1)\pi}{2N} + S(n,k)\sin\frac{r(2k+1)\pi}{2N}$$

$$-\sqrt{\frac{2}{N}}\sum_{y=0}^{r-1} f(n-N+y)\cos\left[\left(\frac{2(y-r)+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$-\sqrt{\frac{2}{N}}\sum_{y=0}^{r-1} f(n+y)\sin\left[\left(\frac{2(y-r)+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]\sin\left[\left(\frac{2k+1}{2}\right)\pi\right]$$

Substituting $y=r\text{-}1\text{-}x$ yields:

$$C(n+r,k) = C(n,k)\cos\frac{r(2k+1)\pi}{2N} + S(n,k)\sin\frac{r(2k+1)\pi}{2N}$$

$$-\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} f(n-N+r-1-x)\cos\left[\left(\frac{-2(x+1)+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$-\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} f(n+r-1-x)\sin\left[\left(\frac{-2(x+1)+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]\sin\left[\left(\frac{2k+1}{2}\right)\pi\right]$$

Therefore,

$$C(n+r,k) = C(n,k)\cos\frac{r(2k+1)\pi}{2N} + S(n,k)\sin\frac{r(2k+1)\pi}{2N}$$

$$-\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} f(n-N+r-1-x)\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+(-1)^k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} f(n+r-1-x)\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right] \qquad (2.41)$$

Equation (2.41) is the *r*-point update equation for DCT-IV.

Similarly for DST-IV, $S(n+r,k)$ where, k=0,…,N-1, represent the updated DST coefficients including the effect of *r*-points of new input data. From equation (2.40):

$$S(n+r,k) = \sqrt{\frac{2}{N}}\sum_{x=0}^{N-1} f(n-N+x+r)\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$\sqrt{\frac{N}{2}}S(n+r,k) = \sum_{x=0}^{N-1-r} f(n-N+x+r)\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+\sum_{x=N-r}^{N-1} f(n-N+x+r)\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

Substituting $y=x+r$ in the first term and $i=x+r-N$ in the second term yields:

$$\sqrt{\frac{N}{2}}S(n+r,k) = \sum_{y=r}^{N-1} f(n-N+y)\sin\left[\left(\frac{2(y-r)+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+\sum_{i=0}^{r-1} f(n+i)\sin\left[\left(\frac{2(i-r+N)+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$= \sum_{y=0}^{N-1} f(n-N+y) \sin\left[\left(\frac{2(y-r)+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$- \sum_{y=0}^{r-1} f(n-N+y) \sin\left[\left(\frac{2(y-r)+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+ \sum_{i=0}^{r-1} f(n+i) \sin\left[\left(\frac{2(i-r+N)+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$= \sum_{y=0}^{N-1} f(n-N+y) \sin\left[\left(\frac{2y+1}{2}-r\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$- \sum_{y=0}^{r-1} f(n-N+y) \sin\left[\left(\frac{2(y-r)+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+ \sum_{i=0}^{r-1} f(n+i) \sin\left[\left(\frac{2i-2r+1}{2}+N\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$= \sum_{y=0}^{N-1} f(n-N+y) \sin\left[\left(\frac{2y+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right] \cos\left[\left(\frac{r(2k+1)}{2}\right)\frac{\pi}{N}\right]$$

$$+ \sum_{y=0}^{N-1} f(n-N+y) \cos\left[\left(\frac{2y+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right] \sin\left[\left(\frac{r(2k+1)}{2}\right)\frac{\pi}{N}\right]$$

$$- \sum_{y=0}^{r-1} f(n-N+y) \sin\left[\left(\frac{2(y-r)+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+ \sum_{i=0}^{r-1} f(n+i) \cos\left[\left(\frac{2i-2r+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right] \sin\left[\left(\frac{2k+1}{2}\right)\pi\right]$$

Therefore,

$$S(n+r,k) = S(n,k)\cos\frac{r(2k+1)\pi}{2N} - C(n,k)\sin\frac{r(2k+1)\pi}{2N}$$

$$- \sqrt{\frac{2}{N}} \sum_{y=0}^{r-1} f(n-N+y) \sin\left[\left(\frac{2(y-r)+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+(-1)^k\sqrt{\frac{2}{N}}\sum_{y=0}^{r-1}f(n+y)\cos\left[\left(\frac{2(y-r)+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

Substituting $y=r\text{-}1\text{-}x$ results in:

$$S(n+r,k)=S(n,k)\cos\frac{r(2k+1)\pi}{2N}-C(n,k)\sin\frac{r(2k+1)\pi}{2N}$$

$$-\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n-N+r-1-x)\sin\left[\left(\frac{-2(x+1)+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+(-1)^k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n+r-1-x)\cos\left[\left(\frac{-2(x+1)+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

Therefore,

$$S(n+r,k)=S(n,k)\cos\frac{r(2k+1)\pi}{2N}-C(n,k)\sin\frac{r(2k+1)\pi}{2N}$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n-N+r-1-x)\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+(-1)^k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n+r-1-x)\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right] \qquad (2.42)$$

Equation (2.42) is the *r*-point equation for the DST-IV but it contains DCT coefficients as well.

2.5.2 Computation of the *r*-point independent update equations for DCT/DST type-IV

To derive the independent update equation for the DCT type-IV we take the *z*-transform of equation (2.41) and (2.42) and eliminate DST coefficient.

Taking *z*-transform of equation (2.41):

$$z^rC(z,k)=C(z,k)E_r+S(z,k)F_r$$

$$-\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}z^{-N+r-1-x}\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]+(-1)^k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}z^{r-1-x}\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$(z^r - E_r)C(z,k) = S(z,k)F_r - \sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} z^{-N+r-1-x}\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+(-1)^k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} z^{r-1-x}\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

Solving for $C(z,k)$:

$$C(z,k) = \frac{F_r}{(z^r - E_r)}S(z,k) - \frac{\sqrt{\frac{2}{N}}}{(z^r - E_r)}\sum_{x=0}^{r-1} z^{-N+r-1-x}\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+\frac{(-1)^k\sqrt{\frac{2}{N}}}{(z^r - E_r)}\sum_{x=0}^{r-1} z^{r-1-x}\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

Therefore,

$$C(z,k) = \frac{F_r z^{-r}}{(1 - E_r z^{-r})}S(z,k) - \frac{\sqrt{\frac{2}{N}}}{(1 - E_r z^{-r})}\sum_{x=0}^{r-1} z^{-N-1-x}\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+\frac{(-1)^k\sqrt{\frac{2}{N}}}{(1 - E_r z^{-r})}\sum_{x=0}^{r-1} z^{-1-x}\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right] \qquad (2.43)$$

Taking z-transform of equation (2.42) results in:

$$z^r S(z,k) = E_r S(z,k) - F_r C(n,k)$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} z^{-N+r-1-x}\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right] + (-1)^k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} z^{r-1-x}\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

Solving further yields:

$$(z^r - E_r)S(z,k) = F_r C(n,k)$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} z^{-N+r-1-x}\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right] + (-1)^k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} z^{r-1-x}\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

Solving for $S(z,k)$ results in:

$$S(z,k) = -\frac{F_r}{(z^r - E_r)} C(n,k) + \frac{\sqrt{\frac{2}{N}}}{(z^r - E_r)} \sum_{x=0}^{r-1} z^{-N+r-1-x} \sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+ \frac{(-1)^k \sqrt{\frac{2}{N}}}{(z^r - E_r)} \sum_{x=0}^{r-1} z^{r-1-x} \cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

Therefore,

$$S(z,k) = -\frac{F_r z^{-r}}{(1 - E_r z^{-r})} C(n,k) + \frac{\sqrt{\frac{2}{N}}}{(1 - E_r z^{-r})} \sum_{x=0}^{r-1} z^{-N-1-x} \sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+ \frac{(-1)^k \sqrt{\frac{2}{N}}}{(1 - E_r z^{-r})} \sum_{x=0}^{r-1} z^{-1-x} \cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right] \qquad (2.44)$$

Eliminating $S(z,k)$ by substituting equation (2.44) in equation (2.43) yields:

$$C(z,k) = -\frac{{F_r}^2 z^{-2r}}{(1 - E_r z^{-r})^2} C(z,k) + \frac{\sqrt{\frac{2}{N} F_r}}{(1 - E_r z^{-r})^2} \sum_{x=0}^{r-1} z^{-N-1-x-r} \sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+ \frac{(-1)^k \sqrt{\frac{2}{N} F_r}}{(1 - E_r z^{-r})^2} \sum_{x=0}^{r-1} z^{-1-x-r} \cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$- \frac{\sqrt{\frac{2}{N}}}{(1 - E_r z^{-r})} \sum_{x=0}^{r-1} z^{-N-1-x} \cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+ \frac{(-1)^k \sqrt{\frac{2}{N}}}{(1 - E_r z^{-r})} \sum_{x=0}^{r-1} z^{-1-x} \sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

Gathering terms in $C(z,k)$ onto the left hand side,

$$[(1 - z^{-r} E_r)^2 + {F_r}^2 z^{-2r}] C(z,k) =$$

$$\sqrt{\frac{2}{N}} F_r \sum_{x=0}^{r-1} z^{-N-1-x-r} \sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+(-1)^k\sqrt{\frac{2}{N}}F_r\sum_{x=0}^{r-1}z^{-1-x-r}\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$-\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}z^{-N-1-x}\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+E_r\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}z^{-N-1-x-r}\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+(-1)^k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}z^{-1-x}\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$-(-1)^k E_r\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}z^{-1-r-x}\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

Multiplying throughout by z^r and taking the inverse z-transform yields:

$$C(n+1,k)+C(n-2r+1,k)-2E_rC(n-r+1,k)=$$

$$\sqrt{\frac{2}{N}}F_r\sum_{x=0}^{r-1}f(n-N-x-r)\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+(-1)^k\sqrt{\frac{2}{N}}F_r\sum_{x=0}^{r-1}f(n-x-r)\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$-\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n-N-x)\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+E_r\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n-N-x-r)\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+(-1)^k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n-x)\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$-(-1)^k E_r\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n-x-r)\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

Therefore,

$$C(n+r,k) = 2E_r C(n,k) - C(n-r,k)$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}\left[F_r f(n-N-x-1) + (-1)^k f(n+r-x-1) - (-1)^k E_r f(n-x-1)\right]$$

$$\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}\left[E_r f(n-N-x-1) + (-1)^k F_r f(n-x-1) - f(n-N+r-x-1)\right]$$

$$\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right] \qquad (2.45)$$

$$\text{for } k = 0,1,\ldots,N-1.$$

Equation (2.45) represents the algorithm to update DCT type-IV independent of the DST coefficients. The C language implementation of the DCT type-IV update equation is given in figure (A.15) of Appendix A.

Similarly the *r*-point update equation for the DST-IV may be derived by substituting equation (2.43) in equation (2.44).

$$S(z,k) = -\frac{{F_r}^2 z^{-2r}}{(1-E_r z^{-r})^2} S(z,k) + \frac{\sqrt{\frac{2}{N}F_r}}{(1-E_r z^{-r})^2}\sum_{x=0}^{r-1} z^{-N-1-x-r}\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$-\frac{(-1)^k F_r\sqrt{\frac{2}{N}}}{(1-E_r z^{-r})^2}\sum_{x=0}^{r-1} z^{-1-x-r}\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+\frac{\sqrt{\frac{2}{N}}}{(1-E_r z^{-r})}\sum_{x=0}^{r-1} z^{-N-1-x}\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+\frac{(-1)^k\sqrt{\frac{2}{N}}}{(1-E_r z^{-r})}\sum_{x=0}^{r-1} z^{-1-x}\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

Gathering terms in *S*(*z*,*k*) onto the left hand side,

$$[(1-z^{-r}E_r)^2 + F_r{}^2 z^{-2r}]S(z,k) =$$

$$\sqrt{\frac{2}{N}}F_r\sum_{x=0}^{r-1} z^{-N-1-x-r}\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$-(-1)^k\sqrt{\frac{2}{N}}F_r\sum_{x=0}^{r-1} z^{-1-x-r}\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} z^{-N-1-x}\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$-E_r\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} z^{-N-1-x-r}\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+(-1)^k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} z^{-1-x}\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$-(-1)^k E_r\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} z^{-1-r-x}\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

Taking inverse *z*-transform of the above equation results in:

$$S(n+1,k)+S(n-2r+1,k)-2E_rS(n-r+1,k) =$$

$$\sqrt{\frac{2}{N}}F_r\sum_{x=0}^{r-1} f(n-N-x-r)\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$-(-1)^k\sqrt{\frac{2}{N}}F_r\sum_{x=0}^{r-1} f(n-x-r)\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} f(n-N-x)\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$-E_r\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1} f(n-N-x-r)\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+(-1)^k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n-x)\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$-(-1)^k E_r\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n-x-r)\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

Therefore,

$$S(n+r,k)=2E_rS(n,k)-S(n-r,k)$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}\left[F_rf(n-N-x-1)+(-1)^k f(n+r-1-x)-(-1)^k E_rf(n-x-1)\right]$$

$$\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}\left[f(n-N+r-1-x)-(-1)^k F_rf(n-x-1)-E_rf(n-N-x-1)\right]$$

$$\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right] \qquad (2.46)$$

$$\text{for } k=0,1,...,N-1.$$

Equation (2.46) represents the algorithm to update DST type-IV independent of DCT coefficients. The C language implementation of the independent *r*-point update equation (2.46) for DST-IV is given in figure (A.18) of Appendix A.

2.6 Derivation of DCT/DST independent equations in time domain

Independent update algorithms derived for DCT/DST type-I through IV in this chapter can be derived in time domain without use of the *z*-transforms. This section includes the derivation for DCT-II and DST-II in time domain. This act as an alternative way of deriving the independent update equations for DCT's and DST's. The equations developed here are identical to independent equations (2.25) and (2.26) derived earlier in this chapter using the *z*-transform.

$C(n+r,k)$ is given by equation (2.19) i.e.:

$$C(n+r,k) = \cos\frac{rk\pi}{N}C(n,k) + \sin\frac{rk\pi}{N}S(n,k)$$

$$+\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{r-1}[(-1)^k f(n+r-1-x) - f(n-N+r-1-x)]\cos\frac{(2x+1)k\pi}{2N}.$$

Similarly calculating $C(n-r,k)$ from equation (2.17) yields:

$$\sqrt{\frac{N}{2}}C(n-r,k) = P_k\sum_{x=0}^{r-1} f(n-N+x-r)\cos\frac{(2x+1)k\pi}{2N}$$

$$+ P_k\sum_{x=r}^{N-1} f(n-N+x-r)\cos\frac{(2x+1)k\pi}{2N}. \qquad (2.47)$$

Substituting $i=x-r-N$ in the first term and $y=x-r$ in the second term results in:

$$\sqrt{\frac{N}{2}}C(n-r,k) = P_k\sum_{y=-r-N}^{-N-1} f(n+i)\cos\frac{(2(i+r+N)+1)k\pi}{2N}$$

$$+ P_k\sum_{y=0}^{N-r-1} f(n-N+y)\cos\frac{(2(y+r)+1)k\pi}{2N}.$$

$$= P_k\sum_{i=-r-N}^{-N-1} f(n+i)\cos\left[\frac{(2i+1)k\pi}{2N} + \frac{(r+N)k\pi}{N}\right]$$

$$+ P_k\sum_{y=0}^{N-r-1} f(n-N+y)\cos\left[\frac{(2y+1)k\pi}{2N} + \frac{rk\pi}{N}\right]$$

$$= P_k\sum_{i=-r-N}^{-N-1} f(n+i)\cos\frac{(2i+1)k\pi}{2N}\cos\frac{(r+N)k\pi}{N}$$

$$- P_k\sum_{i=-r-N}^{-N-1} f(n+i)\sin\frac{(2i+1)k\pi}{2N}\sin\frac{(r+N)k\pi}{N}$$

$$+ P_k\sum_{y=0}^{N-1-r} f(n-N+y)\cos\frac{(2y+1)k\pi}{2N}\cos\frac{rk\pi}{N}$$

$$- P_k \sum_{y=0}^{N-1-r} f(n-N+y) \sin\frac{(2y+1)k\pi}{2N} \sin\frac{rk\pi}{N}$$

$$= (-1)^k \cos\frac{rk\pi}{N} P_k \sum_{i=-r-N}^{-N-1} f(n+i) \cos\frac{(2i+1)k\pi}{2N}$$

$$- (-1)^k \sin\frac{rk\pi}{N} P_k \sum_{i=-r-N}^{-N-1} f(n+i) \sin\frac{(2i+1)k\pi}{2N}$$

$$+ P_k \cos\frac{rk\pi}{N} \sum_{y=0}^{N-1} f(n-N+y) \cos\frac{(2y+1)k\pi}{2N}$$

$$- P_k \cos\frac{rk\pi}{N} \sum_{y=N-r}^{N-1} f(n-N+y) \cos\frac{(2y+1)k\pi}{2N}$$

$$- P_k \sin\frac{rk\pi}{N} \sum_{y=0}^{N-1} f(n-N+y) \sin\frac{(2y+1)k\pi}{2N}$$

$$+ P_k \sin\frac{rk\pi}{N} \sum_{y=N-r}^{N-1} f(n-N+y) \sin\frac{(2y+1)k\pi}{2N}$$

Using the definition of the DCT and the DST as defined in equations (2.17) and (2.18) yields:

$$C(n-r,k) = \cos\frac{rk\pi}{N} C(n,k) - \sin\frac{rk\pi}{N} S(n,k)$$

$$+ (-1)^k \cos\frac{rk\pi}{N} P_k \sqrt{\frac{2}{N}} \sum_{i=-r-N}^{-N-1} f(n+i) \cos\frac{(2i+1)k\pi}{2N}$$

$$- (-1)^k \sin\frac{rk\pi}{N} P_k \sqrt{\frac{2}{N}} \sum_{y=-r-N}^{-N-1} f(n+i) \sin\frac{(2i+1)k\pi}{2N}$$

$$- P_k \cos\frac{rk\pi}{N} \sqrt{\frac{2}{N}} \sum_{y=N-r}^{N-1} f(n-N+y) \cos\frac{(2y+1)k\pi}{2N}$$

$$+ P_k \sin\frac{rk\pi}{N} \sqrt{\frac{2}{N}} \sum_{y=N-r}^{N-1} f(n-N+y) \sin\frac{(2y+1)k\pi}{2N}$$

Substituting i=-x-N-1 and y=-x+N-1 results in:

$$C(n-r,k)=\cos\frac{rk\pi}{N}C(n,k)-\sin\frac{rk\pi}{N}S(n,k)$$
$$+\cos\frac{rk\pi}{N}P_k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n-N-x-1)\cos\frac{(2x+1)k\pi}{2N}$$
$$+\sin\frac{rk\pi}{N}P_k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n-N-x-1)\sin\frac{(2x+1)k\pi}{2N}$$
$$-(-1)^kP_k\cos\frac{rk\pi}{N}\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n-x-1)\cos\frac{(2x+1)k\pi}{2N}$$
$$-(-1)^kP_k\sin\frac{rk\pi}{N}\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}f(n-x-1)\sin\frac{(2x+1)k\pi}{2N} \quad (2.48)$$

Adding equations (2.19) and (2.48) results in cancellation of the DST coefficients and yields the DST independent update equation for DCT-II.

$$C(n+r,k)=2\cos\frac{rk\pi}{N}C(n,k)-C(n-r,k)$$
$$+\sqrt{\frac{2}{N}}P_k\sin\frac{rk\pi}{N}\sum_{x=0}^{r-1}[f(n-N-x-1)-(-1)^kf(n-x-1)]\sin\frac{(2x+1)k\pi}{2N}$$
$$+\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{r-1}[(-1)^kf(n+r-x-1)-f(n-N+r-x-1)]\cos\frac{(2x+1)k\pi}{2N}$$
$$-\sqrt{\frac{2}{N}}P_k\cos\frac{rk\pi}{N}\sum_{x=0}^{r-1}[(-1)^kf(n-x-1)-f(n-N-x-1)]\cos\frac{(2x+1)k\pi}{2N}. \quad (2.49)$$

for $k=0,1,...,N-1$.

Similarly the r-point update equation for the DST may be derived in time domain resulting in:

$$S(n+r,k)=2\cos\frac{rk\pi}{N}S(n,k)-S(n-r,k)$$

$$+\sqrt{\frac{2}{N}}P_k \sin\frac{rk\pi}{N}\sum_{x=0}^{r-1}[f(n-N-x-1)-(-1)^k f(n-x-1)]\cos\frac{(2x+1)k\pi}{2N}$$

$$+\sqrt{\frac{2}{N}}P_k \sum_{x=0}^{r-1}[f(n-N+r-x-1)-(-1)^k f(n+r-x-1)]\sin\frac{(2x+1)k\pi}{2N}$$

$$-\sqrt{\frac{2}{N}}P_k \cos\frac{rk\pi}{N}\sum_{x=0}^{r-1}[f(n-N-x-1)-(-1)^k f(n-x-1)]\sin\frac{(2x+1)k\pi}{2N}. \qquad (2.50)$$

$$\text{for } k = 1,\ldots,N.$$

Analyzing equations (2.49) and (2.50) shows that they are identical to equations (2.25) and (2.26).

CHAPTER 3: DCT/DST TRAPEZOIDAL WINDOWED INDEPENDENT UPDATE ALGORITHMS

3.1 Introduction to independent DCT and DST update algorithms for trapezoidal window

Update algorithms in the presence of trapezoidal or split-triangular window were initially derived by Sherlock and Kakad [22], however these algorithms involved simultaneous update of the DCT and DST coefficients for windowed update data. In this chapter, the algorithms initially derived in [22] are extended to perform independent update of DCT and DST. The independent algorithms are derived for DCT and DST type-I through IV in the presence of split-triangular window. The windowed DCT independent update algorithms do not require the DST coefficients and similarly the windowed DST independent update algorithms do not require the DCT coefficients. The infinite input sequence is windowed with the split-triangular window of figure (3.1). Split-triangular window is preferred over the rectangular window used in chapter 2 because it reduces the edge effects that occur due to the use of rectangular window. The independent DCT and DST split-triangular windowed update of the shifted sequence can be calculated using the algorithms developed in this chapter without using the definition of the transforms. The computational complexity to calculate the split-triangular windowed update of shifted sequence is much less as compared to the computation by the transform definition. Section 3.2 lists the explanation and simultaneous update algorithms developed by Sherlock and Kakad [22], these algorithms are listed here for completeness and understanding of the independent split-triangular windowed update algorithms. In

Section 3.3 independent DCT/DST type-I trapezoidal window update algorithms are developed and sections 3.4, 3.5 and 3.6 focuses on DCT/DST type-II, III and IV algorithms. Appendix A includes the C language implementation of the independent trapezoidal windowed update algorithms for DCT/DST type-I through IV. The algorithms derived here are capable of updating one point at a time i.e. r=1, however the algorithms can be repeated if more than one point update is required.

3.2 DCT/DST split-triangular windowed simultaneous update algorithms

This section lists the simultaneous update equations for split-triangular windowed update as derived by [22, 23]. The windowed update algorithms in this section are given for DCT/DST type-II as it is the most often used transform. For the input signal $f(x)$, x=0,.., N-1, and the split-triangular window $w(x)$ of length N with tail-length n_0 given by equation (3.1), the windowed data is given by $f_w(x) = f(x)w(x)$.

$$w(x) = \begin{cases} x/n_0 & if\ x = 0,...,n_0 \\ 1 & if\ x = n_0+1,...,N-n_0 \\ w(N-x) & if\ x = N-n_0+1,...,N-1 \end{cases} \tag{3.1}$$

When the new data point $f(N)$ is available, $f(0)$ is shifted out and $f(N)$ data point is shifted in. The updated sequence is represented by $f(x+1)$ and the shifted windowed data is given by:

$$f_{w(new)}(x) = f(x+1)w(x). \tag{3.2}$$

which can we rewritten as:

$$f_{w(new)}(x) = f(x+1)\big[w(x) + w(x+1) - w(x+1)\big]$$

$$= f(x+1)w(x+1) + f(x+1)\big[w(x) - w(x+1)\big]$$

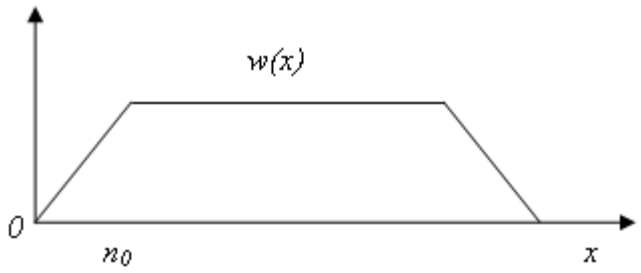

Figure 3.1 Split-triangular window $w(x)$ with tail length n_0

Defining $m(x) = w(x) - w(x+1)$, see figure (3.1), above equation can be written as:

$$f_{w(new)}(x) = f(x+1)w(x+1) + f(x+1)m(x)$$

Therefore,

$$f_{w(new)}(x) = f_w(x+1) + f_{m(new)}(x) \quad \text{for } x=0,\ldots,N-1, \tag{3.3}$$

where, $f_{m(new)}(x) = f(x+1)m(x)$ and $f_w(x+1) = f(x+1)w(x+1)$.

$f_{m(new)}(x)$ can be rewritten as:

$$f_{m(new)}(x) = f(x+1)\left[m(x+1) - m(x+1) + m(x)\right]$$

i.e.

$$= f(x+1)m(x+1) + f(x+1)\left[m(x) - m(x+1)\right] \tag{3.4}$$

Now,

$$m(x) - m(x+1) = \begin{cases} -1/n_0 & \text{if } x = n_0 - 1, \\ -1/n_0 & \text{if } x = N - n_0 - 1, \\ 2/n_0 & \text{if } x = N - 1, \\ 0 & \text{all other } x \text{ in } 0,\ldots, N-1, \end{cases}$$

$$= -\frac{1}{n_0}\delta_{x,n0-1} - \frac{1}{n_0}\delta_{x,N-n0-1} + \frac{2}{n_0}\delta_{x,N-1}, \tag{3.5}$$

Substituting equation (3.5) in equation (3.4) results in:

$$f_{m(new)}(x) = f_m(x+1) + f(x+1)\left[-\frac{1}{n_0}\delta_{x,n0-1} - \frac{1}{n_0}\delta_{x,N-n0-1} + \frac{2}{n_0}\delta_{x,N-1}\right]$$

where, $f_m(x+1) = f(x+1)m(x+1)$

$$f_{m(new)}(x) = f_m(x+1) + \frac{1}{n_0}\left[-f(n_0)\delta_{x,n0-1} - f(N-n_0)\delta_{x,N-n0-1} + 2f(N)\delta_{x,N-1}\right] \tag{3.6}$$

Equations (3.3) and (3.6) represent the windowed update version of $f_w(x)$ and $f_m(x)$ respectively for moving DCT/DST for trapezoidal window. In equation (3.3), $f_w(x+1)$ represents non-windowed update of $f_w(x)$ and the second term $f_{m(new)}(x)$ is a correction factor that converts this non-windowed update of $f_w(x)$ into an update in the presence of the window. Similarly in equation (3.6), $f_m(x+1)$ represents non-windowed update of $f_m(x)$ and the second term converts this into the update in the presence of the window.

Taking DCT-II of equation (3.3) and equation (3.6) yields:

$$C_{w(new)}(x) = C_w(x+1) + C_{m(new)}(x) \tag{3.7}$$

$$C_{m(new)}(x) = C_m(x+1)$$

$$+\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{N-1}\frac{1}{n_0}\left[-f(n_0)\delta_{x,n0-1} - f(N-n_0)\delta_{x,N-n0-1} + 2f(N)\delta_{x,N-1}\right]\cos\frac{(2x+1)k\pi}{2N}$$

Solving the above equation yields:

$$C_{m(new)}(x) = C_m(x+1) + \sqrt{\frac{2}{N}}\frac{P_k}{n_0}\left\{-f(n_0)\cos\frac{[2(n_0-1)+1]k\pi}{2N}\right.$$

$$\left. - f(N-n_0)\cos\frac{[2(N-n_0-1)+1]k\pi}{2N} + 2f(N)\cos\frac{[2(N-1)+1]k\pi}{2N}\right\}$$

Therefore,

$$C_{m(new)}(x) = C_m(x+1) + \sqrt{\frac{2}{N}}\frac{P_k}{n_0}\left\{-f(n_0)\cos\frac{(2n_0-1)k\pi}{2N}\right.$$

$$\left.-(-1)^k f(N-n_0)\cos\frac{(2n_0+1)k\pi}{2N} + 2f(N)(-1)^k\cos\frac{k\pi}{2N}\right\} \qquad (3.8)$$

for k=0,1,…,N-1.

Equations (3.7) and (3.8) can be used to calculate the simultaneous update of the moving DCT for trapezoidal window. $C_w(x+1)$ is the non-windowed DCT update of $f_w(x)$ calculated using equation (2.1) listed below for convenience, and $C_m(x+1)$ is the non-windowed DCT update of $f_m(x)$ calculated using equation (2.1) listed below. Clearly, it can be seen that while performing the windowed DCT update, both the coefficients of DCT and DST are required.

$$C_+(k) = \cos\frac{rk\pi}{N}C(k) + \sin\frac{rk\pi}{N}S(k)$$

$$\left. + P_k\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}[(-1)^k f(N+r-1-x) - f(r-1-x)]\cos\frac{(2x+1)k\pi}{2N}\right].$$

for $k = 0,1,...,N-1$.

Similarly the DST update equation may be derived and is:

$$S_{w(new)}(x) = S_w(x+1) + S_{m(new)}(x) \qquad (3.9)$$

$$S_{m(new)}(x) = S_m(x+1) + \sqrt{\frac{2}{N}}\frac{P_k}{n_0}\left\{-f(n_0)\sin\frac{(2n_0-1)k\pi}{2N}\right.$$

$$\left.+(-1)^k f(N-n_0)\sin\frac{(2n_0+1)k\pi}{2N} - 2f(N)(-1)^k\sin\frac{k\pi}{2N}\right\} \qquad (3.10)$$

for k=1,…, N.

Equations (3.9) and (3.10) can be used to calculate the simultaneous update of the moving DST for trapezoidal window. $S_w(x+1)$ is the non-windowed DST update of $f_w(x)$ calculated using equation (2.2) listed below for convenience, and $S_m(x+1)$ is the non-windowed updated DST of $f_m(x)$ calculated using equation (2.2). Clearly, it can be seen that while performing the windowed DST update both the coefficients of DST and DCT are required.

$$S_+(k) = \cos\frac{rk\pi}{N} S(k) - \sin\frac{rk\pi}{N} C(k)$$

$$- P_k \sqrt{\frac{2}{N}} \sum_{x=0}^{r-1} [(-1)^k f(N+r-1-x) - f(r-1-x)] \sin\frac{(2x+1)k\pi}{2N} \Bigg].$$

$$\text{for } k = 1,2,\ldots,N.$$

where, $S_+(k)$ represents the updated DST coefficients.

In the following sections algorithms are developed that are capable of performing independent moving update of DCT and DST for trapezoidal window.

3.3 Independent DCT/DST type-I split-triangular windowed update algorithms for moving data

The algorithm to calculate $C_{w(new)}$ and $S_{w(new)}$ the windowed DCT-I and DST-I update in the presence of trapezoidal window are derived next and in section 3.3.2 the method to pre-process the data required to update the DCT-I and DST-I used in Section 3.3.1 is derived.

3.3.1. Computation of $C_{w(new)}$ and $S_{w(new)}$

For calculating the independent DCT-I update in the presence of split-triangular window we take DCT-I of equations (3.3) and (3.6).

$$C_{w(new)}(x) = C_w(x+1) + C_{m(new)}(x) \tag{3.11}$$

$$C_{m(new)}(x) = C_m(x+1)$$

$$+\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{N}\frac{1}{n_0}\left[-f(n_0)\delta_{x,n0-1} - f(N-n_0)\delta_{x,N-n0-1} + 2f(N)\delta_{x,N-1}\right]P_x\cos\frac{xk\pi}{N}$$

Solving the above equation yields:

$$C_{m(new)}(x) = C_m(x+1) + \sqrt{\frac{2}{N}}\frac{P_k}{n_0}\left\{-f(n_0)\cos\frac{(n_0-1)k\pi}{N}P_{n_0-1}\right.$$

$$\left. - f(N-n_0)\cos\frac{(N-n_0-1)k\pi}{N} + 2f(N)\cos\frac{(N-1)k\pi}{N}\right\}$$

Therefore for n_0=1:

$$C_{m(new)}(x) = C_m(x+1) + \sqrt{\frac{2}{N}}\frac{P_k}{n_0}\left\{-f(n_0)\frac{1}{\sqrt{2}}\cos\frac{(n_0-1)k\pi}{N}\right.$$

$$\left. - (-1)^k f(N-n_0)\cos\frac{(n_0+1)k\pi}{N} + 2f(N)(-1)^k\cos\frac{k\pi}{N}\right\} \qquad (3.12a)$$

and for $n_0 \neq 1$,

$$C_{m(new)}(x) = C_m(x+1) + \sqrt{\frac{2}{N}}\frac{P_k}{n_0}\left\{-f(n_0)\cos\frac{(n_0-1)k\pi}{N}\right.$$

$$\left. - (-1)^k f(N-n_0)\cos\frac{(n_0+1)k\pi}{N} + 2f(N)(-1)^k\cos\frac{k\pi}{N}\right\} \qquad (3.12b)$$

for k=0,1,…,N.

Equations (3.11), (3.12a) and (3.12b) can be used to calculate the independent update of the moving DCT-I for trapezoidal window. $C_w(x+1)$ is the non-windowed DCT-I update of $f_w(x)$ calculated using equation (2.13) listed below for convenience, and $C_m(x+1)$ is the non-windowed DCT-I update of $f_m(x)$ also calculated using equation (2.13).

$$C(n+r,k) = 2A_r C(n,k) - (A_r{}^2 + P_k B_r{}^2)C(n-r,k)$$

$$+ \sqrt{\frac{2}{N}} P_k B_r \sum_{x=0}^{r-1} [f(n-N-x-1) - (-1)^k f(n-x-1)] \sin\frac{(x+1)k\pi}{N}$$

$$+ P_k \sqrt{\frac{2}{N}} \sum_{x=0}^{r-1} [(-1)^k f(n+r-1-x) - f(n-N+r-1-x) + A_r f(n-N-x-1)$$

$$- A_r (-1)^k f(n-x-1)] \cos\frac{(x+1)k\pi}{N}$$

$$+ \sqrt{\frac{2}{N}} \left(\frac{1}{\sqrt{2}} - 1\right) P_k f(n-N-r) + (-1)^k \sqrt{\frac{2}{N}} \frac{1}{\sqrt{2}} P_k f(n-r)$$

$$+ 2\sqrt{\frac{2}{N}} \left(1 - \frac{1}{\sqrt{2}}\right) P_k A_r f(n-N) - (-1)^k \sqrt{\frac{2}{N}} \sqrt{2} P_k A_r f(n)$$

$$+ \sqrt{\frac{2}{N}} \left(\frac{1}{\sqrt{2}} - 1\right) P_k f(n-N+r) + (-1)^k \sqrt{\frac{2}{N}} \frac{1}{\sqrt{2}} P_k f(n+r)$$

$$\text{for } k = 0,1,\ldots,N.$$

When using the above equation to calculate the non-windowed update we need the current value $C(n,k)$ and the previous value $C(n\text{-}1,k)$. In the case of C_w, the current value is $C_{[f(x)w(x)]}$ and the previous value is $C_{[f(x-1)w(x-1)]}$. However, $C_{[f(x-1)w(x-1)]}$ is not yet available, we instead have $C_{[f(x-1)w(x)]}$ from the previous time-step. Therefore, we need to derive the correction factor that calculates the correct value $C_{[f(x-1)w(x-1)]}$ from $C_{[f(x-1)w(x)]}$. Similarly for C_m, we need to calculate the correction factor to compute $C_{[f(x-1)m(x-1)]}$ from $C_{[f(x-1)m(x)]}$. In section 3.3.2 we derive the formula to calculate the correct values $C_{[f(x-1)w(x-1)]}$ and $C_{[f(x-1)m(x-1)]}$. Appendix A, figure (A.2) lists the C language implementation of DST independent algorithm to implement the DCT-I update in the presence of split-triangular window for moving data.

Similarly, the analogous formulae for DST-I can be derived. Taking the DST-I of equations (3.3) and (3.6):

$$S_{w(new)}(x) = S_w(x+1) + S_{m(new)}(x) \tag{3.13}$$

$$S_{m(new)}(x) = S_m(x+1)$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{N}\frac{1}{n_0}\left[-f(n_0)\delta_{x,n0-1} - f(N-n_0)\delta_{x,N-n0-1} + 2f(N)\delta_{x,N-1}\right]\sin\frac{xk\pi}{N}$$

Solving the above equation yields:

$$S_{m(new)}(x) = S_m(x+1) + \sqrt{\frac{2}{N}}\frac{1}{n_0}\left\{-f(n_0)\sin\frac{(n_0-1)k\pi}{N}\right.$$

$$\left. - f(N-n_0)\sin\frac{(N-n_0-1)k\pi}{N} + 2f(N)\sin\frac{(N-1)k\pi}{N}\right\}$$

$$S_{m(new)}(x) = S_m(x+1) + \sqrt{\frac{2}{N}}\frac{1}{n_0}\left\{-f(n_0)\sin\frac{(n_0-1)k\pi}{N}\right.$$

$$\left. + (-1)^k f(N-n_0)\sin\frac{(n_0+1)k\pi}{N} - 2f(N)(-1)^k\sin\frac{k\pi}{N}\right\} \tag{3.14}$$

for k=1,…,N-1.

Equations (3.13) and (3.14) can be used to calculate the simultaneous update of the moving DST for trapezoidal window. $S_w(x+1)$ is the non-windowed DST-I update of $f_w(x)$ calculated using equation (2.14) listed below for convenience, and $S_m(x+1)$ is the non-windowed updated DST of $f_m(x)$ also calculated using equation (2.14).

$$S(n+r,k) = 2A_k S(n,k) - (A_r^{\,2} + P_k B_r^{\,2})S(n-r,k)$$

$$+\sqrt{\frac{2}{N}}P_k B_r\sum_{x=0}^{r-1}[f(n-1-N-x) - (-1)^k f(n-1-x)]\cos\frac{(x+1)k\pi}{N}$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}[f(n+r-1-N-x)-(-1)^k f(n+r-1-x)+A_r(-1)^k f(n-1-x)$$

$$-A_r f(n-1-N-x)]\sin\frac{(x+1)k\pi}{N}$$

$$+\sqrt{\frac{2}{N}}\left(1-\frac{1}{\sqrt{2}}\right)A_r B_r(1-P_k)f(n-N-r)+(-1)^k\sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}A_r B_r(P_k-1)f(n-r)$$

$$+\sqrt{\frac{2}{N}}\left(1-\frac{1}{\sqrt{2}}\right)(P_k-1)B_r f(n-N)-(-1)^k\sqrt{\frac{2}{N}}(P_k-1)\frac{1}{\sqrt{2}}B_r f(n)$$

$$\text{for } k=1,\ldots,N-1.$$

When using the above equation to calculate the non-windowed update we need the current value $S(n,k)$ and the previous value $S(n\text{-}1,k)$. In the case of S_w, the current value is $S_{[f(x)w(x)]}$ and the previous value is $S_{[f(x-1)w(x-1)]}$. However, $S_{[f(x-1)w(x-1)]}$ is not yet available, we instead have $S_{[f(x-1)w(x)]}$ from the previous time-step. Therefore, we need to derive the correction factor that calculates the correct value $S_{[f(x-1)w(x-1)]}$ from $S_{[f(x-1)w(x)]}$. Similarly for S_m, we need to calculate the correction factor to compute $S_{[f(x-1)m(x-1)]}$ from $S_{[f(x-1)m(x)]}$. In section 3.3.2 we derive the formula to calculate the correct values $S_{[f(x-1)w(x-1)]}$ and $S_{[f(x-1)m(x-1)]}$. Appendix A, figure (A.4) shows the C language implementation of DCT independent algorithm to implement the DST-I update in the presence of split-triangular window for moving data.

3.3.2 Derivation of correction values for oldest time-step

This section derives the correction factor to calculate the correct value $C_{[f(x-1)w(x-1)]}$ from $C_{[f(x-1)w(x)]}$ for DCT-I update algorithm and the correct value of $S_{[f(x-1)w(x-1)]}$ from $S_{[f(x-1)w(x)]}$ for the DST update algorithm.

$$f(x-1)w(x)=f(x-1)\big[w(x)+w(x-1)-w(x-1)\big]$$

$$= f(x-1)w(x-1) - f(x-1)\left[w(x-1) - w(x)\right]$$

$$= f(x-1)w(x-1) - f(x-1)m(x-1),$$

Therefore,

$$f(x-1)w(x-1) = f(x-1)w(x) + f(x-1)m(x-1). \tag{3.15}$$

Similarly, calculating the correction factor to convert $f(x\text{-}1)m(x)$ into the correct value $f(x\text{-}1)m(x\text{-}1)$,

$$f(x-1)m(x) = f(x-1)\left[m(x) + m(x-1) - m(x-1)\right]$$

$$= f(x-1)m(x-1) - f(x-1)\left[m(x-1) - m(x)\right]$$

$$= f(x-1)m(x-1) - f(x-1)m_p(x-1),$$

Therefore,

$$f(x-1)m(x-1) = f(x-1)m(x) + f(x-1)m_p(x-1). \tag{3.16}$$

where,

$$m_p(x) = m(x) - m(x+1) = \begin{cases} -1/n_0 & \text{if } x = n_0\text{-}1, \\ -1/n_0 & \text{if } x = N\text{-}n_0\text{-}1, \\ 2/n_0 & \text{if } x = N\text{-}1, \\ 0 & \text{all other } x \text{ in } 0,\ldots,N\text{-}1 \end{cases}$$

Substituting the value of $m_p(x\text{-}1)$ in equation (3.13) yields:

$$f(x-1)m(x-1) = f(x-1)m(x) + f(x-1)\left[-\frac{1}{n_0}\delta_{x,n0} - \frac{1}{n_0}\delta_{x,N-n0} + \frac{2}{n_0}\delta_{x,0}\right]$$

i.e.

$$f_m(x-1) = f(x-1)m(x) + \frac{1}{n_0}\left[-f(n_0-1)\delta_{x,n0} - f(N-n_0-1)\delta_{x,N-n0} + 2f(-1)\delta_{x,0}\right] \tag{3.17}$$

Taking DCT-I of equation (3.17) and simplifying:

$$C_m old(k) \equiv C_{[f(x-1)m(x-1)]} = C_{[f(x-1)m(x)]}$$

$$+\sqrt{\frac{2}{N}}P_k \sum_{x=0}^{N-1}\frac{1}{n_0}\Big[-f(n_0-1)\delta_{x,n0} - f(N-n_0-1)\delta_{x,N-n0} + \sqrt{2}f(-1)\delta_{x,0}\Big]\cos\frac{xk\pi}{N}$$

for $k=0,1,\ldots,N$.

Therefore,

$$= C_{[f(x-1)m(x)]} + \sqrt{\frac{2}{N}}\frac{P_k}{n_0}\Big\{-f(n_0-1)\cos\frac{n_0 k\pi}{N}$$

$$-f(N-n_0-1)\cos\frac{(N-n_0)k\pi}{N} + \sqrt{2}f(-1)\Big\}$$

$$C_m old(k) \equiv C_{[f(x-1)m(x-1)]} = C_{[f(x-1)m(x)]} + \sqrt{\frac{2}{N}}\frac{P_k}{n_0}\Big\{-f(n_0-1)\cos\frac{n_0 k\pi}{N}$$

$$-(-1)^k f(N-n_0-1)\cos\frac{n_0 k\pi}{N} + \sqrt{2}f(-1)\Big\} \qquad (3.18)$$

Taking the DCT of equation (3.15) yields:

$$C_{[f(x-1)w(x-1)]} = C_{[f(x-1)w(x)]} + C_{[f(x-1)m(x-1)]}. \qquad (3.19)$$

Equation (3.18) and (3.19) together are used to calculate the previous time sequence windowed DCT-I values together with equations (3.11), (3.12) and (2.13) can be used to calculate DCT-I independent windowed update in presence of the split-triangular window.

Similarly, taking the DST-I of equation (3.17) yields:

$$S_m old(k) \equiv S_{[f(x-1)m(x-1)]} = S_{[f(x-1)m(x)]}$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}\frac{1}{n_0}\Big[-f(n_0-1)\delta_{x,n0} - f(N-n_0-1)\delta_{x,N-n0} + \sqrt{2}f(-1)\delta_{x,0}\Big]\sin\frac{xk\pi}{N}$$

for $k=1,\ldots,N-1$,

Therefore,

$$= S_{[f(x-1)m(x)]} + \sqrt{\frac{2}{N}}\frac{1}{n_0}\left\{- f(n_0-1)\sin\frac{n_0 k\pi}{N} - f(N-n_0-1)\sin\frac{(N-n_0)k\pi}{N}\right\}$$

$$S_m old(k) \equiv S_{[f(x-1)m(x-1)]} = S_{[f(x-1)m(x)]} + \sqrt{\frac{2}{N}}\frac{1}{n_0}\left\{- f(n_0-1)\sin\frac{n_0 k\pi}{N}\right.$$

$$\left. + (-1)^k f(N-n_0-1)\sin\frac{n_0 k\pi}{N}\right\} \tag{3.20}$$

Taking the DST-I of equation (3.15) yields:

$$S_w old \equiv S_{[f(x-1)w(x-1)]} = S_{[f(x-1)w(x)]} + S_{[f(x-1)m(x-1)]}. \tag{3.21}$$

Equation (3.20) and (3.21) together are used to calculate the previous time sequence windowed DST values.

3.4 Independent DCT/DST type-II split-triangular windowed update algorithms for moving data

In section 3.4.1 algorithms to calculate $C_{w(new)}$ and $S_{w(new)}$ are developed for the windowed DCT-II and DST-II update in the presence of trapezoidal window and in section 3.4.2 the method to pre-process the data required to update the DCT-II and DST-II used in section 3.4.1 is presented.

3.4.1. Computation of $C_{w(new)}$ and $S_{w(new)}$

For calculating the independent DCT-II update in the presence of split-triangular window we take DCT-II of equations (3.3) and (3.6):

$$C_{w(new)}(x) = C_w(x+1) + C_{m(new)}(x) \tag{3.22}$$

$$C_{m(new)}(x) = C_m(x+1)$$

$$+ \sqrt{\frac{2}{N}} P_k \sum_{x=0}^{N-1}\frac{1}{n_0}\left[- f(n_0)\delta_{x,n0-1} - f(N-n_0)\delta_{x,N-n0-1} + 2f(N)\delta_{x,N-1}\right]\cos\frac{(2x+1)k\pi}{2N}$$

Solving the above equation yields:

$$C_{m(new)}(x) = C_m(x+1) + \sqrt{\frac{2}{N}}\frac{P_k}{n_0}\left\{ - f(n_0)\cos\frac{[2(n_0-1)+1]k\pi}{2N} \right.$$

$$\left. - f(N-n_0)\cos\frac{[2(N-n_0-1)+1]k\pi}{2N} + 2f(N)\cos\frac{[2(N-1)+1]k\pi}{2N} \right\}$$

Therefore,

$$C_{m(new)}(x) = C_m(x+1) + \sqrt{\frac{2}{N}}\frac{P_k}{n_0}\left\{ - f(n_0)\cos\frac{(2n_0-1)k\pi}{2N} \right.$$

$$\left. - (-1)^k f(N-n_0)\cos\frac{(2n_0+1)k\pi}{2N} + 2f(N)(-1)^k \cos\frac{k\pi}{2N} \right\} \qquad (3.23)$$

for k=0,1,…,N-1.

Equations (3.22) and (3.23) can be used to calculate the independent update of the moving DCT-II for trapezoidal window. $C_w(x+1)$ is the non-windowed DCT-II update of $f_w(x)$, calculated using equation (2.25) listed below for convenience, and $C_m(x+1)$ is the non-windowed DCT-II update of $f_m(x)$ also calculated using equation (2.25).

$$C(n+r,k) = 2\cos\frac{rk\pi}{N}C(n,k) - C(n-r,k)$$

$$+\sqrt{\frac{2}{N}}P_k \sin\frac{rk\pi}{N}\sum_{x=0}^{r-1}[f(n-N-x-1) - (-1)^k f(n-x-1)]\sin\frac{(2x+1)k\pi}{2N}$$

$$+\sqrt{\frac{2}{N}}P_k \sum_{x=0}^{r-1}[(-1)^k f(n+r-x-1) - f(n+r-N-x-1)]\cos\frac{(2x+1)k\pi}{2N}$$

$$-\sqrt{\frac{2}{N}}P_k \cos\frac{rk\pi}{N}\sum_{x=0}^{r-1}[(-1)^k f(n-x-1) - f(n-N-x-1)]\cos\frac{(2x+1)k\pi}{2N}$$

for $k = 0,1,...,N-1$.

When using the above equation to calculate the non-windowed update we need the current value $C(n,k)$ and the previous value $C(n\text{-}1,k)$. In the case of C_w, the current value is $C_{[f(x)w(x)]}$ and the previous value is $C_{[f(x-1)w(x-1)]}$. However, $C_{[f(x-1)w(x-1)]}$ is not yet available, we instead have $C_{[f(x-1)w(x)]}$ from the previous time-step. Therefore, we need to derive the correction factor that calculates the correct value $C_{[f(x-1)w(x-1)]}$ from $C_{[f(x-1)w(x)]}$. Similarly for C_m, we need to calculate the correction factor to compute $C_{[f(x-1)m(x-1)]}$ from $C_{[f(x-1)m(x)]}$. In section 3.4.2 the formula to calculate the correct values $C_{[f(x-1)w(x-1)]}$ and $C_{[f(x-1)m(x-1)]}$ are developed. Appendix A, figure (A.6) lists the C language implementation of DST independent algorithm to implement the DCT-II update in the presence of split-triangular window for moving data.

Similarly the analogous formulae for DST-II are obtained by taking DST-II of equations (3.3) and (3.6):

$$S_{w(new)}(x) = S_w(x+1) + S_{m(new)}(x) \tag{3.24}$$

$$S_{m(new)}(x) = S_m(x+1)$$

$$+\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{N-1}\frac{1}{n_0}\left[-f(n_0)\delta_{x,n0-1} - f(N-n_0)\delta_{x,N-n0-1} + 2f(N)\delta_{x,N-1}\right]\sin\frac{(2x+1)k\pi}{2N}$$

Solving the above equation yields:

$$S_{m(new)}(x) = S_m(x+1) + \sqrt{\frac{2}{N}}\frac{P_k}{n_0}\left\{-f(n_0)\sin\frac{[2(n_0-1)+1]k\pi}{2N}\right.$$

$$\left. - f(N-n_0)\sin\frac{[2(N-n_0-1)+1]k\pi}{2N} + 2f(N)\sin\frac{[2(N-1)+1]k\pi}{2N}\right\}$$

Therefore,

$$S_{m(new)}(x) = S_m(x+1) + \sqrt{\frac{2}{N}}\frac{P_k}{n_0}\left\{-f(n_0)\sin\frac{(2n_0-1)k\pi}{2N}\right.$$

$$+(-1)^k f(N-n_0)\sin\frac{(2n_0+1)k\pi}{2N}-2f(N)(-1)^k\sin\frac{k\pi}{2N}\Bigg\} \qquad (3.25)$$

for k=1,…,N.

Equations (3.24) and (3.25) can be used to calculate the independent update of the moving DST-II for trapezoidal window. $S_w(x+1)$ is the non-windowed DST-II update of $f_w(x)$, calculated using equation (2.26) listed below for convenience, and $S_m(x+1)$ is the non-windowed DST-II update of $f_m(x)$ also calculated using equation (2.26).

$$S(n+r,k)=2\cos\frac{rk\pi}{N}S(n,k)-S(n-r,k)$$

$$+\sqrt{\frac{2}{N}}P_k\sin\frac{rk\pi}{N}\sum_{x=0}^{r-1}[f(n-N-x-1)-(-1)^k f(n-x-1)]\cos\frac{(2x+1)k\pi}{2N}$$

$$+\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{r-1}[f(n+r-N-x-1)-(-1)^k f(n+r-x-1)]\sin\frac{(2x+1)k\pi}{2N}$$

$$-\sqrt{\frac{2}{N}}P_k\cos\frac{rk\pi}{N}\sum_{x=0}^{r-1}[f(n-N-x-1)-(-1)^k f(n-x-1)]\sin\frac{(2x+1)k\pi}{2N}.$$

for $k=1,...,N$.

When using the above equation to calculate the non-windowed update we need the current value $S(n,k)$ and the previous value $S(n\text{-}1,k)$. In the case of S_w, the current value is $S_{[f(x)w(x)]}$ and the previous value is $S_{[f(x-1)w(x-1)]}$. However, $S_{[f(x-1)w(x-1)]}$ is not yet available, we instead have $S_{[f(x-1)w(x)]}$ from the previous time-step. Therefore, we need to derive the correction factor that calculates the correct value $S_{[f(x-1)w(x-1)]}$ from $S_{[f(x-1)w(x)]}$. Similarly for S_m, we need to calculate the correction factor to compute $S_{[f(x-1)m(x-1)]}$ from $S_{[f(x-1)m(x)]}$. In section 3.4.2, the formula to calculate the correct values $S_{[f(x-1)w(x-1)]}$ and $S_{[f(x-1)m(x-1)]}$ are developed. Appendix A, figure (A.9) includes the C language implementation

of DCT independent algorithm to implement the DST update in the presence of split-triangular window for moving data.

3.4.2 Derivation of correction values for oldest time-step

The correction factor to calculate the correct value $C_{[f(x-1)w(x-1)]}$ from $C_{[f(x-1)w(x)]}$ for DCT update algorithm, and the correct value of $S_{[f(x-1)w(x-1)]}$ from $S_{[f(x-1)w(x)]}$ are derived here for the DST-II update algorithm.

Taking DCT-II of equation (3.17) and simplifying:

$$C_m old(k) \equiv C_{[f(x-1)m(x-1)]} = C_{[f(x-1)m(x)]}$$

$$+\sqrt{\frac{2}{N}}P_k \sum_{x=0}^{N-1}\frac{1}{n_0}\left[-f(n_0-1)\delta_{x,n0} - f(N-n_0-1)\delta_{x,N-n0} + 2f(-1)\delta_{x,0}\right]\cos\frac{(2x+1)k\pi}{2N}$$

for k=0,1,…,N-1.

Therefore,

$$= C_{[f(x-1)m(x)]} + \sqrt{\frac{2}{N}}\frac{P_k}{n_0}\left\{-f(n_0-1)\cos\frac{2(n_0+1)k\pi}{2N}\right.$$

$$\left. - f(N-n_0-1)\cos\frac{[2(N-n_0)+1]k\pi}{2N} + 2f(-1)\cos\frac{k\pi}{2N}\right\}$$

$$C_m old(k) \equiv C_{[f(x-1)m(x-1)]} = C_{[f(x-1)m(x)]} + \sqrt{\frac{2}{N}}\frac{P_k}{n_0}\left\{-f(n_0-1)\cos\frac{(2n_0+1)k\pi}{2N}\right.$$

$$\left. -(-1)^k f(N-n_0-1)\cos\frac{(2n_0-1)k\pi}{2N} + 2f(-1)\cos\frac{k\pi}{2N}\right\} \qquad (3.26)$$

Taking the DCT-II of equation (3.15) yields:

$$C_{[f(x-1)w(x-1)]} = C_{[f(x-1)w(x)]} + C_{[f(x-1)m(x-1)]}. \qquad (3.27)$$

Equations (3.26) and (3.27) together can be used to calculate the older time sequence windowed DCT-II values.

Similarly, taking the DST-II of equation (3.17) yields:

$$S_m old(k) \equiv S_{[f(x-1)m(x-1)]} = S_{[f(x-1)m(x)]}$$

$$+\sqrt{\frac{2}{N}}P_k \sum_{x=0}^{N-1} \frac{1}{n_0}\left[-f(n_0-1)\delta_{x,n0} - f(N-n_0-1)\delta_{x,N-n0} + 2f(-1)\delta_{x,0}\right]\sin\frac{(2x+1)k\pi}{2N}$$

Therefore,

$$= S_{[f(x-1)m(x)]} + \sqrt{\frac{2}{N}}\frac{P_k}{n_0}\left\{-f(n_0-1)\sin\frac{2(n_0+1)k\pi}{2N}\right.$$

$$\left. - f(N-n_0-1)\sin\frac{[2(N-n_0)+1]k\pi}{2N} + 2f(-1)\sin\frac{k\pi}{2N}\right\}$$

$$S_m old(k) \equiv S_{[f(x-1)m(x-1)]} = S_{[f(x-1)m(x)]} + \sqrt{\frac{2}{N}}\frac{P_k}{n_0}\left\{-f(n_0-1)\sin\frac{(2n_0+1)k\pi}{2N}\right.$$

$$\left. -(-1)^k f(N-n_0-1)\sin\frac{(2n_0-1)k\pi}{2N} + 2f(-1)\sin\frac{k\pi}{2N}\right\} \qquad (3.28)$$

for k=1,…,N.

Taking the DST-II of equation (3.15) yields:

$$S_w old = S_{[f(x-1)w(x-1)]} = S_{[f(x-1)w(x)]} + S_{[f(x-1)m(x-1)]}. \qquad (3.29)$$

Equations (3.28) and (3.29) together can be used to calculate the older time sequence windowed DST-II values.

3.5 Independent DCT/DST type-III split-triangular windowed update algorithms for moving data

The algorithm to calculate $C_{w(new)}$ and $S_{w(new)}$ are derived here for the windowed DCT-III and DST-III update in the presence of trapezoidal window and in section 3.5.2 the method to pre-process the data required to update the DCT-III and DST-III used in section 3.5.1 is presented.

3.5.1. Computation of $C_{w(new)}$ and $S_{w(new)}$

To calculate the independent DCT-III update for trapezoidal window we take DCT-III of equation (3.3) and equation (3.6):

i.e.

$$C_{w(new)}(x) = C_w(x+1) + C_{m(new)}(x) \tag{3.30}$$

$$C_{m(new)}(x) = C_m(x+1)$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1} P_x \frac{1}{n_0}\left[-f(n_0)\delta_{x,n0-1} - f(N-n_0)\delta_{x,N-n0-1} + 2f(N)\delta_{x,N-1}\right]\cos\left(\frac{x(2k+1)\pi}{2N}\right)$$

Solving the above equation yields:

$$C_{m(new)}(x) = C_m(x+1) + \sqrt{\frac{2}{N}}\frac{1}{n_0}\left\{-f(n_0)\cos\left(\frac{(n_0-1)(2k+1)\pi}{2N}\right)\right.$$

$$\left. - f(N-n_0)\cos\left(\frac{(N-n_0-1)(2k+1)\pi}{2N}\right) + 2f(N)\cos\left(\frac{(N-1)(2k+1)\pi}{2N}\right)\right\}$$

Therefore,

$$C_{m(new)}(x) = C_m(x+1) + \sqrt{\frac{2}{N}}\frac{1}{n_0}\left\{-f(n_0)\cos\left(\frac{(n_0-1)(2k+1)\pi}{2N}\right)\right.$$

$$\left. -(-1)^k f(N-n_0)\sin\left(\frac{(n_0+1)(2k+1)\pi}{2N}\right) + 2f(N)(-1)^k \sin\left[\frac{(2k+1)\pi}{2N}\right]\right\} \tag{3.31}$$

for k=0,1,…,N-1.

Equations (3.30) and (3.31) can be used to calculate the independent update of the moving DCT-III for trapezoidal window. $C_w(x+1)$ is the non-windowed DCT-III update of $f_w(x)$, calculated using equation (2.35) listed below for convenience, and $C_m(x+1)$ is the non-windowed DCT-III update of $f_m(x)$ also calculated using equation (2.35).

$$C(n+r,k) = 2E_r C(n,k) - C(n-r,k)$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}\left[(-1)^k f(n+r-1-x) + F_r f(n-N-x-1) - (-1)^k E_r f(n-1-x)\right]$$

$$\sin\frac{(x+1)(2k+1)\pi}{2N}$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}\left[(-1)^k F_r f(n-1-x) - f(n-N-x+r-1) + E_r f(n-N-x-1)\right]$$

$$\cos\frac{(x+1)(2k+1)\pi}{2N}$$

$$-\sqrt{\frac{2}{N}}\left(1-\frac{1}{\sqrt{2}}\right)f(n-N-r) + 2\sqrt{\frac{2}{N}}\left(1-\frac{1}{\sqrt{2}}\right)E_r f(n-N)$$

$$-\sqrt{\frac{2}{N}}\left(1-\frac{1}{\sqrt{2}}\right)f(n-N+r)$$

$$\text{for } k = 0,1,\ldots,N-1.$$

When using the above equation to calculate the non-windowed update we need the current value $C(n,k)$ and the previous value $C(n\text{-}1,k)$. In the case of C_w, the current value is $C_{[f(x)w(x)]}$ and the previous value is $C_{[f(x-1)w(x-1)]}$. However, $C_{[f(x-1)w(x-1)]}$ is not yet available, we instead have $C_{[f(x-1)w(x)]}$ from the previous time-step. Therefore, we need to derive the correction factor that calculates the correct value $C_{[f(x-1)w(x-1)]}$ from $C_{[f(x-1)w(x)]}$. Similarly for C_m, we need to calculate the correction factor to compute $C_{[f(x-1)m(x-1)]}$ from $C_{[f(x-1)m(x)]}$. In section 3.5.2, the formula to calculate the correct values $C_{[f(x-1)w(x-1)]}$ and $C_{[f(x-1)m(x-1)]}$ is presented. Appendix A, figure (A.12) provides the C language implementation of DST independent algorithm to implement the DCT-III update in the presence of split-triangular window for moving data.

Similarly, to calculate the independent DST-III update for trapezoidal window we take DST-III of equation (3.3) and equation (3.6):

i.e.

$$S_{w(new)}(x) = S_w(x+1) + S_{m(new)}(x) \tag{3.32}$$

$$S_{m(new)}(x) = S_m(x+1)$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{N} P_x \frac{1}{n_0}\left[-f(n_0)\delta_{x,n0-1} - f(N-n_0)\delta_{x,N-n0-1} + 2f(N)\delta_{x,N-1}\right]\sin\left(\frac{x(2k+1)\pi}{2N}\right)$$

Solving the above equation yields:

$$S_{m(new)}(x) = S_m(x+1) + \sqrt{\frac{2}{N}}\frac{1}{n_0}\left\{-f(n_0)\sin\left(\frac{(n_0-1)(2k+1)\pi}{2N}\right)\right.$$

$$\left. - f(N-n_0)\sin\left(\frac{(N-n_0-1)(2k+1)\pi}{2N}\right) + 2f(N)\sin\left(\frac{(N-1)(2k+1)\pi}{2N}\right)\right\}$$

Therefore,

$$S_{m(new)}(x) = S_m(x+1) + \sqrt{\frac{2}{N}}\frac{1}{n_0}\left\{-f(n_0)\cos\left(\frac{x(2k+1)\pi}{2N}\right)\right.$$

$$\left. -(-1)^k f(N-n_0)\cos\left(\frac{x(2k+1)\pi}{2N}\right) + 2f(N)(-1)^k\cos\left(\frac{x(2k+1)\pi}{2N}\right)\right\} \tag{3.33}$$

for k=0,1,…,N.

Equations (3.32) and (3.33) can be used to calculate the independent update of the moving DST-III for trapezoidal window. $S_w(x+1)$ is the non-windowed DST-III update of $f_w(x)$, calculated using equation (2.36) listed below for convenience, and $S_m(x+1)$ is the non-windowed DST-III update of $f_m(x)$ also calculated using equation (2.36).

$$S(n+r,k) = 2E_r S(n,k) - S(n-r,k)$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}\left[(-1)^k f(n+r-1-x) + F_r f(n-N-x-1) - (-1)^k E_r f(n-x-1)\right]$$

$$\cos\frac{(x+1)(2k+1)\pi}{2N}$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}\left[f(n-N-x+r-1)-(-1)^k F_r f(n-x-1)-E_r f(n-N-x-1)\right]$$

$$\sin\frac{(x+1)(2k+1)\pi}{2N}$$

$$+(-1)^k\sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}f(n-r)+(-1)^k\sqrt{\frac{2}{N}}\frac{1}{\sqrt{2}}f(n+r)-(-1)^k\sqrt{2}\sqrt{\frac{2}{N}}E_r f(n)$$

$$\text{for } k=1,\ldots,N.$$

When using the above equation to calculate the non-windowed update we need the current value $S(n,k)$ and the previous value $S(n\text{-}1,k)$. In the case of S_w, the current value is $S_{[f(x)w(x)]}$ and the previous value is $S_{[f(x-1)w(x-1)]}$. However, $S_{[f(x-1)w(x-1)]}$ is not yet available, we instead have $S_{[f(x-1)w(x)]}$ from the previous time-step. Therefore, we need to derive the correction factor that calculates the correct value $S_{[f(x-1)w(x-1)]}$ from $S_{[f(x-1)w(x)]}$. Similarly, for S_m, we need to calculate the correction factor to compute $S_{[f(x-1)m(x-1)]}$ from $S_{[f(x-1)m(x)]}$. In section 3.5.2 the formula to calculate the correct values $S_{[f(x-1)w(x-1)]}$ and $S_{[f(x-1)m(x-1)]}$ are developed. Appendix A, figure (A.14) lists the C language implementation of DCT independent algorithm to implement the DST-III update in the presence of split-triangular window for moving data.

3.5.2 Derivation of correction values for oldest time-step

In this section we derive the correction factor to calculate the correct value $C_{[f(x-1)w(x-1)]}$ from $C_{[f(x-1)w(x)]}$ for DCT-III update algorithm and the correct value of $S_{[f(x-1)w(x-1)]}$ from $S_{[f(x-1)w(x)]}$ for the DST update algorithm.

Taking DCT-III of equation (3.17) and simplifying:

$$C_m old(k) \equiv C_{[f(x-1)m(x-1)]} = C_{[f(x-1)m(x)]}$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}P_x\frac{1}{n_0}\left[-f(n_0-1)\delta_{x,n0}-f(N-n_0-1)\delta_{x,N-n0}+2f(-1)\delta_{x,0}\right]\cos\left(\frac{x(2k+1)\pi}{2N}\right)$$

for k=0,1,…,N-1.

Therefore,

$$=C_{[f(x-1)m(x)]}+\sqrt{\frac{2}{N}}\frac{1}{n_0}\left\{-f(n_0-1)\cos\left(\frac{n_0(2k+1)\pi}{2N}\right)\right.$$

$$\left.-f(N-n_0-1)\cos\left(\frac{(N-n_0)(2k+1)\pi}{2N}\right)+2f(-1)\right\}$$

$$C_m old(k)\equiv C_{[f(x-1)m(x-1)]}=C_{[f(x-1)m(x)]}+\sqrt{\frac{2}{N}}\frac{1}{n_0}\left\{-f(n_0-1)\cos\left(\frac{n_0(2k+1)\pi}{2N}\right)\right.$$

$$\left.-(-1)^k f(N-n_0-1)\sin\left(\frac{n_0(2k+1)\pi}{2N}\right)+2f(-1)\right\} \qquad (3.34)$$

Taking the DCT-III of equation (3.15) yields:

$$C_{[f(x-1)w(x-1)]}=C_{[f(x-1)w(x)]}+C_{[f(x-1)m(x-1)]}. \qquad (3.35)$$

Equation (3.34) and (3.35) together are used to calculate the older time sequence windowed DCT-III values.

Similarly taking the DST-III of equation (3.17) yields:

$$S_m old(k)\equiv S_{[f(x-1)m(x-1)]}=S_{[f(x-1)m(x)]}$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{N}\frac{1}{n_0}\left[-f(n_0-1)\delta_{x,n0}-f(N-n_0-1)\delta_{x,N-n0}+2f(-1)\delta_{x,0}\right]\sin\left(\frac{x(2k+1)\pi}{2N}\right)$$

for k=0,1,…,N.

Solving the above equation:

$$=S_{[f(x-1)m(x)]}+\sqrt{\frac{2}{N}}\frac{1}{n_0}\left\{-f(n_0-1)\sin\left(\frac{n_0(2k+1)\pi}{2N}\right)\right.$$

$$- f(N-n_0-1)\sin\left(\frac{(N-n_0)(2k+1)\pi}{2N}\right)\Bigg\}$$

Therefore,

$$S_m old(k) \equiv S_{[f(x-1)m(x-1)]} = S_{[f(x-1)m(x)]} + \sqrt{\frac{2}{N}}\frac{1}{n_0}\Bigg\{- f(n_0-1)\sin\left(\frac{n_0(2k+1)\pi}{2N}\right)$$

$$-(-1)^k f(N-n_0-1)\cos\left(\frac{n_0(2k+1)\pi}{2N}\right)\Bigg\} \qquad (3.36)$$

Taking the DST-III of equation (3.12) yields:

$$S_w old = S_{[f(x-1)w(x-1)]} = S_{[f(x-1)w(x)]} + S_{[f(x-1)m(x-1)]}. \qquad (3.37)$$

Equations (3.36) and (3.37) together are used to calculate the previous time sequence windowed DST values.

3.6 Independent DCT/DST type-IV split-triangular windowed update algorithms for moving data

The algorithm to calculate $C_{w(new)}$ and $S_{w(new)}$ are derived in section 3.6.1 for the windowed DCT-IV and DST-IV update in the presence of trapezoidal window and in section 3.6.2 the method to pre-process the data required to update the DCT-IV and DST-IV used in Section 3.6.1 is derived.

3.6.1. Computation of $C_{w(new)}$ and $S_{w(new)}$

To calculate the independent DCT-IV update for trapezoidal window we take DCT-IV of equation (3.3) and equation (3.6):

i.e.

$$C_{w(new)}(x) = C_w(x+1) + C_{m(new)}(x) \qquad (3.38)$$

$$C_{m(new)}(x) = C_m(x+1)$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}\frac{1}{n_0}\left[-f(n_0)\delta_{x,n0-1} - f(N-n_0)\delta_{x,N-n0-1} + 2f(N)\delta_{x,N-1}\right]\cos\left[\frac{(2x+1)}{2}\right]\left[\frac{(2k+1)}{2}\right]\frac{\pi}{N}$$

Solving the above equation yields:

$$C_{m(new)}(x) = C_m(x+1) + \sqrt{\frac{2}{N}}\frac{1}{n_0}\left\{-f(n_0)\cos\left[\frac{2(n_0-1)+1}{2}\right]\left[\frac{2k+1}{2}\right]\frac{\pi}{N}\right.$$

$$\left. - f(N-n_0)\cos\left[\frac{2(N-n_0-1)+1}{2}\right]\left[\frac{2k+1}{2}\right]\frac{\pi}{N} + 2f(N)\cos\left[\frac{2(N-1)+1}{2}\right]\left[\frac{2k+1}{2}\right]\frac{\pi}{N}\right\}$$

Therefore,

$$C_{m(new)}(x) = C_m(x+1) + \sqrt{\frac{2}{N}}\frac{1}{n_0}\left\{-f(n_0)\cos\left[\frac{2n_0-1}{2}\right]\left[\frac{2k+1}{2}\right]\frac{\pi}{N}\right.$$

$$\left. -(-1)^k f(N-n_0)\sin\left[\frac{2n_0+1}{2}\right]\left[\frac{2k+1}{2}\right]\frac{\pi}{N} + 2f(N)(-1)^k\sin\left[\frac{(2k+1)\pi}{4N}\right]\right\} \qquad (3.39)$$

for k=0,1,…,N-1.

Equations (3.38) and (3.39) can be used to calculate the independent update of the moving DCT-IV for trapezoidal window. $C_w(x+1)$ is the non-windowed DCT-IV update of $f_w(x)$, calculated using equation (2.45) listed below for convenience, and $C_m(x+1)$ is the non-windowed DCT-IV update of $f_m(x)$ also calculated using equation (2.45).

$$C(n+r,k) = 2E_r C(n,k) - C(n-r,k)$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}\left[F_r f(n-N-x-1) + (-1)^k f(n+r-x-1) - (-1)^k E_r f(n-x-1)\right]$$

$$\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}\left[E_r f(n-N-x-1)+(-1)^k F_r f(n-x-1)-f(n-N+r-x-1)\right]$$

$$\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

for $k = 0,1,...,N-1$.

When using the above equation to calculate the non-windowed update we need the current value $C(n,k)$ and the previous value $C(n\text{-}1,k)$. In the case of C_w, the current value is $C_{[f(x)w(x)]}$ and the previous value is $C_{[f(x-1)w(x-1)]}$. However, $C_{[f(x-1)w(x-1)]}$ is not yet available, we instead have $C_{[f(x-1)w(x)]}$ from the previous time-step. Therefore, we need to derive the correction factor that calculates the correct value $C_{[f(x-1)w(x-1)]}$ from $C_{[f(x-1)w(x)]}$. Similarly for C_m, we need to calculate the correction factor to compute $C_{[f(x-1)m(x-1)]}$ from $C_{[f(x-1)m(x)]}$. In section 3.6.2 derives the formula to calculate the correct values $C_{[f(x-1)w(x-1)]}$ and $C_{[f(x-1)m(x-1)]}$ is derived. Appendix A, figure (A.16) provides the C language implementation of DST independent algorithm to implement the DCT-IV update in the presence of split-triangular window for moving data.

Similarly, to calculate the independent DST-IV update for trapezoidal window we take DST-IV of equation (3.3) and equation (3.6):

i.e.

$$S_{w(new)}(x) = S_w(x+1) + S_{m(new)}(x) \tag{3.40}$$

$$S_{m(new)}(x) = S_m(x+1)$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}\frac{1}{n_0}\left[-f(n_0)\delta_{x,n0-1}-f(N-n_0)\delta_{x,N-n0-1}+2f(N)\delta_{x,N-1}\right]\sin\left[\frac{(2x+1)}{2}\right]\left[\frac{(2k+1)}{2}\right]\frac{\pi}{N}$$

Solving the above equation yields:

$$S_{m(new)}(x) = S_m(x+1) + \sqrt{\frac{2}{N}}\frac{1}{n_0}\left\{-f(n_0)\sin\left[\frac{2(n_0-1)+1}{2}\right]\left[\frac{2k+1}{2}\right]\frac{\pi}{N}\right.$$

$$\left. - f(N-n_0)\sin\left[\frac{2(N-n_0-1)+1}{2}\right]\left[\frac{2k+1}{2}\right]\frac{\pi}{N} + 2f(N)\sin\left[\frac{2(N-1)+1}{2}\right]\left[\frac{2k+1}{2}\right]\frac{\pi}{N}\right\}$$

Therefore,

$$S_{m(new)}(x) = S_m(x+1) + \sqrt{\frac{2}{N}}\frac{1}{n_0}\left\{-f(n_0)\sin\left[\frac{2n_0-1}{2}\right]\left[\frac{2k+1}{2}\right]\frac{\pi}{N}\right.$$

$$\left. -(-1)^k f(N-n_0)\cos\left[\frac{2n_0+1}{2}\right]\left[\frac{2k+1}{2}\right]\frac{\pi}{N} + 2f(N)(-1)^k\cos\left[\frac{(2k+1)\pi}{4N}\right]\right\} \qquad (3.41)$$

for k=0,1,…,N-1.

Equations (3.40) and (3.41) can be used to calculate the independent update of the moving DST-IV for trapezoidal window. $S_w(x+1)$ is the non-windowed DST-IV update of $f_w(x)$, calculated using equation (2.46) listed below for convenience, and $S_m(x+1)$ is the non-windowed DST-IV update of $f_m(x)$ also calculated using equation (2.46).

$$S(n+r,k) = 2E_r S(n,k) - S(n-r,k)$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}\left[F_r f(n-N-x-1) + (-1)^k f(n+r-1-x) - (-1)^k E_r f(n-x-1)\right]$$

$$\cos\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{r-1}\left[f(n-N+r-1-x) - (-1)^k F_r f(n-x-1) - E_r f(n-N-x-1)\right]$$

$$\sin\left[\left(\frac{2x+1}{2}\right)\left(\frac{2k+1}{2}\right)\frac{\pi}{N}\right]$$

for $k = 0,1,...,N-1$.

When using the above equation to calculate the non-windowed update we need the current value $S(n,k)$ and the previous value $S(n\text{-}1,k)$. In the case of S_w, the current value is

$S_{[f(x)w(x)]}$ and the previous value is $S_{[f(x-1)w(x-1)]}$. However, $S_{[f(x-1)w(x-1)]}$ is not yet available, we instead have $S_{[f(x-1)w(x)]}$ from the previous time-step. Therefore, we need to derive the correction factor that calculates the correct value $S_{[f(x-1)w(x-1)]}$ from $S_{[f(x-1)w(x)]}$. Similarly, for S_m, we need to calculate the correction factor to compute $S_{[f(x-1)m(x-1)]}$ from $S_{[f(x-1)m(x)]}$. In section 3.6.2 the formula to calculate the correct values $S_{[f(x-1)w(x-1)]}$ and $S_{[f(x-1)m(x-1)]}$ are developed. Appendix A, figure (A.19) includes the C language implementation of DCT independent algorithm to implement the DST-IV update in the presence of split-triangular window for moving data.

3.6.2 Derivation of correction values for oldest time-step

In this section the correction factor to calculate the correct value $C_{[f(x-1)w(x-1)]}$ from $C_{[f(x-1)w(x)]}$ for DCT-IV update algorithm and the correct value of $S_{[f(x-1)w(x-1)]}$ from $S_{[f(x-1)w(x)]}$ for the DST-IV update algorithm are derived.

Taking DCT-IV of equation (3.17) and simplifying:

$$C_m old(k) \equiv C_{[f(x-1)m(x-1)]} = C_{[f(x-1)m(x)]}$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}\frac{1}{n_0}\left[-f(n_0-1)\delta_{x,n0}-f(N-n_0-1)\delta_{x,N-n0}+2f(-1)\delta_{x,0}\right]\cos\left[\frac{(2x+1)}{2}\right]\left[\frac{(2k+1)}{2}\right]\frac{\pi}{N}$$

for k=0,1,…,N-1.

Therefore,

$$= C_{[f(x-1)m(x)]}+\sqrt{\frac{2}{N}}\frac{1}{n_0}\left\{-f(n_0-1)\cos\left[\frac{2n_0+1}{2}\right]\left[\frac{2k+1}{2}\right]\frac{\pi}{N}\right.$$

$$\left.-f(N-n_0-1)\cos\left[\frac{2(N-n_0)+1}{2}\right]\left[\frac{2k+1}{2}\right]\frac{\pi}{N}+2f(-1)\cos\left[\frac{2k+1}{4}\right]\frac{\pi}{N}\right\}$$

$$C_m old(k) \equiv C_{[f(x-1)m(x-1)]} = C_{[f(x-1)m(x)]} + \sqrt{\frac{2}{N}}\frac{1}{n_0}\left\{- f(n_0-1)\cos\left[\frac{2n_0+1}{2}\right]\left[\frac{2k+1}{2}\right]\frac{\pi}{N}\right.$$

$$\left. -(-1)^k f(N-n_0-1)\sin\left[\frac{2n_0-1}{2}\right]\left[\frac{2k+1}{2}\right]\frac{\pi}{N} + 2f(-1)\cos\left[\frac{(2k+1)\pi}{4N}\right]\right\} \qquad (3.42)$$

Taking the DCT-IV of equation (3.15) yields:

$$C_{[f(x-1)w(x-1)]} = C_{[f(x-1)w(x)]} + C_{[f(x-1)m(x-1)]}. \qquad (3.43)$$

Equations (3.42) and (3.43) together are used to calculate the previous time sequence windowed DCT-IV values.

Similarly taking the DST-IV of equation (3.17) yields:

$$S_m old(k) \equiv S_{[f(x-1)m(x-1)]} = S_{[f(x-1)m(x)]}$$

$$+\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}\frac{1}{n_0}\left[- f(n_0-1)\delta_{x,n0} - f(N-n_0-1)\delta_{x,N-n0} + 2f(-1)\delta_{x,0}\right]\sin\left[\frac{(2x+1)}{2}\right]\left[\frac{(2k+1)}{2}\right]\frac{\pi}{N}$$

for k=0,1,…,N-1.

Solving the above equation yields:

$$= S_{[f(x-1)m(x)]} + \sqrt{\frac{2}{N}}\frac{1}{n_0}\left\{- f(n_0-1)\sin\left[\frac{2n_0+1}{2}\right]\left[\frac{2k+1}{2}\right]\frac{\pi}{N}\right.$$

$$\left. - f(N-n_0-1)\sin\left[\frac{2(N-n_0)+1}{2}\right]\left[\frac{2k+1}{2}\right]\frac{\pi}{N} + 2f(-1)\sin\left[\frac{2k+1}{4}\right]\frac{\pi}{N}\right\}$$

Therefore,

$$S_m old(k) \equiv S_{[f(x-1)m(x-1)]} = S_{[f(x-1)m(x)]} + \sqrt{\frac{2}{N}}\frac{1}{n_0}\left\{- f(n_0-1)\sin\left[\frac{2n_0+1}{2}\right]\left[\frac{2k+1}{2}\right]\frac{\pi}{N}\right.$$

$$\left. -(-1)^k f(N-n_0-1)\cos\left[\frac{2n_0-1}{2}\right]\left[\frac{2k+1}{2}\right]\frac{\pi}{N} + 2f(-1)\sin\left[\frac{(2k+1)\pi}{4N}\right]\right\} \qquad (3.44)$$

Taking the DST-IV of equation (3.15) yields:

$$S_w old = S_{[f(x-1)w(x-1)]} = S_{[f(x-1)w(x)]} + S_{[f(x-1)m(x-1)]}. \tag{3.45}$$

Equations (3.44) and (3.45) together are used to calculate the previous time sequence windowed DST-IV values.

CHAPTER 4: MOVING DCT/DST HAMMING, HANNING AND BLACKMAN WINDOWED INDEPENDENT UPDATE ALGORITHMS

4.1 Introduction to independent DCT and DST update algorithms for Hanning, Hamming and Blackman windows

Update algorithms in the presence of Hanning, Hamming and Blackman windows were initially derived by Sherlock and Kakad [22, 23], however these algorithms involved simultaneous update of DCT and DST coefficients for windowed update data. In this chapter, the algorithms initially derived in [22, 23] are extended to perform independent windowed update of DCT and DST in the presence of Hanning, Hamming and Blackman windows. Windowed update for one new data point at a time are derived i.e. r=1, however the algorithms can be repeated if more than one point update is required. Section 4.2 lists the independent windowed update for DCT/DST type-I i.e. when the DCT update of the windowed data is to be performed we do not require the DST coefficients and similarly when the DST update of the windowed data is to be performed we do not require the DCT coefficients. Initially, the simultaneous algorithms derived by Sherlock and Kakad [22, 23] are provided for understanding and completeness of the algorithms and thereafter the modifications for independent update are given. In sections 4.3, 4.4 and 4.5 the derivation of independent windowed update algorithms for DCT/DST type-II, III and IV respectively are given. Appendix A lists the C language code used to calculate the independent windowed update algorithms for DCT/DST-II and IV in the presence of Hanning, Hamming and Blackman windows.

4.2 DCT/DST type-I windowed independent update algorithms in presence of Hanning, Hamming and Blackman windows

Initially the derivation of the simultaneous windowed update developed in [22] is listed and thereafter the modifications required for independent updating is discussed. Simultaneous algorithms developed by [22, 23] are listed to completely understand the suggested independent update algorithms. The general form of Hanning, Hamming and Blackman window function for x=0,…,N-1 is defined as:

$$w(x) = a_0 + a_1 \cos\left(\frac{2\pi x}{N}\right) + a_2 \cos\left(\frac{4\pi x}{N}\right), \tag{4.1}$$

where,

a_0=0.5, a_1=0.5 for Hanning Window,

a_0=0.54, a_1=0.46 for Hamming Window,

and a_0=0.42, a_1=0.5, a_2=0.08 for Blackman Window. (4.2)

For a signal $f(x)$, the windowed signal is:

$$f_w(x) = w(x) f(x) = \left\{ a_0 + a_1 \cos\left(\frac{2\pi x}{N}\right) + a_2 \cos\left(\frac{4\pi x}{N}\right) \right\} f(x). \tag{4.3}$$

Taking the DCT-I of equation (4.3) yields;

$$C_w(n,k) = a_0 C(n,k) + a_1 \sqrt{\frac{2}{N}} P_k \sum_{x=0}^{N} P_x f(x) \cos\left(\frac{2\pi x}{N}\right) \cos\frac{xk\pi}{N}$$

$$+ a_2 \sqrt{\frac{2}{N}} P_k \sum_{x=0}^{N} P_x f(x) \cos\left(\frac{4\pi x}{N}\right) \cos\frac{xk\pi}{N} \tag{4.4}$$

for k=0,1,…,N-1.

Simplifying the second term in equation (4.4) yields:

$$a_1 \sqrt{\frac{2}{N}} P_k \sum_{x=0}^{N} P_x f(x) \cos\left(\frac{2\pi x}{N}\right) \cos\frac{xk\pi}{N}$$

$$= a_1\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{N}P_x f(x)\frac{1}{2}\left[\cos\left(\frac{2\pi x}{N}-\frac{xk\pi}{N}\right)+\cos\left(\frac{2\pi x}{N}+\frac{xk\pi}{N}\right)\right]$$

$$= a_1\sqrt{\frac{2}{N}}\frac{1}{2}P_k\sum_{x=0}^{N}P_x f(x)\left[\cos\left(\frac{(k-2)x\pi}{N}\right)+\cos\left(\frac{(k+2)x\pi}{N}\right)\right]$$

Using the definitions of DCT-I, i.e. equation (2.5) above equation can be re-written as:

$$= \frac{a_1}{2}P_k\left[\frac{C(n,k-2)}{P_{k-2}}+\frac{C(n,k+2)}{P_{k+2}}\right] \tag{4.5}$$

Solving the third terms in equation (4.4) results in:

$$a_2\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{N}P_x f(x)\cos\left(\frac{4\pi x}{N}\right)\cos\frac{xk\pi}{N}$$

$$= a_2\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{N}P_x f(x)\frac{1}{2}\left[\cos\left(\frac{4\pi x}{N}-\frac{xk\pi}{N}\right)+\cos\left(\frac{4\pi x}{N}+\frac{xk\pi}{N}\right)\right]$$

$$= a_2\sqrt{\frac{2}{N}}\frac{1}{2}P_k\sum_{x=0}^{N}P_x f(x)\left[\cos\left(\frac{(k-4)x\pi}{N}\right)+\cos\left(\frac{(k+4)x\pi}{N}\right)\right]$$

Using the definitions of DCT-I, i.e. equation (2.5), above equation can be re-written as:

$$= \frac{a_2}{2}P_k\left[\frac{C(n,k+4)}{P_{k-4}}+\frac{C(n,k-4)}{P_{k-4}}\right] \tag{4.6}$$

Substituting equations (4.5) and (4.6) in equation (4.4) yields:

$$C_w(n,k) = a_0C(n,k)+\frac{a_1}{2}P_k\left[\frac{C(n,k-2)}{P_{k-2}}+\frac{C(n,k+2)}{P_{k+2}}\right]$$

$$+\frac{a_2}{2}P_k\left[\frac{C(n,k+4)}{P_{k-4}}+\frac{C(n,k-4)}{P_{k-4}}\right] \tag{4.7}$$

for k=0,1,…,N.

Clearly, equation (4.7) can be used to independently update the DCT-I independent of DST coefficients. Equation (4.7) developed in [22] used the simultaneous

DCT-I update equation, which required DST coefficients as well, whenever the updated DCT coefficients need to be calculated. However, in our implementation we use the DCT-I independent update equation (2.13) whenever the updated DCT coefficients need to be calculated which is then multiplied by the window. This implementation does not require the DST coefficients. Also to compute the updated DCT using independent update equation (2.13) requires initial processing of data i.e. to calculate the updated independent DCT we require current time-step DCT and one previous time-step DCT. When we start the computation we calculate two time-steps DCT's using the definition and thereafter the last two time-steps DCT's are saved to calculate the updated DCT. Since in equations (4.7) the range of index k is out of defined range of k=0,1,…,N, symmetry properties of DCT-I are used to convert out of range values of k to be within the permissible values [22, 23]. For DCT type-I the symmetry properties are as follows:

$$C(n,-k) = C(n,k); C(n,2N+k) = C(n,2N-k);$$

$$C(n,4N+k) = C(n,k); C(n,N-k) = C(n,N+k) \tag{4.8}$$

Similarly, for deriving the formula to calculate the DST type-I independent windowed update we take the DST-I of equation (4.3) resulting in:

$$S_w(n,k) = a_0 S(n,k) + a_1\sqrt{\frac{2}{N}}\sum_{x=1}^{N-1} f(x)\cos\left(\frac{2\pi x}{N}\right)\sin\frac{xk\pi}{N}$$

$$+ a_2\sqrt{\frac{2}{N}}\sum_{x=1}^{N-1} f(x)\cos\left(\frac{4\pi x}{N}\right)\sin\frac{xk\pi}{N} \tag{4.9}$$

for k=1,2,…,N-1.

Simplifying the second term of equation (4.9):

$$a_1\sqrt{\frac{2}{N}}\sum_{x=1}^{N-1} f(x)\cos\left(\frac{2\pi x}{N}\right)\sin\frac{xk\pi}{N}$$

$$= a_1\sqrt{\frac{2}{N}}\sum_{x=1}^{N-1} f(x)\frac{1}{2}\left[\sin\left(\frac{xk\pi}{N}-\frac{2\pi x}{N}\right)+\sin\left(\frac{xk\pi}{N}+\frac{2\pi x}{N}\right)\right]$$

$$= a_1\sqrt{\frac{2}{N}}\frac{1}{2}\sum_{x=1}^{N-1} f(x)\left[\sin\left(\frac{(k-2)x\pi}{N}\right)+\sin\left(\frac{(k+2)x\pi}{N}\right)\right]$$

Using the definitions of DST-I, i.e. equation (2.6), the above equation can be written as:

$$= \frac{a_1}{2}\left[S(n,k-2)+S(n,k+2)\right] \tag{4.10}$$

Solving the third terms in equation (4.9) results in:

$$= a_2\sqrt{\frac{2}{N}}\sum_{x=1}^{N-1} f(x)\cos\left(\frac{4\pi x}{N}\right)\sin\frac{xk\pi}{N}$$

$$= a_2\sqrt{\frac{2}{N}}\sum_{x=1}^{N-1} f(x)\frac{1}{2}\left[\sin\left(\frac{xk\pi}{N}-\frac{4\pi x}{N}\right)+\sin\left(\frac{xk\pi}{N}+\frac{4\pi x}{N}\right)\right]$$

$$= a_2\sqrt{\frac{2}{N}}\frac{1}{2}\sum_{x=1}^{N-1} f(x)\left[\sin\left(\frac{(k-4)x\pi}{N}\right)+\sin\left(\frac{(k+4)x\pi}{N}\right)\right]$$

Using the definitions of DST-I the above equation can be re-written as:

$$= \frac{a_2}{2}\left[S(n,k-4)+S(n,k+4)\right] \tag{4.11}$$

Substituting equations (4.10) and (4.11) in equation (4.9) results in:

$$S_w(n,k) = a_0 S(n,k)+\frac{a_1}{2}\left[S(n,k+2)+S(n,k-2)\right]+\frac{a_2}{2}\left[S(n,k+4)+S(n,k-4)\right] \tag{4.12}$$

for k=1,2,…,N-1.

Clearly, equation (4.12) can be used to independently update the DST-I independent of the DCT coefficients. Equation (4.12) developed in [22] used the simultaneous DST-I update equation, which required DCT coefficients as well. However, in our implementation we use the DST-I independent update equation (2.14) whenever

the updated DST coefficients need to be calculated, which is then multiplied by the window. This implementation does not require the DCT coefficients. Also to compute the updated DST using independent update equation (2.14) requires initial processing of data i.e. to calculate the updated independent DST we require current time-step DST and one previous time-step DST. When we start the computation we calculate two time-steps DST's using the definition and thereafter the last two time-steps DST's are saved to calculate the updated DST. Examination of equations (4.12) shows that the range of index k is out of the range of the definition, therefore symmetrical properties of DST-I are used to convert out of range values of k to the allowed values [22, 23]. For DST type-I the symmetry properties are as follows:

$$S(n,-k) = -S(n,k); S(n,N+k) = -S(n,N-k); S(n,4N+k) = S(n,k);$$

$$S(n,0) = S(n,N) = 0. \tag{4.13}$$

4.3 DCT/DST type-II windowed independent update algorithms in presence of Hanning, Hamming and Blackman windows

In order to derive the algorithms capable of independently updating the windowed signal for DCT/DST type-II, first the simultaneous windowed update algorithms derived by Sherlock and Kakad [22, 23] are given below and thereafter these algorithms are modified to develop algorithms capable of independently updating the windowed signal in the presence of Hanning, Hamming and Blackman windows. These algorithms are capable of updating one point at a time i.e. for the case r=1. If more than one point update is desired the developed algorithms can be reused r number of times. C language program to validate the derived independent algorithm is included in Appendix A.

Taking the DCT-II of equation (4.3) yields;

$$C_w(n,k) = a_0 C(n,k) + a_1\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{N-1} f(x)\cos\left(\frac{2\pi x}{N}\right)\cos\frac{(2x+1)k\pi}{2N}$$

$$+ a_2\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{N-1} f(x)\cos\left(\frac{4\pi x}{N}\right)\cos\frac{(2x+1)k\pi}{2N} \qquad (4.14)$$

Simplifying the second term of equation (4.14) yields:

$$a_1\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{N-1} f(x)\cos\left(\frac{2\pi x}{N}\right)\cos\frac{(2x+1)k\pi}{2N}$$

$$= a_1\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{N-1} f(x)\frac{1}{2}\left[\cos\left(\frac{2\pi x}{N} - \frac{(2x+1)k\pi}{2N}\right) + \cos\left(\frac{2\pi x}{N} + \frac{(2x+1)k\pi}{2N}\right)\right]$$

$$= a_1\sqrt{\frac{2}{N}}\frac{1}{2}P_k\sum_{x=0}^{N-1} f(x)\left[\cos\left(\frac{(2x+1)(k-2)\pi + 2\pi}{2N}\right) + \cos\left(\frac{(2x+1)(k+2)\pi - 2\pi}{2N}\right)\right]$$

$$= a_1\frac{1}{2}P_k\left[\cos\frac{\pi}{N}\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1} f(x)\cos\left(\frac{(2x+1)(k-2)\pi}{2N}\right)\right.$$

$$- \sin\frac{\pi}{N}\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1} f(x)\sin\left(\frac{(2x+1)(k-2)\pi}{2N}\right)$$

$$+ \cos\frac{\pi}{N}\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1} f(x)\cos\left(\frac{(2x+1)(k+2)\pi}{2N}\right)$$

$$\left.+ \sin\frac{\pi}{N}\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1} f(x)\sin\left(\frac{(2x+1)(k+2)\pi}{2N}\right)\right]$$

Using the definitions of DCT-II and DST-II the above can be re-written as:

$$= \frac{a_1}{2}P_k\left[\cos\frac{\pi}{N}\frac{C(n,k-2)}{P_{k-2}} - \sin\frac{\pi}{N}\frac{S(n,k-2)}{P_{k-2}} + \cos\frac{\pi}{N}\frac{C(n,k+2)}{P_{k+2}} + \sin\frac{\pi}{N}\frac{S(n,k+2)}{P_{k+2}}\right] \qquad (4.15)$$

Solving the third terms in equation (4.14) results in:

$$a_2\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{N-1} f(x)\cos\left(\frac{4\pi x}{N}\right)\cos\frac{(2x+1)k\pi}{2N}$$

$$= a_2\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{N-1}f(x)\frac{1}{2}\left[\cos\left(\frac{4\pi x}{N}-\frac{(2x+1)k\pi}{2N}\right)+\cos\left(\frac{4\pi x}{N}+\frac{(2x+1)k\pi}{2N}\right)\right]$$

$$= a_2\sqrt{\frac{2}{N}}\frac{1}{2}P_k\sum_{x=0}^{N-1}f(x)\left[\cos\left(\frac{(2x+1)(k-4)\pi+2\pi}{2N}\right)+\cos\left(\frac{(2x+1)(k+4)\pi-2\pi}{2N}\right)\right]$$

$$= a_2\frac{1}{2}P_k\left[\cos\frac{2\pi}{N}\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}f(x)\cos\left(\frac{(2x+1)(k-4)\pi}{2N}\right)\right.$$

$$-\sin\frac{2\pi}{N}\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}f(x)\sin\left(\frac{(2x+1)(k-4)\pi}{2N}\right)$$

$$+\cos\frac{2\pi}{N}\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}f(x)\cos\left(\frac{(2x+1)(k+4)\pi}{2N}\right)$$

$$\left.+\sin\frac{2\pi}{N}\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}f(x)\sin\left(\frac{(2x+1)(k+4)\pi}{2N}\right)\right]$$

Using the definitions of DCT-II and DST-II the above equation can be re-written as:

$$= \frac{a_2}{2}P_k\left[\cos\frac{2\pi}{N}\frac{C(n,k+4)}{P_{k+4}}+\sin\frac{2\pi}{N}\frac{S(n,k+4)}{P_{k+4}}+\cos\frac{2\pi}{N}\frac{C(n,k-4)}{P_{k-4}}-\sin\frac{2\pi}{N}\frac{S(n,k-4)}{P_{k-4}}\right] \tag{4.16}$$

Substituting equations (4.15), (4.16) in equation (4.14) results in:

$$C_w(n,k) = a_0C(n,k)+\frac{a_1}{2}P_k\left\{\cos\frac{\pi}{N}\left[\frac{C(n,k+2)}{P_{k+2}}+\frac{C(n,k-2)}{P_{k-2}}\right]\right.$$

$$\left.+\sin\frac{\pi}{N}\left[\frac{S(n,k+2)}{P_{k+2}}-\frac{S(n,k-2)}{P_{k-2}}\right]\right\}$$

$$+\frac{a_2}{2}P_k\left\{\cos\frac{2\pi}{N}\left[\frac{C(n,k+4)}{P_{k+4}}+\frac{C(n,k-4)}{P_{k-4}}\right]\right.$$

$$\left.+\sin\frac{2\pi}{N}\left[\frac{S(n,k+4)}{P_{k+4}}-\frac{S(n,k-4)}{P_{k-4}}\right]\right\}. \tag{4.17}$$

for k=0,1,…,N-1.

Equation (4.17) can be used to calculate the DCT-II windowed update algorithm but it depends on both the DST as well as the DCT coefficients. Therefore we need a way to represent DST coefficients in terms of DCT coefficients only. To do this we consider the non-windowed DCT update equation (2.19) for the special case of r=1:

$$C(n+1,k)=\cos\frac{k\pi}{N}C(n,k)+\sin\frac{k\pi}{N}S(n,k)+\sqrt{\frac{2}{N}}P_k\cos\frac{k\pi}{2N}\left[(-1)^k f(n)-f(n-N)\right] \tag{4.18}$$

Solving equation (4.18) for *S(n,k)* yields:

$$S(n,k)=\frac{1}{\sin\frac{k\pi}{N}}\left[C(n+1,k)-\cos\frac{k\pi}{N}C(n,k)+\sqrt{\frac{2}{N}}P_k\cos\frac{k\pi}{2N}\left\{f(n-N)-(-1)^k f(n)\right\}\right]$$

for k=1,…,N-1. (4.19)

Equation (4.19) is undefined at k=N. Therefore we use equation (2.16), the definition of DST type-II to find the value of DST coefficient at k=N:

$$S(n,N)=\sqrt{\frac{1}{N}}\sum_{x=0}^{N-1}(-1)^x f(n-N+x) \tag{4.20}$$

While using equation (4.17) to calculate the windowed DCT-II update, the DST coefficients when needed, are calculated indirectly using equation (4.19) and (4.20) utilizing DCT coefficients only. Hence equations (4.17), (4.19) and (4.20) can be used simultaneously to calculate the DCT updated windowed coefficients with DCT coefficients only. Dependence of equation (4.17) on indices (k+2), (k-2), (k+4) and (k-4) results in indices being out of the range given in the definitions of DCT-II and DST-II. We use the following symmetry properties of DCT-II and DST-II [22] to extend the range of indices to all integer values of k:

$$C(n,k)=C(n,-k)=-C(n,2N+k)=-C(n,2N-k)=C(n,4N-k)=C(n,4N+k);$$

$$C(n,N-k)=C(n,N+k);C(n,N)=0;$$

$$S(n,k)=-S(n,-k)=S(n,2N-k)=-S(n,2N+k)=-S(n,4N-k)=S(n,4N+k);$$

$$S(n,N+k)=S(n,N-k);S(n,0)=0. \tag{4.21}$$

In Appendix A, figure (A.7), the C code snippet to calculate the DCT-II independent windowed update in the presence of Hanning, Hamming and Blackman windows is included.

Similarly, in order to formulate the analytics to calculate the DST type-II independent windowed update we take the DST-II of equation (4.3) resulting in:

$$S_w(n,k)=a_0S(n,k)+a_1\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{N-1}f(x)\cos\left(\frac{2\pi x}{N}\right)\sin\frac{(2x+1)k\pi}{2N}$$

$$+a_2\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{N-1}f(x)\cos\left(\frac{4\pi x}{N}\right)\sin\frac{(2x+1)k\pi}{2N} \tag{4.22}$$

Simplifying the second term of equation (4.22) yields:

$$a_1\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{N-1}f(x)\cos\left(\frac{2\pi x}{N}\right)\sin\frac{(2x+1)k\pi}{2N}$$

$$=a_1\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{N-1}f(x)\frac{1}{2}\left[\sin\left(\frac{(2x+1)k\pi}{2N}-\frac{2\pi x}{N}\right)+\sin\left(\frac{(2x+1)k\pi}{2N}+\frac{2\pi x}{N}\right)\right]$$

$$=a_1\sqrt{\frac{2}{N}}\frac{1}{2}P_k\sum_{x=0}^{N-1}f(x)\left[\sin\left(\frac{(2x+1)(k-2)\pi-2\pi}{2N}\right)+\sin\left(\frac{(2x+1)(k+2)\pi+2\pi}{2N}\right)\right]$$

$$=a_1\frac{1}{2}P_k\left[\cos\frac{\pi}{N}\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}f(x)\sin\left(\frac{(2x+1)(k-2)\pi}{2N}\right)\right.$$

$$\left.-\sin\frac{\pi}{N}\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}f(x)\cos\left(\frac{(2x+1)(k-2)\pi}{2N}\right)\right.$$

$$+\cos\frac{\pi}{N}\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}f(x)\sin\left(\frac{(2x+1)(k+2)\pi}{2N}\right)$$

$$+\sin\frac{\pi}{N}\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}f(x)\cos\left(\frac{(2x+1)(k+2)\pi}{2N}\right)\Bigg]$$

Using the definitions of DCT-II and DST-II the above equation can be written as:

$$=\frac{a_1}{2}P_k\left[\cos\frac{\pi}{N}\frac{S(n,k-2)}{P_{k-2}}-\sin\frac{\pi}{N}\frac{C(n,k-2)}{P_{k-2}}+\cos\frac{\pi}{N}\frac{S(n,k+2)}{P_{k+2}}+\sin\frac{\pi}{N}\frac{C(n,k+2)}{P_{k+2}}\right] \tag{4.23}$$

Solving the third terms in equation (4.22) results in:

$$a_2\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{N-1}f(x)\cos\left(\frac{4\pi x}{N}\right)\sin\frac{(2x+1)k\pi}{2N}$$

$$=a_2\sqrt{\frac{2}{N}}P_k\sum_{x=0}^{N-1}f(x)\frac{1}{2}\left[\sin\left(\frac{(2x+1)k\pi}{2N}-\frac{4\pi x}{N}\right)+\sin\left(\frac{(2x+1)k\pi}{2N}+\frac{4\pi x}{N}\right)\right]$$

$$=a_2\sqrt{\frac{2}{N}}\frac{1}{2}P_k\sum_{x=0}^{N-1}f(x)\left[\sin\left(\frac{(2x+1)(k-4)\pi-4\pi}{2N}\right)+\sin\left(\frac{(2x+1)(k+4)\pi+4\pi}{2N}\right)\right]$$

$$=a_2\frac{1}{2}P_k\Bigg[\cos\frac{2\pi}{N}\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}f(x)\sin\left(\frac{(2x+1)(k-4)\pi}{2N}\right)$$

$$-\sin\frac{2\pi}{N}\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}f(x)\cos\left(\frac{(2x+1)(k-4)\pi}{2N}\right)$$

$$+\cos\frac{2\pi}{N}\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}f(x)\sin\left(\frac{(2x+1)(k+4)\pi}{2N}\right)$$

$$+\sin\frac{2\pi}{N}\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}f(x)\cos\left(\frac{(2x+1)(k+4)\pi}{2N}\right)\Bigg]$$

Using the definitions of DCT-II and DST-II the above equation can be written as:

$$= \frac{a_2}{2} P_k \left[\cos\frac{2\pi}{N} \frac{S(n,k-4)}{P_{k-4}} - \sin\frac{2\pi}{N} \frac{C(n,k-4)}{P_{k-4}} + \cos\frac{2\pi}{N} \frac{S(n,k+4)}{P_{k+4}} + \sin\frac{2\pi}{N} \frac{C(n,k+4)}{P_{k+4}} \right] \tag{4.24}$$

Substituting equations (4.23) and (4.24) in equation (4.22) yields:

$$\begin{aligned} S_w(n,k) = a_0 S(n,k) + \frac{a_1}{2} P_k \left\{ \cos\frac{\pi}{N} \left[\frac{S(n,k+2)}{P_{k+2}} + \frac{S(n,k-2)}{P_{k-2}} \right] \right. \\ \left. + \sin\frac{\pi}{N} \left[\frac{C(n,k+2)}{P_{k+2}} - \frac{C(n,k-2)}{P_{k-2}} \right] \right\} \\ + \frac{a_2}{2} P_k \left\{ \cos\frac{2\pi}{N} \left[\frac{S(n,k+4)}{P_{k+4}} + \frac{S(n,k-4)}{P_{k-4}} \right] \right. \\ \left. + \sin\frac{2\pi}{N} \left[\frac{C(n,k+4)}{P_{k+4}} - \frac{C(n,k-4)}{P_{k-4}} \right] \right\}, \end{aligned} \tag{4.25}$$

for k=1,2,…,N.

Equation (4.25) can be used to calculate the windowed DST-II update however it requires DST as well as the DCT coefficients. Therefore we need a way to represent DCT coefficients in terms of DST coefficients only. To do this we consider the non-windowed DST-II update equation (2.20) for special case of r=1:

$$S(n+1,k) = \cos\frac{k\pi}{N} S(n,k) - \sin\frac{k\pi}{N} C(n,k) + \sqrt{\frac{2}{N}} P_k \sin\frac{k\pi}{2N} \left[f(n-N) - (-1)^k f(n) \right] \tag{4.26}$$

Solving equation (4.26) for $C(n,k)$ yields:

$$C(n,k) = \frac{1}{\sin\frac{k\pi}{N}} \left[\cos\frac{k\pi}{N} S(n,k) - S(n+1,k) + \sqrt{\frac{2}{N}} P_k \sin\frac{k\pi}{2N} \left\{ f(n-N) - (-1)^k f(n) \right\} \right]$$

for k=1,…,N-1. (4.27)

As equation (4.27) does not yield a value, at $k=0$. We need to find an alternative way of calculating value of $C(n,k)$ at $k=0$. We use equation (2.15), the definition of DCT type-II to find the value at k=0. i.e.,

$$C(n,0) = \sqrt{\frac{1}{N}} \sum_{x=0}^{N-1} f(n-N+x). \tag{4.28}$$

Equations (4.25), (4.27) and (4.28) can then be used to calculate the DST-II update in the presence of Hamming, Hanning and Blackman windows without using the DCT coefficients. While using equation (4.25) to calculate the windowed DST-II update, whenever the DCT coefficients are needed, they are calculated indirectly using equations (4.27) and (4.28). Dependence of equation (4.25) on indices (k+2), (k-2), (k+4) and (k-4) results in indices being out of the range as given in the definitions of DCT-II and DST-II. We use the symmetry properties of DCT and DST [22], given by equation (4.21), to extend the range of indices to all integer values of k. Appendix A, figure (A.10) implements the independent windowed update of DST coefficients for Hamming, Hanning and Blackman windows depending on the values of a_0, a_1 and a_2 as defined in equation (4.2).

4.4 DCT/DST type-III windowed independent update algorithms in presence of Hanning, Hamming and Blackman windows

To find the windowed update algorithm we take the DCT-III of equation (4.3) resulting in:

$$C_w(n,k) = a_0 C(n,k) + a_1 \sqrt{\frac{2}{N}} \sum_{x=0}^{N-1} P_x f(x) \cos\left(\frac{2\pi x}{N}\right) \cos\frac{(2k+1)x\pi}{2N}$$

$$+ a_2 \sqrt{\frac{2}{N}} \sum_{x=0}^{N-1} P_x f(x) \cos\left(\frac{4\pi x}{N}\right) \cos\frac{(2k+1)x\pi}{2N} \tag{4.29}$$

for k=0,1,…,N-1.

Simplifying the second term of equation (4.29) yields:

$$a_1\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}P_x f(x)\cos\left(\frac{2\pi x}{N}\right)\cos\frac{(2k+1)x\pi}{2N}$$

$$=a_1\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}P_x f(x)\cos\left(\frac{4\pi x}{2N}\right)\cos\frac{(2k+1)x\pi}{2N}$$

$$=a_1\sqrt{\frac{2}{N}}\frac{1}{2}\sum_{x=0}^{N-1}P_x f(x)\left[\cos\left(\frac{x(2k+1+4)\pi}{2N}\right)+\cos\left(\frac{x(2k+1-4)\pi}{2N}\right)\right]$$

Using the definitions of DCT-III the above can be written as:

$$=\frac{a_1}{2}\left[C(n,k+4)+C(n,k-4)\right] \tag{4.30}$$

Solving for the third term in equation (4.29) results in:

$$a_2\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}P_x f(x)\cos\left(\frac{4\pi x}{N}\right)\cos\frac{(2k+1)x\pi}{2N}$$

$$=a_2\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}P_x f(x)\cos\left(\frac{8\pi x}{2N}\right)\cos\frac{(2k+1)x\pi}{2N}$$

$$=a_2\sqrt{\frac{2}{N}}\frac{1}{2}\sum_{x=0}^{N-1}P_x f(x)\left[\cos\left(\frac{x(2k+1+8)\pi}{2N}\right)+\cos\left(\frac{x(2k+1-8)\pi}{2N}\right)\right]$$

Using the definitions of DCT-III the above equation can be written as:

$$=\frac{a_2}{2}\left[C(n,k+8)+C(n,k-8)\right] \tag{4.31}$$

Substituting equation (4.30) and (4.31) in equation (4.29) yields the DCT-III independent windowed update equation in the presence of Hanning, Hamming and Blackman window.

$$C_w(n,k) = a_0 C(n,k) + \frac{a_1}{2}\left[C(n,k+4) + C(n,k-4)\right] + \frac{a_2}{2}\left[C(n,k+8) + C(n,k-8)\right] \quad (4.32)$$

Clearly, equation (4.32) developed in [22] can be used to independently update the DCT-III independent of the DST coefficients. Equation (4.32) developed in [22] used the simultaneous DCT-III update equation, which required DST-III coefficients as well. However, in our implementation we use the DCT-III independent update equation (2.35) whenever the updated DCT coefficients need to be calculated which is then multiplied by the window. This implementation does not require the DST coefficients. Also to compute the updated DCT-III using independent update equation (2.35) requires initial processing of data i.e. to calculate the updated independent DCT we require current time-step DCT and one previous time-step DCT. When we start the computation we calculate two time-steps DCT's using the definition and thereafter the last two time-steps DCT's are saved to calculate the updated DCT. Since in equations (4.32) the range of index k is out of defined range of k=0,1,…,N, symmetry properties of DCT-III are used to convert out of range values of k to the permissible values [22, 23]. For DCT type-III the symmetric properties are as follows:

$$C(n,-k) = C(n,k-1); C(n,2N+k) = C(n,k);$$

$$C(n,N+k) = C(n,N-1-k) \quad (4.33)$$

Similarly, in order to formulate the analytics to calculate the DST type-III independent windowed update we take the DST-III of equation (4.3) resulting in:

$$S_w(n,k) = a_0 S(n,k) + a_1\sqrt{\frac{2}{N}}\sum_{x=1}^{N} P_x f(x)\cos\left(\frac{2\pi x}{N}\right)\sin\frac{(2k+1)x\pi}{2N}$$

$$+ a_2\sqrt{\frac{2}{N}}\sum_{x=1}^{N} P_x f(x)\cos\left(\frac{4\pi x}{N}\right)\sin\frac{(2k+1)x\pi}{2N} \quad (4.34)$$

for k=0,1,…,N-1

Simplifying the second term of equation (4.34) results in:

$$a_1\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1}P_x f(x)\cos\left(\frac{2\pi x}{N}\right)\sin\frac{(2k+1)x\pi}{2N}$$

$$=a_1\sqrt{\frac{2}{N}}\frac{1}{2}\sum_{x=1}^{N}P_x f(x)\left[\sin\left(\frac{x(2k+1)\pi+4\pi x}{2N}\right)+\sin\left(\frac{x(2k+1)\pi-4\pi x}{2N}\right)\right]$$

$$=a_1\sqrt{\frac{2}{N}}\frac{1}{2}\sum_{x=1}^{N}P_x f(x)\left[\sin\left(\frac{x(2k+1+4)\pi}{2N}\right)+\sin\left(\frac{x(2k+1-4)\pi}{2N}\right)\right]$$

Using the definitions of DST-III the above equation can be written as:

$$=\frac{a_1}{2}\left[S(n,k+4)+S(n,k-4)\right] \tag{4.35}$$

Solving the third term in equation (4.34) results in:

$$a_2\sqrt{\frac{2}{N}}\sum_{x=1}^{N}P_x f(x)\cos\left(\frac{4\pi x}{N}\right)\sin\frac{(2k+1)x\pi}{2N}$$

$$=a_2\sqrt{\frac{2}{N}}\frac{1}{2}\sum_{x=1}^{N}P_x f(x)\left[\sin\left(\frac{x(2k+1)\pi+4\pi x}{2N}\right)+\sin\left(\frac{x(2k+1)\pi-4\pi x}{2N}\right)\right]$$

$$=a_2\sqrt{\frac{2}{N}}\frac{1}{2}\sum_{x=1}^{N}P_x f(x)\left[\sin\left(\frac{x(2k+1+4)\pi}{2N}\right)+\sin\left(\frac{x(2k+1-4)\pi}{2N}\right)\right]$$

Using the definitions of DST-III the above equation can be written as:

$$=\frac{a_2}{2}\left[S(n,k+8)+S(n,k-8)\right] \tag{4.36}$$

Substituting equations (4.35) and (4.36) in equation (4.34) yields:

$$S_w(n,k)=a_0S(n,k)+\frac{a_1}{2}\left[S(n,k+4)+S(n,k-4)\right]+\frac{a_2}{2}\left[S(n,k+8)+S(n,k-8)\right] \tag{4.37}$$

for k=1,…,N-1.

Clearly, equation (4.37) developed in [22] can be used to independently update the DST-III independent of the DCT coefficients. Equation (4.37) initially developed in [22] used the simultaneous DST-III update equation, which required DCT coefficients as well. However, in our implementation we use the DST-III independent update equation (2.36) whenever the updated DST coefficients need to be calculated which is then multiplied by the window. This implementation does not require the DCT coefficients. Also to compute the updated DST using independent update equation (2.36) requires initial processing of data i.e. to calculate the updated independent DST we require current time-step DST and one previous time-step DST. When we start the computation we calculate two time-steps DST's using the definition and thereafter the last two time-steps DST's are saved to calculate the updated DST. Examination of equations (4.37) shows that the range of index k is out of the permissible range, therefore symmetry properties of DST-III are used to convert out of range values of k to the permissible values [22, 23]. For DST type-III the symmetry properties are as follows:

$$S(n,-k) = -S(n,k-1)$$

$$S(n,k) = -S(n,2N-k-1) \tag{4.38}$$

4.5 DCT/DST type-IV windowed independent update algorithms in presence of Hanning, Hamming and Blackman windows

To find the independent DCT-IV windowed update equation we take DCT-IV of equation (4.3);

$$C_w(n,k) = a_0 C(n,k) + a_1\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1} f(x)\cos\left(\frac{2\pi x}{N}\right)\cos\frac{(2x+1)(2k+1)\pi}{4N}$$

$$+ a_2\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1} f(x)\cos\left(\frac{4\pi x}{N}\right)\cos\frac{(2x+1)(2k+1)\pi}{4N} \tag{4.39}$$

Simplifying the second term in equation (4.39) yields:

$$a_1\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1} f(x)\cos\left(\frac{2\pi x}{N}\right)\cos\frac{(2x+1)(2k+1)\pi}{4N}$$

$$= a_1\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1} f(x)\cos\left(\frac{8\pi x}{4N}\right)\cos\frac{(2x+1)(2k+1)\pi}{4N}$$

$$= a_1\sqrt{\frac{2}{N}}\frac{1}{2}\sum_{x=0}^{N-1} f(x)\left[\cos\left(\frac{(2x+1)(2k+1)\pi+8\pi x}{4N}\right)+\cos\left(\frac{(2x+1)(2k+1)\pi-8\pi x}{4N}\right)\right]$$

$$= a_1\sqrt{\frac{2}{N}}\frac{1}{2}\sum_{x=0}^{N-1} f(x)\left[\cos\left(\frac{(2x+1)(2k+1+4)\pi-4\pi}{4N}\right)+\cos\left(\frac{(2x+1)(2k+1-4)\pi+4\pi}{4N}\right)\right]$$

$$= a_1\sqrt{\frac{2}{N}}\frac{1}{2}\sum_{x=0}^{N-1} f(x)\left\{\cos\left(\frac{(2x+1)(2k+1+4)\pi}{4N}\right)\cos\frac{\pi}{N}+\sin\left(\frac{(2x+1)(2k+1+4)\pi}{4N}\right)\sin\frac{\pi}{N}\right.$$

$$\left.+\cos\left(\frac{(2x+1)(2k+1-4)\pi}{4N}\right)\cos\frac{\pi}{N}-\sin\left(\frac{(2x+1)(2k+1-4)\pi}{4N}\right)\sin\frac{\pi}{N}\right\}$$

Using the definitions of DCT-IV and DST-IV the above equation can be written as:

$$= \frac{a_1}{2}\left[\cos\frac{\pi}{N}C(n,k+4)+\sin\frac{\pi}{N}S(n,k+4)+\cos\frac{\pi}{N}C(n,k-4)-\sin\frac{\pi}{N}S(n,k-4)\right]$$

$$= \frac{a_1}{2}\cos\frac{\pi}{N}\left[C(n,k+4)+C(n,k-4)\right]+\frac{a_1}{2}\sin\frac{\pi}{N}\left[S(n,k+4)-S(n,k-4)\right] \qquad (4.40)$$

Simplifying the second term of equation (4.39) yields:

$$a_2\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1} f(x)\cos\left(\frac{4\pi x}{N}\right)\cos\frac{(2x+1)(2k+1)\pi}{4N}$$

$$= a_2\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1} f(x)\cos\left(\frac{16\pi x}{4N}\right)\cos\frac{(2x+1)(2k+1)\pi}{4N}$$

$$= a_2\sqrt{\frac{2}{N}}\frac{1}{2}\sum_{x=0}^{N-1} f(x)\left[\cos\left(\frac{(2x+1)(2k+1)\pi+16\pi x}{4N}\right)+\cos\left(\frac{(2x+1)(2k+1)\pi-16\pi x}{4N}\right)\right]$$

$$= a_2\sqrt{\frac{2}{N}}\frac{1}{2}\sum_{x=0}^{N-1} f(x)\left[\cos\left(\frac{(2x+1)(2k+1+8)\pi - 8\pi}{4N}\right) + \cos\left(\frac{(2x+1)(2k+1-8)\pi + 8\pi}{4N}\right)\right]$$

$$= a_2\sqrt{\frac{2}{N}}\frac{1}{2}\sum_{x=0}^{N-1} f(x)\left\{\cos\left(\frac{(2x+1)(2k+1+8)\pi}{4N}\right)\cos\frac{2\pi}{N} + \sin\left(\frac{(2x+1)(2k+1+8)\pi}{4N}\right)\sin\frac{2\pi}{N}\right.$$

$$\left. + \cos\left(\frac{(2x+1)(2k+1-8)\pi}{4N}\right)\cos\frac{2\pi}{N} - \sin\left(\frac{(2x+1)(2k+1-8)\pi}{4N}\right)\sin\frac{2\pi}{N}\right\}$$

Using the definitions of DCT-IV and DST-IV the above equation can be written as:

$$= \frac{a_2}{2}\left[\cos\frac{2\pi}{N}C(n,k+8) + \sin\frac{2\pi}{N}S(n,k+8) + \cos\frac{2\pi}{N}C(n,k-8) - \sin\frac{2\pi}{N}S(n,k-8)\right]$$

$$= \frac{a_2}{2}\cos\frac{2\pi}{N}\left[C(n,k+8) + C(n,k-8)\right] + \frac{a_2}{2}\sin\frac{2\pi}{N}\left[S(n,k+8) - S(n,k-8)\right] \qquad (4.41)$$

Substituting equations (4.41) and (4.40) in equation (4.39) yields:

$$C_w(n,k) = a_0 C(n,k) + \frac{a_1}{2}\left\{\cos\frac{\pi}{N}\left[C(n,k+4) + C(n,k-4)\right]\right.$$

$$\left. + \sin\frac{\pi}{N}\left[S(n,k+4) - S(n,k-4)\right]\right\}$$

$$+ \frac{a_2}{2}\left\{\cos\frac{2\pi}{N}\left[C(n,k+8) + C(n,k-8)\right]\right.$$

$$\left. + \sin\frac{2\pi}{N}\left[S(n,k+8) - S(n,k-8)\right]\right\} \qquad (4.42)$$

$$\text{for } k = 0,1,\ldots,N-1.$$

Equation (4.42) developed in [22] can be used to calculate the simultaneous windowed updated coefficients of DCT-IV i.e. it depends on both the DST as well as the DCT coefficients. Therefore we need a way to represent DST coefficients in terms of DCT

coefficients only. To do this we consider the non-windowed DCT-IV update equation (2.41) for the particular case of r=1:

$$C(n+1,k) = \cos\frac{(2k+1)\pi}{2N}C(n,k) + \sin\frac{(2k+1)\pi}{2N}S(n,k)$$

$$-\sqrt{\frac{2}{N}}f(n-N)\cos\frac{(2k+1)\pi}{4N} + (-1)^k\sqrt{\frac{2}{N}}f(n)\sin\frac{(2k+1)\pi}{4N} \qquad (4.43)$$

for k=0,1,…,N-1.

Solving equation (4.43) for $S(n,k)$ yields:

$$S(n,k) = \frac{1}{\sin\frac{(2k+1)\pi}{2N}}\left[C(n+1,k) - \cos\frac{(2k+1)\pi}{2N}C(n,k)\right.$$
$$\left. + \sqrt{\frac{2}{N}}f(n-N)\cos\frac{(2k+1)\pi}{4N} - (-1)^k\sqrt{\frac{2}{N}}f(n)\sin\frac{(2k+1)\pi}{4N}\right] \qquad (4.44)$$

for k=0,…,N-1.

While using equation (4.42) to calculate the windowed DCT-IV update, whenever the DST coefficients are needed, they are calculated indirectly using equation (4.44). Hence equations (4.42) and (4.44) can be used simultaneously to calculate the DCT updated windowed coefficients using DCT coefficients only. However, the dependence of equation (4.42) on indices (k+4), (k-4), (k+8) and (k-8) results in indices out of the range given in the definitions of DCT-IV and DST-IV. We use the symmetry properties of DCT-IV and DST-IV, equation (4.45), to extend the range of indices to all integers k:

$$C(n,-k) = C(n,k-1); C(n,k) = C(n,4N-k-1);$$
$$C(n,k) = -C(n,2N-k-1);$$
$$S(n,-k) = -S(n,4N-k-1);$$
$$S(n,k) = S(n,2N-k-1); \qquad (4.45)$$

Appendix A, figure (A.17), lists the C language snippet to calculate the DCT-IV independent windowed update in the presence of Hanning, Hamming and Blackman windows.

Similarly, for formulating the analytics to calculate the DST type-IV independent windowed update we take the DST-IV of equation (4.3) resulting in:

$$S_w(n,k) = a_0 S(n,k) + a_1\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1} f(x)\cos\left(\frac{2\pi x}{N}\right)\sin\frac{(2x+1)(2k+1)\pi}{4N}$$

$$+ a_2\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1} f(x)\cos\left(\frac{4\pi x}{N}\right)\sin\frac{(2x+1)(2k+1)\pi}{4N} \qquad (4.46)$$

for k=0,…,N-1.

Simplifying the second term of equation (4.46) yields:

$$a_1\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1} f(x)\cos\left(\frac{2\pi x}{N}\right)\sin\frac{(2x+1)(2k+1)\pi}{4N}$$

$$= a_1\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1} f(x)\cos\left(\frac{8\pi x}{4N}\right)\sin\frac{(2x+1)(2k+1)\pi}{4N}$$

$$= a_1\sqrt{\frac{2}{N}}\frac{1}{2}\sum_{x=0}^{N-1} f(x)\left[\sin\left(\frac{(2x+1)(2k+1)\pi + 8\pi x}{4N}\right) + \sin\left(\frac{(2x+1)(2k+1)\pi - 8\pi x}{4N}\right)\right]$$

$$= a_1\sqrt{\frac{2}{N}}\frac{1}{2}\sum_{x=0}^{N-1} f(x)\left[\sin\left(\frac{(2x+1)(2k+1+4)\pi - 4\pi}{4N}\right) + \sin\left(\frac{(2x+1)(2k+1-4)\pi + 4\pi}{4N}\right)\right]$$

$$= a_1\sqrt{\frac{2}{N}}\frac{1}{2}\sum_{x=0}^{N-1} f(x)\left\{\sin\left(\frac{(2x+1)(2k+1+4)\pi}{4N}\right)\cos\frac{\pi}{N} - \cos\left(\frac{(2x+1)(2k+1+4)\pi}{4N}\right)\sin\frac{\pi}{N}\right.$$

$$\left. + \sin\left(\frac{(2x+1)(2k+1-4)\pi}{4N}\right)\cos\frac{\pi}{N} + \cos\left(\frac{(2x+1)(2k+1-4)\pi}{4N}\right)\sin\frac{\pi}{N}\right\}$$

Using the definitions of DCT-IV and DST-IV the above equation can be written as:

$$= \frac{a_1}{2}\left[\cos\frac{\pi}{N}S(n,k+4) - \sin\frac{\pi}{N}C(n,k+4) + \cos\frac{\pi}{N}S(n,k-4) + \sin\frac{\pi}{N}C(n,k-4)\right]$$

$$= \frac{a_1}{2}\cos\frac{\pi}{N}\left[S(n,k+4) + S(n,k-4)\right] + \frac{a_1}{2}\sin\frac{\pi}{N}\left[C(n,k-4) - C(n,k+4)\right] \tag{4.47}$$

Simplifying the second term of equation (4.46):

$$a_2\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1} f(x)\cos\left(\frac{4\pi x}{N}\right)\sin\frac{(2x+1)(2k+1)\pi}{4N}$$

$$= a_2\sqrt{\frac{2}{N}}\sum_{x=0}^{N-1} f(x)\cos\left(\frac{16\pi x}{4N}\right)\sin\frac{(2x+1)(2k+1)\pi}{4N}$$

$$= a_2\sqrt{\frac{2}{N}}\frac{1}{2}\sum_{x=0}^{N-1} f(x)\left[\sin\left(\frac{(2x+1)(2k+1)\pi + 16\pi x}{4N}\right) + \sin\left(\frac{(2x+1)(2k+1)\pi - 16\pi x}{4N}\right)\right]$$

$$= a_2\sqrt{\frac{2}{N}}\frac{1}{2}\sum_{x=0}^{N-1} f(x)\left[\sin\left(\frac{(2x+1)(2k+1+8)\pi - 8\pi}{4N}\right) + \sin\left(\frac{(2x+1)(2k+1-8)\pi + 8\pi}{4N}\right)\right]$$

$$= a_2\sqrt{\frac{2}{N}}\frac{1}{2}\sum_{x=0}^{N-1} f(x)\left\{\sin\left(\frac{(2x+1)(2k+1+8)\pi}{4N}\right)\cos\frac{2\pi}{N} - \cos\left(\frac{(2x+1)(2k+1+8)\pi}{4N}\right)\sin\frac{2\pi}{N}\right.$$

$$\left. + \sin\left(\frac{(2x+1)(2k+1-8)\pi}{4N}\right)\cos\frac{2\pi}{N} + \cos\left(\frac{(2x+1)(2k+1-8)\pi}{4N}\right)\sin\frac{2\pi}{N}\right\}$$

Using the definitions of DCT-IV and DST-IV the above equation can be written as:

$$= \frac{a_2}{2}\left[\cos\frac{2\pi}{N}S(n,k+8) - \sin\frac{2\pi}{N}C(n,k+8) + \cos\frac{2\pi}{N}S(n,k-8) + \sin\frac{2\pi}{N}C(n,k-8)\right]$$

$$= \frac{a_2}{2}\cos\frac{2\pi}{N}\left[S(n,k+8) + S(n,k-8)\right] + \frac{a_2}{2}\sin\frac{2\pi}{N}\left[C(n,k-8) - C(n,k+8)\right] \tag{4.48}$$

Substituting equations (4.47) and (4.48) in equation (4.46) results in:

$$S_w(n,k) = a_0 S(n,k) + \frac{a_1}{2}\left\{\cos\frac{\pi}{N}\left[S(n,k+4) + S(n,k-4)\right]\right.$$

$$+\sin\frac{\pi}{N}\left[C(n,k-4)-C(n,k+4)\right]\Big\}$$

$$+\frac{a_2}{2}\Big\{\cos\frac{2\pi}{N}\left[S(n,k+8)+S(n,k-8)\right]$$

$$+\sin\frac{2\pi}{N}\left[C(n,k-8)-C(n,k+8)\right]\Big\} \tag{4.49}$$

for $k=0,1,\ldots,N-1$.

Equation (4.49) developed in [22] calculates the windowed DST-IV update but depends on the DST as well as the DCT coefficients. Therefore we need a way to represent DCT coefficients in terms of DST coefficients only. To do this we consider the non-windowed DST-IV update equation (2.42) for special case of r=1:

$$S(n+1,k)=\cos\frac{(2k+1)\pi}{2N}S(n,k)-\sin\frac{(2k+1)\pi}{2N}C(n,k)$$

$$+\sqrt{\frac{2}{N}}f(n-N)\sin\frac{(2k+1)\pi}{4N}+(-1)^k\sqrt{\frac{2}{N}}f(n)\cos\frac{(2k+1)\pi}{4N} \tag{4.50}$$

Solving equation (4.50) for $C(n,k)$ yields:

$$C(n,k)=\frac{1}{\sin\dfrac{(2k+1)\pi}{2N}}\Bigg[\cos\frac{(2k+1)\pi}{2N}S(n,k)-S(n+1,k)$$

$$+\sqrt{\frac{2}{N}}f(n-N)\sin\frac{(2k+1)\pi}{4N}+(-1)^k\sqrt{\frac{2}{N}}f(n)\cos\frac{(2k+1)\pi}{4N}\Bigg]$$

for k=1,…,N-1. (4.51)

Equations (4.49) and (4.51) can be used to calculate the DST-IV update in the presence of Hamming, Hanning and Blackman windows without using the DCT coefficients. While using equation (4.49) to calculate the windowed DST-IV update, whenever the DCT coefficients are needed, they are calculated indirectly using equation

(4.51). Dependence of equation (4.49) on indices (k+4), (k-4), (k+8) and (k-8) results in indices being out of the range as given in the definitions of DCT-IV and DST-IV. We use the symmetry properties of DCT-IV and DST-IV [22], given by equation (4.45), to extend the range of indices to all integers k. Appendix A, figure (A.20) implements the independent windowed update of DST-IV coefficients for Hamming, Hanning and Blackman windows depending on the values of a_0, a_1 and a_2 as defined in equation (4.2).

CHAPTER 5: HARDWARE IMPLEMENTATION OF INDEPENDENT DCT-II RECTANGULAR WINDOWED UPDATE ALGORITHM

5.1 Introduction to hardware implementation

This chapter deals with the hardware implementation of independent DCT-II rectangular windowed update algorithm developed in Section 2.3.2 of chapter 2. DCT-II is chosen for the hardware implementation because it is the most often used transform and also it is closest to the optimal KLT. Earlier hardware implementation for the simultaneous rectangular windowed update algorithms of reference [22, 23] was reported in reference [31]. This work is extended to implement the independent update algorithms developed in this research. The hardware implementation is carried out for two cases. In the first case one point update at a time i.e. r=1 (section 5.3) in the original data sequence is considered. In the second case, a four point update at a time i.e. r=4 (section 5.4) in the original data sequence is considered. The infinite data sequence is sampled with a rectangular window of size N=8. The hardware developed is capable of accepting this rectangular windowed data and calculating the update as new data points become available. Appendix B lists the VHDL implementation for these two cases. The hardware designs and test benches to test the correctness of the implemented designs were written and tested on ModelSim. The architecture used to implement the independent DCT-II for one point update i.e. r=1 is given in figure (5.4) and for four point update i.e. r=4 is given in figure (5.9). It can be clearly seen that the DCT update does not require the DST coefficients. The controller designed is stimulated with the test vectors and simulations

results for both the architectures are given in this chapter. RMS errors of the values obtained by the implemented hardware and the analytically obtained expected values using MATLAB, are also discussed and analyzed in this chapter for both these architectures. Although the case of rectangular windowed update is discussed in this chapter, however the architecture for other windows as discussed in chapter 3 and 4, namely the split-triangular window, Hamming, Hanning and Blackman windows can be implemented in a similar manner.

5.2 Explanation of the dataflow

Examining equation (2.25), it is clearly evident that to calculate the DST independent DCT update of a sequence we require the DCT of the current time sequence and one older time sequence. These values are computed using the definition of the DCT and stored in the Look-Up Table (LUT). This is called the pre-computation phase or data preparation phase as these values are needed to calculate the modified transform using the update algorithm. First, the input sequence is sampled by a rectangular window, and the sequence of length N is extracted. Then the controller computes the DCT of this sequence using the definition of the DCT, (i.e. DCT_CONVEN module developed in [31]) and saved in a LUT. Thereafter the sequence is shifted by adding the new points and the DCT is again calculated using the DCT_CONVEN module. Now the data preparation phase is over and data is ready for utilizing the independent update algorithm. Whenever the new data points are available, the state machine enters the update mode where the equations developed in sections 5.3 and 5.4 are used to calculate the DST independent DCT update for one point shift and four point shift respectively. Whenever the new data points become available, the state machine remains in update mode and uses

the last two DCT time-step coefficients to calculate the DCT of the modified sequence. The state machine implements this by storing the last two time-step coefficients in the LUT, and these values are read whenever the next update is to be performed. The dataflow diagram of the architecture implemented for both the cases namely one point update at a time and four point update at a time is given in figure (5.4) and figure (5.9) respectively.

The architecture is designed for the input data bit width of 9. Where the MSB bit is the sign bit weighting -1. The hardware implementation is designed to process black and white images where the pixel values range form 0 to 255 [31], where pure black is represented by 0 and pure white is represented by 255. First the pixel values are shifted by subtracting 128 from each value and thereafter normalized by dividing each value by the factor of 128. Therefore the hardware can accept values between (-128/128) i.e. -1, through (127/128) i.e. 0.9921875. The fractional numbers are converted to binary numbers and given as input data points to the design. Appendix C lists the Java functions written to convert the decimal numbers to binary numbers and binary numbers back to decimal numbers.

Based on the analysis of the equations developed in sections 5.3 and 5.4 to implement the architectures of one point update and four point update, the following modules are required.

5.2.1 The 16-bit subtractor (SUB_2_16)

This module performs the subtraction between 2 operands each of 16 bit width. Upon receiving the enable command (EN_SUB) the 16 bits data at A and B get latched and the results become available at output SUB. The schematic of this block is shown in

figure (5.1). It acts as a 2's complement subtractor, the module consists of XOR gates and full adders, whenever two numbers are to be subtracted the XOR gate inverts the bits and S_AB is asserted and added to the inverted bits that yields the subtraction of two 16 bit operands.

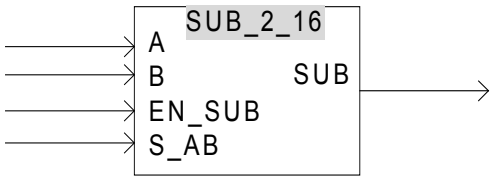

Fig. 5.1: SUB_2_16 Module schematic

5.2.2 Adder Module (ADD_4_16)

This block adds the four 16 bit numbers whenever enable (EN) is asserted. The module consists of full adders in cascaded manner. And the result of the summation goes to the output SUM. Figure (5.2) shows the entity representing this component.

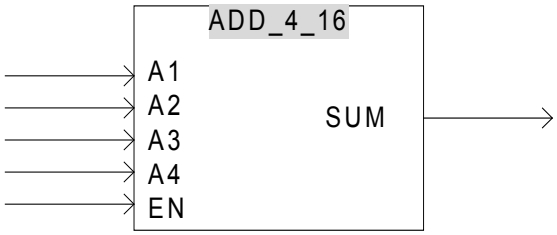

Fig. 5.2: ADD_4_16 Module schematic

5.2.3 Adder Module (ADD_8_16)

The ADD_8_16 module adds 8 numbers of 16 bits width. When the EN signal is asserted, the numbers get latched to the inputs A1 through A8 and the result goes to the output SUM. Figure (5.3) represents the entity ADD_8_16.

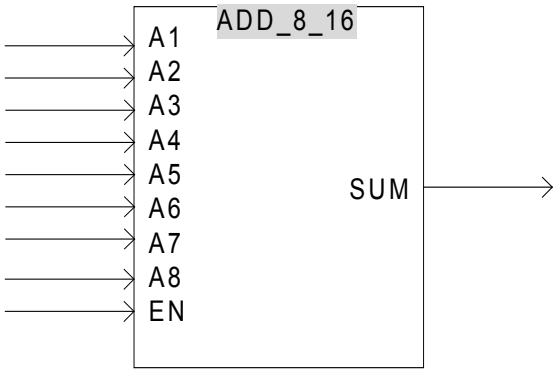

Fig 5.3: ADD_8_16 Module schematic

Other modules used in implementing the two architectures namely the input stage (IP_STAGE), multiplier (MULT), Butterfly stage (BUTTER_FLY) modules, Extension module (EXT_UNIT) developed in [31] are reused in the design.

5.3 Hardware implementation for DCT-II independent one point update algorithm

Let the input data sequence be represented by $f_0, f_1, \ldots, f_n$, and C0 represents the current time-step 0^{th} DCT coefficient, C0M represent the previous time-step 0^{th} DCT coefficient and C0P represents the updated DCT coefficient. Then the equations for DCT independent update can be calculated by substituting r=1, N=8 in equation (2.25) resulting in:

$$C0P = C0 + C0 - C0M + L1*(f_0 - f_8) + L15*(f_1 - f_9) \quad (5.1)$$

$$C1P = L2*\ C1 + L2*\ C1 - C1M + L3*(f_0 + f_8) + L16*(f_1 + f_9) \quad (5.2)$$

$$C2P = L4*\ C2 + L4*\ C2 - C2M + L5*(f_0 - f_8) + L17*(f_1 - f_9) \quad (5.3)$$

$$C3P = L6*\ C3 + L6*\ C3 - C3M + L7*(f_0 + f_8) + L18*(f_1 + f_9) \quad (5.4)$$

$$C4P = L8*(f_0 - f_8) + L19*(f_1 - f_9) - C4M \quad (5.5)$$

$$C5P = L9*\ C5 + L9*\ C5 - C5M + L10*(f_0 + f_8) + L20*(f_1 + f_9) \quad (5.6)$$

$$C6P = L11*\ C6 + L11*\ C6 - C6M + L12*(f_0 - f_8) + L21*(f_1 - f_9) \quad (5.7)$$

$$C7P = L13*\ C7 + L13*\ C7 - C7M + L14*(f_0 + f_8) + L22*(f_1 + f_9) \quad (5.8)$$

Constant	Value	Decimal Value	Binary Value
L1	$1/2\sqrt{2}$	0.3515625	001011010
L2	$\cos(\pi/8)$	0.921875	011101100
L3	$1/2\cos(\pi/16)$	0.48828125	001111101
L4	$\cos(\pi/4)$	0.70703125	010110101
L5	$1/2\cos(\pi/8)$	0.4609375	001110110
L6	$\cos(3\pi/8)$	0.375	001100000
L7	$1/2\cos(3\pi/16)$	0.4140625	001101010
L8	$1/2\cos(\pi/4)$	0.3535	001011010
L9	$\cos(5\pi/8)$	-0.37890625	110011111
L10	$1/2\cos(5\pi/16)$	0.27734375	001000111
L11	$\cos(3\pi/4)$	-0.70703125	101001011
L12	$1/2\cos(3\pi/8)$	0.1875	000110000
L13	$\cos(7\pi/8)$	-0.921875	100010100
L14	$1/2\cos(7\pi/16)$	0.09375	000011000
L15	$-1/2\sqrt{2}$	-0.3515625	110100110
L16	$-1/2\cos(\pi/16)$	-0.48828125	110000011
L17	$-1/2\cos(\pi/8)$	-0.4609375	110001010
L18	$-1/2\cos(3\pi/16)$	-0.4140625	110010110
L19	$-1/2\cos(\pi/4)$	-0.3535	110100101

L20	$-1/2\cos(5\pi/16)$	-0.27734375	110111001
L21	$-1/2\cos(3\pi/8)$	-0.1875	111010000
L22	$-1/2\cos(7\pi/16)$	-0.09375	111101000

Table 2: Binary representation of constants for r=1 point architecture

First the DCT of first eight data points f_0 through f_7 are calculated using the DCT definition and saved in a LUT, thereafter one new data point f_8 is shifted in and oldest data point f_0 is shifted out. The DCT of this shifted sequence is calculated using the definition of the transform and result is saved in the LUT. Thereafter, yet another data point f_9 is shifted in and the update equation for 8 DCT coefficients *C0P* through *C7P* (equations 5.1 through 5.8) derived above are used to calculate the DCT of the modified sequence. As new data points arrive, the hardware iterates the update architecture to calculate the transform of the shifted sequence. RMS errors are calculated between the expected values derived from the DCT definition (matlab values), and the values obtained by the update hardware developed. The formula used to calculate RMS error is given in equation (5.9).

$$\varepsilon = ERROR_{RMS} = \frac{1}{8}\sqrt{\sum_{n=0}^{7}\{E_{DCT}(n) - O_{DCT}(n)\}^2} \tag{5.9}$$

Where E_{DCT} is the expected value of the DCT coefficient and O_{DCT} is the observed value of the DCT from the hardware implementation.

The architecture developed here for computing the independent DCT update can be improved for more accuracy and efficiency. The number of bits used to represent the constant values used can be increased if the application requires more accurate results. Since representing the decimal numbers by 9 bits in binary introduces some error, this

can be reduced if the number or bits are increased. However increasing the bits used to represent the decimal numbers results in space and time overhead in the hardware architecture. Analyzing equation (5.1) shows that whenever (2*C0) needs to be calculate we used adder to speed up the implementation instead of using a multiplier.

The architecture for the split-triangular independent windowed update can be similarly designed. Test bench is used to verify the VHDL controller designed and is given in Appendix B. Test vectors used to stimulate the architecture along with the observed results are given below. Outputs at each steps are listed to understand the implemented design.

The 8 DCT coefficients for the updated sequence are represented by C0P through C7P. The dataflow for the one-point hardware implementation is shows in figure (5.4). Only the C1P coefficient is shown in the dataflow diagram, other coefficients can also be represented in a similar way.

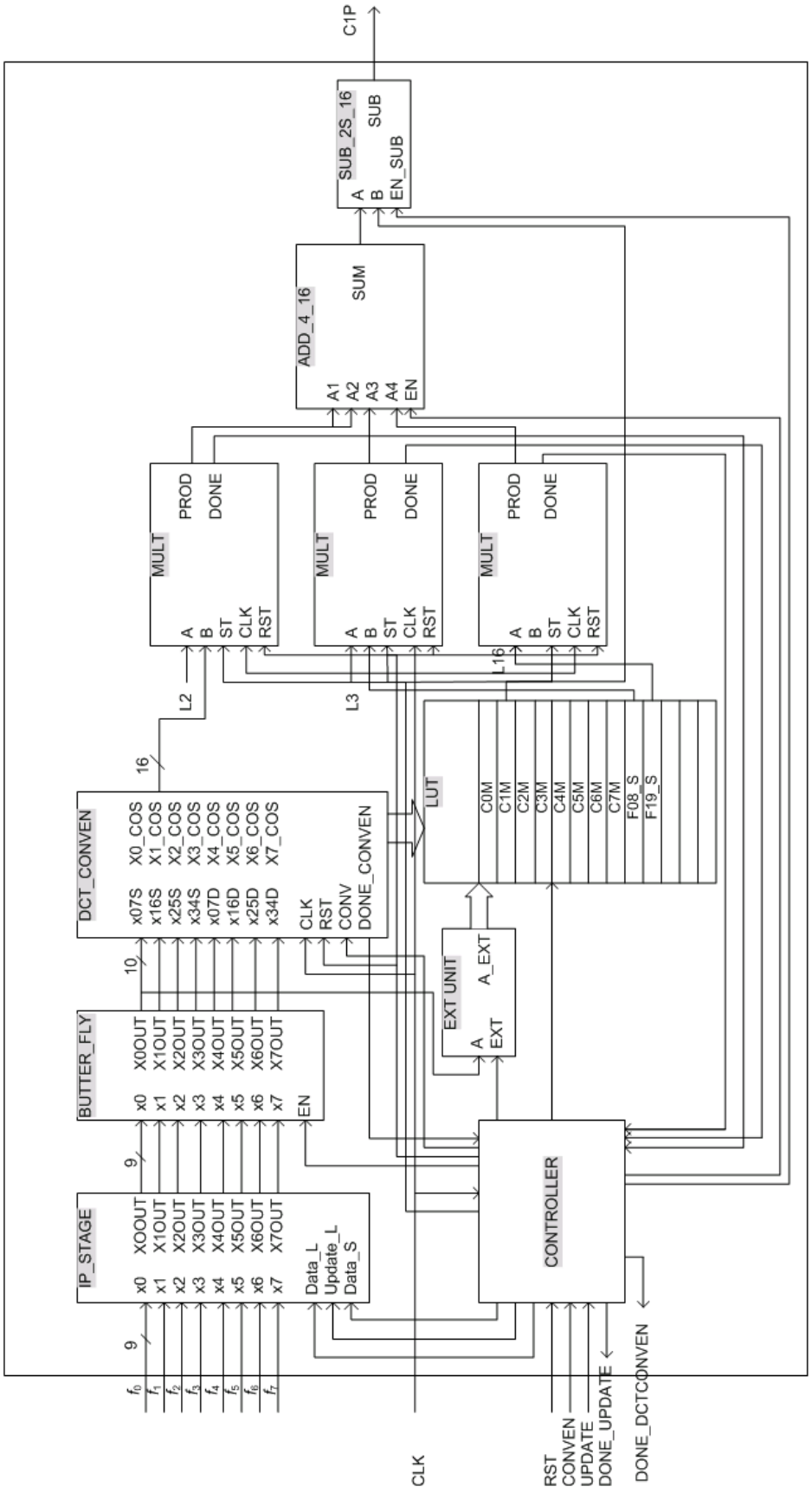

Fig. 5.4 DATAFLOW GRAPH FOR INDEPENDENT DCT-II ONE POINT UPDATE

Step by step explanation of the implemented algorithm for r=1 along with the values obtained, values expected and RMS errors is given below:

Step 1: Computation of DCT of original data sequence (from definition)

	DATA SEQUENCE	Decimal Value	DCT	Decimal Value	Expected Values*	RMS
f_0	111111011	-0.019	0000001010001011	0.156	0.160	
f_1	110100100	-0.359	0000001110000100	0.218	0.222	
f_2	010101100	0.671	1111011011010000	-0.574	-0.572	
f_3	001010101	0.332	1111101010100010	-0.335	-0.336	0.0014
f_4	001101011	0.417	0000000100101101	0.070	0.074	
f_5	101111010	-0.523	0000100001010000	0.519	0.525	
f_6	001010101	0.332	1111110110111000	-0.144	-0.142	
f_7	110011010	-0.398	0000110000111110	0.761	0.767	

* Expected Values calculated from the DCT-II code (definition) from Chapter 2

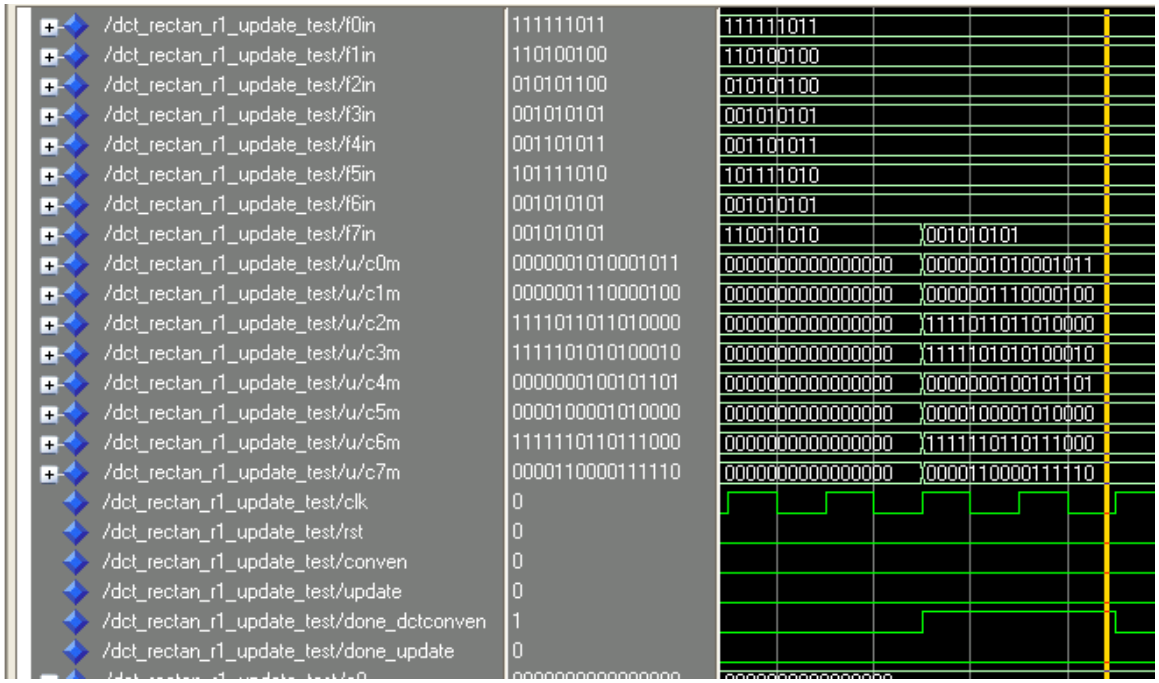

Fig 5.5: The simulation results: C0M through C7M

Step 2: Shifted sequence after 1 point update (DCT calculated from DCT_CONVEN)

	DATA SEQUENCE	Decimal Value	DCT	Decimal Value	Expected Values*	RMS
f_1	110100100	-0.359	0000010010000101	0.281	0.284	0.0016
f_2	010101100	0.671	0000001100011111	0.191	0.197	
f_3	001010101	0.332	1111111101100111	-0.039	-0.038	
f_4	001101011	0.417	1111010110010001	-0.652	-0.652	
f_5	101111010	-0.523	1111100111101010	-0.382	-0.378	
f_6	001010101	0.332	1111101011001110	-0.328	-0.325	
f_7	110011010	-0.398	0000001100011101	0.191	0.195	
f_8	001010101	0.332	1111001011110000	-0.816	-0.825	

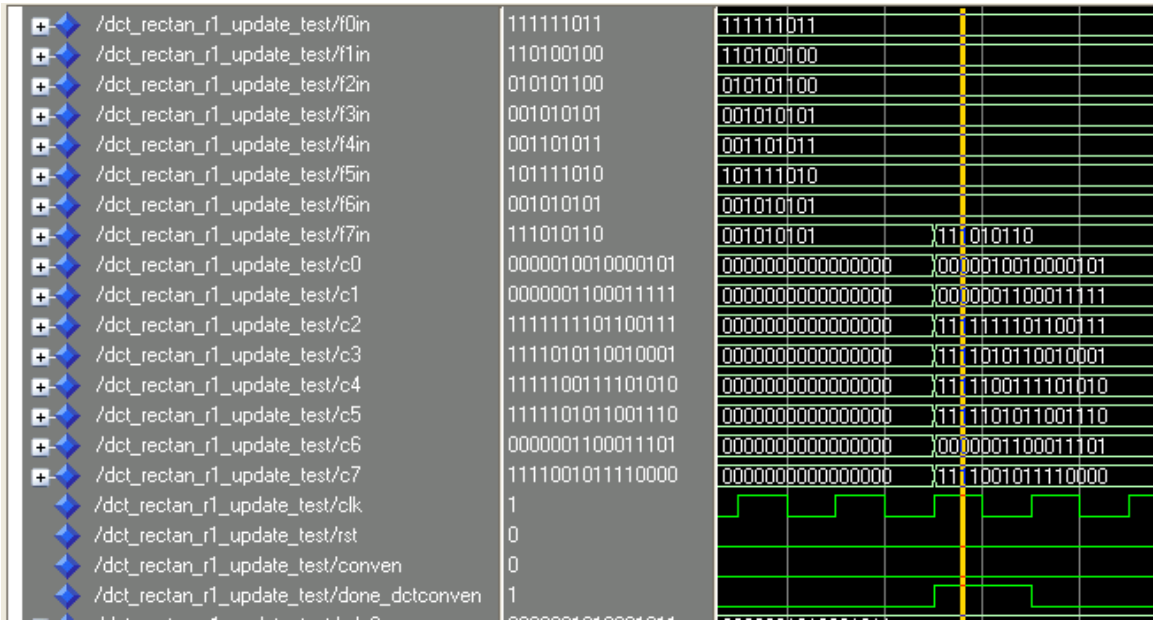

Fig 5.6: The simulation results: C0 through C7

Step 3: Shifted sequence after 1 point update (DCT calculated from independent update module) i.e. coefficients C0P through C7P

	DATA SEQUENCE	Decimal Value	DCT	Decimal Value	Expected Values*	RMS
f_2	010101100	0.671	0000010110011101	0.347	0.353	
f_3	001010101	0.332	0000100011100001	0.554	0.552	
f_4	001101011	0.417	0000011010111101	0.417	0.445	
f_5	101111010	-0.523	0000001100010011	0.191	0.185	0.0046
f_6	001010101	0.332	1111110111110100	-0.128	-0.129	
f_7	110011010	-0.398	1111111101010011	-0.042	-0.044	
f_8	001010101	0.332	1111110101101000	-0.164	-0.164	
f_9	111010110	-0.164	0000110100010111	0.816	0.839	

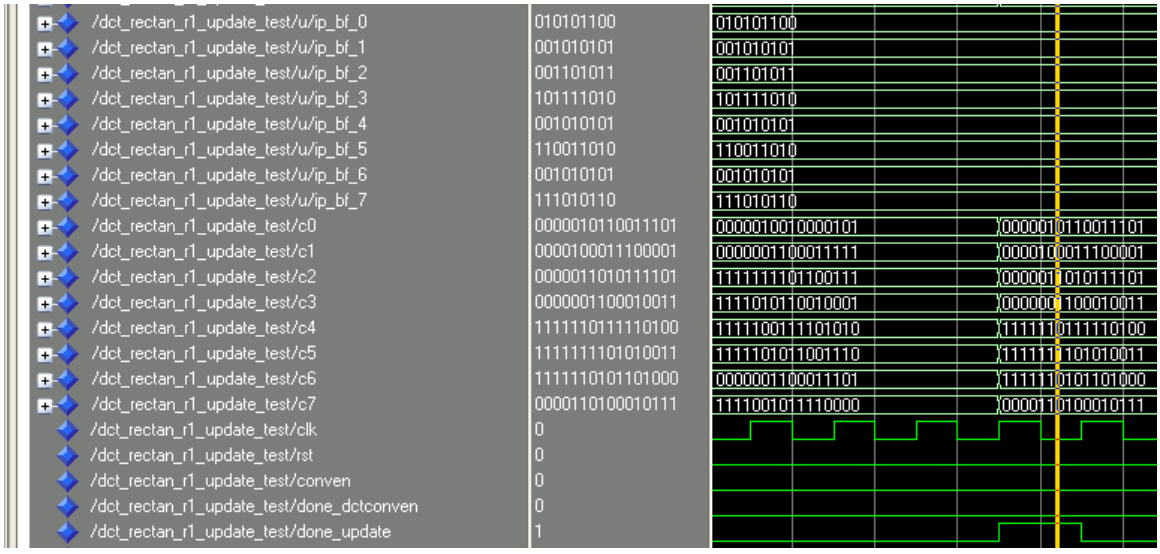

Fig 5.7: The simulation results from update module

Step 4: Yet another 1 point update (DCT calculated from update module second iteration)

	DATA SEQUENCE	Decimal Value	DCT	Decimal Value	Expected Values*	RMS
f_3	001010101	0.332	1111111011010110	-0.074	-0.071	
f_4	001101011	0.417	0000011111011011	0.488	0.498	
f_5	101111010	-0.523	1111111110001111	-0.031	0.023	
f_6	001010101	0.332	0000100001010110	0.519	0.518	0.0080
f_7	110011010	-0.398	1111111000100100	-0.117	-0.115	
f_8	001010101	0.332	0000001011000001	0.171	0.174	
f_9	111010110	-0.164	1111110001011011	-0.230	-0.230	
f_{10}	101111000	-0.531	1111001111101101	-0.757	-0.790	

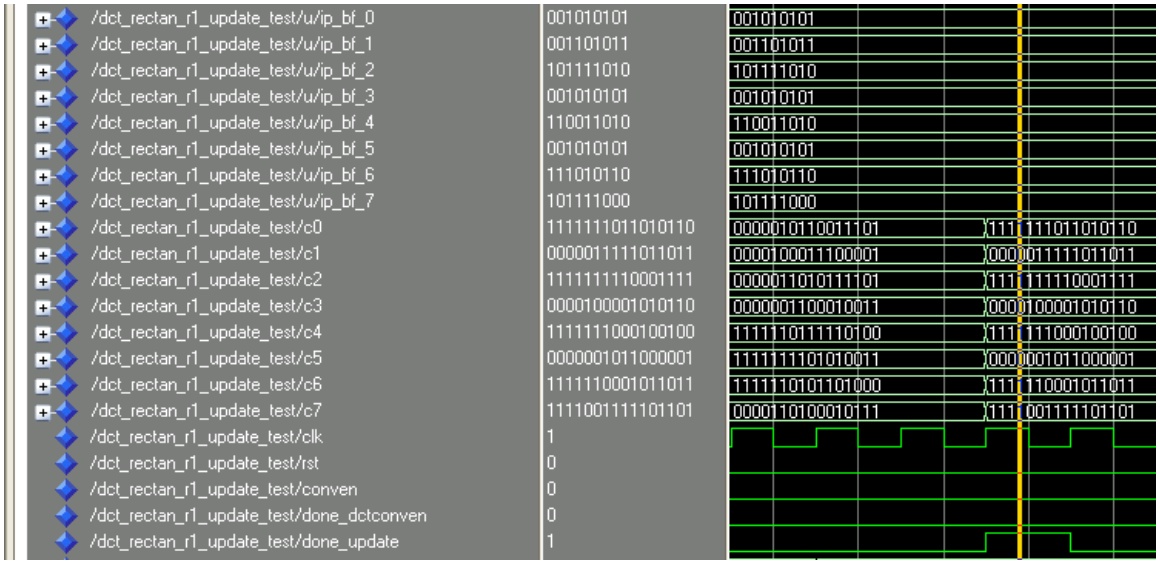

Fig 5.8: The simulation results from second iteration in update module

5.4 Hardware implementation for DCT-II independent four point update algorithm

Similar to the one point independent architecture the DCT-II independent four point update algorithm can be calculated by substituting r=4, N=8 in equation (2.25) resulting in equations (5.10) through (5.17):

$$C0P = C0 + C0 - C0M + L19*(f_7-f_{15}) + L19*(f_6-f_{14}) + L19*(f_5 - f_{13}) + L19*(f_4 - f_{12}) + L1*(f_3 - f_{11}) + L1*(f_2 - f_{10}) + L1*(f_1 - f_9) + L1*(f_0 - f_8) \quad (5.10)$$

$$C1P = - C1 + L2*(f_3 + f_{11}) + L3*(f_2 + f_{10}) + L4*(f_1 + f_9) + L5*(f_0 + f_8) + L6*(f_{15} + f_7) + L7*(f_{14} + f_6) + L8*(f_{13} + f_5) + L9*(f_{12} + f_4) \quad (5.11)$$

$$C2P = - C2 - C2 - C2M + L20*(f_7 - f_{15}) + L21*(f_6 - f_{14}) + L22*(f_5 - f_{13}) + L23*(f_4 - f_{12}) + L20*(f_3 - f_{11}) + L21*(f_2 - f_{10}) + L22*(f_1-f_9) + L23*(f_0 - f_8) \quad (5.12)$$

$$C3P = - C3M + L14*(f_3 - f_{11}) + L15*(f_2 - f_{10}) + L16*(f_1 - f_9) + L4*(f_0 - f_8) + L7*(f_{15} + f_7) + L2*(f_{14} + f_6) + L5*(f_{13} + f_5) + L3*(f_{12} + f_4) \quad (5.13)$$

$$C4P = C4 + C4 - C4M + L17*(f_7 - f_{15}) + L24*(f_6 - f_{14}) + L24*(f_5 - f_{13}) + L17*(f_4 - f_{12}) + L24*(f_3 - f_{11}) + L17*(f_2 - f_{10}) + L17*(f_1 - f_9) + L24*(f_0 - f_8) \quad (5.14)$$

$$C5P = - C5M + L4*(f_3 - f_{11}) + L2*(f_2 - f_{10}) + L15*(f_1 - f_9) + L3*(f_0 - f_8) + L8*(f_{15} + f_7) + L5*(f_{14} + f_6) + L16*(f_{13} + f_5) + L18*(f_{12} + f_4) \quad (5.15)$$

$$C6P = - C6 - C6 - C6M + L21*(f_7 - f_{15}) + L23*(f_6 - f_{14}) + L20*(f_5 - f_{13}) + L22*(f_4 - f_{12}) + L21*(f_3 - f_{11}) + L23*(f_2 - f_{10}) + L20*(f_1 - f_9) + L22*(f_0 - f_8) \quad (5.16)$$

$$C7P = - C7M + L15*(f_3 - f_{11}) + L4*(f_2 - f_{10}) + L14*(f_1 - f_9) + L2*(f_0 - f_8) + L9*(f_{15} + f_7) + L3*(f_{14} + f_6) + L18*(f_{13} + f_5) + L5*(f_{12} + f_4) \quad (5.17)$$

Constant	Value	Decimal Value	Binary Value
L1	$1/2\sqrt{2}$	0.3515625	001011010
L2	$1/2\sin(\pi/16)$	0.09375	000011000
L3	$1/2\sin(3\pi/16)$	0.27734375	001000111
L4	$1/2\sin(5\pi/16)$	0.4140625	001101010
L5	$1/2\sin(7\pi/16)$	0.48828125	001111101
L6	$-1/2\cos(\pi/16)$	-0.48828125	110000011
L7	$-1/2\cos(3\pi/16)$	-0.4140625	110010110
L8	$-1/2\cos(5\pi/16)$	-0.27734375	110111001
L9	$-1/2\cos(7\pi/16)$	-0.09375	111101000
L10	$1/2\cos(\pi/8)$	0.4609375	001110110
L11	$1/2\cos(3\pi/8)$	0.1875	000110000
L12	$1/2\cos(5\pi/8)$	-0.19140625	111001111
L13	$1/2\cos(7\pi/8)$	-0.46484375	110001001
L14	$1/2\sin(3\pi/2)\sin(3\pi/16)$	-0.28125	110111000
L15	$1/2\sin(3\pi/2)\sin(9\pi/16)$	-0.4921875	110000010
L16	$1/2\sin(3\pi/2)\sin(15\pi/16)$	-0.09765625	111100111
L17	$1/2\cos(3\pi/4)$	-0.35546875	110100101
L18	$-1/2\cos(35\pi/16)$	-0.41796875	110010101
L19	$-1/2\sqrt{2}$	-0.3515625	110100110

L20	$-1/2\cos(\pi/8)$	-0.4609375	110001010
L21	$-1/2\cos(3\pi/8)$	-0.1875	111010000
L22	$-1/2\cos(5\pi/8)$	0.19140625	000110001
L23	$-1/2\cos(7\pi/8)$	0.46484375	001110111
L24	$-1/2\cos(3\pi/4)$	0.35546875	001011011

Table 3: Binary representation of constants for *r*=4 point update architecture

The 8 DCT coefficients for the updated sequence are represented by C0P through C7P. The dataflow for the four-point hardware implementation is shows in figure (5.9). Only the C0P coefficient is shown in the dataflow diagram, other coefficients can also be represented in a similar way.

The RMS error values are calculated between the values obtained through the implemented hardware and the expected values obtained using MATLAB. Step 1 consists on latching the initial 8 data points to the architecture and the DCT values are calculated using definition and saved in the LUT. Thereafter the four new data points are shifted in and the DCT calculated using the definition. As four new input data points arrive the controller enters the update mode of operation and using the equations (5.10) to (5.17) and previous DCT values saved in the LUT are used to calculate the updated DCT. Whenever four new data points become available the controller iterates in the update mode of operation and the DCT of the shifted sequence is calculated. The test bench used to stimulate the controller designed to implement the four point update architecture is given in Appendix B. The step-by-step implementation of the controller along with the test vectors and outputs are given in tables below.

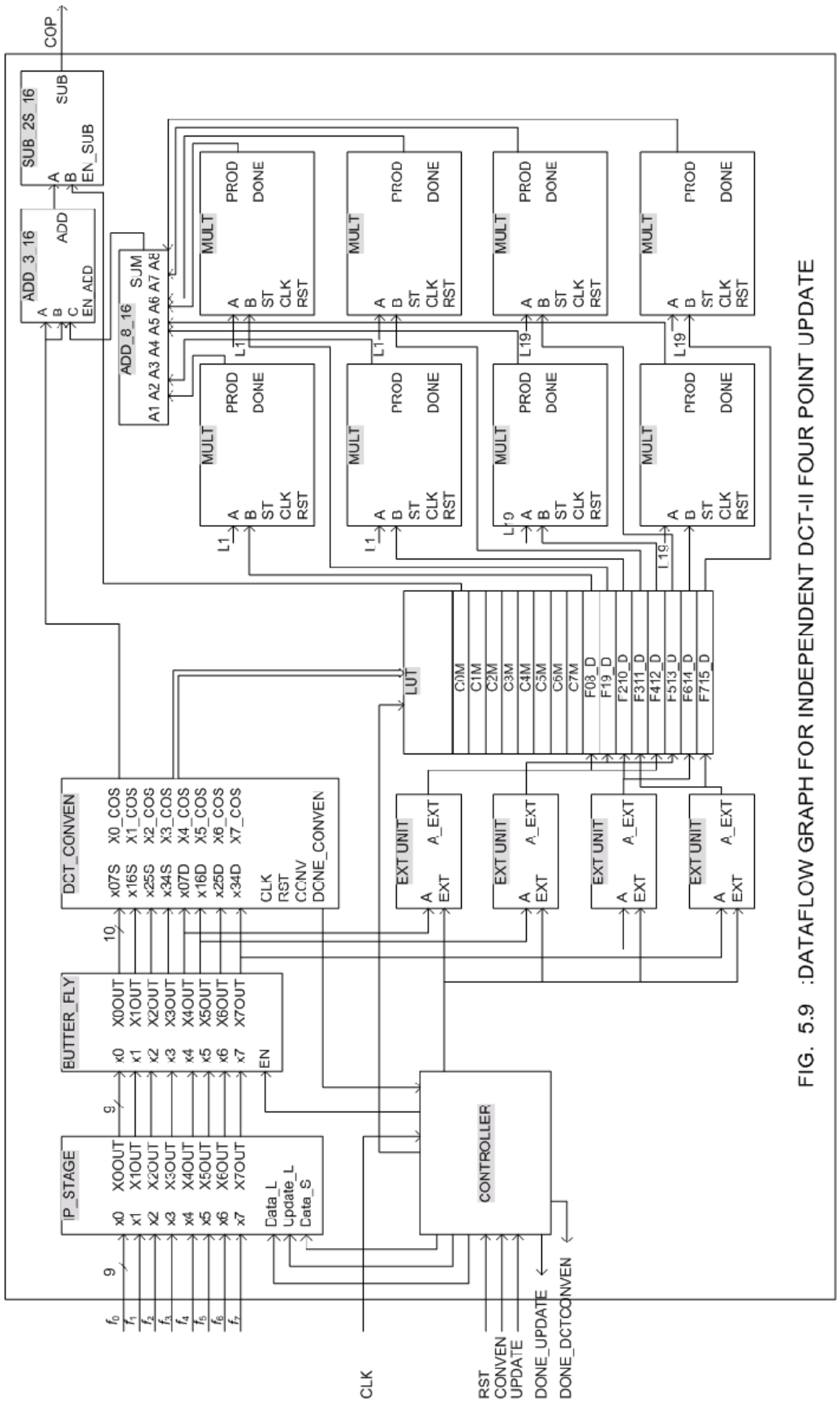

FIG. 5.9 :DATAFLOW GRAPH FOR INDEPENDENT DCT-II FOUR POINT UPDATE

Step by step explanation of the implemented algorithm for r=4 along with the actual values obtained, values expected and RMS errors is given below:

Step 1: Computation of DCT of original data sequence (from definition)

	DATA SEQUENCE	Decimal Value	DCT	Decimal Value	Expected Values*	RMS
f_0	101010010	-0.679	0000011101010100	0.457	0.455	0.00172
f_1	111010110	-0.164	1111100100001010	-0.435	-0.434	
f_2	011010011	0.824	1111010101011001	-0.666	-0.666	
f_3	010101011	0.667	1111100000110100	-0.488	-0.488	
f_4	101100011	-0.613	1111000001110001	-0.972	-0.977	
f_5	011101001	0.910	0000101010010101	0.660	0.671	
f_6	001110101	0.457	0000100000101001	0.509	0.503	
f_7	111100011	-0.113	1111011101001010	-0.544	-0.546	

* Expected values calculated using matlab.

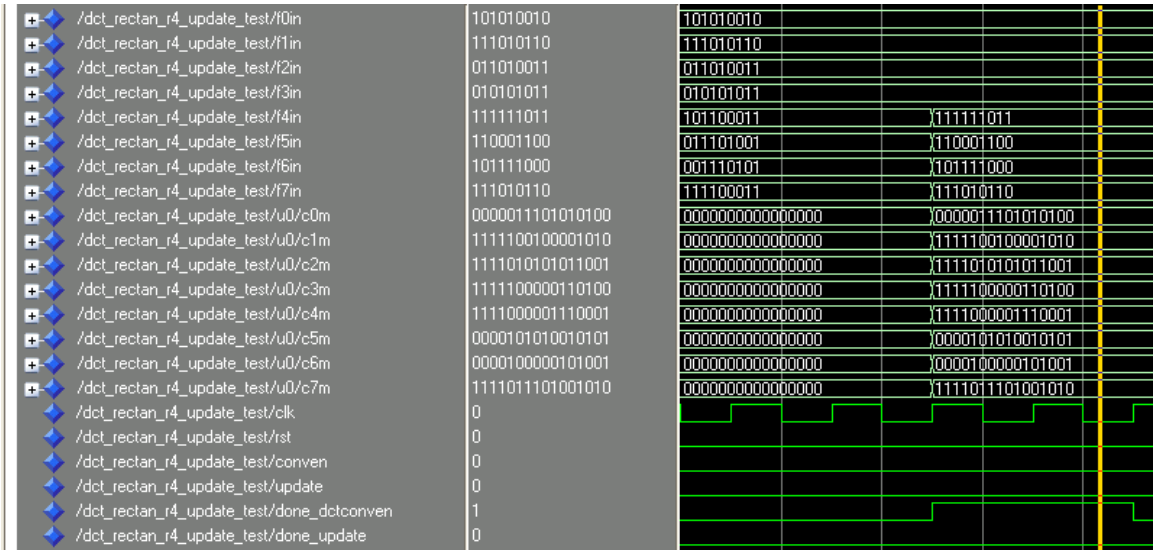

Fig 5.10: ModelSim simulation for Step 1 four point update

Step 2: Data after 4 pt. shift (DCT calculated from DCT_CONVEN module)

	DATA SEQUENCE	Decimal Value	DCT	Decimal Value	Expected Values*	RMS
f_4	101100011	-0.613	1111110100000000	-0.187	-0.186	
f_5	011101001	0.910	0000100111101110	0.619	0.622	
f_6	001110101	0.457	1111110001100101	-0.226	-0.226	
f_7	111100011	-0.113	1111010000010100	-0.746	-0.747	0.00073
f_8	111111011	-0.019	1111100010101011	-0.458	-0.456	
f_9	110001100	-0.453	1111001110000101	-0.781	-0.781	
f_{10}	101111000	-0.531	1111101101001011	-0.294	-0.296	
f_{11}	111010110	-0.164	1111111110100001	-0.023	-0.019	

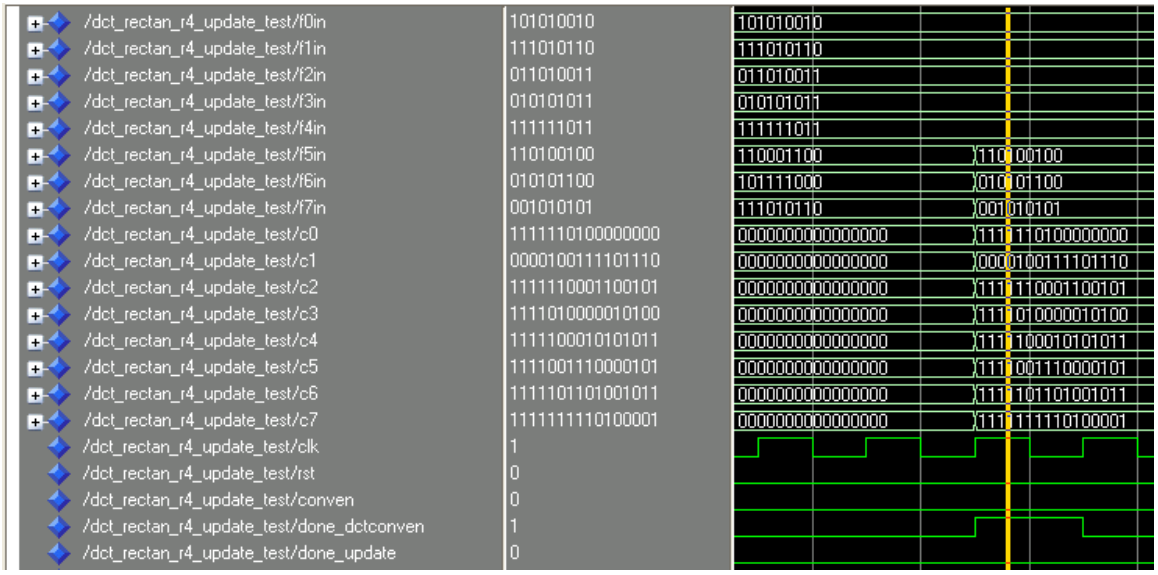

Fig 5.11: Simulation results for Step 2 four point update

Step 3: Shifted sequence after 4 point update (DCT calculated from independent update module) i.e. coefficients C0P through C7P

	DATA SEQUENCE	Decimal Value	DCT	Decimal Value	Expected Values*	RMS
f_8	111111011	-0.019	1111110011001010	-0.201	-0.191	0.002375
f_9	110001100	-0.453	1111010011000101	-0.703	-0.701	
f_{10}	101111000	-0.531	0000011100000111	0.437	0.441	
f_{11}	111010110	-0.164	0000000101001000	0.080	0.088	
f_{12}	111111011	-0.019	0000010001101010	0.275	0.283	
f_{13}	110100100	-0.359	0000011000101000	0.384	0.376	
f_{14}	010101100	0.671	1111100100111011	-0.423	-0.416	
f_{15}	001010101	0.332	0000010001110101	0.277	0.277	

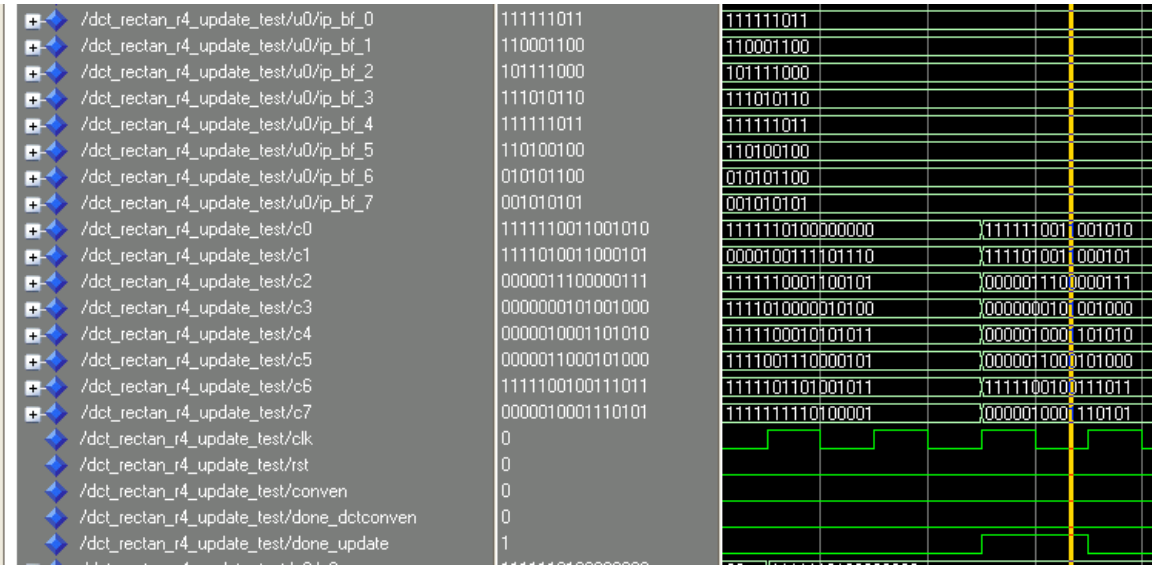

Fig 5.12: The simulation results from update module

Step 4: Yet another 4 pt. shift (from independent update module second iteration)

	DATA SEQUENCE	Decimal Value	DCT	Decimal Value	Expected Values*	RMS
f_{12}	111111011	-0.019	0000001001000010	0.140	0.160	
f_{13}	110100100	-0.359	0000001110001110	0.220	0.222	
f_{14}	010101100	0.671	1111011011000100	-0.578	-0.572	
f_{15}	001010101	0.332	1111101010001100	-0.343	-0.336	0.00400
f_{16}	001101011	0.417	0000000011100010	0.054	0.074	
f_{17}	101111010	-0.523	0000100001000100	0.515	0.525	
f_{18}	001010101	0.332	1111110111010000	-0.136	-0.142	
f_{19}	110011010	-0.398	0000110001001010	0.765	0.767	

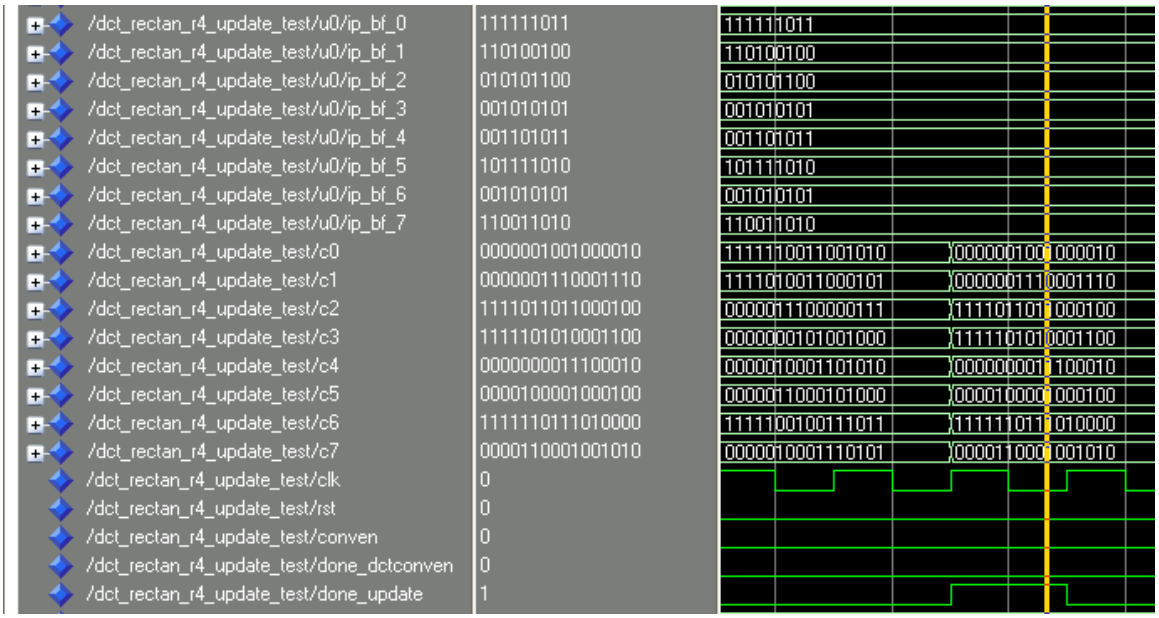

Fig 5.13: The simulation results from update module second iteration

Whenever the DCT computation completes using the update mode of operation the “done_update” flag is asserted.

CHAPTER 6: CONCLUSION AND FUTURE WORK

In this research new fast efficient algorithms are developed that are capable of independently updating the windowed DCT and the DST for a real time input data sequence. The sequence is constantly updated by shifting in the new data points and removing the old data points. Independent algorithms are developed for DCT/DST type-I through IV of which DCT-II and DST-II are the most widely used transforms in the field of signal processing. Update algorithms significantly reduce the computational complexity by calculating the transform of the shifted sequence indirectly rather than using standard fast transform algorithms. It is particularly useful in real time processing of the incoming data as it is computationally more efficient to calculate the DCT of the new shifted sequence using the update algorithm than directly calculating the transform of the new shifted sequence. Update algorithms are widely used in applications such as real-time analysis of financial market data and data compression.

Firstly, the *r*-point independent windowed update algorithms in the presence of rectangular window are derived for DCT and DST type-I through IV. These independent algorithms derived to use only the DCT (DST) coefficients to calculate the updated DCT (DST) coefficients, without using the DST (DCT) coefficients. The commonly used rectangular window to sample the input data sequence results in undesired edge effects such as ringing. These ringing effects can be reduced by applying a more appropriate window such as split-triangular i.e. trapezoidal window or Hamming, Hanning and

Blackman windows. Therefore, the algorithms are presented for independent windowed updates for DCT (DST) coefficients using any of these more desirable windows like split-triangular, Hanning, Hamming and Blackman windows. The algorithms developed in this research are an extension of the previously developed update algorithms as they do not require the simultaneous update of both the DCT coefficients and the DST coefficients. Also, these algorithms constitute an easier implementation as compared with the previous algorithms as we do not require to retain both the DCT coefficients and DST coefficients. The DCT independent update algorithms developed in this research utilize the DCT coefficients of the current time sequence and one previous time sequence, new input data points and the old data points to calculate the transform of the modified sequence independent of the corresponding discrete sine coefficients. These algorithms are particularly useful for the applications where only the DCT coefficients or only the DST coefficients are available. The independent update algorithms developed in this research are capable of updating r new data points at a time for the case of rectangular windowed update and one-point for the case of split triangular, Hamming, Hanning and Blackman windows. However if more than one point update is required these algorithms can be repeated r times to calculate the r-point update. All algorithms developed are of computational order N, whereas calculating the transform via fast DCT/DST algorithms is of order $N \log_2 N$.

Software implementations are presented to verify the analytically derived algorithms. C language functions for the windowed update in the presence of rectangular, trapezoidal, Hamming, Hanning and Blackman windows for DCT and DST type-I through IV are included in Appendix A. These C language implementation is interactive

and prompts the user to specify the values of sequence length N, and the new data points. Utilizing these new data points, the transform coefficients of the shifted sequence are computed using both the update algorithm and using the definitions. RMS error between these values is computed using the update algorithm and the fast transform definition to validate the update algorithm.

State machines for the hardware implementation are written in VHDL for the independent DCT-II windowed update in the presence of rectangular window. DCT-II was chosen for the implementation because it is the most commonly used transform. Hardware implementation for other types of DCT and DST can be easily written in a similar way. Hardware implementation was carried out and tested for the case of one point update at a time i.e. r=1, and four point update at a time i.e. r=4. Both the state machines were fully implemented and simulated on ModelSim for testing the hardware design implementation. The VHDL controller presented can be divided into two main parts. The first part prepares data for update transform where the DCT coefficients of the current and one previous time sequences are calculated using the definitions, and the second part codes the independent update algorithm which uses the previously obtained values to find the modified DCT of the updated sequence. The VHDL controller written to implement the one-point and four-point independent update algorithms along with the test benches used are included in Appendix B. Fixed point arithmetic is used to represent the data points as it results in easier hardware implementation. Although the window size selected for implementation was N=8, a larger window size can be easily implemented using the similar architecture. The constants used in the update equations were implemented in 9 bit format and stored in LUT's. Thereafter, the performance evaluation

in terms of RMS errors between the DCT-II update coefficients obtained using the implemented hardware and the expected values from the transform definition (using matlab) is carried out.

Suggestions for future work include deriving the independent update algorithm i.e. the DCT-only algorithms for ODCT (Odd DCT's) and DST-only independent algorithms for ODST (Odd DST's) type-I through IV. These odd DCT's and odd DST's are also sometimes referred to as type-V to VIII DCT's and DST's. The update algorithms for ODCT's and ODST's for rectangular window can be derived initially and thereafter the idea can be extended to calculate the update in the presence of more appropriate windows like trapezoidal, Hamming, Hanning and Blackman windows. Also the performance evaluation for other available windows such as Bartlett window, Bartlett-Hann, Kaiser, Tukey, Flat top window etc. can be studied. The performance evaluation for each of the windowed update algorithm can be analyzed and the complexity can be compared to see the effectiveness of each window.

To extend the work further the hardware implementation for the case of independent update for DCT and DST in the presence of split-triangular window, Hamming, Hanning and Blackman windowed can be implemented. These can be implemented with some minor modifications to the rectangular windowed hardware implementation for DCT-II developed in this research work. Hardware implementation can be carried out for different lengths of windows and the performance of each of them can be compared to find the optimal length of the window sequence that should be used for the practical applications. The efficiency of the design in terms of the area requirements and power consumption can be further enhanced by using more efficient

multipliers. Also, the bit lengths to represent the coefficients can be increased for more accurate results, for which we have to reach a compromise between efficiency on one hand and space and speed requirements on the other.

REFERENCES

[1] V. Britanak, P. C. Yip, K. R. Rao, "Discrete Cosine and Sine Transforms: General Properties, Fast Algorithms and Integer Approximations", Academic Press 2006.

[2] Ray W.D., Driver, R.M. “Further Decomposition of the Karhunen-Loéve Series Representation of a Stationary Random Process”, IEEE Trans., 1970, IT-16, pp 12-13.

[3] Clarke R.J. “Performance of Karhunen-Loéve and Discrete Cosine Transforms for data having widely varying values of inter-sample correlation coefficient”, Electronics Letters, 1983, Vol.19 Issue:7, pp 251-253.

[4] Clarke R.J. “On the relation between the Karhunen-Loéve and cosine transforms”, IEEE Proc. F, Communication., Radar & Signal Process, 1981, 128, (6), pp 359-360.

[5] Clarke R.J. “Relation between the Karhunen-Loéve and sine transforms”, IEEE Proc. F, Commun., Radar & Signal Process, 1984, Vol.20 Issue:1, pp 12-13.

[6] A.K. Jain, "A sinusoidal family of unitary transforms," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-1, October 1979, pp. 356-365.

[7] N. Ahmed, T. Natarajan, and K.R. Rao, "Discrete cosine transform," IEEE Trans. Comput., vol. C-23, pp. 90-94, Jan. 1974.

[8] N.R. Murthy and M.N.S. Swamy, "On the computation of running discrete Cosine and Sine transforms," IEEE Trans. Signal Processing, vol. 40, pp. 1430-1437, June 1992.

[9] P.Yip and K.R. Rao, "On the shift properties of DCT's and DST's," IEEE Trans. Signal Processing, vol. 35, pp. 404-406, Mar.1987.

[10] B.G. Sherlock and D.M. Monro, “Moving Fast Fourier Transform”, IEEE Proceedings., Vol. 139, No. 4, August 1992, pp 279-282.

[11] B.G. Sherlock, “Windowed Discrete Fourier Transform for Shifting Data”, Signal Processing, 1999, Vol.74, pp 169-177.

[12] B.G. Sherlock and Y.P. Kakad, “Rapid update of discrete Fourier transform for real-time signal processing,” Proceedings of SPIE, Volume 4379, pp.490-500, Orlando, Florida, April 2001.

[13] N.R. Murthy and M.N.S. Swamy, “Efficient algorithms for the computation of running discrete cosine and sine transforms,” Circuits and Systems, IEEE International Sympoisum. pp. 634-637 vol. 1, 1991.

[14] S.C. Chan and K.L. Ho, "Direct methods for computing discrete sinusoidal transforms," IEE Proceedings-Radar and Signal Processing, Vol. 137, No. 6, 1990, pp. 433-442.

[15] Y.H. Chan and W.C. Siu, "Mixed-radix discrete cosine transform", IEEE Transactions on Signal Processing, 1993, Vol.41, No. 11,pp 3157-3161.

[16] A.N. Skodras and A.G. Constantinides, "Radix-3 fast discrete cosine transform algorithm," Proceedings of the International Conference on Digital Signal Processing, Limassol, Cyprus, 1995, pp. 819-824.

[17] Hassan El-Banna, Alaa A. EL-Fattah. "An Efficient Implementation of the ID DCT using FPGA Technology", IEEE computer society, pp 356-360 2004.

[18] C.S.Choy, W.K. Cham and L.Lee, "An LSI implementation of integer cosine transform," Proceedings of the International Conference on Circuits and Communication Systems (ICCS'88), Singapore, November 1988, pp. 17.5.1-17.5.5.

[19] T.C. Tan, G. Bi and H.N. Tan, "Fast recursive algorithms for 2-D discrete cosine transform," Signal Processing, Vol. 80, No. 9, 2000, pp. 1917-1935.

[20] J. Canaris, "A VLSI architecture for real time computation of discrete trigonometric transforms," Journal of VLSI Signal Processing, Vol. 5, 1993, pp. 95-104.

[21] Proakis J.G., Manolakis D.G.,"Digital Signal Processing", Prentice Hall International Inc. 1996.

[22] B.G. Sherlock and Y.P. Kakad, "Windowed discrete cosine and sine transforms for shifting data," Signal Processing, vol. 81, Issue 7, pp. 1465-1478, July 2001.

[23] B.G. Sherlock, Y.P. Kakad, "Transform domain technique for windowing the DCT and DST," Journal of the Franklin Institute, vol. 339, Issue 1, pp. 111-120, April 2002.

[24] W.K. Cham and Y.T. Chan., "Integer discrete cosine transforms," Proceedings of the International Symposium on Signal Processing, Theories, Implementations, and Applications (ISSPA'87), Brisbane, Australia, August 1987, pp. 674-676.

[25] W.K.Cham and P.C.Yip, "Integer sinusoidal transforms for image processing," International Journal of Electronics, Vol. 70, 1991, pp. 1015-1030.

[26] Z. Wang, "Pruning the fast discrete cosine transform", IEEE Transactions on Communications, 1991, Vol.39, No. 5,pp 640-643.

[27] A.N. Skodras, "Fast discrete cosine transform pruning," IEEE Transactions on Signal Processing, Vol. 42, No. 7, 1994, pp. 1833-1837.

[28] Pei S.C. and Yeh M.H., "The discrete fractional cosine and sine transforms," IEEE Transactions on Signal Processing, Vol. 49, No. 6, 2001, pp. 1198-1207.

[29] Klappenecker A. and Rotteler M., "The fractional discrete cosine transform," IEEE Transactions on Signal Processing, Vol. 50, No. 4, 2002, pp. 902-911.

[30] Jiangtao Xi, J. F. Chicharo, "Computing Running DCT's and DST's Based on Their Second-Order Shift Properties," IEEE Trans. on Circuits and Systems-I, vol. 47, No. 5, 779-783, May 2000.

[31] Hazem Bassam Alassaly, “A Hardware Implementation and analysis of the Discrete Cosine Transform Update Algorithms” 2005, PhD Dissertation, UNC-Charlotte.

[32] N.C. Geckinli, D. Yavuz, "Some novel windows and a concise tutorial comparison of window families," IEEE Trans. Acoust., Speech Signal Process. ASSP vol. 26, Issue 6, pp. 501-507, June 1978.

[33] B.G. Sherlock and Y.P. Kakad, "Real Time Rapid Update of Type-I DCT and DST for Moving Window," 8th International Conference on Advances in Communications and Control (COMCON 8), Crete, Greece, 25 - 29 June 2001.

APPENDIX A: C FUNCTIONS FOR INDEPENDENT DCT/DST UPDATE ALGORITHMS

This appendix lists the C language functions used to test the DCT/DST I-IV windowed update algorithms derived in Chapter 2, 3 and 4. The implementation of the update algorithms in the presence of rectangular, split-triangular, Hamming, Hanning and Blackman windows are given below. The simultaneous windowed update C function originally given in [22, 33] are modified to implement the independent windowed update algorithms derived in this research.

/* Program to calculate *r*-point update for any given signal length *N* and using independent updating of DCT type-I coefficients in the presence of the rectangular window */

```
void update_transform_I(double C[], double Cold[], int N, double
                        fnew[], double fold[], int r)

/*     C[]        input      DCT of the previous time unit sequence
       C[]        output     Updated DCT calculated using DCT values of
                             older sequences
       N          input      the input data sequence length
       r          input      the number of new points to shift in
       fnew[]     input      the previous data point shifted in
       fold[]     input      most resent data point shifted out    */

{
   double k;
   int u, m1tu;
   double theta;
   double costheta;
   double sintheta;
   double root2bN;
   double p1, p2, p3;
   double t, coeff;
   int m, x;
   double root2,term1,sine,cosine,angle;
   double Csave;

   root2bN=sqrt(2.0/N);
   root2=sqrt(2.0);
   m1tu= -1;

   for(u=0;u<=N;u++){
       theta = r*u*M_PI/N;
```

```
        costheta = cos(theta);
        sintheta = sin(theta);
        m1tu= -m1tu;
        p1=0.0;  p2=0.0; p3=0.0;

             for(m=0; m<=(r-1); m++) {
                   angle=((m+1)*u*M_PI)/N;
                   sine=sin(angle);
                   cosine=cos(angle);

                   p1 += (fold[r-m-1] - m1tu*fnew[r-m-1])*sine*sintheta;
                   p2 += ((m1tu*fnew[2*r-m-1]) - fold[2*r-m-1]
                         +(costheta*fold[r-m-1])
                         -(costheta*m1tu*fnew[r-m-1]))*cosine;
                   }
                   p3 = ((((1.0/root2)-1.0)*fold[0])
                         +(m1tu*(1.0/root2)*fnew[0])
                         +(2.0*(1.0-(1.0/root2))*costheta*fold[r])
                         -(m1tu*root2*costheta*fnew[r])
                         +(((1.0/root2)- 1.0)*fold[2*r])
                         +(m1tu*(1.0/root2)*fnew[2*r]));

        term1= root2bN*(p1 + p2 + p3);
        if((u%N)==0) { term1 /= root2; }       // multiply term1 by k
        Csave=C[u];
        t=(sintheta*sintheta);
        if((u%N)==0) { t=t/root2; }
        coeff= (costheta*costheta+t);
        C[u]=((2.0*costheta*C[u]) - (coeff*Cold[u]) + term1);
        Cold[u]=Csave;
      }
}
```

Fig. A.1 C function to calculate *r*-point rectangular windowed update for DCT-I

/* Program to calculate DCT type-I windowed update in the presence of split-triangular window */

```
void update_windowed_I(double Cw[], double Cwold[], double Cm[], double
                       Cmold[], int N, int n0, double fwnew[], double
                       fwold[], double fmnew[], double fmold[], double
                       fn0, double fNmn0, double fN, double fm1, double
                       fn0m1, double fNmn0p1, int r){

        Variables used:
        Cw[]  0..N-1      Input - DCT1 of old windowed data
                          Output- DCT1 of new windowed data
        Cm[]  0..N-1      Input - DCT1 of old data_times_m
                          Output- DCT1 of new data_times_m
        N                 signal and window length
        n0                tail length of split-triangular window
        fwnew[] 0..r-1    new data shifted in times old window
```

```
     fwold[] 0..r-1    oldest data shifted out times old window
     fmnew[] 0..r-1    new data shifted in times old m
     fmold[] 0..r-1    r old data points times old m
     fm1               the number fold1[0];
     fn0m1             the number f[n0-1]
     fNmn0m1           the number f[N-n0-1]
     fn0               the number f[n0]
     fNmn0             the number f[N-n0]
     fN                the number f[N]  (= fnew[0])
     r                 number of new data points coming in
     Cmold[0..N-1]     Input: DCT of data_times_m from previous
                       timestep
                       Output:A copy of Cm as input
     Cwold[0..N-1]     Input: DCT of windowed data from previous time
                       step
                       Output: A copy of Cw as input
    */

int m1tu;
double root2bNbn0,theta;
double Cmcorrection, Cold_correction;
int u,pk;
double *rec;

rec = (double *)calloc(N+1,sizeof(double));
if (rec==NULL){
    printf("Unable to allocate array\n");
    exit(1);
}
if(r!=1){
  printf("Update_windowed_I works for r=1, not r=%d\n",r);
    exit(1);
}
root2bNbn0 = ((sqrt(2.0/N))/((double)n0));
N= (double)N;

m1tu=1;
for(u=0; u<=N; u++){
     theta= (u*M_PI)/N;
     if (n0==1){
     pk=1.0/sqrt(2.0);
    }
    else pk=1;
    Cold_correction = root2bNbn0*(-(fn0m1*cos(n0*theta))
                                  -(pk*fNmn0*m1tu*cos((n0-1)*theta))
                                  +(2.0*fm1*(1.0/sqrt(2.0))));

    if ((u==0)||(u==N)) {Cold_correction /= sqrt(2.0);}
    Cmold[u] += Cold_correction;
        // Adds correction factor to Cmold to get DCT of f(x-1)m(x-1)
    Cwold[u] += Cmold[u];
    m1tu = -m1tu;
}

IDCT1(Cmold, rec, N);
pntarray("Before update_transform a c f(x-1)m(x-1) Cmold is =", rec,
         N+1);
```

```
  update_transform_I(Cw, Cwold, N, fwnew, fwold, r);
                                        // Unwindowed update
  update_transform_I(Cm, Cmold, N, fmnew, fmold, r) ;
                                        // Unwindowed update

  m1tu=1;
  for(u=0; u<=N; u++){
      theta=(u*M_PI)/N;
      if (n0==1){ pk=1.0/sqrt(2.0);}
      else pk=1;
      Cmcorrection = root2bNbn0*( -pk*fn0*cos((n0-1)*theta)
                                  -fNmn0p1*m1tu*cos(n0*theta)
                                  +2.0*fN*m1tu*(1.0/sqrt(2.0)) );

      if ((u==0)||(u==N))Cmcorrection /= sqrt(2.0);
      Cm[u] += Cmcorrection;
      Cw[u] += Cm[u];
      m1tu = -m1tu;
      IDCT1(Cm, rec, N);
      pntarray("Signal C(X+1) X M(X) is =", rec, N+1);

  }
}
```

Fig. A.2 C function implementing DCT-I split-triangular windowed update

/* Program to calculate *r*-point update for any given signal length *N* and using independent updating of DST type-I coefficients in the presence of the rectangular window */

```
void update_transform_I(double S[], double Sold[], double sig[],
                        int N, double fnew1[], double fnew2[],
                        double fold1[], double fold2[], int r)

/*      S[]         input       DST of the previous time unit sequence
        S[]         output      Updated DST calculated using DST values of
                                older sequences
        sig[]       input       the input data points
        N           input       the input data sequence length
        r           input       the number of new points to shift in
        fnew1[]     input       the previous data point shifted in
        fnew2[]     input       the newest data point shifted in
        fold1[]     input       most resent data point shifted out
        fold2[]     input       the older data point shifted out      */

{
   double k;
   int u, m1tu;
   double theta;
   double costheta;
```

```
    double sintheta;
    double root2bN;
    double p1, p2, p3, p4;
    double t, coeff;
    int m, x;
    double root2,term1,sine,cosine,angle;
    double Ssave;

    root2bN=sqrt(2.0/N);
    root2=sqrt(2.0);
    m1tu= -1;
    for(u=0;u<=N;u++){
        theta = r*u*M_PI/N;
        costheta = cos(theta);
        sintheta = sin(theta);
        m1tu= -m1tu;
        p1=0.0; p2=0.0; p3=0.0; p4=0.0;

    for(m=0; m<=(r-1); m++){
          angle=((m+1)*u*M_PI)/N;
          sine=sin(angle);
          cosine=cos(angle);

          p1 += (fold1[r-1-m] - m1tu*fnew1[r-1-m])*cosine*sintheta;
          p2 += (fold2[r-m-1] -(m1tu*fnew2[r-m-1])+
                (costheta*m1tu*fnew1[r-1-m])-
                (costheta*fold1[r-1-m]))*sine;
          p3 = ((((1.0/root2)-1.0)*fold1[0])*costheta*sintheta +
               ((m1tu*(1.0/root2)*fnew1[0])*costheta*sintheta)+
               ((1.0-(1.0/root2))*sintheta*fold2[0])
                -(m1tu*(1.0/root2)*sintheta*fnew2[0]));
          p4 = (((1.0-(1.0/root2))*fold1[0])*costheta*sintheta -
               ((m1tu*(1.0/root2)*fnew1[0])*costheta*sintheta)-
               ((1.0-(1.0/root2))*sintheta*fold2[0])
               +(m1tu*(1.0/root2)*sintheta*fnew2[0]));
 }
        term1= (p1+p3);
        if((u%N)==0) { term1 /= root2; }         // multiply term1 by k
        term1= root2bN*(term1 + p2 + p4);
        Ssave=S[u];

        t=(sintheta*sintheta);
        if((u%N)==0) { t=t/root2; }
        coeff= (costheta*costheta+t);
        S[u]=((2.0*costheta*S[u]) - (coeff*Sold[u]) + term1);
        Sold[u]=Ssave;
  }
}
```

Fig. A.3 C function to calculate *r*-point rectangular windowed update for DST-I

/* Program to calculate DST type-I windowed update in the presence of split-triangular window */

```
void update_windowed_I(double Sw[], double Swold[], double Sm[],
                       double Smold[], int N, int n0, double fwnew1[],
                       double fwnew2[], double fwold1[],
                       double fwold2[], double fmnew1[],
                       double fmnew2[], double fmold1[],
                       double fmold2[], double fn0, double fNmn0,
                       double fN, double fm1, double fn0m1, double
                       fNmn0m1, double fw[], double fm[], int r){
/*
       Variables used:
       Sw[]  0..N-1      Input – DST1 of old windowed data
                         Output- DST1 of new windowed data
       Sm[]  0..N-1      Input – DST1 of old data_times_m
                         Output- DST1 of new data_times_m
       N                 signal and window length
       n0                tail length of split-triangular window
       fwnew1[] 0..r-1   last but one data to be shifted in times old
                         window
       fwnew2[] 0..r-1   new data shifted in times old window
       fwold1[] 0..r-1   oldest data shifted out times old window
       fwold2[] 0..r-1   last old data shifted out times old window
       fmnew1[] 0..r-1   last but one data to be shifted in times old m
       fmnew2[] 0..r-1   new data shifted in times old m
       fmold1[] 0..r-1   oldest data shifted times old m
       fmold2[] 0..r-1   r old data points times old m
       fm1               the number fold1[0]
       fn0m1             the number f[n0-1]
       fNmn0m1           the number f[N-n0-1]
       fn0               the number f[n0]
       fNmn0             the number f[N-n0]
       fN                the number f[N]  (= fnew[0])
       r                 number of new data points coming in
       Smold[0..N-1]     Input - DST of data_times_m from previous
                         timestep
                         Output - A copy of Sm as input
       Swold[0..N-1]     Input - DST of windowed data from previous
                         timestep
                         Output - A copy of Sw as input
                                  */

  int m1tu;
  double *rec;
  double twoN,root2bNbn0, theta;
  double Smcorrection, Sold_correction;
  int u;
    rec = (double *)calloc(N+1,sizeof(double));
     if (rec==NULL){
       printf("Unable to allocate array\n");
       exit(1);
     }
    if(r!=1){
      printf("Update_windowed_I works for r=1, not
```

```
            r=%d\n",r);
      exit(1);
     }
  root2bNbn0 = ((sqrt(2.0/(double)N))/((double)n0));
  m1tu=1;
  for(u=0; u<=N; u++)
  {
       theta= ((u*M_PI)/(double)N);
       Sold_correction = root2bNbn0*(-fn0m1*sin(n0*theta)
                                     +fNmn0m1*m1tu*sin(n0*theta));
            // Correction factor to get f(x-1)m(x-1) from f(x-1)m(x)
            // Correction factor to get f(x-1)w(x-1) from f(x-1)w(x)

       Smold[u] += Sold_correction;
         // Adds correction factor to Smold2 to get DST of f(x-1)m(x-1)
       Swold[u] += Smold[u];
       m1tu = -m1tu;
  }

 update_transform_I(Sw,Swold,fw,N,fwnew1,fwnew2,fwold1,fwold2,r);
                      // Unwindowed update
 update_transform_I(Sm,Smold,fm,N,fmnew1,fmnew2,fmold1,fmold2,r);
                      // Unwindowed update

 m1tu=1;
 for(u=0; u<=N; u++){
       theta = ((u*M_PI)/(double)N);
       Smcorrection = root2bNbn0*(-fn0*sin((n0-1)*theta)
                                  +fNmn0*m1tu*sin((n0+1)*theta)
                                  -2.0*fN*m1tu*sin(theta));
       Sm[u] += Smcorrection;
       Sw[u] += Sm[u];
       m1tu = -m1tu;
  }
}
```

Fig. A.4 C function implementing DCT-I split-triangular windowed update

/* Program to calculate *r*-point rectangular windowed update for any given signal length *N* and using independent updating of DCT-II coefficients */

```
void update_transform(double C[], double Cold[], int N, double fnew1[],
                      double fnew2[], double fold1[], double fold2[],
                      int r)

/*     C[]        input      DCT of the previous time unit sequence
       C[]        output     Updated DCT calculated using DCT values of
                             older sequences
       N          input      the input data sequence length
       r          input      the number of new points to shift in
       fnew1[]    input      the previous data point shifted in
```

```
        fnew2[]    input      the newest data point shifted in
        fold1[]    input      most resent data point shifted out
        fold2[]    input      the older data point shifted out      */

{
   double k;
   int u;
   int m1tu;
   double theta;
   double costheta;
   double sintheta;
   double root2bN;
   double p1, p2, p3;
   double t1, t2;
   int m;
   int x;
   double root2,term1,sine,cosine,angle;
   double Csave;

   root2bN=sqrt(2.0/N);
   root2=sqrt(2.0);
   m1tu= -1;
   for(u=0;u<=N-1;u++){
       theta = r*u*M_PI/N;
       costheta = cos(theta);
       sintheta = sin(theta);
       m1tu= -m1tu;

       p1=0.0;  p2=0.0; p3=0.0; t1=0.0;
       t2=0.0;
       for(m=0; m<=(r-1); m++){
         angle= (m+m+1)*u*M_PI/(N+N);
         sine=sin(angle);
         cosine=cos(angle);

         t1 = m1tu*fnew1[r-1-m] - fold1[r-1-m];
         t2 = m1tu*fnew2[r-1-m] - fold2[r-1-m];
         p1 -= (t1)*sine;
         p2 += (t2)*cosine;
         p3 += (t1)*cosine;
       }
       term1= root2bN*(sintheta*p1 + p2 - costheta*p3);
       if((u%N)==0) term1 /= root2;         // multiply term1 by k
       Csave=C[u];
       C[u]=2.0*costheta*C[u] - Cold[u] + term1;
       Cold[u]=Csave;
  }
}

void update_signal(double sig[], int N, double fnew1[], double vnew[],
                 double fnew2[], double fold1[], double fold2[], int r)
{
  int x;

  for(x=0;x<=(r-1);x++)
       {
             fold1[x]=fold2[x];
```

```
        }
  for(x=0;x<=(r-1);x++)
        {
              fold2[x]=sig[x];
        }
  for(x=0;x<=(r-1);x++)
        {
              fnew1[x]=fnew2[x];
        }
  for(x=0;x<=(r-1);x++)
        {
              fnew2[x]=vnew[x];
        }
  for(x=r;x<=N-1;x++)
        {
             sig[x-r] = sig[x];
        }
  for(x=0; x<=r-1; x++)
        {
             sig[N-r+x]=vnew[x];
        }
}


void dctrect()
{
  int N,r;
  int i;
  double *sig;
  double *C;
  double *Cold;
  double *rec;
  double *fnew1;
  double *fnew2;
  double *fold1;
  double *fold2;
  double *vnew;
  int x;
  double rmsC;

  printf("\n");
  printf("Enter signal length N:");
  scanf("%d",&N);
  printf("\n");

  sig = (double *)calloc(N,sizeof(double));
  if (sig==NULL)
        {
              printf("Unable to allocate array\n");
              exit(1);
        }    //initialise signal
  for (x=0;x<=N-1;x++)
        {
              sig[x] = x+1;
        }
   sig[2]=42.0;
```

```
printf("\n");
for (x=0;x<=N-1;x++)
       {
             printf("Original Signal sig : %d: %lf\n",x,sig[x]);
       }
printf("\n");

printf("Enter r, the number of new data points to shift in (0 to
        %d):",N-1);
scanf("%d",&r);
printf("\n");

C = (double *)calloc(N+1,sizeof(double));
if (C==NULL)
       {
             printf("Unable to allocate array\n");
             exit(1);
       }

Cold = (double *)calloc(N+1,sizeof(double));
if (Cold==NULL)
       {
             printf("Unable to allocate array\n");
             exit(1);
       }

rec = (double *)calloc(N,sizeof(double));
    if (rec==NULL)
       {
             printf("Unable to allocate array\n");
             exit(1);
       }

fnew1 = (double *)calloc(r, sizeof(double));
    if (fnew1==NULL)
          {
                printf("Unable to allocate array \n");
                exit(1);
          }

fnew2 = (double *)calloc(r, sizeof(double));
if (fnew2==NULL)
          {
                printf("Unable to allocate array \n");
                exit(1);
          }
fold1 = (double *)calloc(r, sizeof(double));
if (fold1==NULL)
          {
                printf("Unable to allocate array \n");
                exit(1);
          }
fold2 = (double *)calloc(r, sizeof(double));
if (fold2==NULL)
          {
                printf("Unable to allocate array \n");
                exit(1);
```

```
          }

vnew = (double *)calloc(r, sizeof(double));
if (vnew==NULL)
          {
                printf("Unable to allocate array \n");
                exit(1);
          }

for(x=0;x<=r-1;x++)
          {
                printf("Enter the %d th new point ",x);
                scanf("%lf",&vnew[x]);
          }
printf("\n");
for (x=0;x<=r-1;x++)
          {
                printf("\n");
                printf("vnew signal:  vnew[%d]= %lf",x,vnew[x]);
          }
     printf("\n");

printf("Initializing State Begin to calculate Cold and C:\n");
printf("_____________________________________________________\n");

DCT2(sig,Cold,N);

pntarray("DCT using definition Cold is=", Cold, N);

update_signal(sig, N, fnew1, vnew, fnew2, fold1, fold2, r);

for (x=0;x<=N-1;x++)
          {
             printf("\nAfter %d point update sig is %d: %lf",r,
             x,sig[x]);
          }

printf("\n");

DCT2(sig,C,N);

printf("\n");

pntarray("DCT using definition C is=", C, N);

printf("Initializing complete\n");
printf("_____________________________________________\n");
while(1==1)
 {
      for(x=0;x<=r-1;x++)
          {
                printf("Enter the %d th new point ",x);
                scanf("%lf",&vnew[x]);
          }
      printf("\n");
      for (x=0;x<=r-1;x++)
          {
```

```
                printf("\n");
                printf("vnew signal:  vnew[%d]= %lf",x,vnew[x]);
          }
       printf("\n");

       update_signal(sig,N,fnew1, vnew, fnew2,fold1, fold2,r);

       update_transform(C, Cold, N, fnew1, fnew2, fold1, fold2, r);

       for (x=0;x<=N-1;x++)
            {
               printf("After %d point update sig is %d:
                       %lf\n",r,x,sig[x]);
            }
       printf("\n");

       pntarray("DCT using update transform is",C,N);

       IDCT2(C,rec,N);

       printf("Signal recovered from update transform DCT coefficients
               is");
       printf("\n");
       rmsC=0.0;

       for (x=0;x<=N-1;x++)
             {
                   printf("%d: %lf\n",x,rec[x]);
                   rmsC += (sig[x]-rec[x])*(sig[x]-rec[x]);
             }

       rmsC=sqrt(rmsC);
       printf("\n");
       printf("RMS error on reconstruction from DCT =%lf\n",rmsC);

       printf("Enter %d new points to shift in\n",r);
       printf("\n");

 }

  free(fnew1);
  free(fnew2);
  free(fold1);
  free(fold2);
  free(vnew);
  free(sig);
  free(Cold);
  free(C);
  free(rec);
}
```

Fig. A.5 C program to calculate *r*-point rectangular windowed update for DCT-II

/* Program to calculate DCT type-II windowed update in the presence of split-triangular window */

```
void update_windowed_II(double Cw[], double Cwold[], double Cm[],
                        double Cmold[],int N, int n0, double fwnew1[],
                        double fwnew2[], double fwold1[],
                        double fwold2[], double fmnew1[], double
                        fmnew2[], double fmold1[], double fmold2[],
                        double fn0, double fNmn0, double fN, double
                        fm1, double fn0m1, double fNmn0m1,int r){

/*    Split-triangular windowed update for DCT-II
      Window length N with tail-length n0.


  Variables used:
       Cw[]  0..N-1      Input - DCT2 of old windowed data
                         Output- DCT2 of new windowed data
       Cm[]  0..N-1      Input - DCT2 of old data_times_m
                         Output- DCT2 of new data_times_m
       N                 signal and window length
       n0                tail length of split-triangular window
       fwnew1[] 0..r-1   last but one data to be shifted in times old
                         window
       fwnew2[] 0..r-1   new data shifted in times old window
       fwold1[] 0..r-1   oldest data shifted out times old window
       fwold2[] 0..r-1   last old data shifted out times old window
       fmnew1[] 0..r-1   last but one data to be shifted in times old m
       fmnew2[] 0..r-1   new data shifted in times old m
       fmold1[] 0..r-1   oldest data shifted times old m
       fmold2[] 0..r-1   r old data points times old m
       fm1               the number fold1[0];
       fn0m1             the number f[n0-1]
       fNmn0m1           the number f[N-n0-1]
       fn0               the number f[n0]
       fNmn0             the number f[N-n0]
       fN                the number f[N]  (= fnew[0])
       r                 number of new data points coming in
       Cmold[0..N-1]     Input: DCT of data_times_m from previous
                         timestep
                         Output:A copy of Cm as input
       Cwold1[0..N-1]    Input: DCT of windowed data from previous time
                         step
                         Output: A copy of Cw as input
      */

  int m1tu;
  double twoN, root2bNbn0, theta;
  double Cmcorrection, Cold_correction;
  int u;

  if(r!=1){
    printf("Update_windowed_II works for r=1,
              r=%d\n",r);
```

```
      exit(1);
  }
  root2bNbn0 = sqrt(2.0/(double)N)/(double)n0;
  twoN= (double)N+N;

  m1tu=1;
  for(u=0; u<=N-1; u++){
    theta= u*M_PI/twoN;
    Cold_correction = root2bNbn0*(  -fn0m1*cos((n0+n0+1)*theta)
                                    -fNmn0m1*m1tu*cos((n0+n0-1)*theta)
                                    +2.0*fm1*cos(theta));

    if (u==0) Cold_correction /= sqrt(2.0);
    Cmold[u] += Cold_correction;    // Adds correction factor to Cmold
                                       to get DCT of f(x-1)m(x-1)
    Cwold[u] += Cmold[u];
    m1tu = -m1tu;
  }

    pntarray("fwnew1 in V = ",fwnew1,r);
    pntarray("fwnew2 = ",fwnew2,r);
    pntarray("fwold1 = ",fwold1,r);
    pntarray("fwold2 in V = ",fwold2,r);

  update_transform_II(Cw, Cwold, N, fwnew1, fwnew2, fwold1, fwold2, r);
                      // Unwindowed update
  update_transform_II(Cm, Cmold, N, fmnew1, fmnew2, fmold1, fmold2, r);
                      // Unwindowed update

  pntarray("Cw after update_transform is =", Cw, N);
  pntarray("Cm after update_transform is =", Cm, N);

  m1tu=1;
  for(u=0; u<=N-1; u++){
    theta=u*M_PI/twoN;
      Cmcorrection = root2bNbn0*( -fn0*cos((n0+n0-1)*theta)
                                  -fNmn0*m1tu*cos((n0+n0+1)*theta)
                                  +2.0*fN*m1tu*cos(theta) );

      if ((u==0)||(u==N))Cmcorrection /= sqrt(2.0);
      Cm[u] += Cmcorrection;
      Cw[u] += Cm[u];
      m1tu = -m1tu;
  }
}

void apply_windows( int N, int r, double w[], double m[],
                    double f[], double fw[], double fm[],
                    double fnew1[], double fnew2[],
                    double fold1[], double fold2[],
                    double fwnew1[], double fwnew2[],
                    double fwold1[], double fwold2[],
                    double fmnew1[], double fmnew2[],
                    double fmold1[], double fmold2[] ){

   /*  Variables used:
```

```
        N length of the signal                                    (input)
        r                  number of new data points              (input)
        w[0..N-1]          the window function                    (input)
        m[0..N-1]          the m-function                         (input)
        f[0..N-1]          the input signal                       (input)
        fw[0..N-1]         the windowed signal                    (output)
        fm[0..N-1]         the m-ed signal                        (output)
        fnew1[0..r-1]      last but one data to be shifted in     (input)
        fnew2[0..r-1]      new data shifting in                   (input)
        fold1[0..r-1]      oldest data shifting out, two time domains
                           earlier                                (input)
        fold2[0..r-1]      last old data shifting out             (input)
        fwnew1[] 0..r-1    last but one data to be shifted in times old
                           window                                 (output)
        fwnew2[] 0..r-1    new data shifted in times old window   (output)
        fwold1[] 0..r-1    oldest data shifted out times old window
                                                                  (output)
        fwold2[] 0..r-1    last old data shifted out times old window
                                                                  (output)
        fmnew1[] 0..r-1    last but one data to be shifted in times old m
                                                                  (output)
        fmnew2[] 0..r-1    new data shifted in times old m        (output)
        fmold1[] 0..r-1    oldest data shifted times old m        (output)
        fmold2[] 0..r-1    r old data points times old m          (output)
  */
  int x;

  for(x=0;x<=N-1;x++){
    fw[x]= f[x]*w[x];
    fm[x]= f[x]*m[x];
  }

  for(x=0;x<=r-1;x++){
      fwold1[x] = w[N-1]*fold1[x];
      fwold2[x] = w[x]*fold2[x];
      fmold1[x] = m[N-1]*fold1[x];
      fmold2[x] = m[x]*fold2[x];

      fwnew1[x] = w[N-1]*fnew1[x];
      fwnew2[x] = w[x]*fnew2[x];
      fmnew1[x] = m[N-1]*fnew1[x];
      fmnew2[x] = m[x]*fnew2[x];
}
```

Fig. A.6 C program to calculate split-triangular windowed update for DCT-II

/* Program to calculate DST coefficients indirectly for Hamming, Hanning and Blackman windows */

```
for(m=0;m<=N;m++){
  if((m%2) == 0)    {  Sold1[N]=Sold1[N]+ sig[m]; }
  else              {  Sold1[N]=Sold1[N]- sig[m]; }
  }
```

```
  Sold1[N]=1.0/sqrt(N)*Sold1[N];

 m1tu=-1;
 for(u=1;u<=N-1;u++){
         if((u%N)==0)  { k = 1.0/sqrt(2.0);  }
         else          { k=1.0;  }

         theta = (u*M_PI)/N;
         costheta = cos(theta);
         sintheta = sin(theta);
         costhetaby2=cos(theta/2.0);
         Sold1[u]=Cupdate[u]-(costheta*Cold1[u])+
                  k*costhetaby2/root_Nv2*(fold2[0]- m1tu*fnew2[0]);
         Sold1[u]=Sold1[u]/sintheta;
         m1tu=-m1tu;
}
```

Fig. A.7 C code snippet to calculate independent windowed update for DCT-II in presence of Hanning, Hamming and Blackman windows

/* Program to calculate *r*-point rectangular windowed update for any given signal length *N* for DST-II */

```
void update_transform(double S[], double Sold[], int N,  double
                      fnew1[], double fnew2[], double fold1[], double
                      fold2[], int r)

/*     S[]        input      DST of the previous time unit sequence
       S[]        output     Updated DST calculated using DST values of
                             older sequences
       N          input      the input data sequence length
       r          input      the number of new points to shift in
       fnew1[]    input      the previous data point shifted in
       fnew2[]    input      the newest data point shifted in
       fold1[]    input      most resent data point shifted out
       fold2[]    input      the older data point shifted out      */

{
   double k;
   int u;
   int m1tu;
   double theta;
   double costheta;
   double sintheta;
   double root2bN;
   double p1, p2, p3, t1, t2;
   int m;
   int x;
   double root2,term1,sine,cosine,angle;
   double Ssave;
```

```
   root2bN=sqrt(2.0/N);
   root2=sqrt(2.0);
   m1tu= -1;
   for(u=0;u<=N;u++){
       theta = r*u*M_PI/N;
       costheta = cos(theta);
       sintheta = sin(theta);
       m1tu= -m1tu;
       p1=0.0;  p2=0.0;  p3=0.0;  t1=0.0;  t2=0.0;

       for(m=0; m<=(r-1); m++){
         angle= (m+m+1)*u*M_PI/(N+N);
         sine=sin(angle);
         cosine=cos(angle);
         t1= (fold1[r-1-m] - m1tu*fnew1[r-1-m]);
         t2= (fold2[r-1-m] - m1tu*fnew2[r-1-m]);
         p1 += (t1)*cosine;
         p2 += (t2)*sine;
         p3 += (t1)*sine;
         }
         term1= root2bN*(sintheta*p1 + p2 - costheta*p3);
         if((u%N)==0) term1 /= root2;          // multiply term1 by k
         Ssave=S[u];
         S[u]=2.0*costheta*S[u] - Sold[u] + term1;
         Sold[u]=Ssave;
   }
}
```

Fig. A.8 C program to calculate *r*-point rectangular windowed update for DST-II

/* Program to calculate DST type-II windowed update in the presence of split-triangular window */

```
void update_windowed_II(double Sw[], double Swold[], double Sm[],
                        double Smold[], int N, int n0,
                        double fwnew1[], double fwnew2[], double
                        fwold1[], double fwold2[], double fmnew1[],
                        double fmnew2[], double fmold1[], double
                        fmold2[], double fn0, double fNmn0, double fN,
                        double fm1, double fn0m1, double fNmn0m1,
                        int r){

       Variables used:
       Sw[]  0..N-1      Input - DST2 of old windowed data
                         Output- DST2 of new windowed data
       Sm[]  0..N-1      Input - DST2 of old data_times_m
                         Output- DST2 of new data_times_m
       N                 signal and window length
       n0                tail length of the window used
       fwnew1[] 0..r-1   last but one data to be shifted in times old
                         window
       fwnew2[] 0..r-1   new data shifted in times old window
       fwold1[] 0..r-1   oldest data shifted out times old window
```

```
      fwold2[] 0..r-1   last old data shifted out times old window
      fmnew1[] 0..r-1   last but one data to be shifted in times old m
      fmnew2[] 0..r-1   new data shifted in times old m
      fmold1[] 0..r-1   oldest data shifted times old m
      fmold2[] 0..r-1   r old data points times old m
      fm1               the number fold1[0]
      fn0m1             the number f[n0-1]
      fNmn0m1           the number f[N-n0-1]
      fn0               the number f[n0]
      fNmn0             the number f[N-n0]
      fN                the number f[N]  (= fnew[0])
      r                 number of new data points coming in
      Smold[0..N-1]     Input - DST of data_times_m from previous
                        timestep
                        Output - A copy of Sm as input
      Swold[0..N-1]     Input - DST of windowed data from previous
                        timestep
                        Output - A copy of Sw as input
     */

 int m1tu;
 double twoN, root2bNbn0, theta;
 double Smcorrection, Sold_correction;
 int u;

 if(r!=1){
     printf("Update_windowed_II works for r=1, not
             r=%d\n",r);
     exit(1);
}

 root2bNbn0 = sqrt(2.0/(double)N)/(double)n0;
 twoN= (double)N+N;

 m1tu=1;
 for(u=1; u<=N+1; u++)
 {
      theta= u*M_PI/twoN;
      Sold_correction = root2bNbn0*(-fn0m1*sin((n0+n0+1)*theta)
                                    -fNmn0m1*m1tu*sin((n0+n0-1)*theta)
                                    +2.0*fm1*sin(theta));
           // Correction factor to get f(x-1)m(x-1) from f(x-1)m(x)
           // Correction factor to get f(x-1)w(x-1) from f(x-1)w(x)

      if (u==0||u==N){  Sold_correction /= sqrt(2.0);  }

      Smold[u] += Sold_correction;
        // Adds correction factor to Smold2 to get DST of f(x-1)m(x-1)
      Swold[u] += Smold[u];
      m1tu = -m1tu;
 }

 update_transform(Sw, Swold, N, fwnew1, fwnew2, fwold1, fwold2, r);
                                  // Unwindowed update
 update_transform(Sm, Smold, N, fmnew1, fmnew2, fmold1, fmold2, r) ;
                                  // Unwindowed update
```

```
  m1tu=1;
  for(u=1; u<=N; u++){            // later, N
      theta = u*M_PI/twoN;
      Smcorrection = root2bNbn0*( -fn0*sin((n0+n0-1)*theta)
                                  -fNmn0*m1tu*sin((n0+n0+1)*theta)
                                  +2.0*fN*m1tu*sin(theta)  );

      if (u==0||u==N){  Smcorrection /= sqrt(2.0);  }
            Sm[u] += Smcorrection;
            Sw[u] += Sm[u];
      m1tu = -m1tu;
  }
}
```

Fig. A.9 C function to calculate split-triangular windowed update for DST-II

/* Code snippet to calculate DCT-II coefficients indirectly from DST-II coefficients for Hanning, Hamming and Blackman windows */

```
for(m=0;m<=N-1;m++){
  Cold1[0]=Cold1[0]+sig[m];
  }
  Cold1[0]=1.0/sqrt(N)*Cold1[0];

 for(u=1;u<=N-1;u++){
         if((u%N)==0)  { k = 1.0/sqrt(2.0);  }
         else { k=1.0;  }
         theta = (u*M_PI)/N;
         costheta = cos(theta);
         sintheta = sin(theta);
         sinthetaby2=sin(theta/2.0);
         m1tu=pow(-1,u);
         Cold1[u]=(costheta*Sold1[u])-
                   Supdate[u]+(((1/root_Nv2)*k*sinthetaby2)*(fold2[0]-
                   m1tu*fnew2[0]));
         Cold1[u]=Cold1[u]*((1.0)/sintheta);
}
```

Fig. A.10 C code snippet to calculate independent windowed update for DST-II in presence of Hanning, Hamming and Blackman windows

/* Program to calculate *r*-point rectangular windowed update for any given signal length *N* for DCT-III */

```
void update_transform(double C[], double Cold[], int N, double fnew[],
                      double fold[], int r)
```

```
/*     C[]        input      DCT of the previous time unit sequence
       C[]        output     Updated DCT calculated using DCT values of
                             older sequences
       N          input      the input data sequence length
       r          input      the number of new points to shift in
       fnew[]     input      the newest data point shifted in
       fold[]     input      the older data point shifted out      */

{
   double k;
   int u;
   int m1tu;
   double root2bN;
   double p1, p2, p3;
   int m;
   int x;
   double root2,term1,sine,cosine,angle,angle2,Fr,Er;
   double Csave;

   root2bN=sqrt(2.0/N);
   root2=sqrt(2.0);
   m1tu= -1;
   for(u=0;u<=N-1;u++){
       m1tu= -m1tu;
       p1=0.0; p2=0.0; p3=0.0;
       angle2=r*(u+u+1)*M_PI/(N+N);
       Fr=sin(angle2);
       Er=cos(angle2);

       for(m=0; m<=(r-1); m++){
         angle=(m+1)*(u+u+1)*M_PI/(N+N);
         sine=sin(angle);
         cosine=cos(angle);

         p1 += (m1tu*fnew[2*r-1-m] + Fr*fold[r-m-1]
                - m1tu*Er*fnew[r-m-1])*sine;
         p2 += (m1tu*Fr*fnew[r-m-1] - fold[2*r-m-1]
                + Er*fold[r-m-1])*cosine;  }
         p3 =  (1.0-(1.0/root2))*(fold[0]
                - (2.0*Er*fold[r]) + fold[2*r]);
       term1= root2bN*(p1 + p2 - p3);
       Csave=C[u];
       C[u]=((2.0*Er*C[u]) - Cold[u] + term1);
       Cold[u]=Csave;
  }
}
```

Fig. A.11 C function to calculate *r*-point rectangular windowed update for DCT-III

/* Program to calculate DCT type-III windowed update in the presence of split-triangular window */

```
void update_windowed_III(double Cw[], double Cwold[], double Cm[],
                         double Cmold[], double f[], int N, int n0,
                         double fwnew[], double fwold[], double
                         fmnew[], double fmold[], double fn0, double
                         fNmn0, double fN, double fm1, double fn0m1,
                         double fNmn0m1, double fw[], double fm[],
                         int r){

  Variables used:
       Cw[]  0..N-1      Input – DCT3 of old windowed data
                         Output- DCT3 of new windowed data
       Cm[]  0..N-1      Input – DCT3 of old data_times_m
                         Output- DCT3 of new data_times_m
       N                 signal and window length
       n0                tail length of split-triangular window
       fwnew[] 0..r-1    last but one data to be shifted in times old
                         window
       fwold[] 0..r-1    oldest data shifted out times old window
       fmnew[] 0..r-1    new data shifted in times old m
       fmold[] 0..r-1    r old data points times old m
       fm1               the number fold1[0];
       fn0m1             the number f[n0-1]
       fNmn0m1           the number f[N-n0-1]
       fn0               the number f[n0]
       fNmn0             the number f[N-n0]
       fN                the number f[N]  (= fnew[0])
       r                 number of new data points coming in
       Cmold[0..N-1]     Input: DCT of data_times_m from previous
                         timestep
                         Output:A copy of Cm as input
       Cwold[0..N-1]     Input: DCT of windowed data from previous time
                         step
                         Output: A copy of Cw as input
*/

  int m1tu;
  double twoN, root2bNbn0, theta, pk;
  double Cmcorrection, Cold_correction;
  int u,x;
  double *rec;

  rec = (double *)calloc(N+1,sizeof(double));
  if (rec==NULL){
      printf("Unable to allocate array\n");
      exit(1);
  }
  r=1;
  root2bNbn0 = sqrt(2.0/(double)N)/(double)n0;
  twoN= (double)N+N;

  IDCT3(Cmold, rec, N);
```

```
  pntarray("signal f(x-1)m(x) is =", rec, N);

  m1tu=1;
  for(u=0; u<=N-1; u++){
      theta= ((u+u+1)*M_PI)/twoN;
      Cold_correction = root2bNbn0*( -(fn0m1*cos(n0*theta))
                                     -(fNmn0m1*m1tu*sin(n0*theta))
                                     +((1.0/sqrt(2.0))*(2.0*fm1)) );

      Cmold[u] += Cold_correction;
          // Adds correction factor to Cmold to get DCT of f(x-1)m(x-1)
      Cwold[u] += Cmold[u];
      m1tu = -m1tu;
  }
  update_transform(Cw, Cwold, N, fwnew, fwold, r);
                                  // Unwindowed update
  update_transform(Cm, Cmold, N, fmnew, fmold, r) ;
                                  // Unwindowed update

  m1tu=1;
  for(u=0; u<=N-1; u++){
      theta= ((u+u+1)*M_PI)/twoN;
      if (n0==1) {
         pk=1.0/sqrt(2.0);}
      else pk =1;
      Cmcorrection = root2bNbn0*( -(pk*fn0*cos((n0-1.0)*theta))
                                  -(fNmn0*m1tu*sin((n0+1.0)*theta))
                                  +(2.0*fN*m1tu*sin(theta)) );
      Cm[u] += Cmcorrection;
      Cw[u] += Cm[u];
      m1tu = -m1tu;
  }
}
```

Fig. A.12 C function to calculate split-triangular windowed update for DCT-III

/* Program to calculate *r*-point rectangular windowed update for any given signal length *N* for DST-III */

```
void update_transform(double S[], double Sold[], int N, double fnew[],
                      double fold[], int r)

/*     S[]        input      DST of the previous time unit sequence
       S[]        output     Updated DST calculated using DST values of
                             older sequences
       N          input      the input data sequence length
       r          input      the number of new points to shift in
       fnew[]     input      the previous data point shifted in
       fold[]     input      most resent data point shifted out    */

{
```

```
   double k;
   int u;
   int m1tu;
   double root2bN;
   double p1, p2, p3;
   int m;
   int x;
   double root2,term,sine,cosine,angle,angle2,Fr,Er;
   double Ssave;

   root2bN=sqrt(2.0/N);
   root2=sqrt(2.0);
   m1tu= -1;
   for(u=0;u<=N;u++){
       m1tu= -m1tu;
       p1=0.0; p2=0.0; p3=0.0;

       angle2=r*((u+u+1)*M_PI)/(N+N);
       Fr=sin(angle2);
       Er=cos(angle2);

       for(m=0; m<=(r-1); m++){
         angle=((m+1)*(u+u+1)*M_PI)/(N+N);
         sine=sin(angle);
         cosine=cos(angle);

         p1 += ((m1tu*fnew[2*r-m-1]) + (Fr*fold[r-1-m])-
                          (m1tu*Er*fnew[r-m-1]))*cosine;
         p2 += (fold[2*r-m-1] -(m1tu*Fr*fnew[r-m-1])-
                          (Er*fold[r-1-m]))*sine;  }
         p3 = ((m1tu*(1.0/root2)*fnew[0])+(m1tu*(1.0/root2)*fnew[2*r])
                           -(m1tu*root2*Er*fnew[r]));

       term= root2bN*(p1+p2+p3);
       Ssave=S[u];
       S[u]=((2.0*Er*S[u]) - Sold[u] + term);
       Sold[u]=Ssave;
  }
}
```

Fig. A.13 C function to calculate *r*-point rectangular windowed update for DST-III

/* Program to calculate DST type-III windowed update in the presence of split-triangular window */

```
void update_windowed_III(double Sw[], double Swold[], double Sm[],
                         double Smold[], int N, int n0,
                         double fwnew[], double fwold[],
                         double fmnew[], double fmold[], double fn0,
                         double fNmn0, double fN, double fm1, double
                         fn0m1, double fNmn0p1, int r){
       Variables used:
```

```
     Sw[]  0..N-1      Input – DST3 of old windowed data
                       Output- DST3 of new windowed data
     Sm[]  0..N-1      Input – DST3 of old data_times_m
                       Output- DST3 of new data_times_m
     N                 signal and window length
     n0                tail length of split-triangular window
     fwnew[] 0..r-1    new data shifted in times old window
     fwold[] 0..r-1    last old data shifted out times old window
     fmnew[] 0..r-1    new data shifted in times old m
     fmold[] 0..r-1    r old data points times old m
     fm1               the number fold1[0]
     fn0m1             the number f[n0-1]
     fNmn0m1           the number f[N-n0-1]
     fn0               the number f[n0]
     fNmn0             the number f[N-n0]
     fN                the number f[N]  (= fnew[0])
     r                 number of new data points coming in
     Smold[0..N-1]     Input - DST of data_times_m from previous
                               timestep
                       Output - A copy of Sm as input
     Swold[0..N-1]     Input - DST of windowed data from previous
                               timestep
                       Output - A copy of Sw as input
     */

int m1tu;
double twoN,root2bNbn0, theta;
double Smcorrection, Sold_correction;
int u,pk,x;
double *rec;

rec = (double *)calloc(N+1,sizeof(double));
if (rec==NULL){
    printf("Unable to allocate array\n");
    exit(1);
}
if(r!=1){
    printf("Update_windowed_II only works for r=1, you gave
            r=%d\n",r);
    exit(1);
 }
root2bNbn0 = sqrt(2.0/(double)N)/(double)n0;
twoN= (double)N+N;

m1tu=1;
for(u=0; u<=N; u++)
{
     theta= ((u+u+1)*M_PI)/twoN;
     if (n0==1){
      pk=1.0/sqrt(2.0);
      }
    else pk=1;
     Sold_correction = root2bNbn0*(-(fn0m1*sin(n0*theta))
                                -(pk*fNmn0*m1tu*cos((n0-1)*theta)));

          // Correction factor to get f(x-1)m(x-1) from f(x-1)m(x)
          // Correction factor to get f(x-1)w(x-1) from f(x-1)w(x)
```

```
        Smold[u] += Sold_correction;
          // Adds correction factor to Smold2 to get DST of f(x-1)m(x-1)
        Swold[u] += Smold[u];
        m1tu = -m1tu;
  }

  IDST3(Smold,rec,N);
  update_transform(Sw, Swold, N, fwnew, fwold, r);
                                  // Unwindowed update
  update_transform(Sm, Smold, N, fmnew, fmold, r) ;
                                  // Unwindowed update
  m1tu=1;
  for(u=0; u<=N; u++){  // later, N
      theta= ((u+u+1)*M_PI)/twoN;

      if (n0==1){  pk=1.0/sqrt(2.0);  }
      else pk=1;
            Smcorrection = root2bNbn0*(-pk*fn0*sin((n0-1)*theta)
                                       -fNmn0p1*m1tu*cos(n0*theta)
                                       +2.0*fN*m1tu*(1.0/sqrt(2.0)));
            Sm[u] += Smcorrection;
            Sw[u] += Sm[u];
            m1tu = -m1tu;
  }
}
```

Fig. A.14 C function to calculate split-triangular windowed update for DST-III

/* Program to calculate *r*-point rectangular windowed update for any given signal length *N* for DCT type IV */

```
void update_transform(double C[], double Cold[], int N, double fnew1[],
                      double fnew2[], double fold1[], double fold2[],
                      int r)

/*     C[]        input      DCT of the previous time unit sequence
       C[]        output     Updated DCT calculated using DCT values of
                             older sequences
       N          input      the input data sequence length
       r          input      the number of new points to shift in
       fnew1[]    input      the previous data point shifted in
       fnew2[]    input      the newest data point shifted in
       fold1[]    input      most resent data point shifted out
       fold2[]    input      the older data point shifted out      */

{
   double k;
   int u;
   int m1tu;
   double root2bN;
   double p1;
   double p2;
```

```
   int m;
   int x;
   double root2,term1,sine,cosine,angle,angle2,Fr,Er;
   double Csave;

   root2bN=sqrt(2.0/N);
   m1tu= -1;
   for(u=0;u<=N;u++){
       m1tu= -m1tu;
       p1=0.0;  p2=0.0;
       angle2=r*(u+u+1)*M_PI/(N+N);
       Fr=sin(angle2);
       Er=cos(angle2);

       for(m=0; m<=(r-1); m++){
         angle= (m+m+1)*(u+u+1)*M_PI/(4*N);
         sine=sin(angle);
         cosine=cos(angle);

         p1 += (Fr*fold1[r-1-m] + m1tu*fnew2[r-1-m]
                                - m1tu*Er*fnew1[r-1-m])*sine;
         p2 += (Er*fold1[r-1-m] + m1tu*Fr*fnew1[r-1-m]
                                - fold2[r-1-m])*cosine;
     }
    term1= root2bN*(p1 + p2);
    Csave=C[u];
    C[u]=2.0*Er*C[u] - Cold[u] + term1;
    Cold[u]=Csave;
  }
}
```

Fig. A.15 C function to calculate *r*-point rectangular windowed update for DCT-IV

/* Function to calculate DCT type-IV windowed update in the presence of split-triangular window */

```
void update_windowed_IV(double Cw[], double Cwold[], double Cm[],
                        double Cmold[],int N, int n0,
                        double fwnew1[], double fwnew2[], double
                        fwold1[], double fwold2[], double fmnew1[],
                        double fmnew2[], double fmold1[],
                        double fmold2[], double fn0, double fNmn0,
                        double fN, double fm1, double fn0m1,
                        double fNmn0m1, int r){

/*    Split-triangulr DCT-IV of length N signal.
      Window length N with tail length n0.
      algorithm works for r=1.

  Variables used:
       Cw[]  0..N-1      Input - DCT-IV of old windowed data
                         Output- DCT-IV of new windowed data
       Cm[]  0..N-1      Input - DCT-IV of old data_times_m
```

```
                        Output- DCT-IV of new data_times_m
     N                  signal and window length
     n0                 tail length of split-triangular window
     fwnew1[] 0..r-1    last but one data to be shifted in times old
                        window
     fwnew2[] 0..r-1    new data shifted in times old window
     fwold1[] 0..r-1    oldest data shifted out times old window
     fwold2[] 0..r-1    last old data shifted out times old window
     fmnew1[] 0..r-1    last but one data to be shifted in times old m
     fmnew2[] 0..r-1    new data shifted in times old m
     fmold1[] 0..r-1    oldest data shifted times old m
     fmold2[] 0..r-1    r old data points times old m
     fm1                the number fold1[0];
     fn0m1              the number f[n0-1]
     fNmn0m1            the number f[N-n0-1]
     fn0                the number f[n0]
     fNmn0              the number f[N-n0]
     fN                 the number f[N]  (= fnew[0])
     r                  number of new data points coming in
     Cmold[0..N-1]      Input: DCT of data_times_m from previous
                        timestep
                        Output:A copy of Cm as input
     Cwold[0..N-1]      Input: DCT of windowed data from previous time
                        step
                        Output: A copy of Cw as input
    */

int m1tu;
double twoN,root2bNbn0,theta;
double Cmcorrection, Cold_correction;
int u;

if(r!=1){
 printf("Update_windowed_IV works for r=1, not r=%d\n",r);
    exit(1);
}
root2bNbn0 = (sqrt(2.0/(double)N)/(double)n0);
twoN = (double)N+N;
m1tu=1;
for(u=0; u<=N-1; u++){
    theta= ((u+u+1)*M_PI)/twoN;
    Cold_correction = root2bNbn0*
                       (-fn0m1*cos(((n0+n0+1)/2.0)*theta))
                        -(m1tu*fNmn0m1*sin(((n0+n0-1)/2.0)*theta))
                        +(2.0*fm1*cos(theta/2.0)));
  Cmold[u] += Cold_correction;
       // Adds correction factor to Cmold to get DCT of f(x-1)m(x-1)
  Cwold[u] += Cmold[u];
    m1tu = -m1tu;
}
update_transform(Cw, Cwold, N, fwnew1, fwnew2, fwold1, fwold2, r);
                                    // Unwindowed update
update_transform(Cm, Cmold, N, fmnew1, fmnew2, fmold1, fmold2, r) ;
                                    // Unwindowed update

m1tu=1;
for(u=0; u<=N-1; u++){
```

```
      theta= ((u+u+1)*M_PI)/twoN;
      Cmcorrection = root2bNbn0*(-(fn0*cos(((n0+n0-1)/2.0)*theta))
                                 -(m1tu*fNmn0*sin(((n0+n0+1)/2.0)*theta))
                                 +(2.0*fN*m1tu*sin(theta/2.0)));
      Cm[u] += Cmcorrection;
      Cw[u] += Cm[u];
      m1tu = -m1tu;
  }
}
```

Fig. A.16 C function to calculate split-triangular windowed update for DST-IV

/* Code snippet to calculate DST-IV coefficients indirectly from DCT coefficients for Hanning, Hamming and Blackman windows */

```
for(u=0;u<=N-1;u++){
         theta = ((u+u+1)*M_PI)/(4*N);
         costheta = cos(theta);
         sintheta = sin(theta);
         cos2theta = cos(2*theta);
         sin2theta = sin(2*theta);
         m1tu=pow(-1,u);
         S[u]=C[u]-(cos2theta*Cold[u])+ ((costheta*fold2[0]-
                    sintheta*m1tu*fnew2[0])/root_Nv2);
         S[u]=S[u]/sin2theta;
}
```

Fig. A.17 C code snippet to calculate independent windowed update for DCT-IV in presence of Hanning, Hamming and Blackman windows

/* Program to calculate *r*-point rectangular windowed update for any given signal length *N* for DST type IV */

```
void update_transform(double S[], double Sold[], int N,  double
                      fnew1[], double fnew2[], double fold1[], double
                      fold2[], int r)

/*     S[]        input      DST of the previous time unit sequence
       S[]        output     Updated DST calculated using DST values of
                             older sequences
       N          input      the input data sequence length
       r          input      the number of new points to shift in
       fnew1[]    input      the previous data point shifted in
       fnew2[]    input      the newest data point shifted in
       fold1[]    input      most resent data point shifted out
       fold2[]    input      the older data point shifted out      */

{
```

```
   double k;
   int u;
   int m1tu;
   double root2bN;
   double p1;
   double p2;
   int m;  int x;
   double root2,term1,sine,cosine,angle,angle2, Fr, Er;
   double Ssave;

   root2bN=sqrt(2.0/N);
   m1tu= -1;
   for(u=0;u<=N;u++){
       m1tu= -m1tu;
       p1=0.0;  p2=0.0;
       angle2=r*(u+u+1)*M_PI/(N+N);
       Fr=sin(angle2);
       Er=cos(angle2);

       for(m=0; m<=(r-1); m++){
         angle= (m+m+1)*(u+u+1)*M_PI/(4*N);
         sine=sin(angle);
         cosine=cos(angle);
         p1 += (Fr*fold1[r-1-m] + m1tu*fnew2[r-1-m]
                              - m1tu*Er*fnew1[r-1-m])*cosine;
         p2 += (fold2[r-1-m] - m1tu*Fr*fnew1[r-1-m]
                              - Er*fold1[r-1-m])*sine;
         }
         term1= root2bN*(p1 + p2);
         Ssave=S[u];
         S[u]=2.0*Er*S[u] - Sold[u] + term1;
         Sold[u]=Ssave;
   }
}
```

Fig. A.18 C function to calculate *r*-point rectangular windowed update for DST-IV

/* Function to calculate DST type-IV windowed update in the presence of split-triangular window */

```
void update_windowed_IV(double Sw[], double Swold[], double Sm[],
                        double Smold[], int N, int n0, int r,
                        double fwnew1[], double fwnew2[], double
                        fwold1[], double fwold2[], double fmnew1[],
                        double fmnew2[], double fmold1[], double
                        fmold2[], double fn0, double fNmn0, double fN,
                        double fm1, double fn0m1, double fNmn0m1) {

       Variables used:
       Sw[]  0..N-1      Input - DST type-IV of old windowed data
                         Output- DST type-IV of new windowed data
       Sm[]  0..N-1      Input - DST type-IV of old data_times_m
```

```
                          Output- DST type-IV of new data_times_m
        N                 signal and window length
        n0                tail length of split-triangular window
        fwnew1[] 0..r-1   last but one data to be shifted in times old
                          window
        fwnew2[] 0..r-1   new data shifted in times old window
        fwold1[] 0..r-1   oldest data shifted out times old window
        fwold2[] 0..r-1   last old data shifted out times old window
        fmnew1[] 0..r-1   last but one data to be shifted in times old m
        fmnew2[] 0..r-1   new data shifted in times old m
        fmold1[] 0..r-1   oldest data shifted times old m
        fmold2[] 0..r-1   r old data points times old m
        fm1               the number fold1[0]
        fn0m1             the number f[n0-1]
        fNmn0m1           the number f[N-n0-1]
        fn0               the number f[n0]
        fNmn0             the number f[N-n0]
        fN                the number f[N]  (= fnew[0])
        r                 number of new data points coming in
        Smold[0..N-1]     Input - DST of data_times_m from previous
                          timestep
                          Output - A copy of Sm as input
        Swold[0..N-1]     Input - DST of windowed data from previous
                          timestep
                          Output - A copy of Sw as input
*/

  int m1tu;
  double twoN,root2bNbn0, theta;
  double Smcorrection, Sold_correction;
  int u;

  if(r!=1){
      printf("Update_windowed_IV only works for r=1, you gave
              r=%d\n",r);
      exit(1);
 }
  root2bNbn0 = (sqrt(2.0/(double)N)/(double)n0);
  twoN= (double)N+N;
  m1tu=1;
  for(u=0; u<=N-1; u++)  {
       theta = ((u+u+1)*M_PI)/twoN;
       Sold_correction = root2bNbn0*(
                           -(fn0m1*sin(((n0+n0+1)/2.0)*theta))
                           -(fNmn0m1*m1tu*cos(((n0+n0-1)/2.0)*theta))
                           +(2.0*fm1*sin(theta/2.0))  );

      // Correction factor to get f(x-1)m(x-1) from f(x-1)m(x)
      // Correction factor to get f(x-1)w(x-1) from f(x-1)w(x)

       Smold[u] += Sold_correction;
         // Adds correction factor to Smold2 to get DST of f(x-1)m(x-1)
       Swold[u] += Smold[u];
       m1tu = -m1tu;
  }
  update_transform(Sw, Swold, N, fwnew1, fwnew2, fwold1, fwold2, r);
                           // Unwindowed update
```

```
  update_transform(Sm, Smold, N, fmnew1, fmnew2, fmold1, fmold2, r) ;
                            // Unwindowed update

  m1tu=1;
  for(u=0; u<=N-1; u++){  // later, N
      theta = ((u+u+1)*M_PI)/twoN;
      Smcorrection = root2bNbn0*(-(fn0*sin(((n0+n0-1)/2.0)*theta))
                                 -(fNmn0*m1tu*cos(((n0+n0+1)/2.0)*theta))
                                 +(2.0*fN*m1tu*cos(theta/2.0))  );
            Sm[u] += Smcorrection;
            Sw[u] += Sm[u];
            m1tu = -m1tu;
  }
}
```

Fig. A.19 C function to calculate split-triangular windowed update for DST-IV

/* Code snippet to calculate DCT-IV coefficients indirectly from DST coefficients for Hanning, Hamming and Blackman windows */

```
 for(u=0;u<=N-1;u++){
         theta = ((u+u+1)*M_PI)/(4*N);
         costheta = cos(theta);
         sintheta = sin(theta);
         cos2theta = cos(2*theta);
         sin2theta = sin(2*theta);
         m1tu=pow(-1,u);
         C[u]=(cos2theta*Sold[u])- S[u] + ((sintheta*fold2[0]+
               costheta*m1tu*fnew2[0])/root_Nv2);
         C[u]=C[u]/sin2theta;
}
```

Fig. A.20 C code snippet to calculate independent windowed update for DST-IV in presence of Hanning, Hamming and Blackman windows

APPENDIX B: VHDL CODE FOR HARDWARE IMPLEMENTATION OF INDEPENDENT DCT-II ALGORITHM

```
-- Entity: DCT_RECTAN_R1_UPDATE for independent updating of DCT type-II

-- One-point update controller for DCT-II rectangular update algorithm
-- developed in Chapter 2. The components DCT_DST_CONVEN, BUTTER_FLY,
-- IP_STAGE, MULT, EXT_UNIT used in this controller are given in [31].
-- The simultaneous update hardware implementation initially given in
-- [31] is extended to implement the independent update algorithm.

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_unsigned.all;
use ieee.math_real.all;
use ieee.numeric_std.all;

entity DCT_RECTAN_R1_UPDATE is
port(f0, f1, f2, f3, f4, f5, f6, f7: in std_logic_vector(8 downto
0):=(others=>'0');
clk, rst, conven, update: in std_logic;
C0, C1, C2, C3, C4, C5, C6, C7: buffer std_logic_vector(15 downto
0):=(others=>'0');
DONE_DCTCONVEN: out std_logic:='1';
DONE_UPDATE: out std_logic:='1');
end DCT_RECTAN_R1_UPDATE;

architecture DCT_RECTAN_R1_UPDATE_arch of DCT_RECTAN_R1_UPDATE is

component DCT_DST_CONVEN
port(x07_S, x16_S, x25_S, x34_S, x07_D, x16_D, x25_D, x34_D: in
std_logic_vector(9 downto 0):=(others=>'0');
clk, rst, conv: in std_logic;
X0_COS, X1_COS, X2_COS, X3_COS, X4_COS, X5_COS, X6_COS, X7_COS: out
std_logic_vector(15 downto 0):=(others=>'0');
X1_SIN, X2_SIN, X3_SIN, X4_SIN, X5_SIN, X6_SIN, X7_SIN, X8_SIN: out
std_logic_vector(15 downto 0):=(others=>'0');
DONE_CONVEN: out std_logic:='1');
end component;

component BUTTER_FLY
port(x0, x1, x2, x3, x4, x5, x6, x7: in std_logic_vector(8 downto 0);
EN: in std_logic;
X0OUT, X1OUT, X2OUT, X3OUT, X4OUT, X5OUT, X6OUT, X7OUT: out
std_logic_vector(9 downto 0):=(others=>'0'));
end component;

component IP_STAGE
port(x0, x1, x2, x3, x4, x5, x6, x7: in std_logic_vector(8 downto 0);
Data_L, Update_L, Data_S: in std_logic;
X0OUT, X1OUT, X2OUT, X3OUT, X4OUT, X5OUT, X6OUT, X7OUT: buffer
std_logic_vector(8 downto 0):=(others=>'0'));
end component;
```

```
component MULT
port(A: in std_logic_vector(8 downto 0);
B: in std_logic_vector(15 downto 0);
ST, clk, rst: in std_logic;
PROD: out std_logic_vector(15 downto 0):=(others=>'0');
DONE: out std_logic:='1');
end component;

component EXT_UNIT
port(A: in std_logic_vector(9 downto 0);
EXT: in std_logic;
A_EXT: out std_logic_vector(15 downto 0):=(others=>'0'));
end component;

component ADD_2_16
port(A1, A2: in std_logic_vector(15 downto 0);
EN: in std_logic;
SUM: out std_logic_vector(17 downto 0):=(others=>'0'));
end component;

component ADD_4_16
port(A1, A2, A3, A4: in std_logic_vector(15 downto 0);
EN: in std_logic;
SUM: out std_logic_vector(17 downto 0):=(others=>'0'));
end component;

component ADD_5_16
port(A1, A2, A3, A4, A5: in std_logic_vector(15 downto 0);
EN: in std_logic;
SUM: out std_logic_vector(17 downto 0):=(others=>'0'));
end component;

component SUB_2_16
port(A, B: in std_logic_vector(15 downto 0);
     S_AB: in std_logic:='1';
     EN: in std_logic;
     SUB: out std_logic_vector(16 downto 0):=(others=>'0'));
end component;

type STATE is (s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12,
s13, s14, s15, s16, s17, s18, s19, s20, s21, s22, s23, s24, s25, s26,
s27, s28, s29, s30, s31, s32, s33, s34, s35, s36, s37, s38, s39, s40,
s41, s42, s43, s44, s45, s46, s47, s48, s49, s50, s51, s52, s53, s54,
s55, s56  );

signal PS, NS: STATE;
signal trig: std_logic:='0';

signal rst_DCT_DST_CONVEN: std_logic:='1';
signal conv_DCT_DST_CONVEN: std_logic:='0';
signal DONE_DCT_DST_CONVEN: std_logic:='0';

signal Data_LAT: std_logic:='0';
signal Update_LAT: std_logic:='0';
signal Data_SHIFT: std_logic:='0';
```

```
signal EN_BUTTER_FLY: std_logic:='0';

signal MULT_ST: std_logic:='0';

signal MULT_rst: std_logic:='0';

signal EN_ADD: std_logic:='0';

signal EXT: std_logic:='0';

signal EN_SUB: std_logic:='0';

signal EN_SUB_P: std_logic:='0';

signal f08_S_EXT: std_logic_vector(15 downto 0):=(others=>'0');
signal f19_S_EXT: std_logic_vector(15 downto 0):=(others=>'0');

signal f08_D_EXT: std_logic_vector(15 downto 0):=(others=>'0');
signal f19_D_EXT: std_logic_vector(15 downto 0):=(others=>'0');

signal F08_S_BUF: std_logic_vector(15 downto 0):=(others=>'0');
signal F19_S_BUF: std_logic_vector(15 downto 0):=(others=>'0');

signal F08_D_BUF: std_logic_vector(15 downto 0):=(others=>'0');
signal F19_D_BUF: std_logic_vector(15 downto 0):=(others=>'0');

signal F_C0_1, F_C0_2: std_logic:='0';
signal F_C1_1, F_C1_2, F_C1_3: std_logic:='0';
signal F_C2_1, F_C2_2, F_C2_3: std_logic:='0';
signal F_C3_1, F_C3_2, F_C3_3: std_logic:='0';
signal F_C4_1, F_C4_2: std_logic:='0';
signal F_C5_1, F_C5_2, F_C5_3: std_logic:='0';
signal F_C6_1, F_C6_2, F_C6_3: std_logic:='0';
signal F_C7_1, F_C7_2, F_C7_3: std_logic:='0';

signal F_C0: std_logic:='0';
signal F_C1: std_logic:='0';
signal F_C2: std_logic:='0';
signal F_C3: std_logic:='0';
signal F_C4: std_logic:='0';
signal F_C5: std_logic:='0';
signal F_C6: std_logic:='0';
signal F_C7: std_logic:='0';

signal F_C: std_logic:='0';

signal IP_BF_0: std_logic_vector(8 downto 0):=(others=>'0');
signal IP_BF_1: std_logic_vector(8 downto 0):=(others=>'0');
signal IP_BF_2: std_logic_vector(8 downto 0):=(others=>'0');
signal IP_BF_3: std_logic_vector(8 downto 0):=(others=>'0');
signal IP_BF_4: std_logic_vector(8 downto 0):=(others=>'0');
signal IP_BF_5: std_logic_vector(8 downto 0):=(others=>'0');
signal IP_BF_6: std_logic_vector(8 downto 0):=(others=>'0');
signal IP_BF_7: std_logic_vector(8 downto 0):=(others=>'0');

signal BF_CONVEN_0: std_logic_vector(9 downto 0):=(others=>'0');
signal BF_CONVEN_1: std_logic_vector(9 downto 0):=(others=>'0');
```

```
signal BF_CONVEN_2: std_logic_vector(9 downto 0):=(others=>'0');
signal BF_CONVEN_3: std_logic_vector(9 downto 0):=(others=>'0');
signal BF_CONVEN_4: std_logic_vector(9 downto 0):=(others=>'0');
signal BF_CONVEN_5: std_logic_vector(9 downto 0):=(others=>'0');
signal BF_CONVEN_6: std_logic_vector(9 downto 0):=(others=>'0');
signal BF_CONVEN_7: std_logic_vector(9 downto 0):=(others=>'0');

signal C0M: std_logic_vector(15 downto 0):=(others=>'0');
signal C1M: std_logic_vector(15 downto 0):=(others=>'0');
signal C2M: std_logic_vector(15 downto 0):=(others=>'0');
signal C3M: std_logic_vector(15 downto 0):=(others=>'0');
signal C4M: std_logic_vector(15 downto 0):=(others=>'0');
signal C5M: std_logic_vector(15 downto 0):=(others=>'0');
signal C6M: std_logic_vector(15 downto 0):=(others=>'0');
signal C7M: std_logic_vector(15 downto 0):=(others=>'0');

signal C0_L: std_logic_vector(15 downto 0):=(others=>'0');
signal C1_L: std_logic_vector(15 downto 0):=(others=>'0');
signal C2_L: std_logic_vector(15 downto 0):=(others=>'0');
signal C3_L: std_logic_vector(15 downto 0):=(others=>'0');
signal C4_L: std_logic_vector(15 downto 0):=(others=>'0');
signal C5_L: std_logic_vector(15 downto 0):=(others=>'0');
signal C6_L: std_logic_vector(15 downto 0):=(others=>'0');
signal C7_L: std_logic_vector(15 downto 0):=(others=>'0');

signal C0_LX: std_logic_vector(17 downto 0):=(others=>'0');
signal C1_LX: std_logic_vector(17 downto 0):=(others=>'0');
signal C2_LX: std_logic_vector(17 downto 0):=(others=>'0');
signal C3_LX: std_logic_vector(17 downto 0):=(others=>'0');
signal C4_LX: std_logic_vector(17 downto 0):=(others=>'0');
signal C5_LX: std_logic_vector(17 downto 0):=(others=>'0');
signal C6_LX: std_logic_vector(17 downto 0):=(others=>'0');
signal C7_LX: std_logic_vector(17 downto 0):=(others=>'0');

signal F0_INT: std_logic_vector(17 downto 0):=(others=>'0');
signal F1_INT: std_logic_vector(17 downto 0):=(others=>'0');
signal F2_INT: std_logic_vector(17 downto 0):=(others=>'0');
signal F3_INT: std_logic_vector(17 downto 0):=(others=>'0');
signal F4_INT: std_logic_vector(17 downto 0):=(others=>'0');
signal F5_INT: std_logic_vector(17 downto 0):=(others=>'0');
signal F6_INT: std_logic_vector(17 downto 0):=(others=>'0');
signal F7_INT: std_logic_vector(17 downto 0):=(others=>'0');

signal C0_1, C0_2: std_logic_vector(15 downto 0):=(others=>'0');
signal C1_1, C1_2, C1_3: std_logic_vector(15 downto 0):=(others=>'0');
signal C2_1, C2_2, C2_3: std_logic_vector(15 downto 0):=(others=>'0');
signal C3_1, C3_2, C3_3: std_logic_vector(15 downto 0):=(others=>'0');
signal C4_1, C4_2: std_logic_vector(15 downto 0):=(others=>'0');
signal C5_1, C5_2, C5_3: std_logic_vector(15 downto 0):=(others=>'0');
signal C6_1, C6_2, C6_3: std_logic_vector(15 downto 0):=(others=>'0');
signal C7_1, C7_2, C7_3: std_logic_vector(15 downto 0):=(others=>'0');

signal L1: std_logic_vector(8 downto 0):="001011010";--(0.3515625)
signal L2: std_logic_vector(8 downto 0):="011101100";--(0.921875)
signal L3: std_logic_vector(8 downto 0):="010000011";--(0.48828125)
signal L4: std_logic_vector(8 downto 0):="010110101";--(0.70703125)
signal L5: std_logic_vector(8 downto 0):="010001010";--(0.4609375)
```

```
signal L6: std_logic_vector(8 downto 0):="001100000";--(0.375)
signal L7: std_logic_vector(8 downto 0):="001101010";--(0.4140625)
signal L8: std_logic_vector(8 downto 0):="001011010";--(0.3535)
signal L9: std_logic_vector(8 downto 0):="110011111";--(-0.37890625)
signal L10: std_logic_vector(8 downto 0):="001000111";--(0.27734375)
signal L11: std_logic_vector(8 downto 0):="101001011";--(-0.70703125)
signal L12: std_logic_vector(8 downto 0):="000110000";--(0.1875)
signal L13: std_logic_vector(8 downto 0):="100010100";--(-0.921875)
signal L14: std_logic_vector(8 downto 0):="000011000";--(0.09375)
signal L15: std_logic_vector(8 downto 0):="110100110";--(-0.3515625)
signal L16: std_logic_vector(8 downto 0):="110000011";--(-0.48828125)
signal L17: std_logic_vector(8 downto 0):="110001010";--(-0.4609375)
signal L18: std_logic_vector(8 downto 0):="110010110";--(-0.4140625)
signal L19: std_logic_vector(8 downto 0):="110100101";--(-0.3535)
signal L20: std_logic_vector(8 downto 0):="110111001";--(-0.27734375)
signal L21: std_logic_vector(8 downto 0):="111010000";--(-0.1875)
signal L22: std_logic_vector(8 downto 0):="111101000";--(-0.09375)


begin

L1<="001011010";--(0.3515625)
L2<="011101100";--(0.921875)
L3<="010000011";--(0.48828125)
L4<="010110101";--(0.70703125)
L5<="010001010";--(0.4609375)
L6<="001100000";--(0.375)
L7<="001101010";--(0.4140625)
L8<="001011010";--(0.3535)
L9<="110011111";--(-0.37890625)
L10<="001000111";--(0.27734375)
L11<="101001011";--(-0.70703125)
L12<="000110000";--(0.1875)
L13<="100010100";--(-0.921875)
L14<="000011000";--(0.09375)
L15<="110100110";--(-0.3515625)
L16<="110000011";--(-0.48828125)
L17<="110001010";--(-0.4609375)
L18<="110010110";--(-0.4140625)
L19<="110100101";--(-0.3535)
L20<="110111001";--(-0.27734375)
L21<="111010000";--(-0.1875)
L22<="111101000";--(-0.09375)


F_C0<=F_C0_1 and F_C0_2;
F_C1<=F_C1_1 and F_C1_2 and F_C1_3;
F_C2<=F_C2_1 and F_C2_2 and F_C2_3;
F_C3<=F_C3_1 and F_C3_2 and F_C3_3;
F_C4<=F_C4_1 and F_C4_2;
F_C5<=F_C5_1 and F_C5_2 and F_C5_3;
F_C6<=F_C6_1 and F_C6_2 and F_C6_3;
F_C7<=F_C7_1 and F_C7_2 and F_C7_3;

F_C<=F_C0 and F_C1 and F_C2 and F_C3 and F_C4 and F_C5 and F_C6 and
F_C7;
```

```
process(clk, rst)
begin
if(rst='1')then
PS<=s0;
elsif(rising_edge(clk))then
PS<=NS;
trig<=not(trig);
end if;
end process;

process(PS, trig, conven, update, DONE_DCT_DST_CONVEN, F_C)
begin
case PS is

    -- prepartion stage I

when s0=>       -- previous time-unit DCT calculation
rst_DCT_DST_CONVEN<='1';
conv_DCT_DST_CONVEN<='0';
MULT_rst<='1';
MULT_ST<='0';
EN_ADD<='0';
EN_BUTTER_FLY<='0';
Data_LAT<='0';
Update_LAT<='0';
Data_SHIFT<='0';
EXT<='0';
DONE_DCTCONVEN<='0';
DONE_UPDATE<='0';

if(conven='1')then
NS<=s1;
elsif(update='1')then
NS<=s33;
else
NS<=s0;
end if;

when s1=>
Data_LAT<='1';
DONE_DCTCONVEN<='0';
rst_DCT_DST_CONVEN<='0';
NS<=s2;

when s2=>
NS<=s3;

when s3=>
Data_LAT<='0';
EN_BUTTER_FLY<='1';
NS<=s4;

when s4=>
NS<=s5;

when s5=>
EN_BUTTER_FLY<='0';
```

```
NS<=s6;

when s6=>
conv_DCT_DST_CONVEN<='1';
NS<=s7;

when s7=>
NS<=s8;

when s8=>
conv_DCT_DST_CONVEN<='0';
if(DONE_DCT_DST_CONVEN='1')then
NS<=s9;
else
NS<=s8;
end if;

when s9=>
C0M<=C0_L;
C1M<=C1_L;
C2M<=C2_L;
C3M<=C3_L;
C4M<=C4_L;
C5M<=C5_L;
C6M<=C6_L;
C7M<=C7_L;
DONE_DCTCONVEN<='1';
NS<=s10;
            -- prepartion stage II
when s10=>       -- current time-unit DCT calculation
rst_DCT_DST_CONVEN<='1';
conv_DCT_DST_CONVEN<='0';
NS<=s11;

when s11=>
Update_LAT<='1';
DONE_DCTCONVEN<='0';
rst_DCT_DST_CONVEN<='0';
NS<=s12;

when s12=>
NS<=s13;

when s13=>
Update_LAT<='0';
NS<=s14;

when s14=>
NS<=s15;

when s15=>
EN_BUTTER_FLY<='1';
NS<=s16;

when s16=>
NS<=s17;
```

```
when s17=>
EN_BUTTER_FLY<='0';
NS<=s18;

when s18=>
NS<=s19;

when s19=>
EXT<='1';
NS<=s20;

when s20=>
NS<=s21;

when s21=>
EXT<='0';
F08_S_BUF<=f08_S_EXT;
F08_D_BUF<=f08_D_EXT;
NS<=s22;

when s22=>
NS<=s23;

when s23=>
Data_SHIFT<='1';
NS<=s24;

when s24=>
NS<=s25;

when s25=>
Data_SHIFT<='0';
NS<=s26;

when s26=>
EN_BUTTER_FLY<='1';
NS<=s27;

when s27=>
NS<=s28;

when s28=>
EN_BUTTER_FLY<='0';
NS<=s29;

when s29=>
conv_DCT_DST_CONVEN<='1';
NS<=s30;

when s30=>
NS<=s31;

when s31=>
conv_DCT_DST_CONVEN<='0';
if(DONE_DCT_DST_CONVEN='1')then
NS<=s32;
else
```

```
NS<=s31;
end if;

when s32=>
C0<=C0_L;
C1<=C1_L;
C2<=C2_L;
C3<=C3_L;
C4<=C4_L;
C5<=C5_L;
C6<=C6_L;
C7<=C7_L;
DONE_DCTCONVEN<='1';
NS<=s0;

--   Independent update mode

when s33=>
Update_LAT<='1';
DONE_UPDATE<='0';
MULT_rst<='0';
NS<=s34;

when s34=>
NS<=s35;

when s35=>
Update_LAT<='0';
NS<=s36;

when s36=>
EN_BUTTER_FLY<='1';
NS<=s37;

when s37=>
EN_BUTTER_FLY<='0';
EXT<='1';
NS<=s38;

when s38=>
EXT<='0';
F19_S_BUF<=f08_S_EXT;
F19_D_BUF<=f08_D_EXT;
NS<=s39;

when s39=>
NS<=s40;

when s40=>
MULT_ST<='1';
NS<=s41;

when s41=>
NS<=s42;

when s42=>
MULT_ST<='0';
```

```
if(F_C='1')then
NS<=s43;
else
NS<=s42;
end if;

when s43=>
EN_ADD<='1';
NS<=s44;

when s44=>
NS<=s45;

when s45=>
EN_ADD<='0';
NS<=s46;

when s46=>
NS<=s47;

when s47=>
EN_SUB<='1';
NS<=s48;

when s48=>
NS<=s49;

when s49=>
EN_SUB<='0';
NS<=s50;

when s50=>
NS<=s51;

when s51=>
Data_SHIFT<='1';
NS<=s52;

when s52=>
NS<=s53;

when s53=>
Data_SHIFT<='0';
NS<=s54;

when s54=>
f08_S_BUF<=F19_S_BUF;
f08_D_BUF<=F19_D_BUF;

C0M<=C0;
C1M<=C1;
C2M<=C2;
C3M<=C3;
C4M<=C4;
C5M<=C5;
C6M<=C6;
C7M<=C7;
```

```
NS<=s55;

when s55=>
NS<=s56;

when s56=>
C0<=C0_LX(15 downto 0);
C1<=C1_LX(15 downto 0);
C2<=C2_LX(15 downto 0);
C3<=C3_LX(15 downto 0);
C4<=C4_LX(15 downto 0);
C5<=C5_LX(15 downto 0);
C6<=C6_LX(15 downto 0);
C7<=C7_LX(15 downto 0);
DONE_UPDATE<='1';
NS<=s0;

end case;
end process;


U0: IP_STAGE
port map(x0=>f0, x1=>f1, x2=>f2, x3=>f3, x4=>f4, x5=>f5, x6=>f6,
x7=>f7,
Data_L=>Data_LAT, Update_L=>Update_LAT, Data_S=>Data_SHIFT,
X0OUT=>IP_BF_0, X1OUT=>IP_BF_1, X2OUT=>IP_BF_2, X3OUT=>IP_BF_3,
X4OUT=>IP_BF_4, X5OUT=>IP_BF_5, X6OUT=>IP_BF_6, X7OUT=>IP_BF_7);

U1: BUTTER_FLY
port map(x0=>IP_BF_0, x1=>IP_BF_1, x2=>IP_BF_2,
x3=>IP_BF_3,x4=>IP_BF_4, x5=>IP_BF_5, x6=>IP_BF_6, x7=>IP_BF_7,
EN=>EN_BUTTER_FLY, X0OUT=>BF_CONVEN_0, X1OUT=>BF_CONVEN_1,
X2OUT=>BF_CONVEN_2, X3OUT=>BF_CONVEN_3, X4OUT=>BF_CONVEN_4,
X5OUT=>BF_CONVEN_5, X6OUT=>BF_CONVEN_6, X7OUT=>BF_CONVEN_7);

U2: DCT_DST_CONVEN
port map(x07_S=>BF_CONVEN_0, x16_S=>BF_CONVEN_1, x25_S=>BF_CONVEN_2,
x34_S=>BF_CONVEN_3, x07_D=>BF_CONVEN_4, x16_D=>BF_CONVEN_5,
x25_D=>BF_CONVEN_6, x34_D=>BF_CONVEN_7,
clk=>clk, rst=>rst_DCT_DST_CONVEN, conv=>conv_DCT_DST_CONVEN,
X0_COS=>C0_L, X1_COS=>C1_L, X2_COS=>C2_L, X3_COS=>C3_L, X4_COS=>C4_L,
X5_COS=>C5_L, X6_COS=>C6_L, X7_COS=>C7_L,
DONE_CONVEN=>DONE_DCT_DST_CONVEN);

U3: EXT_UNIT
port map(A=>BF_CONVEN_0, EXT=>EXT, A_EXT=>f08_S_EXT);

U4: EXT_UNIT
port map(A=>BF_CONVEN_4, EXT=>EXT, A_EXT=>f08_D_EXT);

-------------------------------------------

U5: MULT
port map(A=>L1, B=>f08_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C0_1, DONE=>F_C0_1);

U6: MULT
```

```
port map(A=>L15, B=>f19_D_BUF, ST=>MULT_ST, clk=>clk,
rst=>MULT_rst,PROD=>C0_2, DONE=>F_C0_2);

U7: ADD_4_16
port map(A1=>C0, A2=>C0, A3=>C0_1, A4=>C0_2, EN=>EN_ADD, SUM=>F0_INT);

U8: SUB_2_16
port map(A=>F0_INT(15 downto 0), B=>C0M,  S_AB=>'1', EN=>EN_SUB,
SUB=>C0_LX(16 downto 0));

-------------------------------------------

U9: MULT
port map(A=>L2, B=>C1, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,PROD=>C1_1,
DONE=>F_C1_1);

U10: MULT
port map(A=>L3, B=>f08_S_BUF, ST=>MULT_ST, clk=>clk,
rst=>MULT_rst,PROD=>C1_2, DONE=>F_C1_2);

U11: MULT
port map(A=>L16, B=>f19_S_BUF, ST=>MULT_ST, clk=>clk,
rst=>MULT_rst,PROD=>C1_3, DONE=>F_C1_3);

U12: ADD_4_16
port map(A1=>C1_1, A2=>C1_1, A3=>C1_2, A4=>C1_3, EN=>EN_ADD,
SUM=>F1_INT);

U13: SUB_2_16
port map(A=>F1_INT(15 downto 0), B=>C1M,  S_AB=>'1', EN=>EN_SUB,
SUB=>C1_LX(16 downto 0));

-------------------------------------------

U14: MULT
port map(A=>L4, B=>C2, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,PROD=>C2_1,
DONE=>F_C2_1);

U15: MULT
port map(A=>L5, B=>f08_D_BUF, ST=>MULT_ST, clk=>clk,
rst=>MULT_rst,PROD=>C2_2, DONE=>F_C2_2);

U16: MULT
port map(A=>L17, B=>f19_D_BUF, ST=>MULT_ST, clk=>clk,
rst=>MULT_rst,PROD=>C2_3, DONE=>F_C2_3);

U17: ADD_4_16
port map(A1=>C2_1, A2=>C2_1, A3=>C2_2, A4=>C2_3, EN=>EN_ADD,
SUM=>F2_INT);

U18: SUB_2_16
port map(A=>F2_INT(15 downto 0), B=>C2M,  S_AB=>'1', EN=>EN_SUB,
SUB=>C2_LX(16 downto 0));

-------------------------------------------

U19: MULT
```

```
port map(A=>L6, B=>C3, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,PROD=>C3_1,
DONE=>F_C3_1);

U20: MULT
port map(A=>L7, B=>f08_S_BUF, ST=>MULT_ST, clk=>clk,
rst=>MULT_rst,PROD=>C3_2, DONE=>F_C3_2);

U21: MULT
port map(A=>L18, B=>f19_S_BUF, ST=>MULT_ST, clk=>clk,
rst=>MULT_rst,PROD=>C3_3, DONE=>F_C3_3);

U22: ADD_4_16
port map(A1=>C3_1, A2=>C3_1, A3=>C3_2, A4=>C3_3, EN=>EN_ADD,
SUM=>F3_INT);

U23: SUB_2_16
port map(A=>F3_INT(15 downto 0), B=>C3M,  S_AB=>'1', EN=>EN_SUB,
SUB=>C3_LX(16 downto 0));

-----------------------------------------

U24: MULT
port map(A=>L8, B=>f08_D_BUF, ST=>MULT_ST, clk=>clk,
rst=>MULT_rst,PROD=>C4_1, DONE=>F_C4_1);

U25: MULT
port map(A=>L19, B=>f19_D_BUF, ST=>MULT_ST, clk=>clk,
rst=>MULT_rst,PROD=>C4_2, DONE=>F_C4_2);

U26: ADD_2_16
port map(A1=>C4_1, A2=>C4_2, EN=>EN_ADD, SUM=>F4_INT);

U27: SUB_2_16
port map(A=>F4_INT(15 downto 0), B=>C4M,  S_AB=>'1', EN=>EN_SUB,
SUB=>C4_LX(16 downto 0));

-----------------------------------------

U28: MULT
port map(A=>L9, B=>C5, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,PROD=>C5_1,
DONE=>F_C5_1);

U29: MULT
port map(A=>L10, B=>f08_S_BUF, ST=>MULT_ST, clk=>clk,
rst=>MULT_rst,PROD=>C5_2, DONE=>F_C5_2);

U30: MULT
port map(A=>L20, B=>f19_S_BUF, ST=>MULT_ST, clk=>clk,
rst=>MULT_rst,PROD=>C5_3, DONE=>F_C5_3);

U31: ADD_4_16
port map(A1=>C5_1, A2=>C5_1, A3=>C5_2, A4=>C5_3, EN=>EN_ADD,
SUM=>F5_INT);

U32: SUB_2_16
port map(A=>F5_INT(15 downto 0), B=>C5M,  S_AB=>'1', EN=>EN_SUB,
SUB=>C5_LX(16 downto 0));
```

```
-------------------------------------------

U33: MULT
port map(A=>L11, B=>C6, ST=>MULT_ST, clk=>clk,
rst=>MULT_rst,PROD=>C6_1, DONE=>F_C6_1);

U34: MULT
port map(A=>L12, B=>f08_D_BUF, ST=>MULT_ST, clk=>clk,
rst=>MULT_rst,PROD=>C6_2, DONE=>F_C6_2);

U35: MULT
port map(A=>L21, B=>f19_D_BUF, ST=>MULT_ST, clk=>clk,
rst=>MULT_rst,PROD=>C6_3, DONE=>F_C6_3);

U36: ADD_4_16
port map(A1=>C6_1, A2=>C6_1, A3=>C6_2, A4=>C6_3, EN=>EN_ADD,
SUM=>F6_INT);

U37: SUB_2_16
port map(A=>F6_INT(15 downto 0), B=>C6M,  S_AB=>'1', EN=>EN_SUB,
SUB=>C6_LX(16 downto 0));

-------------------------------------------

U38: MULT
port map(A=>L13, B=>C7, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C7_1, DONE=>F_C7_1);

U39: MULT
port map(A=>L14, B=>f08_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C7_2, DONE=>F_C7_2);

U40: MULT
port map(A=>L22, B=>f19_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C7_3, DONE=>F_C7_3);

U41: ADD_4_16
port map(A1=>C7_1, A2=>C7_1, A3=>C7_2, A4=>C7_3, EN=>EN_ADD,
SUM=>F7_INT);

U42: SUB_2_16
port map(A=>F7_INT(15 downto 0), B=>C7M, S_AB=>'1', EN=>EN_SUB,
SUB=>C7_LX(16 downto 0));

-------------------------------------------
end DCT_RECTAN_R1_UPDATE_arch;


-- The testbench for independent DCT-II one-point update circuit

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_unsigned.all;
use ieee.math_real.all;
```

```
use ieee.numeric_std.all;
entity DCT_RECTAN_R1_UPDATE_test is
end DCT_RECTAN_R1_UPDATE_test;

architecture DCT_RECTAN_R1_UPDATE_test_arch of
DCT_RECTAN_R1_UPDATE_test is

component DCT_RECTAN_R1_UPDATE
port(f0, f1, f2, f3, f4, f5, f6, f7: in std_logic_vector(8 downto
0):=(others=>'0');
clk, rst, conven, update: in std_logic;
C0, C1, C2, C3, C4, C5, C6, C7: buffer std_logic_vector(15 downto
0):=(others=>'0');
DONE_DCTCONVEN: out std_logic:='0';
DONE_UPDATE: out std_logic:='0');
end component;

signal f0in: std_logic_vector(8 downto 0):=(others=>'0');
signal f1in: std_logic_vector(8 downto 0):=(others=>'0');
signal f2in: std_logic_vector(8 downto 0):=(others=>'0');
signal f3in: std_logic_vector(8 downto 0):=(others=>'0');
signal f4in: std_logic_vector(8 downto 0):=(others=>'0');
signal f5in: std_logic_vector(8 downto 0):=(others=>'0');
signal f6in: std_logic_vector(8 downto 0):=(others=>'0');
signal f7in: std_logic_vector(8 downto 0):=(others=>'0');

signal C0: std_logic_vector(15 downto 0):=(others=>'0');
signal C1: std_logic_vector(15 downto 0):=(others=>'0');
signal C2: std_logic_vector(15 downto 0):=(others=>'0');
signal C3: std_logic_vector(15 downto 0):=(others=>'0');
signal C4: std_logic_vector(15 downto 0):=(others=>'0');
signal C5: std_logic_vector(15 downto 0):=(others=>'0');
signal C6: std_logic_vector(15 downto 0):=(others=>'0');
signal C7: std_logic_vector(15 downto 0):=(others=>'0');

signal CLK: std_logic:='0';
signal RST: std_logic:='0';
signal CONVEN: std_logic:='0';
signal UPDATE: std_logic:='0';
signal DONE_DCTCONVEN: std_logic:='0';
signal DONE_UPDATE: std_logic:='0';

begin

CLK<=not(CLK) after 50 ns;
RST<='1', '0' after 300 ns;

process
begin

CONVEN<='0';
UPDATE<='0';

wait on RST until (RST'event and RST='0');

f0in<="111111011";
f1in<="110100100";
```

```
f2in<="010101100";
f3in<="001010101";
f4in<="001101011";
f5in<="101111010";
f6in<="001010101";
f7in<="110011010";

wait for 100 ns;

CONVEN<='1';
wait for 200 ns;
CONVEN<='0';

wait on DONE_DCTCONVEN until (DONE_DCTCONVEN='1');

f7in<="001010101";--f8

wait on DONE_DCTCONVEN until (DONE_DCTCONVEN='1');

f7in<="111010110";--f9

wait for 100 ns;

UPDATE<='1';
wait for 200 ns;
UPDATE<='0';

wait on DONE_UPDATE until (DONE_UPDATE='1');

f7in<="101111000";--f10

wait for 100 ns;

UPDATE<='1';
wait for 200 ns;
UPDATE<='0';

end process;

U: DCT_RECTAN_R1_UPDATE
port map(f0=>f0in, f1=>f1in, f2=>f2in, f3=>f3in, f4=>f4in,
f5=>f5in, f6=>f6in, f7=>f7in,
clk=>CLK, rst=>RST, conven=>CONVEN, update=>UPDATE,
C0=>C0, C1=>C1, C2=>C2, C3=>C3,
C4=>C4, C5=>C5, C6=>C6, C7=>C7,
DONE_DCTCONVEN=>DONE_DCTCONVEN,
DONE_UPDATE=>DONE_UPDATE);

end DCT_RECTAN_R1_UPDATE_test_arch;
-- Entity: DCT_RECTAN_R4_UPDATE  for independent four point update of
DCT type II
```

```
-- Four-point update controller for DCT-II rectangular update algorithm
-- developed in Chapter 2. The components DCT_DST_CONVEN, BUTTER_FLY,
-- IP_STAGE, MULT, EXT_UNIT used in this controller are given in [31].

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_unsigned.all;
use ieee.math_real.all;
use ieee.numeric_std.all;

entity DCT_RECTAN_R4_UPDATE is
port(f0, f1, f2, f3, f4, f5, f6, f7: in std_logic_vector(8 downto
0):=(others=>'0');
clk, rst, conven, update: in std_logic;
C0, C1, C2, C3, C4, C5, C6, C7: buffer std_logic_vector(15 downto
0):=(others=>'0');
DONE_DCTCONVEN: out std_logic:='1';
DONE_UPDATE: out std_logic:='1');
end DCT_RECTAN_R4_UPDATE;

architecture DCT_RECTAN_R4_UPDATE_arch of DCT_RECTAN_R4_UPDATE is

component DCT_DST_CONVEN
port(x07_S, x16_S, x25_S, x34_S, x07_D, x16_D, x25_D, x34_D: in
std_logic_vector(9 downto 0):=(others=>'0');
clk, rst, conv: in std_logic;
X0_COS, X1_COS, X2_COS, X3_COS, X4_COS, X5_COS, X6_COS, X7_COS: out
std_logic_vector(15 downto 0):=(others=>'0');
X1_SIN, X2_SIN, X3_SIN, X4_SIN, X5_SIN, X6_SIN, X7_SIN, X8_SIN: out
std_logic_vector(15 downto 0):=(others=>'0');
DONE_CONVEN: out std_logic:='1');
end component;

component BUTTER_FLY
port(x0, x1, x2, x3, x4, x5, x6, x7: in std_logic_vector(8 downto 0);
EN: in std_logic;
X0OUT, X1OUT, X2OUT, X3OUT, X4OUT, X5OUT, X6OUT, X7OUT: out
std_logic_vector(9 downto 0):=(others=>'0'));
end component;

component IP_STAGE_R4
port(x0, x1, x2, x3, x4, x5, x6, x7: in std_logic_vector(8 downto 0);
Data_L, Update_L, Data_S: in std_logic;
X0OUT, X1OUT, X2OUT, X3OUT, X4OUT, X5OUT, X6OUT, X7OUT: buffer
std_logic_vector(8
downto 0):=(others=>'0'));
end component;

component MULT
port(A: in std_logic_vector(8 downto 0);
B: in std_logic_vector(15 downto 0);
ST, clk, rst: in std_logic;
```

```
PROD: out std_logic_vector(15 downto 0):=(others=>'0');
DONE: out std_logic:='1');
end component;

component EXT_UNIT
port(A: in std_logic_vector(9 downto 0);
EXT: in std_logic;
A_EXT: out std_logic_vector(15 downto 0):=(others=>'0'));
end component;

component ADD_3_16
port(A1, A2, A3: in std_logic_vector(15 downto 0);
EN: in std_logic;
SUM: out std_logic_vector(17 downto 0):=(others=>'0'));
end component;

component ADD_8_16
port(A1, A2, A3, A4, A5, A6, A7, A8: in std_logic_vector(15 downto 0);
EN: in std_logic;
SUM: out std_logic_vector(17 downto 0):=(others=>'0'));
end component;

component SUB_2_16
port(A, B: in std_logic_vector(15 downto 0);
S_AB: in std_logic:='1';
EN: in std_logic;
SUB: out std_logic_vector(16 downto 0):=(others=>'0'));
end component;

type STATE is (s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12,
s13, s14, s15, s16, s17, s18, s19, s20, s21, s22, s23, s24, s25, s26,
s27, s28, s29, s30, s31, s32, s33, s34, s35, s36, s37, s38, s39, s40,
s41, s42, s43, s44, s45, s46, s47, s48, s49, s50, s51, s52, s53, s54,
s55, s56, s57, s58);

signal PS, NS: STATE;
signal trig: std_logic:='0';
signal rst_DCT_DST_CONVEN: std_logic:='1';
signal conv_DCT_DST_CONVEN: std_logic:='0';
signal DONE_DCT_DST_CONVEN: std_logic:='0';
signal Data_LAT: std_logic:='0';
signal Update_LAT: std_logic:='0';
signal Data_SHIFT: std_logic:='0';
signal EN_BUTTER_FLY: std_logic:='0';
signal MULT_ST: std_logic:='0';
signal MULT_rst: std_logic:='0';
signal EN_ADD1: std_logic:='0';
signal EN_ADD2: std_logic:='0';
signal EXT: std_logic:='0';
signal EN_SUB: std_logic:='0';

signal f08_S_EXT: std_logic_vector(15 downto 0):=(others=>'0');
signal f19_S_EXT: std_logic_vector(15 downto 0):=(others=>'0');
signal f210_S_EXT: std_logic_vector(15 downto 0):=(others=>'0');
signal f311_S_EXT: std_logic_vector(15 downto 0):=(others=>'0');

signal f08_D_EXT: std_logic_vector(15 downto 0):=(others=>'0');
```

```
signal f19_D_EXT: std_logic_vector(15 downto 0):=(others=>'0');
signal f210_D_EXT: std_logic_vector(15 downto 0):=(others=>'0');
signal f311_D_EXT: std_logic_vector(15 downto 0):=(others=>'0');

signal f08_S_BUF: std_logic_vector(15 downto 0):=(others=>'0');
signal f19_S_BUF: std_logic_vector(15 downto 0):=(others=>'0');
signal f210_S_BUF: std_logic_vector(15 downto 0):=(others=>'0');
signal f311_S_BUF: std_logic_vector(15 downto 0):=(others=>'0');
signal f412_S_BUF: std_logic_vector(15 downto 0):=(others=>'0');
signal f513_S_BUF: std_logic_vector(15 downto 0):=(others=>'0');
signal f614_S_BUF: std_logic_vector(15 downto 0):=(others=>'0');
signal f715_S_BUF: std_logic_vector(15 downto 0):=(others=>'0');

signal f08_D_BUF: std_logic_vector(15 downto 0):=(others=>'0');
signal f19_D_BUF: std_logic_vector(15 downto 0):=(others=>'0');
signal f210_D_BUF: std_logic_vector(15 downto 0):=(others=>'0');
signal f311_D_BUF: std_logic_vector(15 downto 0):=(others=>'0');
signal f412_D_BUF: std_logic_vector(15 downto 0):=(others=>'0');
signal f513_D_BUF: std_logic_vector(15 downto 0):=(others=>'0');
signal f614_D_BUF: std_logic_vector(15 downto 0):=(others=>'0');
signal f715_D_BUF: std_logic_vector(15 downto 0):=(others=>'0');

signal F_C0_1, F_C0_2, F_C0_3, F_C0_4, F_C0_5, F_C0_6, F_C0_7,
F_C0_8:std_logic:='0';

signal F_C1_1, F_C1_2, F_C1_3, F_C1_4, F_C1_5, F_C1_6, F_C1_7,
F_C1_8:std_logic:='0';

signal F_C2_1, F_C2_2, F_C2_3, F_C2_4, F_C2_5, F_C2_6, F_C2_7,
F_C2_8:std_logic:='0';

signal F_C3_1, F_C3_2, F_C3_3, F_C3_4, F_C3_5, F_C3_6, F_C3_7,
F_C3_8:std_logic:='0';

signal F_C4_1, F_C4_2, F_C4_3, F_C4_4, F_C4_5, F_C4_6, F_C4_7,
F_C4_8:std_logic:='0';

signal F_C5_1, F_C5_2, F_C5_3, F_C5_4, F_C5_5, F_C5_6, F_C5_7,
F_C5_8:std_logic:='0';

signal F_C6_1, F_C6_2, F_C6_3, F_C6_4, F_C6_5, F_C6_6, F_C6_7,
F_C6_8:std_logic:='0';

signal F_C7_1, F_C7_2, F_C7_3, F_C7_4, F_C7_5, F_C7_6, F_C7_7,
F_C7_8:std_logic:='0';

signal F_C0: std_logic:='0';
signal F_C1: std_logic:='0';
signal F_C2: std_logic:='0';
signal F_C3: std_logic:='0';
signal F_C4: std_logic:='0';
signal F_C5: std_logic:='0';
signal F_C6: std_logic:='0';
signal F_C7: std_logic:='0';
signal F_C: std_logic:='0';

signal IP_BF_0: std_logic_vector(8 downto 0):=(others=>'0');
```

```
signal IP_BF_1: std_logic_vector(8 downto 0):=(others=>'0');
signal IP_BF_2: std_logic_vector(8 downto 0):=(others=>'0');
signal IP_BF_3: std_logic_vector(8 downto 0):=(others=>'0');
signal IP_BF_4: std_logic_vector(8 downto 0):=(others=>'0');
signal IP_BF_5: std_logic_vector(8 downto 0):=(others=>'0');
signal IP_BF_6: std_logic_vector(8 downto 0):=(others=>'0');
signal IP_BF_7: std_logic_vector(8 downto 0):=(others=>'0');

signal BF_CONVEN_0: std_logic_vector(9 downto 0):=(others=>'0');
signal BF_CONVEN_1: std_logic_vector(9 downto 0):=(others=>'0');
signal BF_CONVEN_2: std_logic_vector(9 downto 0):=(others=>'0');
signal BF_CONVEN_3: std_logic_vector(9 downto 0):=(others=>'0');
signal BF_CONVEN_4: std_logic_vector(9 downto 0):=(others=>'0');
signal BF_CONVEN_5: std_logic_vector(9 downto 0):=(others=>'0');
signal BF_CONVEN_6: std_logic_vector(9 downto 0):=(others=>'0');
signal BF_CONVEN_7: std_logic_vector(9 downto 0):=(others=>'0');


signal C0M: std_logic_vector(15 downto 0):=(others=>'0');
signal C1M: std_logic_vector(15 downto 0):=(others=>'0');
signal C2M: std_logic_vector(15 downto 0):=(others=>'0');
signal C3M: std_logic_vector(15 downto 0):=(others=>'0');
signal C4M: std_logic_vector(15 downto 0):=(others=>'0');
signal C5M: std_logic_vector(15 downto 0):=(others=>'0');
signal C6M: std_logic_vector(15 downto 0):=(others=>'0');
signal C7M: std_logic_vector(15 downto 0):=(others=>'0');

signal C0_L: std_logic_vector(15 downto 0):=(others=>'0');
signal C1_L: std_logic_vector(15 downto 0):=(others=>'0');
signal C2_L: std_logic_vector(15 downto 0):=(others=>'0');
signal C3_L: std_logic_vector(15 downto 0):=(others=>'0');
signal C4_L: std_logic_vector(15 downto 0):=(others=>'0');
signal C5_L: std_logic_vector(15 downto 0):=(others=>'0');
signal C6_L: std_logic_vector(15 downto 0):=(others=>'0');
signal C7_L: std_logic_vector(15 downto 0):=(others=>'0');

signal C0_LX: std_logic_vector(17 downto 0):=(others=>'0');
signal C1_LX: std_logic_vector(17 downto 0):=(others=>'0');
signal C2_LX: std_logic_vector(17 downto 0):=(others=>'0');
signal C3_LX: std_logic_vector(17 downto 0):=(others=>'0');
signal C4_LX: std_logic_vector(17 downto 0):=(others=>'0');
signal C5_LX: std_logic_vector(17 downto 0):=(others=>'0');
signal C6_LX: std_logic_vector(17 downto 0):=(others=>'0');
signal C7_LX: std_logic_vector(17 downto 0):=(others=>'0');

signal C0_LXS: std_logic_vector(17 downto 0):=(others=>'0');
signal C1_LXS: std_logic_vector(17 downto 0):=(others=>'0');
signal C2_LXS: std_logic_vector(17 downto 0):=(others=>'0');
signal C3_LXS: std_logic_vector(17 downto 0):=(others=>'0');
signal C4_LXS: std_logic_vector(17 downto 0):=(others=>'0');
signal C5_LXS: std_logic_vector(17 downto 0):=(others=>'0');
signal C6_LXS: std_logic_vector(17 downto 0):=(others=>'0');
signal C7_LXS: std_logic_vector(17 downto 0):=(others=>'0');

signal C0_LXSS: std_logic_vector(17 downto 0):=(others=>'0');
signal C1_LXSS: std_logic_vector(17 downto 0):=(others=>'0');
signal C2_LXSS: std_logic_vector(17 downto 0):=(others=>'0');
```

```
signal C3_LXSS: std_logic_vector(17 downto 0):=(others=>'0');
signal C4_LXSS: std_logic_vector(17 downto 0):=(others=>'0');
signal C5_LXSS: std_logic_vector(17 downto 0):=(others=>'0');
signal C6_LXSS: std_logic_vector(17 downto 0):=(others=>'0');
signal C7_LXSS: std_logic_vector(17 downto 0):=(others=>'0');

signal C0_1, C0_2, C0_3, C0_4, C0_5, C0_6, C0_7, C0_8:
std_logic_vector(15
downto 0):=(others=>'0');

signal C1_1, C1_2, C1_3, C1_4, C1_5, C1_6, C1_7, C1_8:
std_logic_vector(15
downto 0):=(others=>'0');

signal C2_1, C2_2, C2_3, C2_4, C2_5, C2_6, C2_7, C2_8:
std_logic_vector(15
downto 0):=(others=>'0');

signal C3_1, C3_2, C3_3, C3_4, C3_5, C3_6, C3_7, C3_8:
std_logic_vector(15
downto 0):=(others=>'0');

signal C4_1, C4_2, C4_3, C4_4, C4_5, C4_6, C4_7, C4_8:
std_logic_vector(15
downto 0):=(others=>'0');

signal C5_1, C5_2, C5_3, C5_4, C5_5, C5_6, C5_7, C5_8:
std_logic_vector(15
downto 0):=(others=>'0');

signal C6_1, C6_2, C6_3, C6_4, C6_5, C6_6, C6_7, C6_8:
std_logic_vector(15
downto 0):=(others=>'0');

signal C7_1, C7_2, C7_3, C7_4, C7_5, C7_6, C7_7, C7_8:
std_logic_vector(15
downto 0):=(others=>'0');

signal L1: std_logic_vector(8 downto 0):="001011010";--(0.3515625)
signal L2: std_logic_vector(8 downto 0):="000011000";--(0.921875)
signal L3: std_logic_vector(8 downto 0):="001000111";--(0.27734375)
signal L4: std_logic_vector(8 downto 0):="001101010";--(0.4140625)
signal L5: std_logic_vector(8 downto 0):="001111101";--(0.48828125)
signal L6: std_logic_vector(8 downto 0):="110000011";--(-0.48828125)
signal L7: std_logic_vector(8 downto 0):="110010110";--(-0.4140625)
signal L8: std_logic_vector(8 downto 0):="110111001";--(-0.27734375)
signal L9: std_logic_vector(8 downto 0):="111101000";--(-0.09375)
signal L10: std_logic_vector(8 downto 0):="001110110";--(0.4609375)
signal L11: std_logic_vector(8 downto 0):="000110000";--(0.1875)
signal L12: std_logic_vector(8 downto 0):="111001111";--(-0.19140625)
signal L13: std_logic_vector(8 downto 0):="110001001";--(-0.46484375)
signal L14: std_logic_vector(8 downto 0):="110111000";--(-0.28125)
signal L15: std_logic_vector(8 downto 0):="110000010";--(-0.4921875)
signal L16: std_logic_vector(8 downto 0):="111100111";--(-0.09765625)
signal L17: std_logic_vector(8 downto 0):="110100101";--(-0.35546875)
signal L18: std_logic_vector(8 downto 0):="110010101";--(-0.41796875)
signal L19: std_logic_vector(8 downto 0):="110100110";--(-0.3515625)
```

```
signal L20: std_logic_vector(8 downto 0):="110001010";--(-0.4609375)
signal L21: std_logic_vector(8 downto 0):="111010000";--(-0.1875)
signal L22: std_logic_vector(8 downto 0):="000110001";--(0.19140625)
signal L23: std_logic_vector(8 downto 0):="001110111";--(0.46484375)
signal L24: std_logic_vector(8 downto 0):="001011011";--(0.35546875)

begin
L1<="001011010";--(0.3515625)
L2<="000011000";--(0.921875)
L3<="001000111";--(0.27734375)
L4<="001101010";--(0.4140625)
L5<="001111101";--(0.48828125)
L6<="110000011";--(-0.48828125)
L7<="110010110";--(-0.4140625)
L8<="110111001";--(-0.27734375)
L9<="111101000";--(-0.09375)
L10<="001110110";--(0.4609375)
L11<="000110000";--(0.1875)
L12<="111001111";--(-0.19140625)
L13<="110001001";--(-0.46484375)
L14<="110111000";--(-0.28125)
L15<="110000010";--(-0.4921875)
L16<="111100111";--(-0.09765625)
L17<="110100101";--(-0.35546875)
L18<="110010101";--(-0.41796875)
L19<="110100110";--(-0.3515625)
L20<="110001010";--(-0.4609375)
L21<="111010000";--(-0.1875)
L22<="000110001";--(0.19140625)
L23<="001110111";--(0.46484375)
L24<="001011011";--(0.35546875)

F_C0<=F_C0_1 and F_C0_2 and F_C0_3 and F_C0_4 and F_C0_5 and F_C0_6 and
F_C0_7 and F_C0_8;
F_C1<=F_C1_1 and F_C1_2 and F_C1_3 and F_C1_4 and F_C1_5 and F_C1_6 and
F_C1_7 and F_C1_8;
F_C2<=F_C2_1 and F_C2_2 and F_C2_3 and F_C2_4 and F_C2_5 and F_C2_6 and
F_C2_7 and F_C2_8;
F_C3<=F_C3_1 and F_C3_2 and F_C3_3 and F_C3_4 and F_C3_5 and F_C3_6 and
F_C3_7 and F_C3_8;
F_C4<=F_C4_1 and F_C4_2 and F_C4_3 and F_C4_4 and F_C4_5 and F_C4_6 and
F_C4_7 and F_C4_8;
F_C5<=F_C5_1 and F_C5_2 and F_C5_3 and F_C5_4 and F_C5_5 and F_C5_6 and
F_C5_7 and F_C5_8;
F_C6<=F_C6_1 and F_C6_2 and F_C6_3 and F_C6_4 and F_C6_5 and F_C6_6 and
F_C6_7 and F_C6_8;
F_C7<=F_C7_1 and F_C7_2 and F_C7_3 and F_C7_4 and F_C7_5 and F_C7_6 and
F_C7_7 and F_C7_8;

F_C<= F_C0 and F_C1 and F_C2 and F_C3 and F_C4 and F_C5 and F_C6 and
F_C7;

process(clk, rst)
begin
if(rst='1')then
PS<=s0;
elsif(rising_edge(clk))then
```

```
PS<=NS;
trig<=not(trig);
end if;
end process;

process(PS, trig, conven, update, DONE_DCT_DST_CONVEN, F_C)
begin
case PS is

-- data-preparation stage I

when s0=>
rst_DCT_DST_CONVEN<='1';
conv_DCT_DST_CONVEN<='0';
MULT_rst<='1';
MULT_ST<='0';
EN_ADD1<='0';
EN_ADD2<='0';
EN_BUTTER_FLY<='0';
Data_LAT<='0';
Update_LAT<='0';
Data_SHIFT<='0';
EXT<='0';
DONE_DCTCONVEN<='0';
DONE_UPDATE<='0';
if(conven='1')then
NS<=s1;
elsif(update='1')then
NS<=s33;
else
NS<=s0;
end if;

when s1=>
Data_LAT<='1';
DONE_DCTCONVEN<='0';
rst_DCT_DST_CONVEN<='0';
NS<=s2;

when s2=>
NS<=s3;

when s3=>
Data_LAT<='0';
EN_BUTTER_FLY<='1';
NS<=s4;

when s4=>
NS<=s5;

when s5=>
EN_BUTTER_FLY<='0';
NS<=s6;

when s6=>
conv_DCT_DST_CONVEN<='1';
NS<=s7;
```

```
when s7=>
NS<=s8;

when s8=>
conv_DCT_DST_CONVEN<='0';
if(DONE_DCT_DST_CONVEN='1')then
NS<=s9;
else
NS<=s8;
end if;


when s9=>    -- previous time unit DCT coefficients
C0M<=C0_L;
C1M<=C1_L;
C2M<=C2_L;
C3M<=C3_L;
C4M<=C4_L;
C5M<=C5_L;
C6M<=C6_L;
C7M<=C7_L;
DONE_DCTCONVEN<='1';
NS<=s10;

when s10=>
rst_DCT_DST_CONVEN<='1';
conv_DCT_DST_CONVEN<='0';
NS<=s11;

-- data-preparation stage II
-- current time unit DCT coefficient calculation

when s11=>
Update_LAT<='1';
DONE_DCTCONVEN<='0';
rst_DCT_DST_CONVEN<='0';
NS<=s12;

when s12=>
NS<=s13;

when s13=>
Update_LAT<='0';
NS<=s14;

when s14=>
NS<=s15;

when s15=>
EN_BUTTER_FLY<='1';
NS<=s16;

when s16=>
NS<=s17;

when s17=>
```

```
EN_BUTTER_FLY<='0';
NS<=s18;

when s18=>
NS<=s19;

when s19=>
EXT<='1';
NS<=s20;

when s20=>
NS<=s21;

when s21=>
EXT<='0';
F08_S_BUF<=f08_S_EXT;
F19_S_BUF<=f19_S_EXT;
F210_S_BUF<=f210_S_EXT;
F311_S_BUF<=f311_S_EXT;
F08_D_BUF<=f08_D_EXT;
F19_D_BUF<=f19_D_EXT;
F210_D_BUF<=f210_D_EXT;
F311_D_BUF<=f311_D_EXT;
NS<=s22;

when s22=>
NS<=s23;

when s23=>
Data_SHIFT<='1';
NS<=s24;

when s24=>
NS<=s25;

when s25=>
Data_SHIFT<='0';
NS<=s26;

when s26=>
EN_BUTTER_FLY<='1';
NS<=s27;

when s27=>
NS<=s28;

when s28=>
EN_BUTTER_FLY<='0';
NS<=s29;

when s29=>
conv_DCT_DST_CONVEN<='1';
NS<=s30;

when s30=>
NS<=s31;
```

```
when s31=>
conv_DCT_DST_CONVEN<='0';
if(DONE_DCT_DST_CONVEN='1')then
NS<=s32;
else
NS<=s31;
end if;

when s32=>
C0<=C0_L;
C1<=C1_L;
C2<=C2_L;
C3<=C3_L;
C4<=C4_L;
C5<=C5_L;
C6<=C6_L;
C7<=C7_L;
DONE_DCTCONVEN<='1';
NS<=s0;

-- Update mode of operation

when s33=>
Update_LAT<='1';
DONE_DCTCONVEN<='0';
MULT_rst<='0';
NS<=s34;

when s34=>
NS<=s35;

when s35=>
Update_LAT<='0';
EN_BUTTER_FLY<='1';
NS<=s36;

when s36=>
NS<=s37;

when s37=>
EN_BUTTER_FLY<='0';
EXT<='1';
NS<=s38;

when s38=>
EXT<='0';
F412_S_BUF<=f08_S_EXT;
F513_S_BUF<=f19_S_EXT;
F614_S_BUF<=f210_S_EXT;
F715_S_BUF<=f311_S_EXT;
F412_D_BUF<=f08_D_EXT;
F513_D_BUF<=f19_D_EXT;
F614_D_BUF<=f210_D_EXT;
F715_D_BUF<=f311_D_EXT;
NS<=s39;

when s39=>
```

```
NS<=s40;

when s40=>
MULT_ST<='1';
NS<=s41;

when s41=>
NS<=s42;

when s42=>
MULT_ST<='0';
if(F_C='1')then
NS<=s43;
else
NS<=s42;
end if;

when s43=>
EN_ADD1<='1';
NS<=s44;

when s44=>
NS<=s45;

when s45=>
EN_ADD1<='0';
NS<=s46;

when s46=>
EN_ADD2<='1';
NS<=s47;

when s47=>
NS<=s48;

when s48=>
EN_ADD2<='0';
NS<=s49;

when s49=>
NS<=s50;

when s50=>
EN_SUB<='1';
NS<=s51;

when s51=>
EN_SUB<='0';
NS<=s52;

when s52=>
NS<=s53;

when s53=>
Data_SHIFT<='1';
NS<=s54;
```

```
when s54=>
NS<=s55;

when s55=>
Data_SHIFT<='0';
NS<=s56;

when s56=>
F08_S_BUF<=F412_S_BUF;
F19_S_BUF<=F513_S_BUF;
F210_S_BUF<=F614_S_BUF;
F311_S_BUF<=F715_S_BUF;
F08_D_BUF<=F412_D_BUF;
F19_D_BUF<=F513_D_BUF;
F210_D_BUF<=F614_D_BUF;
F311_D_BUF<=F715_D_BUF;

C0M<=C0;
C1M<=C1;
C2M<=C2;
C3M<=C3;
C4M<=C4;
C5M<=C5;
C6M<=C6;
C7M<=C7;
NS<=s57;

when s57=>
NS<=s58;

when s58=>
C0<=C0_LX(15 downto 0);
C1<=C1_LX(15 downto 0);
C2<=C2_LX(15 downto 0);
C3<=C3_LX(15 downto 0);
C4<=C4_LX(15 downto 0);
C5<=C5_LX(15 downto 0);
C6<=C6_LX(15 downto 0);
C7<=C7_LX(15 downto 0);
DONE_UPDATE<='1';
NS<=s0;

end case;
end process;

U0: IP_STAGE_R4
port map(x0=>f0, x1=>f1, x2=>f2, x3=>f3, x4=>f4, x5=>f5, x6=>f6,
x7=>f7,
Data_L=>Data_LAT, Update_L=>Update_LAT, Data_S=>Data_SHIFT,
X0OUT=>IP_BF_0, X1OUT=>IP_BF_1, X2OUT=>IP_BF_2, X3OUT=>IP_BF_3,
X4OUT=>IP_BF_4, X5OUT=>IP_BF_5, X6OUT=>IP_BF_6, X7OUT=>IP_BF_7);

U1: BUTTER_FLY
port map(x0=>IP_BF_0, x1=>IP_BF_1, x2=>IP_BF_2, x3=>IP_BF_3,
x4=>IP_BF_4, x5=>IP_BF_5, x6=>IP_BF_6, x7=>IP_BF_7,
EN=>EN_BUTTER_FLY,
X0OUT=>BF_CONVEN_0, X1OUT=>BF_CONVEN_1, X2OUT=>BF_CONVEN_2,
```

```
X3OUT=>BF_CONVEN_3,
X4OUT=>BF_CONVEN_4, X5OUT=>BF_CONVEN_5, X6OUT=>BF_CONVEN_6,
X7OUT=>BF_CONVEN_7);

U2: DCT_DST_CONVEN
port map(x07_S=>BF_CONVEN_0, x16_S=>BF_CONVEN_1, x25_S=>BF_CONVEN_2,
x34_S=>BF_CONVEN_3, x07_D=>BF_CONVEN_4, x16_D=>BF_CONVEN_5,
x25_D=>BF_CONVEN_6, x34_D=>BF_CONVEN_7,
clk=>clk, rst=>rst_DCT_DST_CONVEN, conv=>conv_DCT_DST_CONVEN,
X0_COS=>C0_L, X1_COS=>C1_L, X2_COS=>C2_L, X3_COS=>C3_L, X4_COS=>C4_L,
X5_COS=>C5_L, X6_COS=>C6_L, X7_COS=>C7_L,
DONE_CONVEN=>DONE_DCT_DST_CONVEN);

U3: EXT_UNIT
port map(A=>BF_CONVEN_0, EXT=>EXT,A_EXT=>f08_S_EXT);

U4: EXT_UNIT
port map(A=>BF_CONVEN_1, EXT=>EXT,A_EXT=>f19_S_EXT);

U5: EXT_UNIT
port map(A=>BF_CONVEN_2, EXT=>EXT,A_EXT=>f210_S_EXT);

U6: EXT_UNIT
port map(A=>BF_CONVEN_3, EXT=>EXT,A_EXT=>f311_S_EXT);

U7: EXT_UNIT
port map(A=>BF_CONVEN_4, EXT=>EXT,A_EXT=>f08_D_EXT);

U8: EXT_UNIT
port map(A=>BF_CONVEN_5, EXT=>EXT,A_EXT=>f19_D_EXT);

U9: EXT_UNIT
port map(A=>BF_CONVEN_6, EXT=>EXT,A_EXT=>f210_D_EXT);

U10: EXT_UNIT
port map(A=>BF_CONVEN_7, EXT=>EXT,A_EXT=>f311_D_EXT);
----------------------------------------
U11: MULT
port map(A=>L19, B=>f715_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C0_1, DONE=>F_C0_1);

U12: MULT
port map(A=>L19, B=>f614_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C0_2, DONE=>F_C0_2);

U13: MULT
port map(A=>L19, B=>f513_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C0_3, DONE=>F_C0_3);

U14: MULT
port map(A=>L19, B=>f412_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C0_4, DONE=>F_C0_4);

U15: MULT
port map(A=>L1, B=>f311_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C0_5, DONE=>F_C0_5);
```

```
U16: MULT
port map(A=>L1, B=>f210_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C0_6, DONE=>F_C0_6);

U17: MULT
port map(A=>L1, B=>f19_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C0_7, DONE=>F_C0_7);

U18: MULT
port map(A=>L1, B=>f08_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C0_8, DONE=>F_C0_8);

U19: ADD_8_16
port map(A1=>C0_1, A2=>C0_2, A3=>C0_3, A4=>C0_4, A5=>C0_5, A6=>C0_6,
A7=>C0_7, A8=>C0_8, EN=>EN_ADD1, SUM=>C0_LXSS);

U20: ADD_3_16
port map(A1=>C0, A2=>C0_LXSS(15 downto 0), A3=>C0, EN=>EN_ADD2,
SUM=>C0_LXS);

U21: SUB_2_16
port map(A=>C0_LXS(15 downto 0), B=>C0M, S_AB=>'1', EN=>EN_SUB,
SUB=>C0_LX(16 downto 0));
-------------------------------------------
U22: MULT
port map(A=>L2, B=>f311_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C1_1, DONE=>F_C1_1);

U23: MULT
port map(A=>L3, B=>f210_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C1_2, DONE=>F_C1_2);

U24: MULT
port map(A=>L4, B=>f19_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C1_3, DONE=>F_C1_3);

U25: MULT
port map(A=>L5, B=>f08_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C1_4, DONE=>F_C1_4);

U26: MULT
port map(A=>L6, B=>f715_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C1_5, DONE=>F_C1_5);

U27: MULT
port map(A=>L7, B=>f614_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C1_6, DONE=>F_C1_6);

U28: MULT
port map(A=>L8, B=>f513_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C1_7, DONE=>F_C1_7);

U29: MULT
port map(A=>L9, B=>f412_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C1_8, DONE=>F_C1_8);

U30: ADD_8_16
```

```
port map(A1=>C1_1, A2=>C1_2, A3=>C1_3, A4=>C1_4, A5=>C1_5, A6=>C1_6,
A7=>C1_7, A8=>C1_8, EN=>EN_ADD1, SUM=>C1_LXSS);

U31: SUB_2_16
port map(A=>C1_LXSS(15 downto 0), B=>C1M, S_AB=>'1', EN=>EN_SUB,
SUB=>C1_LX(16 downto 0));
-----------------------------------------
U32: MULT
port map(A=>L20, B=>f715_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C2_1, DONE=>F_C2_1);

U33: MULT
port map(A=>L21, B=>f614_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C2_2, DONE=>F_C2_2);

U34: MULT
port map(A=>L22, B=>f513_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C2_3, DONE=>F_C2_3);

U35: MULT
port map(A=>L23, B=>f412_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C2_4, DONE=>F_C2_4);

U36: MULT
port map(A=>L20, B=>f311_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C2_5, DONE=>F_C2_5);

U37: MULT
port map(A=>L21, B=>f210_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C2_6, DONE=>F_C2_6);

U38: MULT
port map(A=>L22, B=>f19_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C2_7, DONE=>F_C2_7);

U39: MULT
port map(A=>L23, B=>f08_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C2_8, DONE=>F_C2_8);

U40: ADD_8_16
port map(A1=>C2_1, A2=>C2_2, A3=>C2_3, A4=>C2_4, A5=>C2_5, A6=>C2_6,
A7=>C2_7, A8=>C2_8, EN=>EN_ADD1, SUM=>C2_LXSS);

U41: ADD_3_16
port map(A1=>C2, A2=>C2, A3=>C2M, EN=>EN_ADD2, SUM=>C2_LXS);

U42: SUB_2_16
port map(A=>C2_LXSS(15 downto 0), B=>C2_LXS(15 downto 0), S_AB=>'1',
EN=>EN_SUB, SUB=>C2_LX(16 downto 0));
-----------------------------------------
U43: MULT
port map(A=>L14, B=>f311_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C3_1, DONE=>F_C3_1);

U44: MULT
port map(A=>L15, B=>f210_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C3_2, DONE=>F_C3_2);
```

```
U45: MULT
port map(A=>L16, B=>f19_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C3_3, DONE=>F_C3_3);

U46: MULT
port map(A=>L4, B=>f08_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C3_4, DONE=>F_C3_4);

U47: MULT
port map(A=>L7, B=>f715_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C3_5, DONE=>F_C3_5);

U48: MULT
port map(A=>L2, B=>f614_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C3_6, DONE=>F_C3_6);

U49: MULT
port map(A=>L5, B=>f513_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C3_7, DONE=>F_C3_7);

U50: MULT
port map(A=>L3, B=>f412_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C3_8, DONE=>F_C3_8);

U51: ADD_8_16
port map(A1=>C3_1, A2=>C3_2, A3=>C3_3, A4=>C3_4, A5=>C3_5, A6=>C3_6,
A7=>C3_7, A8=>C3_8, EN=>EN_ADD1, SUM=>C3_LXSS);

U52: SUB_2_16
port map(A=>C3_LXSS(15 downto 0), B=>C3M, S_AB=>'1', EN=>EN_SUB,
SUB=>C3_LX(16 downto 0));
-----------------------------------------
U53: MULT
port map(A=>L17, B=>f715_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C4_1, DONE=>F_C4_1);

U54: MULT
port map(A=>L24, B=>f614_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C4_2, DONE=>F_C4_2);

U55: MULT
port map(A=>L24, B=>f513_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C4_3, DONE=>F_C4_3);

U56: MULT
port map(A=>L17, B=>f412_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C4_4, DONE=>F_C4_4);

U57: MULT
port map(A=>L24, B=>f311_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C4_5, DONE=>F_C4_5);

U58: MULT
port map(A=>L17, B=>f210_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C4_6, DONE=>F_C4_6);
```

```
U59: MULT
port map(A=>L17, B=>f19_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C4_7, DONE=>F_C4_7);

U60: MULT
port map(A=>L24, B=>f08_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C4_8, DONE=>F_C4_8);

U61: ADD_8_16
port map(A1=>C4_1, A2=>C4_2, A3=>C4_3, A4=>C4_4, A5=>C4_5, A6=>C4_6,
A7=>C4_7, A8=>C4_8, EN=>EN_ADD1, SUM=>C4_LXSS);

U62: ADD_3_16
port map(A1=>C4, A2=>C4, A3=>C4_LXSS(15 downto 0), EN=>EN_ADD2,
SUM=>C4_LXS);

U63: SUB_2_16
port map(A=>C4_LXS(15 downto 0), B=>C4M, S_AB=>'1', EN=>EN_SUB,
SUB=>C4_LX(16 downto 0));
-----------------------------------------
U64: MULT
port map(A=>L4, B=>f311_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C5_1, DONE=>F_C5_1);

U65: MULT
port map(A=>L2, B=>f210_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C5_2, DONE=>F_C5_2);

U66: MULT
port map(A=>L15, B=>f19_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C5_3, DONE=>F_C5_3);

U67: MULT
port map(A=>L3, B=>f08_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C5_4, DONE=>F_C5_4);

U68: MULT
port map(A=>L8, B=>f715_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C5_5, DONE=>F_C5_5);

U69: MULT
port map(A=>L5, B=>f614_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C5_6, DONE=>F_C5_6);

U70: MULT
port map(A=>L16, B=>f513_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C5_7, DONE=>F_C5_7);

U71: MULT
port map(A=>L18, B=>f412_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C5_8, DONE=>F_C5_8);

U72: ADD_8_16
port map(A1=>C5_1, A2=>C5_2, A3=>C5_3, A4=>C5_4, A5=>C5_5, A6=>C5_6,
A7=>C5_7, A8=>C5_8, EN=>EN_ADD1, SUM=>C5_LXSS);

U73: SUB_2_16
```

```
port map(A=>C5_LXSS(15 downto 0), B=>C5M, S_AB=>'1', EN=>EN_SUB,
SUB=>C5_LX(16 downto 0));
-----------------------------------------
U74: MULT
port map(A=>L21, B=>f715_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C6_1, DONE=>F_C6_1);

U75: MULT
port map(A=>L23, B=>f614_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C6_2, DONE=>F_C6_2);

U76: MULT
port map(A=>L20, B=>f513_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C6_3, DONE=>F_C6_3);

U77: MULT
port map(A=>L22, B=>f412_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C6_4, DONE=>F_C6_4);

U78: MULT
port map(A=>L21, B=>f311_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C6_5, DONE=>F_C6_5);

U79: MULT
port map(A=>L23, B=>f210_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C6_6, DONE=>F_C6_6);

U80: MULT
port map(A=>L20, B=>f19_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C6_7, DONE=>F_C6_7);

U81: MULT
port map(A=>L22, B=>f08_D_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C6_8, DONE=>F_C6_8);

U82: ADD_8_16
port map(A1=>C6_1, A2=>C6_2, A3=>C6_3, A4=>C6_4, A5=>C6_5, A6=>C6_6,
A7=>C6_7, A8=>C6_8, EN=>EN_ADD1, SUM=>C6_LXSS);

U83: ADD_3_16
port map(A1=>C6, A2=>C6, A3=>C6M, EN=>EN_ADD2, SUM=>C6_LXS);

U84: SUB_2_16
port map(A=>C6_LXSS(15 downto 0), B=>C6_LXS(15 downto 0), S_AB=>'1',
EN=>EN_SUB, SUB=>C6_LX(16 downto 0));
-----------------------------------------
U85: MULT
port map(A=>L15, B=>f311_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C7_1, DONE=>F_C7_1);

U86: MULT
port map(A=>L4, B=>f210_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C7_2, DONE=>F_C7_2);

U87: MULT
port map(A=>L14, B=>f19_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C7_3, DONE=>F_C7_3);
```

```
U88: MULT
port map(A=>L2, B=>f08_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C7_4, DONE=>F_C7_4);

U89: MULT
port map(A=>L9, B=>f715_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C7_5, DONE=>F_C7_5);

U90: MULT
port map(A=>L3, B=>f614_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C7_6, DONE=>F_C7_6);

U91: MULT
port map(A=>L18, B=>f513_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C7_7, DONE=>F_C7_7);

U92: MULT
port map(A=>L5, B=>f412_S_BUF, ST=>MULT_ST, clk=>clk, rst=>MULT_rst,
PROD=>C7_8, DONE=>F_C7_8);

U93: ADD_8_16
port map(A1=>C7_1, A2=>C7_2, A3=>C7_3, A4=>C7_4, A5=>C7_5, A6=>C7_6,
A7=>C7_7, A8=>C7_8, EN=>EN_ADD1, SUM=>C7_LXSS);

U_94: SUB_2_16
port map(A=>C7_LXSS(15 downto 0), B=>C7M, S_AB=>'1', EN=>EN_SUB,
SUB=>C7_LX(16 downto 0));
-----------------------------------------
end DCT_RECTAN_R4_UPDATE_arch;

--The testbench for the four-point independent update circuit

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_signed.all;
use ieee.std_logic_unsigned.all;
use ieee.math_real.all;
use ieee.numeric_std.all;

entity DCT_RECTAN_R4_UPDATE_test is
end DCT_RECTAN_R4_UPDATE_test;

architecture DCT_RECTAN_R4_UPDATE_test_arch of
DCT_RECTAN_R4_UPDATE_test is

component DCT_RECTAN_R4_UPDATE
port(f0, f1, f2, f3, f4, f5, f6, f7: in std_logic_vector(8 downto
0):=(others=>'0');
clk, rst, conven, update: in std_logic;
C0, C1, C2, C3, C4, C5, C6, C7: buffer
std_logic_vector(15 downto 0):=(others=>'0');
DONE_DCTCONVEN: out std_logic:='0';
DONE_UPDATE:out std_logic:='0');
end component;
```

```
signal f0in: std_logic_vector(8 downto 0):=(others=>'0');
signal f1in: std_logic_vector(8 downto 0):=(others=>'0');
signal f2in: std_logic_vector(8 downto 0):=(others=>'0');
signal f3in: std_logic_vector(8 downto 0):=(others=>'0');
signal f4in: std_logic_vector(8 downto 0):=(others=>'0');
signal f5in: std_logic_vector(8 downto 0):=(others=>'0');
signal f6in: std_logic_vector(8 downto 0):=(others=>'0');
signal f7in: std_logic_vector(8 downto 0):=(others=>'0');

signal C0: std_logic_vector(15 downto 0):=(others=>'0');
signal C1: std_logic_vector(15 downto 0):=(others=>'0');
signal C2: std_logic_vector(15 downto 0):=(others=>'0');
signal C3: std_logic_vector(15 downto 0):=(others=>'0');
signal C4: std_logic_vector(15 downto 0):=(others=>'0');
signal C5: std_logic_vector(15 downto 0):=(others=>'0');
signal C6: std_logic_vector(15 downto 0):=(others=>'0');
signal C7: std_logic_vector(15 downto 0):=(others=>'0');

signal CLK: std_logic:='0';
signal RST: std_logic:='0';
signal CONVEN: std_logic:='0';
signal UPDATE: std_logic:='0';
signal DONE_DCTCONVEN: std_logic:='0';
signal DONE_UPDATE: std_logic:='0';

begin
CLK<=not(CLK) after 50 ns;
RST<='1', '0' after 300 ns;

process
begin
CONVEN<='0';
UPDATE<='0';
wait on RST until (RST'event and RST='0');

f0in<="101010010";--f0
f1in<="111010110";--f1
f2in<="011010011";--f2
f3in<="010101011";--f3
f4in<="101100011";--f4
f5in<="011101001";--f5
f6in<="001110101";--f6
f7in<="111100011";--f7

wait for 100 ns;
CONVEN<='1';
wait for 200 ns;
CONVEN<='0';

wait on DONE_DCTCONVEN until (DONE_DCTCONVEN='1');

f4in<="111111011";--f8
f5in<="110001100";--f9
f6in<="101111000";--f10
f7in<="111010110";--f11

wait on DONE_DCTCONVEN until (DONE_DCTCONVEN='1');
```

```
f4in<="111111011";--f12
f5in<="110100100";--f13
f6in<="010101100";--f14
f7in<="001010101";--f15

wait for 100 ns;
UPDATE<='1';
wait for 200 ns;
UPDATE<='0';

wait on DONE_UPDATE until (DONE_UPDATE='1');


f4in<="001101011";--f16
f5in<="101111010";--f17
f6in<="001010101";--f18
f7in<="110011010";--f19

wait for 100 ns;
UPDATE<='1';
wait for 200 ns;
UPDATE<='0';

end process;

U0: DCT_RECTAN_R4_UPDATE
port map(f0=>f0in, f1=>f1in, f2=>f2in, f3=>f3in, f4=>f4in, f5=>f5in,
f6=>f6in, f7=>f7in,
clk=>CLK, rst=>RST, conven=>CONVEN, update=>UPDATE,
C0=>C0, C1=>C1, C2=>C2, C3=>C3,
C4=>C4, C5=>C5, C6=>C6, C7=>C7,
DONE_DCTCONVEN=>DONE_DCTCONVEN,
DONE_UPDATE=>DONE_UPDATE);
end DCT_RECTAN_R4_UPDATE_test_arch;
```

APPENDIX C: JAVA CODE TO CONVERT DECIMAL NUMBER TO BINARY AND VICE VERSA

/* The following code converts the Decimal number to Binary number */

```
public class DecimalToBinary {
      public static void main(String args[]){

            Double inputVariable = -0.5;
            //Value of str:100000000

            int maxValue = 9;

            System.out.println("Value of inputVariable: " +
                                                inputVariable);

            String str = null;

            if(inputVariable>0)
            {
                  str = "0";
            }
            else
            {
                  str = "1";
                  inputVariable = 1.0 + inputVariable;
            }

            Double dbl = inputVariable;

            for (int i = 0; i < maxValue; i++)
            {
                  dbl = dbl * 2;

                  if(Double.parseDouble(dbl.toString()) ==1)
                  {
                        str = str + "1";
                        break;
                  }
                  else if(Double.parseDouble(dbl.toString()) > 1)
                  {
                        dbl = dbl -1;
                        str = str + "1";
                  }
                  else
                  {
                        str = str + "0";
                  }
            }


            int difference = maxValue-str.length();
```

```
            for (int i = 0; i < difference; i++) {
                  str = str + "0";
            }

            System.out.println("Value of str:" + str);
      }

}
```

/* The following code converts the Binary number to Decimal number */

```
public class BinaryToDecimal {
      public static void main(String[] args) {

            String inputVariable = "111001110";
            //Value of str:100000000

            System.out.println("Value of inputVariable: " +
                                                inputVariable);

            char[] character = inputVariable.toCharArray();
            double dbl = 0.0;

            boolean flag = false;

            for (int i = 0; i < character.length; i++) {
                  if(i==0)
                  {
                        if(character[0] == '0')
                        {
                              flag = false;
                        }
                        else
                        {
                              flag = true;
                        }

                        continue;
                  }
                  else
                  {
        dbl = dbl +
        Integer.parseInt(String.valueOf(character[i]))*Math.pow(2,-i);
                  }
            }

            if(flag)
            {
                  dbl = dbl - 1.0;        }

            System.out.println("Value of dbl: " + dbl);
      }

}
```