Fakultet elektrotehnike i računarstva
Poslijediplomski studij

Predmet: Multimedijski računalni sustavi

DISCRETE COSINE TRANSFORM ALGORITHMS FOR FPGA DEVICES

Domagoj Babić

Zagreb, 11. April 2003

Contents

List of Figures

List of Tables

Listings

1 Motivation

Discrete cosine transform (DCT) is widely used transform in image processing, especially for compression. Some of the applications of two-dimensional DCT involve still image compression and compression of individual video frames, while multidimensional DCT is mostly used for compression of video streams and volume spaces. Transform is also useful for transferring multidimensional data to DCT frequency domain, where different operations, like spread-spectrum data watermarking, can be performed in easier and more efficient manner. A countless number of papers discussing DCT algorithms is strongly witnessing about its importance and applicability.

Hardware implementations are especially interesting for the realization of highly parallel algorithms that can achieve much higher throughput than software solutions. In addition, a special purpose DCT hardware discharges the computational load from the processor and therefore improves the performance of complete multimedia system. The throughput is directly influencing the quality of experience of multimedia content. Another important factor that influences the quality of is the finite register length effect on the accuracy of the forward-inverse transformation process.

Hence, the motivation for investigating hardware specific DCT algorithms is clear. As 2-D DCT algorithms are the most typical for multimedia applications, the main focus of this thesis will be on the efficient hardware implementations of 2-D DCT. As the number of applications that require higher-dimensional DCT algorithms is growing, a special attention will be payed to the algorithms that are easily extensible to higher dimensional cases.

A class of transforms, called polynomial transforms, have been used heavily for the realization of efficient multidimensional algorithms in digital signal processing. Some of the examples of significant computational savings achieved by using the results from number theory and polynomial transforms include multidimensional discrete Fourier transforms, convolutions and also a discrete cosine transform. The application of polynomial transforms to DCT is not so straightforward as it is the case with discrete Fourier transform and convolutions. A suitable polynomial transform based multidimensional DCT algorithm has emerged very recently and it will be later introduced as MPTDCT algorithm. According to the best of author's knowledge neither hardware implementation has been made nor any accuracy measurements performed.

The goal of this thesis will be to research computational savings, accuracy improvements and chip area savings that result from the application of polynomial transforms to DCT.

2 Discrete Cosine Transform

2.1 Transforms

Mathematical transforms can be defined as operators that map functions from one functional space to another. It's important to introduce the notion of functional to understand how transforms can be constructed.

Functional is defined as an operation that associates a real number to every function from a selected class. Integration is an example of functional:

$$I(x) = \int_a^b x(t)dt, \tag{2.1}$$

where $x(t)$ is an integrable function defined on interval $[a, b]$. Transform can be created by multiplying any subintegral function of functional (integral in this case, but it can be also derivative) by a kernel containing a parameter that determines the result of functional. Effectively, we obtain transform from functional by using different kernels, which determine transform properties. Integral transforms are often used for the reduction of complexity of mathematical problems. The Fourier transform is certainly one of the best known of the integral transforms and its direct and inverse forms are given by:

$$\mathcal{F}\left[x\left(t\right)\right] = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft}dt \tag{2.2}$$

$$\mathcal{F}^{-1}\left[X\left(f\right)\right] = \int_{-\infty}^{\infty} X(F)e^{j2\pi ft}df, \tag{2.3}$$

where $x(t)$ is an absolutely integrable function on interval $(-\infty, \infty)$ and $2\pi f$ is angular frequency. Transform kernel is $e^{-j2\pi ft}$.

2.2 Fourier Transform

In the early 1800s French mathematician Joseph Fourier has introduced Fourier series for the representation of continuous-time periodic signals:

$$x(t) = \sum_{k=-\infty}^{\infty} c_k e^{j2\pi k f_0 t} \tag{2.4}$$

$$c_k = \frac{1}{T_p} \int_{T_p} x(t) e^{-j2\pi k f_0 t} dt, \tag{2.5}$$

where $T_p = 1/f_0$ is the period of signal $x(t)$. The signal can be decomposed to a linear weighted sum of harmonically related complex exponentials. This weighted sum represents the frequency content of signal called spectrum. When the signal becomes aperiodic, its period becomes infinite and its spectrum becomes continuous. This special case represents Fourier transform for continuous-time aperiodic signals, defined as shown in Eq. 2.2 on the preceding page. A detailed explanation and proof can be found in [29].

From continuous form one can obtain the form for discrete-time signals. Before proceeding to discrete Fourier transform, some properties of continuous Fourier transform need to be mentioned:

- Linearity
- Invertibility
- Symmetry
- Scaling
- Translation
- Convolution.

Only the first two will be explained in somewhat more detail because they will be occasionally referenced to later. More details about others can be found in the large body of literature. An especially good overview is given in [27].

Linearity property makes the Fourier transform suitable for the analysis of linear systems. It means that the Fourier transform of a linear combination of two or more signals is equal to the same linear combination of the Fourier transforms of individual signals. A detailed explanation of the term "linear combination" can be found in almost any linear algebra book. The property can be expressed as:

$$\mathcal{F}\left[\alpha f + \beta g\right] = \alpha\mathcal{F}\left[f\right] + \beta\mathcal{F}\left[g\right]. \tag{2.6}$$

Invertibility means that the Fourier transform and the inverse Fourier transforms are operational inverses, thus:

$$\begin{aligned} \psi = \mathcal{F}\left[\phi\right] &\Leftrightarrow \mathcal{F}^{-1}\left[\psi\right] = \phi \\ \mathcal{F}^{-1}\left[\mathcal{F}\left[f\right]\right] &= f. \end{aligned} \tag{2.7}$$

2.3 Discrete Fourier Transform

The Fourier series representation of a continuous-time periodic signal can contain a countably finite number of frequency components because the frequency range of continuous-time signals can extend between $-\infty$ to ∞. The frequency spacing between two adjacent components is $1/T_p$. Discrete-time signals have also infinite frequency range, but it is periodic, so one period is sufficient for the complete reconstruction of discrete signal. Thus, we can say that frequency range is in the interval $(-\pi, \pi)$ or $(0, 2\pi)$. If discrete signal is periodic with the fundamental period N, then its adjacent frequency components are separated by $2\pi/N$ radians. In conclusion, Fourier series of discrete-time signal can contain at most N unique frequency components.

If $x(n)$ is a periodic sequence with period N, Fourier series is defined as:

$$x(n) = \sum_{k=0}^{N-1} c_k e^{j2\pi kn/N} \tag{2.8}$$

where c_k are Fourier coefficients:

$$c_k = \frac{1}{N}\sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N}. \tag{2.9}$$

In the same way as Fourier transform for aperiodic continuous-time signals can be derived from Fourier series of continuous-time periodic signal, we can obtain discrete Fourier transform (DFT) of discrete-time aperiodic signal from discrete Fourier series. The relation between continuous and discrete Fourier transform is described in the literature, and especially detailed explanation is given in [16]. Direct and inverse DFT equations are shown in Eq. 2.10 and 2.11.

$$X(\omega) = \sum_{n=-\infty}^{\infty} x(n)\, e^{-j\omega n} \tag{2.10}$$

$$x(n) = \frac{1}{2\pi} \int_{0}^{2\pi} X(\omega)\, e^{j\omega n} d\omega \tag{2.11}$$

If the discrete signal in previous equation is periodical (or we assume periodicity), then we can limit DFT range to N points:

$$X(\omega) = \sum_{n=0}^{N-1} x(n) e^{-j\omega n}. \tag{2.12}$$

From the linearity property and the fact that DFT contains a finite number of frequency components N, it can be deduced that DFT can be represented as a linear operator. Therefore, the DFT operation can be realized as a matrix multiplication with vector. Matrix represents the transform kernel coefficients and the vector represents the samples of input signal. Further, from linearity and invertibility properties it follows that coefficient matrix must be regular, i.e. it must be invertible. This isomorphism has far reaching consequences and it also applies to transforms derived from DFT. It has spurred the development of a large number of algorithms that rely on the properties of coefficient matrix.

2.4 Multidimensional Transforms

Fourier series and transform can be easily extended to a multidimensional case. For example, two dimensional DFT is defined by the following equation:

$$X(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2)\, e^{-j2\pi\left(\frac{n_1 k_1}{N_1} + \frac{n_2 k_2}{N_2}\right)} \tag{2.13}$$

$$k_1 = 0, ..., N_1 - 1, \qquad k_2 = 0, ..., N_2 - 1$$

The transform kernel can be written as:

$$e^{-j2\pi\left(\frac{n_1 k_1}{N_1} + \frac{n_2 k_2}{N_2}\right)} = e^{-j2\pi \frac{n_1 k_1}{N_1}}\, e^{-j2\pi \frac{n_2 k_2}{N_2}}, \tag{2.14}$$

and if we use substitution:

$$W_1 = e^{-j2\pi/N_1}, \qquad W_2 = e^{-j2\pi/N_2}, \tag{2.15}$$

then we can rewrite Eq. 2.13 as:

$$X(k_1, k_2) = \sum_{n_1=0}^{N_1-1} W_1^{n_1 k_1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) W_2^{n_2 k_2} \tag{2.16}$$

Hence, the two-dimensional DFT can be computed by performing one-dimensional DFT on the result of another one-dimensional DFT. This important property is called separability and it also applies to other transforms derived from Fourier transform. It means that the 2D DFT of two-dimensional signal can be computed by computing a one-dimensional DFT of the rows of input signal matrix followed by the computation of one-dimensional DFT of the columns. This simple procedure for computing multidimensional separable transforms is called row-column decomposition. It follows that any separable multidimensional transform can be computed by a series of one-dimensional transforms. Later it will be shown that row-column decomposition is not an ideal way of computing multi-dimensional transforms, actually it's a rather naive approach. More complex algorithms for computing multidimensional transforms rely on the properties of the transform itself to compute the result directly without decomposition.

2.5 Discrete Cosine Transform

Although the invention of Fourier series was motivated by the problem of heat conduction, Fourier series and transform have found a vast number of applications and were a basis for development of other transforms, like discrete cosine transform (DCT).

2.5.1 Fourier cosine transform

The Fourier transform kernel is complex valued. Fourier cosine transform is obtained by using only a real part of complex kernel:

$$Re\left[e^{j\omega t}\right] = \cos(\omega t) = \frac{1}{2}\left[e^{j\omega t} + e^{-j\omega t}\right] \tag{2.17}$$

where ω is angular frequency. So, Fourier cosine transform of real or complex valued function $f(t)$, which is defined over $t \geq 0$, for $\omega \geq 0$, is written as:

$$\mathcal{F}_c[f(t)] = \int_0^\infty f(t) \cos \omega t dt. \tag{2.18}$$

The close relationship between Fourier transform and the cosine transform is apparent. Given the extended $f(t)$ function defined on the interval $(-\infty, \infty)$ so that f is an even function:

$$f_c(t) = f(|t|), \quad t \in \mathbb{R}. \tag{2.19}$$

Its Fourier transform is shown in Eq. 2.20. As f is an even function, integral 2.20 can be written as 2.21. The relation between transforms follows in Eq. 2.22.

$$\mathcal{F}[f_c(t)] = \int_{-\infty}^{\infty} f_c(t) e^{-j\omega t} dt, \quad t \in \mathcal{R} \tag{2.20}$$

$$\mathcal{F}[f_c(t)] = \left[\int_0^\infty f_c(t) e^{j\omega t} dt + \int_0^\infty f_c(t) e^{-j\omega t} dt \right] \tag{2.21}$$

$$\mathcal{F}[f_c(t)] = 2\mathcal{F}_c[f(t)] \tag{2.22}$$

Eq. 2.22 describes the relation between continuous Fourier and cosine transforms. For discrete case, DCT can be obtained from DFT of the mirrored original N-point sequence (effectively a 2N-point sequence). DCT is simply the first N points of the resulting 2N-point DFT. This relation between discrete cosine and discrete Fourier transform was used for computing DCT before efficient DCT algorithms have been developed.

Fourier cosine transform inherits many properties from Fourier transform, although many of them are less elegant and more complex. Linearity, invertibility and separability properties are directly inherited, others, like convolution, are much more complex. Despite this inelegancy, cosine transform, especially discrete cosine transform, has found many applications. DCT is most well known after its usage in multimedia systems for lossy compression. An in-depth survey of other applications can be found in [38].

2.5.2 Basis vectors

Transform kernel of Fourier cosine transform $\mathcal{K}_c$ of Eq. 2.18, evidently an even function, is denoted as:

$$\mathcal{K}_c(\omega, t) = \cos(\omega t). \tag{2.23}$$

The kernel of discrete cosine transform can be obtained by sampling angular frequency and time. If δf and δt represent the unit sample intervals for frequency and time, then the sampled angular frequency and time (ω and t) can be written as $\omega_m = 2\pi m \delta f$ and $t_n = n\delta t$ yielding:

$$\begin{aligned} \mathcal{K}_c(\omega, t) = \mathcal{K}_c(2\pi m\delta f, n\delta t) &= \cos(2\pi mn\delta f\delta t) = \mathcal{K}_c(m, n) \\ \mathcal{K}_c(m, n) &= \cos\left(\frac{\pi mn}{N}\right), \end{aligned} \tag{2.24}$$

where m, n and $N = \frac{1}{2\delta f\delta t}$ are integers. As explained before, linear discrete transforms can be represented by a kernel coefficient matrix. The coefficient matrix of simple one-dimensional $(N+1)$-point cosine transform, named symmetric cosine transform (SCT), shown below:

$$X(m) = \sum_{n=0}^{N} x(n) \cos\left(\frac{\pi mn}{N}\right) \qquad m, n = 0, 1, ..., N \tag{2.25}$$

is given by:

$$[\mathcal{M}]_{mn} = \cos\left(\frac{\pi mn}{N}\right) \qquad m, n = 0, 1, ..., N. \tag{2.26}$$

The vectors in $\mathcal{M}$ coefficient matrix are called basis vectors. Basis vectors of SCT are orthogonal, but not normalized, and coefficient matrix is symmetric[1].

The basic vectors of one-dimensional 8-point DCT-II[2] are shown in Fig. 2.1. Simply put, the forward transform computes the dot product of every single basis vector and the input data with the purpose of extracting its frequency information.

Two-dimensional transforms have basis matrices instead of vectors. The basis matrices of DCT-II are shown in Fig. 2.2 on the following page. It should be noticed that the frequency of variation increases from top to bottom and from left to right.

[1] Symmetric matrices have the property that the transpose of the matrix is equal to the matrix itself.

[2] DCT-II is a type of DCT, see equation 2.27 on page 16.

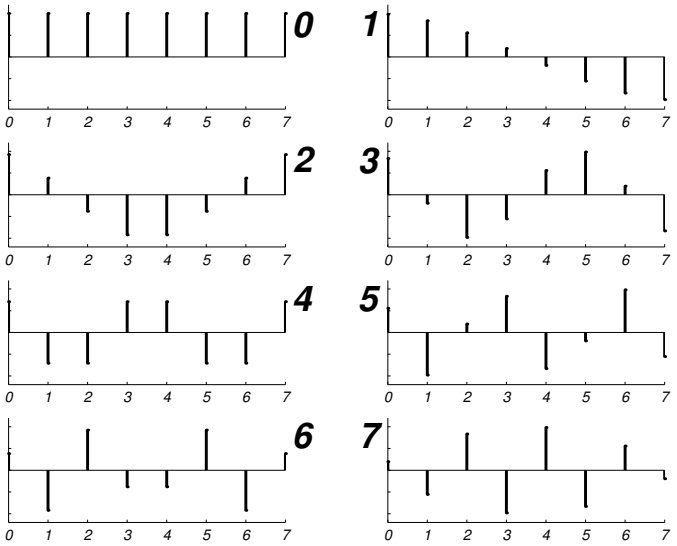


Figure 2.1: The basis vectors for 8-point DCT

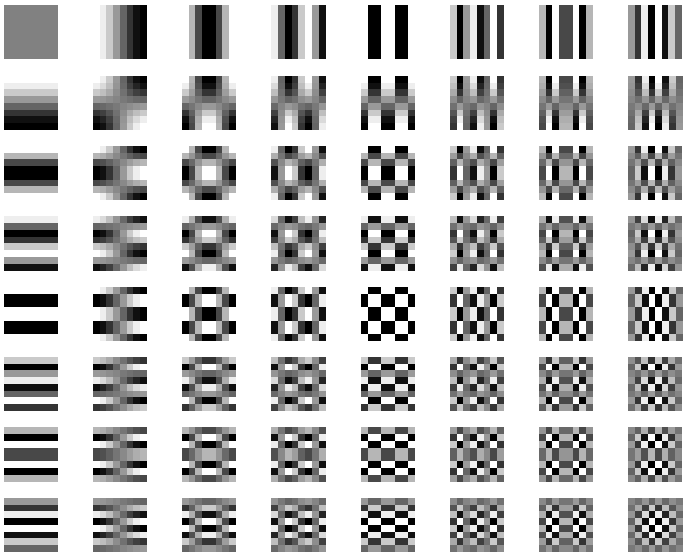


Figure 2.2: The basis matrices for 8 x 8 DCT

2.5.3 Karhunen-Loéve transform

Prior to introducing different types of DCT and orthonormalization of kernel coefficient matrix, the foundations for the application of DCT have to be explained.

The most important application of DCT stems from its similarity to Karhunen-Loéve transform (KLT), first discussed by K. Karhunen 1947. KLT is optimal transform for energy decorrelation of the signals of the stochastic Markov-1 process types. In order to give a simple informal explanation, the KLT of simple sinusoid can be considered. A signal, more accurately a sinusoid, is transmitted sequentially sending sampled values. More samples means that it is possible to reconstruct the waveform more precisely. Anyway, it is not needed to send all samples, it would be enough to send information about magnitude, phase, frequency, starting time and the fact that it is a sinusoidal waveform. Thus, only five pieces of information are needed to reconstruct the given waveform at the receiver. Because the sampled values of sinusoid are highly correlated, the information content is low. Therefore, if we would be able to decorrelate the input signal, we would ideally get exactly the minimum amount of information needed to reconstruct the transmitted signal. In the given example, it would be only five parameters. KLT is performing ideal decorrelation of input data when the transmitted signal is of Markov-1 type.

There's another way to look at this process [3]. A two-dimensional input data 8 x 8 matrix can be seen as a set of eight vectors in the eight dimensional space. Let this matrix represent an image block, and every individual vector a single pixel row in that block. Usually, pixels, as well as rows (or columns), are highly correlated. Therefore, the set of those eight vectors represents a small, more or less homogeneous, cluster in the eight-dimensional space. It is obvious that there is some redundancy and the most important question is whether these vectors can be represented in less dimensional space.

By rotating this cluster and aligning it along some of the coordinate axes, the cluster can be represented only with the information about chosen axis and the distance of individual points. KLT is ideal in the sense that it always finds the flattest possible direction so that the information can be coded in the smallest amount of data. In other words, KLT achieves optimum energy concentration. Basis vectors of KLT are obtained as the eigenvectors of the corresponding auto-covariance matrix of a data vector.

Although KLT is ideal transform, it is not very practical. The main drawbacks are:

- KLT is data-dependent
- KLT is not separable for image blocks
- transform matrix cannot be factored into sparse matrices.

Hence, other simpler and more practical transforms that would have similar effect had to be found. DCT is very close to ideal KLT for 1st order stationary

Markov sequence when correlation parameter ρ is close to 1 and therefore it can be applied to the same purpose of signal decorrelation. DCT performs the best for highly correlated data, with $\rho \geq 0.5$. When the correlation parameter is $-0.5 \leq \rho \leq 0.5$, discrete sine transform is a better choice.

2.5.4 Discrete cosine transform types

According to previous discussion, two-dimensional discrete cosine transform can be seen as a rotation operator in multidimensional space, where the number of dimensions depends on the size of transform. Rotation operator must be an orthonormalized matrix which has a property that a transposed matrix is equal to its inverse (antisymmetrical). Thus, the inverse operator is simply a transposed version of it. The rotation axis is an eigenvector of the operator matrix. If the rotation is performed around axis for an angle π then the rotation operator is also symmetrical. As SCT basis vectors are not normalized, it cannot be a rotation operator. A simple way to orthonormalize SCT basis vectors is to multiply individual coefficients with correction factors. It can be easily shown that the first type DCT (according to classification in [38]), called DCT-I and representing SCT with correction factors, has orthonormalized basis vectors.

Altogether, there are four types of DCT, denoted by $\mathcal{M}_{size}^{type}$:
DCT-I:

$$\left[\mathcal{M}_{N+1}^{I}\right]_{mn} = \left(\frac{2}{N}\right)^{1/2} \left[k_m k_n \cos\left(\frac{mn\pi}{N}\right)\right] \qquad m,n = 0,1,...N$$

DCT-II:

$$\begin{aligned} \left[\mathcal{M}_{N}^{II}\right]_{mn} &= \left(\frac{2}{N}\right)^{1/2} \left[k_m \cos\left(\frac{m\left(n+\frac{1}{2}\right)\pi}{N}\right)\right] \\ m,n &= 0,1,...N-1 \end{aligned} \tag{2.27}$$

DCT-III:

$$\left[\mathcal{M}_{N}^{III}\right]_{mn} = \left(\frac{2}{N}\right)^{1/2} \left[k_n \cos\left(\frac{\left(m+\frac{1}{2}\right)n\pi}{N}\right)\right] \qquad m,n = 0,1,...N-1$$

DCT-IV:

$$\left[\mathcal{M}_{N}^{IV}\right]_{mn} = \left(\frac{2}{N}\right)^{1/2} \cos\left[\frac{\left(m+\frac{1}{2}\right)\left(n+\frac{1}{2}\right)\pi}{N}\right] \qquad m,n = 0,1,...N-1$$

$$k_j = \begin{cases} 1 & if \quad j \neq 0 \text{ or } N \\ \frac{1}{\sqrt{2}} & if \quad j = 0 \text{ or } N \end{cases}$$

The correction factors are chosen so as to normalize coefficient matrix $\mathcal{M}$, which is already orthogonal. This new orthonormalized matrix represents the rotation operator in multidimensional space. Orthonormalized DCT matrix dotted with its transpose gives an identity matrix. Accordingly, another important role of these correction coefficients is that the energy level of signal is maintained after forward and inverse transforms are performed.

Further on, the focus will be on DCT-II, because this type is the most frequent in different applications.

3 Polynomial Transform

The elementary basics of number theory need to be explained before introducing polynomial transform (PT). Despite the abundance of number theory and polynomial algebra literature, it is hard to find books about application of that theory to digital signal processing. Two notable exceptions are [25, 17]. Therefore, this introduction section relies heavily on those sources.

The section begins with fast sweep over some basic terms from number theory and than proceeds to Chinese remainder theorem (CRT) which is one of the most important theorems for application of number theory to digital signal processing. After integer CRT is explained, a short introduction to polynomial CRT will be given.

Polynomial transform is explained in somewhat more detail, as it is the core of efficient multidimensional digital signal processing algorithms that will be mentioned in the final subsection about applications.

3.1 Chinese Remainder Theorem

3.1.1 Greatest common divisor

Two integers a and b, $a \geq b$ can be written as:

$$a = bq + r, \quad 0 \leq r < b \tag{3.1}$$

where q is quotient and r is remainder. If $r = 0$, it is said that b and q are factors or divisors of a, in other words, b and q divide a. This relation is usually marked with symbol $b|a$. In the case that the only factor of a is 1, a is a prime number. A notion of prime number is the basis for understanding CRT.

Further, we can define the greatest common divisor (GCD) as the largest positive integer that divides two integers a and b, and we denote it with braces (...) :

$$d = (a, b) \tag{3.2}$$

If $(a, b) = 1$, a and b are relatively prime, i.e. their greatest common divisor is 1.

A simple algorithm for computing GCD is called Euclid algorithm and it is based on modulo operation. Modulo operation $a \; mod \; b$ produces as a

result the remainder of the division of a by b. Two integers c and d are said to be congruent modulo b if:

$$c \equiv d \mod b. \tag{3.3}$$

Actually, integers c and d have the same residues when divided by b. A simpler way to represent it is by using $\langle\ \rangle$ symbol:

$$\langle c \rangle_b = \langle d \rangle_b \tag{3.4}$$

```
int gcd(int u, int v) {
    int t;
    while (u != 0) {
        t = u mod v;
        u = v;
        v = t;
    }
    return u;
}
```

Listing 3.1: Euclid algorithm C code

C code implementing Euclid algorithm is given in Lst. 3.1. From the code and Eq. 3.1 on the previous page, it can be seen that GCD of two numbers, a and b, can be represented as a linear combination, where m and n are integers:

$$(a, b) = ma + nb. \tag{3.5}$$

This fact is used for the analysis of solvability and finding solutions of Diophantine equations. It can be shown that Diophantine equation with integer coefficients a, b and c :

$$ax + by = c \tag{3.6}$$

can be solved if and only if $(a, b)|c$.

Operations, like addition and multiplication, can be performed directly on residues, while division is not defined:

$$\begin{aligned} \langle c + d \rangle &= \langle \langle c \rangle + \langle d \rangle \rangle \\ \langle cd \rangle &= \langle \langle c \rangle \langle d \rangle \rangle. \end{aligned} \tag{3.7}$$

Such modulo equations are called congruence relations. An example of linear congruence equation is Diophantine equation in which all terms are defined modulo b:

$$ax \equiv c \mod b. \tag{3.8}$$

Eq. 3.8 can be easily seen as an ordinary Diophantine equation and solved in the same way, as shown in 3.9.

$$\begin{aligned} ax - q_1 b &= c - q_2 b \\ ax + b(q_2 - q_1) &= c \\ ax + by &= c \end{aligned} \tag{3.9}$$

In the special case, when $(a, b) = 1$, Eq. 3.8 has a unique solution that can be obtained more elegantly by using Euler's theorem. As solving this special case will be a part of CRT, it is important to give a short introduction to Euler's function and theorem.

3.1.2 Euler's function

Congruence can be understood as an equivalence of residues of two expressions modulo some integer, as described before. It follows that modulo operation actually maps integers into equivalence classes. The number of classes of *mod* M operation is exactly M, where M is an integer. Simply, if M divides an arbitrary integer a, the result of modulo operation is 0. The largest possible result is $M - 1$ and it is obvious that the range of solutions is $\{0, ..., M - 1\}$ - a set of M members.

Another important property of modulo operation, beside partitioning into equivalence classes, is permutation. Having a set $\mathcal{S} = \{0, ..., M - 1\}$, let n_i represent i-th member and a an integer relatively prime with M. By multiplying n_i with a modulo M, we obtain M distinct b_i integer results having values from set $\mathcal{S}$, but in a different order.

$$b_i \equiv a n_i \mod M \tag{3.10}$$

A simple proof by contradiction follows. In the case that modulo operation would map n_j, n_k multiplied with a to $b_j = b_k$, we would have:

$$b_j - b_k \equiv a(n_j - n_k) \mod M \tag{3.11}$$

$$b_j - b_k = 0 \tag{3.12}$$

$$a(n_j - n_k) \equiv 0 \mod M. \tag{3.13}$$

And because of $(a, M) = 1$, a must be relatively prime with M and $n_j - n_k$ is by definition less then M, so Eq. 3.13 is not possible. Therefore our initial assumption about $b_j = b_k$ is wrong and we have proved that modulo operation performs permutation, distinctively mapping n_i into b_i.

An integer M partitions integers into M equivalence classes. Euler function is defined as a number of members of those equivalence classes that are relatively prime with M and denoted with $\phi(M)$. A number of examples is given in the Table 3.1.

n	$\phi(n)$	n	$\phi(n)$	n	$\phi(n)$	n	$\phi(n)$
1	1	6	2	11	10	16	8
2	1	7	6	12	4	17	16
3	2	8	4	13	12	18	6
4	2	9	6	14	6	19	18
5	4	10	4	15	8	20	8

Table 3.1: Euler function for $n \leq 20$

Prime numbers are relatively prime with all smaller numbers, because they don't have any common divisor except 1, therefore for prime p:

$$\phi(p) = p - 1. \tag{3.14}$$

It can be shown that the linear congruence Eq. 3.8 on the previous page, which will be an essential part of solving CRT, can be easily solved when $(a, b) = 1$ and that its solution is unique. The solution is given by:

$$x \equiv ca^{\phi(b)-1} \mod b. \tag{3.15}$$

By substituting with 3.14 when b is prime we obtain:

$$x \equiv ca^{b-2} \mod b. \tag{3.16}$$

3.1.3 Chinese remainder theorem

Previous discussion explained the basic terms needed to understand Chinese remainder theorem. Suppose that we have k positive integers $m_i > 1$ that are relatively prime in pairs, then the set of linear congruence equations:

$$x \equiv r_i \mod m_i \tag{3.17}$$

has a unique solution modulo M, where $M = \prod_{i=1}^{k} m_i$.

The problem boils down to reconstructing an integer having only its residues modulo m_i and it is solved by introducing T_i:

$$\left(\frac{M}{m_i}\right) T_i \equiv 1 \mod m_i, \tag{3.18}$$

that is used in the reconstruction of x as shown:

$$x \equiv \sum_{i=1}^{k} \left(\frac{M}{m_i}\right) r_i T_i \mod M. \tag{3.19}$$

The connection between Eq. 3.17 and 3.18, follows from the fact that m_i are relatively prime and therefore m_i and M/m_i are also relatively prime because

$$\frac{M}{m_i} = \prod_{j \neq i}^{k} m_j \tag{3.20}$$

doesn't contain any factors that would have GCD > 1 with m_i. When we reduce[1] Eq. 3.19 with m_u, all factors in the sum, except M/m_u, become equal to zero because they all contain m_u in the product. Thus, Eq. 3.19 reduces to:

$$x \equiv \left(\frac{M}{m_u}\right) r_u T_u \mod m_u, \tag{3.21}$$

and since T_i is introduced as shown in Eq. 3.18, previous equation equals to:

$$x \equiv r_i \mod m_i, \tag{3.22}$$

so we have obtained the equations needed to reconstruct the integer knowing only its residues modulo relatively prime integers m_i.

Effectively, the problem of reconstruction has been decomposed to solving a set of simple congruence relations (Eq. 3.19) that can be easily solved by using either Euclid's or more elegant Euler's algorithm, because $\left(\frac{M}{m_i}, m_i\right) = 1$.

Equipped with the understanding of simple CRT, we can proceed to polynomial CRT. But first, some introduction to polynomial algebra should be given.

[1]The application of modulo operation is usually called reduction.

3.1.4 Polynomial CRT

It is said that a polynomial $P(z)$ divides a polynomial $H(z)$ if there exists a polynomial $D(z)$ such that:

$$H(z) = P(z)D(z). \tag{3.23}$$

If $P(z)$ is not a divisor of $H(z)$, it produces a residue polynomial $R(z)$:

$$H(z) = P(z)D(z) + R(z). \tag{3.24}$$

Modulo operation is defined pretty much the same as with integers:

$$R(z) \equiv H(z) \mod P(z). \tag{3.25}$$

$P(z)$ also maps polynomials into equivalence classes. Further, if we can decompose $P(z)$ into factors it is said to be reducible, otherwise it is irreducible. Polynomials that are irreducible in the field of rational numbers (i.e. it is not possible to find any factors that have rational polynomial coefficients) are called cyclotomic polynomials. Depending on the variety of $P(z)$ polynomial, mathematical structures that consist of a set of polynomials with defined operations of addition and multiplication modulo $P(z)$ are called a ring if $P(z)$ is reducible and a field if $P(z)$ is irreducible.

The polynomial equivalent of CRT is quite similar to its integer version. If $P_i(z)$ are relatively prime polynomials (i.e. they have no common factors) and we define $P(z)$ by:

$$P(z) = \prod_{i=1}^{k} P_i(z), \tag{3.26}$$

then CRT is expressed with:

$$H(z) \equiv \sum_{i=1}^{k} S_i(z)H_i(z) \mod P(z). \tag{3.27}$$

This can be seen as a problem of reconstruction of $H(z)$ knowing its residues H_i by polynomials $P_i(z)$. $S_u(z)$ is defined as:

$$S_u(z) = T_u(z) \prod_{j \neq u}^{k} P_j(z) \tag{3.28}$$

To find a solution, we have introduced $T_u(z)$ such that:

$$T_u(z) \prod_{j \neq u}^{k} P_j(z) \equiv 1 \mod P_u(z). \tag{3.29}$$

Thus, the problem of reconstruction of the polynomial $H(z)$ can be solved by solving a set of equations 3.29. The proof is quite similar to the proof of integer CRT and it relies on the fact that:

$$S_u(z) \equiv \begin{cases} 0 \mod P_i(z), & i \neq u \\ 1 \mod P_u(z) \end{cases} \tag{3.30}$$

which follows from the definition of $P(z)$.

Solving for $T_u(z)$ can be a tedious job. A symbolic mathematical package, like *Wolfram's Mathematica*[2] can be of great help. For example, let us denote $\prod_{j \neq u}^{k} P_j(z)$ in Eq. 3.29 with $M_u(z)$. Then we can simply solve for $T_u(z)$ by using instructions in Lst. 3.2.

```
Solve[
    PolynomialRemainder[Tu(z)*Mu(z),Pu(z),z]==1,Tu(z)
]
```

Listing 3.2: Solving a system of polynomial congruence relations

Now, when CRT is outlined, we can reason about its applications. In the 3.3 section, an example of its usage will be given. The crux of idea is to partition the problem into smaller ones, perform necessary operations, and then reconstruct the final solution by CRT. Other outlined applications, namely fast multidimensional DFT and DCT algorithms, are based on polynomial reduction. Basically idea is the same - to break the problem into smaller pieces, but the final solution reconstruction is different as it will be seen. The middle stages of all applications that will be explained are based on polynomial transforms.

[2] **http://www.wolfram.com/**

3.2 Polynomial Transforms

3.2.1 Basic definition

After surveying modulo arithmetic and CRT, we can broach the subject of polynomial transforms (PT). Polynomial transform was introduced by Nussbaumer [22, 23] to map multidimensional convolutions into one-dimensional. Higher dimensions are obtained by polynomial products and additions needed to implement polynomial transform and Chinese remainder reconstruction, effectively decreasing computational complexity. Polynomial product can be performed by scalar coefficient multiplication, but there's a more efficient method based on combination of direct and inverse polynomial transform.

PT can also be applied to reduce multidimensional DFTs to one-dimensional and a number of additions for implementing polynomial transform. Many digital signal processing operations, like DCT and correlation, are closely related to DFT and convolution and therefore PT can be also applied to reduce their computational complexity. Even more, PT can be applied to single-dimensional convolution and DFT, as will be described later. By using polynomial transforms we also avoid matrix transpositions that are needed in naive implementations of separable transforms. Another important property of PT based DFT and convolution algorithms is that dynamic range limitation and round-off errors are avoided. PT based algorithms are also suitable for hardware implementation because they can be easily parallelized.

In the most general case [10], DFT can also be considered as a very simple polynomial transform corresponding to projection of an input data sequence onto the family of monomials and evaluated at the roots of unity $\mathcal{K}_{DFT} = e^{j2\pi kn/N}$, $k = 0, ...N-1$. But further on, we will use only a class of polynomial transforms of the form:

$$Y_k(z) \equiv \sum_{m=0}^{N-1} X_m(z) \left[G(z)\right]^{mk} \mod P(z), \quad k = 0, ..., N-1, \tag{3.31}$$

where $G(z)$ is root of transform kernel and z is just an auxiliary variable. All members of the class have to satisfy:

$$\left[G(z)\right]^N \equiv 1 \mod P(z) \tag{3.32}$$

$$S(q) \equiv \sum_{k=0}^{N-1} \left[G(z)\right]^{qk} \mod P(z) \equiv \begin{cases} 0 & for \quad q \not\equiv 0 \mod N \\ N & for \quad q \equiv 0 \mod N \end{cases} \tag{3.33}$$

Additional condition is that N and $G(z)$ must have inverses $mod\ P(z)$. Inverse PT is defined as:

$$X_m(z) \equiv \frac{1}{N} \sum_{k=0}^{N-1} Y_k(z) \left[G(z)\right]^{-mk} \quad mod\ \ P(z). \tag{3.34}$$

As long as the three mentioned conditions are satisfied, we can also use composite roots of the type $WG(z)$, $W \in \mathbb{C}$.

3.2.2 Computation

In general, algorithms based on polynomial transform require computation of polynomial reductions, polynomial transforms and Chinese remainder reconstruction. In many cases the first two operations can be completely computed without multiplications. Reconstruction is a bit more complex and in some cases it can be computed without multiplications with the reorganization of computation.

A few examples will be given to demonstrate polynomial reduction, which is simply a modulo operation. The type of operations needed to compute the reduction depends on a polynomial that defines a polynomial ring. Examples of the most often used cases are given in Table 3.2, where p stands for an odd prime and $W \in \mathbb{C}$.

Type	Ring	Computed by
$R_I(z)$	$z-1$	additions
$R_{II}(z)$	$z-W$	complex multiplications and additions
$R_{III}(z)$	z^p-1	additions
$R_{IV}(z)$	$(z^p-1)/(z-1)$	additions

Table 3.2: Polynomial reduction

For a polynomial:

$$X(z) = \sum_{k=0}^{N-1} x_k z^k, \tag{3.35}$$

it is easy to see that:

$$X(z) \quad mod\ R_I(z) = \sum_{k=0}^{N-1} x_k. \tag{3.36}$$

Auxiliary variable z is substituted with 1, and $N-1$ additions are needed to compute the result. The same applies to the second type, with the difference that z is substituted with complex variable W resulting in $N-1$ complex multiplications by powers of W and $N-1$ additions. Third type can be factored into the first and the fourth so as to decompose a problem into two smaller ones. Basically, if $p \geq N$ the reduction has no effect. If $p < N$, it maps coefficients of degree $d > N$ into equivalence classes of degree $d \ mod \ N$, and thus it can be computed without multiplications.

For the third type, we will assume that $N = p$. It follows that:

$$X(z) \quad mod \ R_{III}(z) = \sum_{k=0}^{p-2} (x_k - x_{p-1})\, z^k. \tag{3.37}$$

While we were talking about Euler's function in section 3.1.2, it was explained how a set of integers can be permuted using modulo operation. The same principle applies to polynomial transforms. Generalizing a third type from the Table 3.2 on the preceding page, a polynomial transform, with $N \in \mathbb{R}$:

$$X_m(z) = \sum_{k=0}^{N-1} x_{k,m} z^k \tag{3.38}$$

$$Y_k(z) = \sum_{m=0}^{N-1} X_m(z) z^{mk} \quad mod \ z^N - 1 \tag{3.39}$$

can be viewed as a permutation of $X_m(z)$ polynomial. When $mk = 1$, permutation is a one-word polynomial rotation followed by a sign inversion of the overflow words. One-word rotation means that if polynomial is of degree k, than a coefficient of z^k would be rotated to z^0 position.

Computational complexity of polynomial transformation is a bit harder to explain, but essentially it is performed by a series of reductions. An example of computational complexity analysis for polynomial transforms can be found in [25].

The computation of Chinese remainder reconstruction will be shortly explained in the discussion about applications of polynomial transform in the next section.

3.3 Application of PTs

3.3.1 Convolution

Convolution has many applications in digital signal processing. One of especially important applications is in design of finite impulse response filters. It is a well known fact that a form of convolution, named circular convolution, can be computed by multiplication of two DFT sequences:

$$y_k = DFT^{-1}\left(DFT(x_m)DFT(h_n)\right). \tag{3.40}$$

Circular convolution is the same as an ordinary convolution with the index defined modulo N, as shown:

$$y_m = \sum_{n=0}^{N-1} x_n h_{\langle m-n\rangle_N}. \tag{3.41}$$

Ordinary linear convolution can be obtained from circular one by padding input sequences with a sufficient number of zeros [29].

Brute force computation of convolution of length N requires N^2 multiplications. From Eq. 3.40 it is clear that circular convolution can be computed using DFT which can be computed by Fast Fourier transform (FFT) algorithm. Multiplicative complexity of FFT is $Nlog_2N$. Hence, the benefit of using FFT for the computation of convolution is obvious.

Another way to compute one-dimensional real convolution is by using reductions and Chinese remainder reconstruction iteratively [23]. The problem is decomposed into two smaller ones, more precisely into polynomial products and smaller convolution. Polynomial products can be computed efficiently using polynomial transform. The multiplicative complexity of such an approach is also $Nlog_2N$ making a direct comparison difficult, especially because there is a large number of different FFT algorithms. Taking roundoff error and auxiliary operations into consideration makes this comparison even harder.

But in the case of multidimensional convolutions, polynomial transform based approach has a clear advantage in many cases over the usage of naive row-column FFT algorithms and even over Winograd nesting Fourier transform algorithm (WFT). In the case when coefficients $H(k,l) = DFT2(h_{n,m})$ are precomputed, NxN WFT can be implemented via M^2 complex multiplications, where M represents a number of multiplications needed to implement

one-dimensional DFT of length N. Each complex multiplication can be implemented either with 4 real multiplications and 2 real additions, or with 3 multiplications and 5 additions[3]:

$$\begin{aligned}(a+ib)(c+id) &= (k_0-k_1)+i(k_2-k_1-k_0)\\ k_0 &= ac\\ k_1 &= bd\\ k_2 &= (a+b)(c+d)\end{aligned}$$

Hence, $3\left(Nlog_2N\right)^2$ real multiplications are needed for the realization of WFT. FFT based approach requires $2N^2log_2N^2+2N^2$ [11] real multiplications. Polynomial transform based approach for computing NxN convolution, where N is prime requires only $2N^2-N-2$ real multiplications, and it can be proved that it is the theoretical minimum. Comparison is given in the Table 3.3.

Algorithm	Real multiplications
PT	$2N^2-N-2$
WFT	$3\left(Nlog_2N\right)^2$
FFT	$2N^2log_2N^2+2N^2$

Table 3.3: 2D convolution multiplicative complexity

The multiplicative complexity analysis provides a strong motivation for using polynomial transform based algorithms. The constraint that N has to be prime was used only to simplify the following example, but it is possible also to design PT algorithms when the length is not a prime number. PT algorithms can be easily applied to multidimensional convolutions.

Two-dimensional circular convolution can be written as a polynomial:

$$Y_l \equiv \sum_{m=0}^{N-1}\sum_{n=0}^{N-1}\sum_{u=0}^{N-1} h_{n,m}x_{u-n,l-m}z^u \quad mod\ (z^N-1). \tag{3.42}$$

It should be noted that modulo operation is superfluous in the above equation, but it will make further presentation easier to follow[4]. To simplify Eq. 3.42, let us write:

[3]Although it might not be worth the effort on the machines with fast multiplication, it might save significant area on chip, depending on the implementation of multiplication circuit.

[4]As the highest degree of a polynomial Y_l is less than N, this modulo operation does not have any effect.

$$H_m(z) = \sum_{n=0}^{N-1} h_{n,m} z^n, \quad m = 0, ..., N-1 \tag{3.43}$$

$$X_r(z) = \sum_{s=0}^{N-1} x_{s,r} z^s, \quad r = 0, ..., N-1 \tag{3.44}$$

$$Y_l(z) \equiv \sum_{m=0}^{N-1} H_m(z) X_{l-m}(z) \quad mod\ (z^N - 1). \tag{3.45}$$

The proof simply follows from the periodicity of the sequence defined modulo $z^N - 1$. Now, we shall constrain N to be an odd prime $N = p$, so that $z^p - 1$ can be factored into two cyclotomic polynomials.

$$z^p - 1 = (z-1)P(z) \tag{3.46}$$

$$P(z) = z^{p-1} + z^{p-2} + ... + 1 \tag{3.47}$$

$Y_{2,l}$ is computed by reduction by $z - 1$ and p point convolution.

$$Y_{1,l}(z) = Y_l(z) \quad mod\ P(z) \tag{3.48}$$

$$Y_{2,l}(z) = Y_l(z) \quad mod\ (z-1) \tag{3.49}$$

$$Y_{2,l}(z) = \sum_{m=0}^{p-1} H_{2,m} X_{2,l-m} \quad l = 0, ..., p-1 \tag{3.50}$$

Computing $Y_{1,l}$ is slightly more complex and it is based on the properties of polynomial transform:

$$Y_{1,l} \equiv \sum_{m=0}^{p-1} \sum_{r=0}^{p-1} H_{1,m}(z) X_{1,r}(z) \frac{1}{p} \sum_{k=0}^{p-1} z^{qk} \quad mod\ P(z), \tag{3.51}$$

where $q = m + r - l$ and $X_{1,r}(z) \equiv X_r(z)\ \ mod\ P(z)$. Since all the exponents in the last sum are defined modulo p, and q is relatively prime with p, it is easy to see that if $q \not\equiv 0$ the last sum is a polynomial with the exponents $\{0, 1, ..., p-1\}$. This polynomial reduces to zero modulo $P(z)$. In the case $q \equiv 0$ the last sum computes to p. If $q \equiv 0$ then $r \equiv l - m$ and therefore Eq. 3.51 can be written as a circular convolution:

$$Y_{1,l} \equiv \sum_{m=0}^{p-1} H_{1,m}(z) X_{1,l-m}(z)\ \ mod\ P(z). \tag{3.52}$$

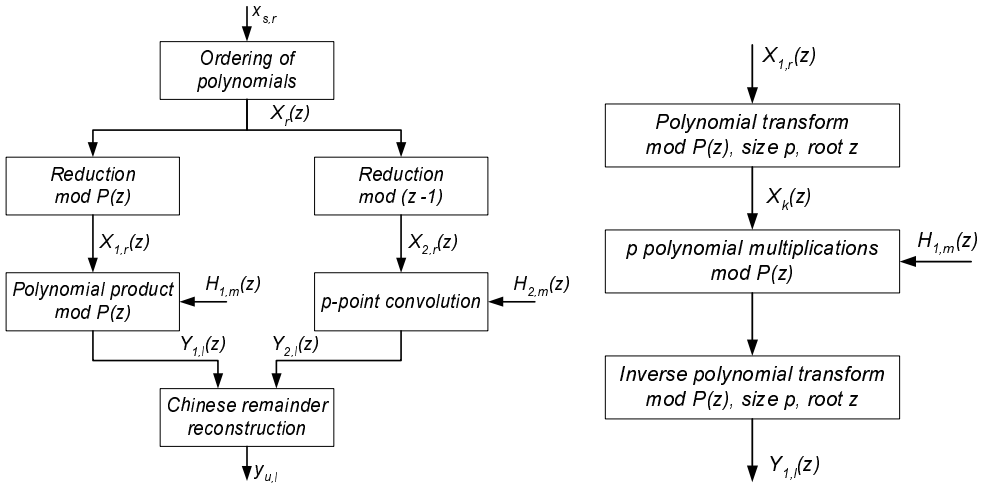


Figure 3.1: Block diagram of PT based 2-D convolution

A block diagram of main computational steps is shown in Fig. 3.1. Fig. 3.1(a) represents the basic flow of operations, while Fig. 3.1(b) shows the realization of polynomial products.

Polynomial product in Eq. 3.51 is actually an inverse polynomial transform of the product of polynomial transforms of $X_{1,r}(z)$ and $H_{1,m}(z)$ modulo $P(z)$:

$$Y_{1,l} = \frac{1}{p}\sum_{k=0}^{p-1}\left(\sum_{r=0}^{p-1} X_{1,r}(z)z^{rk}\sum_{m=0}^{p-1} H_{1,m}(z)z^{mk}\right)z^{-lk}. \tag{3.53}$$

The final result can be obtained by Chinese remainder reconstruction. Solving a set of polynomial congruence relations is relatively simple when the ring is defined modulo $z^p - 1$, because S_i polynomials can be simply computed as shown below.

$$\begin{aligned} Y_l(z) &\equiv S_1(z)Y_{1,l}(z) + S_2(z)Y_{2,l} \quad mod\ (z^p - 1) \\ S_1(z) &= \frac{p - P(z)}{p} \\ S_2(z) &= \frac{P(z)}{p} \end{aligned} \tag{3.54}$$

In order to decrease the number of multiplications even further, a part of

Chinese remainder reconstruction can be precomputed together with $H_{1,m}(z)$, but this is not shown in the block diagrams.

3.3.2 DFT

DFT and DCT are closely related. Some of the early algorithms for multidimensional DCT were based on multidimensional DFT algorithms. Thus, an example of application of polynomial transforms to the computation of multidimensional DFT will make the presentation more complete and easier to follow.

Two important properties of polynomial reduction have to be mentioned before proceeding with PT based two-dimensional DFT algorithm:
I. If $Q(z) = z - a$, then:

$$P(z) \ mod\, Q(z) = P(a). \tag{3.55}$$

II. When $Q(z)$ can be factored, then we can write:

$$\begin{aligned} Q(z) &= Q_1(z)Q_2(z) && (3.56)\\ P(z)\ mod\, Q_1(z) &= (P(z)\ mod\, Q(z))\ mod\, Q_1(z). && (3.57) \end{aligned}$$

Two-dimensional DFT of size NxN is defined as:

$$\begin{aligned} \hat{X}_{k_1,k_2} &= \sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x_{n_1,n_2} W^{n_1k_1} W^{n_2k_2} && (3.58)\\ k_1, k_2 &= 0, ..., N-1, && (3.59) \end{aligned}$$

where $W = e^{-j2\pi/N}$. Using the properties of polynomial reduction, previous equation can be rewritten as:

$$\hat{X}_{k_1,k_2} = \sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x_{n_1,n_2} W^{n_1k_1} z^{n_2} \ \ mod\ \left(z - W^{k_2}\right), \tag{3.60}$$

or more formally:

$$\bar{X}_{k_1}(z) \equiv \sum_{n_1=0}^{N-1} X_{n_1}(z) W^{n_1k_1} \ \ mod\ (z^N - 1) \tag{3.61}$$

$$X_{n_1}(z) = \sum_{n_2=0}^{N-1} x_{n_1,n_2} z^{n_2} \tag{3.62}$$

$$\hat{X}_{k_1,k_2} \equiv \bar{X}_{k_1}(z) \ mod \ \left(z - W^{k_2}\right). \tag{3.63}$$

It should be noted that modulo operation in Eq. 3.61 is not required, but it makes further exposition clearer. Now, let us constraint N to be an odd prime number p so that $z^p - 1$ can be factored into two cyclotomic polynomials as shown in 3.46 and 3.47 on page 30. As a polynomial of degree N has N complex roots, the chosen polynomial $z^p - 1$ has p roots and each W^{k_2} is a root of the chosen polynomial. For $k_2 = 0$, we get DFT of length N by $X_{n_1} \ mod \ (z-1)$:

$$\hat{X}_{k_1,0} = \sum_{n_1=0}^{p-1} \left(\sum_{n_2=0}^{p-1} x_{n_1,n_2} \right) W^{n_1 k_1}. \tag{3.64}$$

For $k_2 \neq 0$, we can write 3.58 as:

$$\bar{X}^1_{k_1}(z) \equiv \sum_{n_1=0}^{p-1} X^1_{n_1}(z) W^{n_1 k_1} \ mod \ P(z), \ \ k_1 = 0, ..., p-1 \tag{3.65}$$

$$X^1_{n_1}(z) = \sum_{n_2=0}^{p-2} (x_{n_1,n_2} - x_{n_1,p-1}) z^{n_2} \equiv X_{n_1}(z) \ mod \ P(z) \tag{3.66}$$

$$\hat{X}_{k_1,k_2} \equiv \bar{X}^1_{k_1}(z) \ mod \ (z - W^{k_2}), \ \ k_2 = 1, ..., p-1. \tag{3.67}$$

If $k_2 \neq 0$, W^{k_2} are roots of $P(z)$ and DFT can be obtained by a series of polynomial reductions and one-dimensional DFTs.

$$\hat{X}_{k_1,k_2} \equiv \left\{ \left[\bar{X}_{k_1}(z) \ mod \ (z^p - 1) \right] \ mod \ P(z) \right\} \ mod \ (z - W^{k_2}) \tag{3.68}$$

As k_2 is relatively prime with p, it can be introduced in the exponent in order to eliminate W factor in Eq. 3.65. The result is simply a permuted sequence.

$$\bar{X}^1_{k_1 k_2}(z) \equiv \sum_{n_1=0}^{p-1} X^1_{n_1}(z) W^{k_1 k_2 n_1} \ mod \ P(z) \tag{3.69}$$

$$\hat{X}_{k_1 k_2, k_2} \equiv \bar{X}^1_{k_1 k_2}(z) \ mod \ \left(z - W^{k_2}\right) \tag{3.70}$$

By substituting z by W^{k_2} we obtain a polynomial transform of length p which can be computed without multiplications.

$$\bar{X}^1_{k_1 k_2} \equiv \sum_{n_1=0}^{p-1} X^1_{n_1}(z) z^{k_1 n_1} \ \ mod\ P(z) \tag{3.71}$$

Since the previous equation is defined modulo $P(z)$, the highest exponent is $p-2$ and the equations can be written as:

$$\bar{X}^1_{k_1 k_2} = \sum_{l=0}^{p-2} y_{k_1,l} z^l \tag{3.72}$$

$$\hat{X}_{k_1 k_2, k_2} = \sum_{l=0}^{p-2} y_{k_1,l} W^{k_2 l}, \quad k_2 = 1, ...p-1. \tag{3.73}$$

The last equation represents p DFTs of length p. Hence, pxp DFT has been reduced to $(p+1)$ p-length DFTs, while row-column decomposition would require $2p$ DFTs. Further, p DFTs of length p can be converted into a convolution using chirp z-transform. Even more efficient way is to convert DFTs to circular convolution using Rader's algorithm [6]. Circular convolution can be computed using polynomial transforms yielding a very efficient way for computing multidimensional DFTs. A block diagram for computing pxp can be seen in Fig. 3.2.

The left part of block diagram 3.2 can be implemented as follows in diagram 3.3. As $p-1$ is always an even number, one of the factors of $z^{p-1}-1$ is surely z^2-1.

Finally, we should compare polynomial transform approach with other more traditional approaches like WFT and FFT. Computational complexity of polynomial transform based NxN DFT is given in the Table 3.4 on the next page. Numbers p, p_1 and p_2 are prime and $t \in \mathbb{N}$. Comparison of the number of multiplications and additions, denoted with μ and α respectively, can be viewed in Table 3.5 on page 36. Both tables are adapted from [26].

Obviously, polynomial transform based approach for computing multi-dimensional DFTs results in significant computational savings. It has been shown in paper [8] that this approach saves 46% of logic resources for 512x512 DFT implemented in Xilinx[5] 4025 FPGA [36] over conventional row-column decomposition approach, while maintaining the same throughput.

[5] **http://www.xilinx.com/**

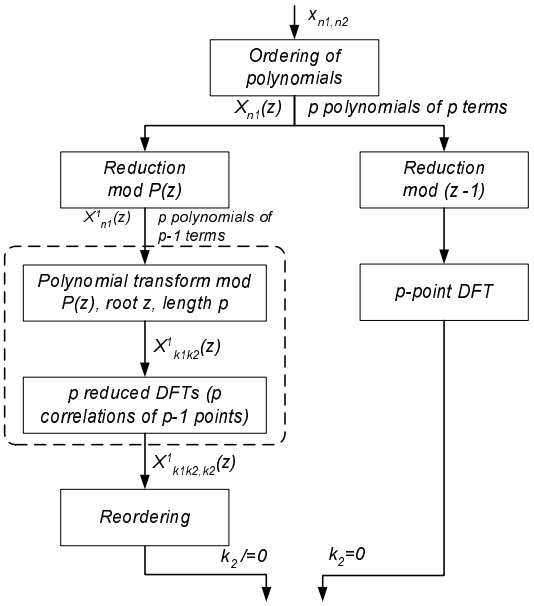


Figure 3.2: Block diagram of PT based 2-D DFT

Size	Multiplications	Number of additions
p	1 DFT of p terms, p correlations of $p-1$ terms	p^3+p^2-5p+4
p^2	p^2+p correlations of p^2-p terms, 1 DFT of p terms, p^2+2p correlations of dimension $p-1$	$2p^5+p^4-5p^3+p^2+6$
2^t	$3(2^{t-1})$ reduced DFTs of dimension 2^t, 1 DFT of dimension $2^t/2$x$2^t/2$	$(3t+5)2^{2(t-1)}$
p_1p_2	$(p_1p_2+p_1+p_2)$ correlations of (p_1-1)x(p_2-1) terms, p_1 correlations of p_1-1 terms, p_2 correlations of p_2-1 terms, 1 DFT of p_1p_2 terms	$p_1^2p_2^2(p_1+p_2+2)-$ $-5p_1p_2(p_1+p_2)+$ $+4(p_1^2+p_2^2)$

Table 3.4: Computational complexity of PT based DFT

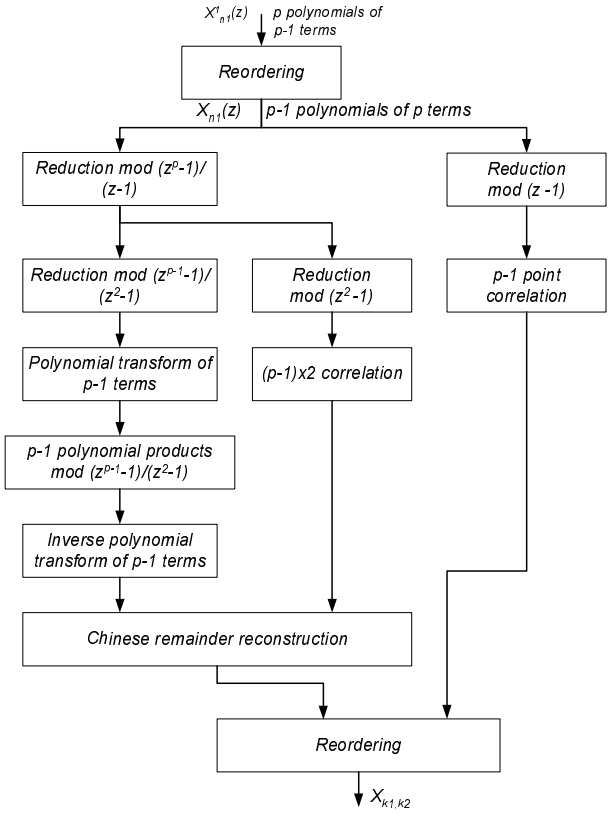


Figure 3.3: Realization of DFT via circular convolution

Size	PT DFT		WFT		FFT	
	μ	α	μ	α	μ	α
2x2	4(4)	8	4(4)	8	8(8)	8
3x3	9(1)	43	9(1)	36	18(6)	36
4x4	16(16)	64	16(16)	64	32(32)	64
5x5	31(1)	221	36(1)	187	60(10)	170
7x7	65(1)	635	81(1)	576	126(14)	504
8x8	64(40)	408	64(36)	416	128(96)	416
9x9	105(1)	785	121(1)	880	198(18)	792
16x16	304(88)	2264	324(64)	2516	576(256)	2368

Table 3.5: DFT computational complexity comparision

4 Previous Work

4.1 Computational Complexity

Feig and Winograd were the first to report the theoretical lower bound on the multiplicative complexity of multidimensional DCT transform [14]. The lower bound for one-dimensional DCT was established a few years earlier by Duhamel and H'Mida [13].

For one-dimensional DCT of the size $N = 2^n$, the minimum number of nonrational multiplications needed for computation is determined to be equal to:

$$\mu\left(K_{2^n}\right) = 2^{n+1} - n - 2. \tag{4.1}$$

The lower multiplicative bound for two-dimensional MxN DCT, where $M = 2^m$ and $N = 2^n$ with $m \leq n$, is given in Eq. 4.2.

$$\mu\left(K_M \bigotimes K_N\right) = 2^m(2^{n+1} - n - 2) \tag{4.2}$$

Symbol $\bigotimes$ denotes a tensor product and $K_M \bigotimes K_N$ represents 2D DCT coefficient matrix. More generally, for N-dimensional DCT, where $M_j = 2^{m_j}$ and $m_1 \leq m_2 \leq m_3 \leq \cdots \leq m_N$, we obtain:

$$\mu\left(K_{M_1} \bigotimes K_{M_2} \bigotimes \cdots \bigotimes K_{M_N}\right) = 2^{(m_1+\cdots+m_{N-1})}\left(2^{m_N+1} - m_N - 2\right). \tag{4.3}$$

Complexity results are important for evaluation and design of practical algorithms. The minimization of the number of multiplications, without introducing an excessive number of additions, is crucial for the hardware implementations. For software implementations, one has to consider the percentage of the total application run-time that is needed for the execution of the target algorithm before searching for a more advanced one. Even if the percentage of time is significant, it is questionable whether an algorithm with smaller number of multiplications (and usually higher number or additions) will shorten execution time at all. Other effects, like memory and cache behavior, total number of operations, code scheduling and processor efficiency in execution of different types of operations might be more important. The number of multiplications for software implementation has become even more irrelevant since the ratio $\frac{multiplication\ time}{addition\ time}$ is close to 1 on most processors. As

the topic of this thesis are DCT algorithms for hardware implementation, multiplicative complexity is an important issue that needs to be considered.

4.2 One-dimensional Algorithms

DCT is probably most widely used transform in image processing, especially for compression. Many algorithms have been proposed in the literature and it is virtually impossible to give a detailed survey. Probably the most complete DCT survey is given in [38].

Discrete cosine transform was invented 1974. [2]. At first, it was computed using FFT algorithm that was newly rediscovered at the time. For DCT of length N, $2N$ FFT had to be used, like described before. One of the early papers on DCT algorithms [21] reported an improved way to compute DCT via FFT of length N using $(3/4)(Nlog_2N - N + 2)$ real multiplications (if each complex multiplication is computed by 3 real multiplications). Many FFT algorithms were optimized for complex data, while in most practical purposes the transformation is performed on real data. The research in DCT and FFT algorithms for real data resulted in an efficient recursive algorithm for computing DCT of real sequence [32].

Other research direction was based on the analysis of coefficient matrix properties. For instance, in [4] a new algorithm based on alternation of cosine and sine butterfly matrices with binary ones was reported. By reordering the matrix elements, they obtained a form that preserves a recognizable bit-reversed patterns.

A general prime factor decomposition algorithm that is very suitable for hardware implementation was proposed in [37]. The algorithm decomposed DCT of length $N = N_1N_2$, where N_1 and N_2 are relatively prime, into smaller DCTs of length N_1 and N_2 respectively.

Important work on DCT matrix factorization has been done by Feig and Winograd in [15]. The result was an efficient recursive DCT algorithm, that is used for one-dimensional DCT implementation in this thesis. It can be realized by 13 multiplications and 29 additions for 1-D DCT of length 8. Although not optimal, this algorithm is particularly effective for FPGA implementation when combined with distributed arithmetic algorithms. In the same paper they also present an algorithm for 8x8 DCT which uses 94 multiplications and 454 additions. Theoretical minimum is 88 multiplications.

Algorithm proposed by Loeffler in [19] has been devised using graph transformations and equivalence relations. For 8-point DCT, the algorithm is

optimal and requires only 11 multiplications and 29 additions, achieving theoretical minimum number of multiplications, while for larger input widths it yields the best effort algorithm.

Computation of inner products is essential in many DCT algorithms. When one of the vectors is fixed, inner products can be efficiently computed by reformulating the algorithm at the bit level and it is exactly an approach used in distributed arithmetic reformulation. Inner product can be efficiently implemented by look-up tables and accumulators. This approach is particularly suitable for FPGA architectures if serial input is acceptable. A detailed overview of distributed arithmetic and its applications can be found in [33]. Previously mentioned DCT matrix factorization approach can be combined with distributed arithmetic [39] resulting in a very regular algorithm. This algorithm with partial DCT matrix factorization is used later for the implementation of the reference 8x8 DCT and it will be called DADCT algorithm. An implementation of DADCT in FPGA will be used for comparison with polynomial transform based approach.

Another very interesting and innovative algorithm is based on the representation of DCT by a finite series of Chebyshev polynomials [20]. The order of the series can be halved by using successive factorization. This method computes DCT with $(1/2)Nlog_2N$ real multiplications and $(3/2)Nlog_2N$ additions. For 8-point DCT that means 12 multiplications and 36 additions.

Other approaches to the computation of DCT include CORDIC (COordinated Rotation DIgital Computer) algorithms [18] and systolic arrays. DCT CORDIC based algorithms can be implemented using only addition and shift operations as in [41]. Comparison between these different classes of algorithms is hard, and author is not aware of any detailed published work on the subject.

4.3 Early Multidimensional Algorithms

Some of the early approaches for computing multidimensional DCT transforms used its separable nature to compute first dimension, transpose the result and compute DCT of the transposed matrix. If we have NxN DCT and one-dimensional DCT of length N is computed with M multiplications, then $2NM$ multiplications are needed for this naive approach. Examples of such approach are described in [31, 30]. Each dimension is computed by partially factorizing DCT matrix, while inner products are realized with distributed arithmetic. After first dimension is computed, data matrix is

transposed and fed into second dimension.

Nussbaumer proposed in [24] one of the first 2-D DCT algorithms that treated both dimensions at once (i.e. without computing each dimension separately). The algorithm was based on the permutation of input sequence as follows:

$$y_{n_1,n_2} = \begin{cases} x_{2n_1,2n_2} & 0 \le n_1 < \frac{N_1}{2}, \quad 0 \le n_2 < \frac{N_2}{2} \\ x_{2N_1-2n_1-1,2n_2} & \frac{N_1}{2} \le n_1 < N_1, \quad 0 \le n_2 < \frac{N_2}{2} \\ x_{2n_1,2N_2-2n_2-1} & 0 \le n_1 < \frac{N_1}{2}, \quad \frac{N_2}{2} \le n_2 < N_2 \\ x_{2N_1-2n_1-1,2N_2-2n_2-1} & \frac{N_1}{2} \le n_1 < N_1, \quad \frac{N_2}{2} \le n_2 < N_2. \end{cases} \tag{4.4}$$

This permutation is further used to derive N_1xN_2 DCT without correction factors:

$$X_{k_1,k_2} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n_1,n_2} cos\left[\frac{2\pi(2n_1+1)k_1}{4N_1}\right] cos\left[\frac{2\pi(2n_2+1)k_2}{4N_2}\right] \tag{4.5}$$

from N_1xN_2 DFT:

$$\bar{Y}(k_1,k_2) = W_1^{k_1} W_2^{k_2} \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} y_{n_1,n_2} W_1^{4n_1k_1} W_2^{4n_2k_2}, \tag{4.6}$$

by simple additions:

$$\begin{aligned} X_{0,0} &= 4\bar{Y}(0,0) \\ X_{0,k_2} &= 2\left(\bar{Y}(0,k_2) + j\bar{Y}(0,N_2-k_2)\right) \\ X_{k_1,0} &= 2\left(\bar{Y}(k_1,0) + j\bar{Y}(N_1-k_1,0)\right) \end{aligned} \tag{4.7}$$

$$X_{k_1,k_2} = \bar{Y}(k_1,k_2) + j\bar{Y}(N_1-k_1,k_2) + j\bar{Y}(k_1,N_2-k_2) - \bar{Y}(N_1-k_1,N_2-k_2).$$

The result is obtained by computing DFT, which can be computed by using polynomial transforms, and multiplying the result with $W_1^{k_1} W_2^{k_2}$. Even smaller number of operations can be achieved by applying polynomial transform modulo $z^{N_1} - j$, where j is $\sqrt{-1}$, directly to 4.6. The algorithm can be calculated by $2N_1N_2(2 + log_2N_1)$ multiplications and $N_1N_2(8+3log_2N_1+2log_2N_2)$ additions if a simple radix-2 FFT is used. Nussbaumer claimed 50% reduction of the number of multiplications, but other algorithm used for comparison was a very inefficient one. In brief, the first proposed polynomial transform DCT algorithm was rather inefficient, especially when compared

with naive row-column decomposition implementation based on Loeffler's [19] algorithm. For instance, 8x8 DCT can be computed with 176 multiplications by using 16 8-point DCTs based on Loeffler's algorithm. Proposed polynomial transform based DCT requires 640 multiplications.

Although the advantage of application of polynomial transforms to multidimensional convolution and DFT was clear, more effort was needed to achieve really efficient multidimensional DCT algorithm.

4.4 Advanced Multidimensional Algorithms

Many researchers have been working on 2-D DCT algorithms that are used mostly for image compression. Higher-dimensional algorithms were rarely considered. New applications that require 3-D and even 4-D DCT algorithms are emerging lately. Three dimensional image and video systems, like 3-D TV [7], use DCT for lossy compression of integral 3-D image data. In [35] 3-D DCT is used for watermarking volume data by applying spread-spectrum technique in frequency DCT domain in order to hide watermark trace in multiple frequencies.

A very efficient 2-D DCT algorithm was proposed by Cho and Lee in [5]. Their algorithm requires only NM multiplications for NxN DCT, where M is the number of multiplications required for 1-D N-point DCT. The algorithm decomposes multidimensional DCT to one-dimensional DCTs and a number of additions. They reported multiplicative complexity of $(N^2/2)log_2N$, but they used non-optimal 1-D algorithm requiring $(N/2)log_2N$ multiplications. If optimal one-dimensional DCT is used for computing the first dimension in their algorithm, resulting 2-D algorithm will be also optimal [34]. Hence, for 8x8 DCT it is possible to achieve the theoretical minimum of 88 multiplications by using Loeffler's algorithm for the first dimension and compute the second dimension by Cho's algorithm. It is not clear whether their approach could be extended to multidimensional DCT.

4.4.1 Duhamel's 2D algorithm

Duhamel and Guillemot [12] have used a bit different approach to map 2-D DCT to complex polynomials. The approach is quite complex, especially for forward transform. Inverse DCT (IDCT) algorithm is somewhat easier to prove.

In order to demonstrate the basic steps of Duhamel's IDCT algorithm, let us now constraint that $N_1 = N_2 = N$. Input data permutation 4.4 can be rewritten as follows.

$$\begin{aligned} y_{n_1,n_2} &= x_{2n_1,2n_2} \\ y_{N-n_1-1,n_2} &= x_{2n_1+1,2n_2} \\ y_{n_1,N-n_2-1} &= x_{2n_1,2n_2+1} \\ y_{N-n_1-1,N-n_2-1} &= x_{2n_1+1,2n_2+1} \\ n_1, n_2 &= 0, ..., \frac{N}{2} - 1 \end{aligned} \tag{4.8}$$

And from 2-D DCT equation 4.5 on page 40 we get:

$$\hat{X}_{k_1,k_2} = (X_{k_1,k_2} - X_{N-k_1,N-k_2}) + j\,(X_{N-k_1,k_2} + X_{k_1,N-k_2}) \tag{4.9}$$

$$= \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} y_{n_1,n_2} W_{4N}^{-(4n_1+1)k_1} W_{4N}^{-(4n_2+1)k_2}, \tag{4.10}$$

where W_{4N} is defined as:

$$W_{4N} = e^{\frac{-j2\pi}{4N}}. \tag{4.11}$$

An equivalent to 2-D IDCT plus a few superfluous additive terms can be obtained from previous equation. The proof is not trivial.

$$\begin{aligned} z_{n_1,n_2} &= \sum_{k_1=0}^{N-1} \sum_{k_2=0}^{N-1} \hat{X}_{k_1,k_2} W_{4N}^{(4n_1+1)k_1} W_{4N}^{(4n_2+1)k_2} \\ &= 4y_{n_1,n_2} - 2\sum_{k_1=0}^{N-1} \hat{X}_{k_1,0} cos\left(\frac{2\pi}{4N}(4n_1+1)\,k_1\right) - \\ &\quad -2\sum_{k_2=0}^{N-1} \hat{X}_{0,k_2} cos\left(\frac{2\pi}{4N}(4n_2+1)\,k_2\right) + \hat{X}_{0,0} \end{aligned} \tag{4.12}$$

A simple substitution is used in order to remove additive terms in previous equation.

$$\begin{aligned} \hat{Y}_{k_1,0} &= 2\hat{X}_{k_1,0}, \quad k_1 = 0, 1, ..., N-1 \\ \hat{Y}_{k_1,k_2} &= \hat{X}_{k_1,k_2}, \quad k_1, k_2 = 1, 2, ..., N-1 \\ \hat{Y}_{0,k_2} &= 2\hat{X}_{0,k_2}, \quad k_2 = 0, 1, ..., N-1 \end{aligned} \tag{4.13}$$

Finally, 2-D IDCT can be written as:

$$z_{n_1,n_2} = 4y_{n_1,n_2} = \sum_{k_1=0}^{N-1}\sum_{k_2=0}^{N-1} \hat{Y}_{k_1,k_2} W_{4N}^{(4n_1+1)k_1} W_{4N}^{(4n_2+1)k_2}. \tag{4.14}$$

By using polynomial reduction property (Eq. 3.47 on page 30) above equation can be expressed modulo complex polynomial.

$$\hat{Y}_{k_1}(z) = \sum_{k_2=0}^{N-1} \hat{Y}_{k_1,k_2} z^{k_2} \tag{4.15}$$

$$z_{n_1,n_2} = \sum_{k_1=0}^{N-1} \hat{Y}_{k_1}(z) W_{4N}^{(4n_1+1)k_1} \ \ mod\ \left(z - W_{4N}^{4n_2+1}\right) \tag{4.16}$$

Now we can apply the same properties of polynomial reduction to rewrite previous equations. As $\left(z - W_{4N}^{4n_2+1}\right)$ is a factor of $(z^N + j)$ it follows:

$$\hat{Y}_{k_1}(z) = \sum_{k_2=0}^{N-1} \hat{Y}_{k_1,k_2} z^{k_2} \tag{4.17}$$

$$\breve{Y}_{n_1}(z) = \sum_{k_1=0}^{N-1} \hat{Y}_{k_1}(z) W_{4N}^{(4n_1+1)k_1} \ \ mod\ (z^N + j) \tag{4.18}$$

$$z_{n_1,n_2} = \breve{Y}_{n_1}(z) \ \ mod\ \left(z - W_{4N}^{4n_2+1}\right) \tag{4.19}$$

In section 3.3.2 it was shown how to eliminate W factor from equation by permuting the sequence (Eq. 3.69 on page 33). The same approach can be used here. As $(4n_2 + 1)$ is relatively prime with $4N$, exponent in Eq. 4.18 can be replaced with:

$$4n_1' + 1 \equiv (4n_1 + 1)(4n_2 + 1) \ \ mod\ 4N, \tag{4.20}$$

to obtain:

$$\breve{Y}_{n_1'}(z) = \sum_{k_1=0}^{N-1} \hat{Y}_{k_1}(z) W_{4N}^{(4n_1+1)k_1(4n_2+1)} \ \ mod\ (z^N + j). \tag{4.21}$$

The root of unity $W_{4N}^{(4n_1'+1)k_1}$ includes the same factor as that involved in modulo operation in Eq. 4.19. Therefore, a complete computation of 2-D IDCT can be expressed as follows.

$$\hat{Y}_{k_1}(z) = \sum_{k_2=0}^{N-1} \hat{Y}_{k_1,k_2} z^{k_2} \tag{4.22}$$

$$\breve{Y}_{n_1}(z) = \sum_{k_1=0}^{N-1} \hat{Y}_{k_1}(z) z^{(4n_1+1)k_1} \mod (z^N + j) \tag{4.23}$$

$$z_{n_1',n_2} = \breve{Y}_{n_1}(z) \mod \left(z - W_{4N}^{4n_2+1}\right) \tag{4.24}$$

First equation does not involve any practical operation, second one is a polynomial transform that can be computed without multiplications. The reduction in the last Eq. 4.24 is actually a computation of N IDCTs of length N.

This algorithm requires NM multiplications, where M is a number of multiplications for the realization of 1-D DCT. Hence, this algorithm is also optimal for 2-D DCT if an optimal 1-D DCT is used.

In [28], Prado and Duhamel have refined previous work by discovering symmetries in different stages of the algorithm. Although they have reduced the number of additions by approximately 50%, this has made already complex algorithm even more complex and irregular. If Loeffler's algorithm is used for computing the first dimension of 8x8 DCT, this algorithm requires the same number of multiplications (88) and additions (466) as Cho's algorithm mentioned before. Hence, both algorithms are optimal for the given dimensions of the transform, but cannot be simply extended to higher dimensions.

Correction factors are neglected in both algorithms and therefore their DCT matrix is not orthonormalized.

Prado and Duhamel have found symmetries at all stages of their algorithm. Input and output symmetries are relatively easy to prove, but proving symmetries for inner stages of the algorithm is very tedious. Even more, their notation is extremely complex.

4.4.2 Multidimensional PT algorithm

In [40] a group of authors have described a new algorithm based on polynomial transforms for computing multidimensional DCT transform. It has significant advantages in front of all other mentioned algorithms because it can be used for M-dimensional algorithms and symmetries in higher dimensions are easily and elegantly expressible. We will call this algorithm MPT-DCT. It is also optimal if the first dimension is computed via optimal 1-D DCT algorithm. As polynomial transform can be computed by only using

additions, higher-dimensions are implemented with a butterfly-like addition structure, similar to FFT butterfly. MPTDCT has simpler addition butterfly than previously discussed Duhamel's algorithm. Additional advantage of MPTDCT is that it does not require permutation of output data, since values are already in the correct order. At the moment, author is not aware of any hardware implementation of MPTDCT, so it will be interesting to compare hardware implementations of MPTDCT and Duhamel's algorithm. Also, an analysis of finite register length effects of hardware implementation will be made. In addition, advantage of MPTDCT over classical row-column approach will be scrutinized. To understand and appreciate this algorithm, it is important to give its detailed description, which is taken from [40] in order to provide the reader with the knowledge needed for understanding the implementation of the algorithm.

It is a well-known fact that polynomial transforms defined modulo z^N+1 can be implemented with a reduced number of additions by using a radix-2 FFT-type algorithm. Symmetry of different stages of algorithm is used in the derivation of FFT algorithm and that symmetry is easily expressible. Hence, it was a good idea to find a similar way to map DCT into real polynomials in such a manner that symmetries can be elegantly written and that addition butterfly is as regular as possible. It is exactly what MPTDCT does.

We will consider two dimensional NxN case of the algorithm, although it can be used for M-dimensional case M_1xM_2x...xM_n where each M_i is a power of 2. This will somewhat simplify the presentation.

First, let us use a simple permutation in Eq. 4.8 on page 42 of input values to obtain $4m$ in the nominator of cosine function parameter.

$$\begin{aligned} \hat{X}_{k,l} &= \sum_{n=0}^{N-1}\sum_{m=0}^{N-1} y_{n,m} \cos\left(\frac{\pi\,(4n+1)\,k}{2N}\right) \cos\left(\frac{\pi\,(4m+1)\,l}{2N}\right) \qquad (4.25)\\ k,l &= 0,1,...,N-1 \end{aligned}$$

Now, we will split previous equation in two parts defined as follows:

$$X(k,l) = \frac{1}{2}\left[A(k,l)+B(k,l)\right] \qquad (4.26)$$

$$A_{k,l} = \sum_{p=0}^{N-1}\sum_{m=0}^{N-1} y_{p(m),m} \cos\left[\frac{\pi\,(4p(m)+1)\,k}{2N} + \frac{\pi\,(4m+1)\,l}{2N}\right] \qquad (4.27)$$

$$B_{k,l} = \sum_{p=0}^{N-1}\sum_{m=0}^{N-1} y_{p(m),m} \cos\left[\frac{\pi\,(4p(m)+1)\,k}{2N} - \frac{\pi\,(4m+1)\,l}{2N}\right] \qquad (4.28)$$

where $p(m)$ is defined as:

$$p(m) = [(4p+1)m + p] \ \ mod\ N. \tag{4.29}$$

By simple algebraic manipulation preceding equation can be rewritten:

$$4p(m) + 1 \equiv (4m+1)(4p+1) \ \ mod\ 4N. \tag{4.30}$$

Now we substitute $p(m)$ in previous equations and rewrite the cosine of the sum of two factors.

$$A_{k,l} = \sum_{p=0}^{N-1}\sum_{m=0}^{N-1} y_{p(m),m} \cos\left[\frac{\pi\,(4m+1)\,(4p+1)\,k + l}{2M}\right] \tag{4.31}$$

$$B_{k,l} = \sum_{p=0}^{N-1}\sum_{m=0}^{N-1} y_{p(m),m} \cos\left[\frac{\pi\,(4m+1)\,(4p+1)\,k - l}{2M}\right]. \tag{4.32}$$

Obtained expressions can be simplified by using substitution:

$$V_p(j) = \sum_{m=0}^{N-1} y\,(p(m), m) \cos\left(\frac{\pi(4m+1)j}{2N}\right) \tag{4.33}$$

$$p, j = 0, 1, ..., N-1.$$

Finally, $A_{k,l}$ and $B_{k,l}$ are defined as:

$$A_{k,l} = \sum_{p=0}^{N-1} V_p\,((4p+1)k + l) \tag{4.34}$$

$$B_{k,l} = \sum_{p=0}^{N-1} V_p\,((4p+1)k - l)\,. \tag{4.35}$$

Although not immediately visible, Eq. 4.33 can be seen as 1-D DCT. This is clearer after introducing a new permutation that can be incorporated into previously explained Nussbaumer's permutation 4.8 on page 42. Hence, no additional operations have to be performed.

$$\hat{y}_p(2m) = y\,(p(m), m) \tag{4.36}$$

$$\hat{y}_p(2m+1) = y\,(p(M-1-m), M-1-m) \tag{4.37}$$

$$m = 0, 1, ..., \frac{N}{2} - 1$$

And the result is 1-D DCT:

$$V_p(j) = \sum_{m=0}^{N-1} \hat{y}_p(m) \cos\left(\frac{\pi(2m+1)j}{2N}\right). \tag{4.38}$$

The nominator in the parameter of cosine function has been changed to $2m$ again. In addition, input symmetries are readily expressible as:

$$\begin{aligned} V_p(j+u2M) &= (-1)^u V_p(j) && (4.39)\\ V_p(2M-j) &= -V_p(j) && (4.40)\\ V_p(M) &= 0, && (4.41) \end{aligned}$$

and then by applying them to $A_{k,l}$ and $B_{k,l}$, we obtain:

$$\begin{aligned} A_{k,0} &= B_{k,0} && (4.42)\\ A_{0,l} &= B_{0,l} && (4.43)\\ A_{k,2M-l} &= -B_{k,l}. && (4.44) \end{aligned}$$

So, 2-D DCT has just been mapped to N 1-D DCTs and it can be computed that way, but a significant savings can be achieved by using polynomial transform. Let us start by constructing a polynomial:

$$\begin{aligned} B_k(z) &= \sum_{l=0}^{N-1} B_{k,l} z^l - \sum_{l=N}^{2N-1} A_{k,2N-l} z^l && (4.45)\\ B_k(z) &= \sum_{l=0}^{2N-1} \sum_{p=0}^{N-1} V_p\left((4p+1)k - l\right) z^l && (4.46)\\ B_k(z) &\equiv \sum_{p=0}^{N-1} \sum_{l=0}^{2N-1} V_p\left((4p+1)k - l\right) z^l \ \ mod\ (z^{2N}+1) && (4.47) \end{aligned}$$

Further, by the introduction of new substitutions:

$$\begin{aligned} U_p(z) &\equiv \sum_{j=0}^{2N-1} V_p(j) z^j \ \ mod\ (z^{2N}+1) && (4.48)\\ \hat{z} &\equiv z^4 \ \ mod\ (z^{2N}+1) && (4.49)\\ C_k(z) &\equiv \sum_{p=0}^{N-1} U_p(z) \hat{z}^{pk} \ \ mod\ (z^{2N}+1) && (4.50)\\ k &= 0, 1, ..., N-1, \end{aligned}$$

previous equations can be expressed as:

$$\begin{aligned} B_k(z) &\equiv \sum_{p=0}^{N-1}\sum_{l=0}^{2N-1} V_p\left(l-(4p+1)k\right) z^l \ \ mod\ (z^{2N}+1) && (4.51)\\ &\equiv \sum_{p=0}^{N-1}\sum_{l=0}^{2N-1} V_p\left(l\right) z^{l+(4p+1)k} \ \ mod\ (z^{2N}+1) && (4.52)\\ &\equiv \left(\sum_{p=0}^{N-1} U_p(z)\hat{z}^{pk}\right) z^k \ \ mod\ (z^{2N}+1) && (4.53)\\ &\equiv C_k(z)z^k \ \ mod\ (z^{2N}+1). && (4.54) \end{aligned}$$

Fast polynomial transform is based on the properties of polynomial ring and can be computed with butterfly-like addition stage. Previous equations satisfy conditions for application of polynomial transform because it can be easily shown that:

$$\begin{aligned} \hat{z}^N &\equiv 1 \ \ mod\ (z^{2N}+1) && (4.55)\\ \hat{z}^{N/2} &\equiv -1 \ \ mod\ (z^{2N}+1). && (4.56) \end{aligned}$$

By using symmetries on different stages of the algorithm, we can halve the number of additions. $U_p(z)$ and $C_k(z)$ symmetric properties follow from the input symmetries.

$$\begin{aligned} U_p(z) &\equiv U_p(z^{-1}) \ \ mod\ (z^{2N}+1) && (4.57)\\ C_{N-k}(z) &\equiv C_k(z^{-1}) \ \ mod\ (z^{2N}+1) && (4.58) \end{aligned}$$

By noting that $C_k(z)$ can be decomposed in similar way as DFT:

$$\begin{aligned} C_k(z) &= \sum_{p=0}^{N/2-1} U_{2p}(z)\hat{z}^{2pk} + \hat{z}^k \sum_{p=0}^{N/2-1} U_{2p+1}(z)\hat{z}^{2pk} && (4.59)\\ C_{k+N/2}(z) &= \sum_{p=0}^{N/2-1} U_{2p}(z)\hat{z}^{2pk} - \hat{z}^k \sum_{p=0}^{N/2-1} U_{2p+1}(z)\hat{z}^{2pk}, && (4.60) \end{aligned}$$

we can express existing symmetries as:

$$
\begin{aligned}
C^{j}_{\bar{n}2^j+\bar{k}}(z) &\equiv C^{j-1}_{\bar{n}2^j+\bar{k}}(z) + \qquad (4.61)\\
&\quad C^{j-1}_{\bar{n}2^j+\bar{k}+2^{j-1}}(z)\hat{z}^q \ \ mod\ \ (z^{2N}+1)\\
C^{j}_{\bar{n}2^j+\bar{k}+2^{j-1}}(z) &\equiv C^{j-1}_{\bar{n}2^j+\bar{k}}(z) - \qquad (4.62)\\
&\quad C^{j-1}_{\bar{n}2^j+\bar{k}+2^{j-1}}(z)\hat{z}^q \ \ mod\ \ (z^{2N}+1)
\end{aligned}
$$

$$
\begin{aligned}
\bar{k} &= k_{j-2}\cdots k_0 = 0, 1, ..., 2^{j-1}-1\\
\bar{n} &= n_0\cdots n_{t-j-1} = 0, 1, ..., 2^{t-j}-1
\end{aligned}
$$

$\bar{k}$ and $\bar{n}$ are simply an aggregate of bits of binary representation of n and k:

$$
\begin{aligned}
n &= n_{t-1}2^{t-1} + n_{t-2}2^{t-2} + \cdots + n_0\\
k &= k_{t-1}2^{t-1} + k_{t-2}2^{t-2} + \cdots + k_0.
\end{aligned}
$$

j is denoting the stage of algorithm which has $log_2 N$ stages altogether. q is computed as shown:

$$q = \sum_{l=0}^{j-2} k_l 2^{t-j+l}. \qquad (4.63)$$

Some $C_k(z)$ polynomials have symmetric property within themselves:

$$
\begin{aligned}
C^{j}_{\bar{n}2^j}(z^{-1}) &\equiv C^{j}_{\bar{n}2^j}(z)\ \ mod\ \ (z^{2N}+1) \qquad (4.64)\\
C^{j}_{\bar{n}2^j+2^{j-1}}(z^{-1}) &\equiv C^{j}_{\bar{n}2^j+2^{j-1}}(z)\ \ mod\ \ (z^{2N}+1). \qquad (4.65)
\end{aligned}
$$

Polynomials of type $Y(z^{-1})$ can be computed simply by noting that such a polynomial can be expressed as:

$$Y_{n_1}(z^{-1}) = \frac{P(z)}{z^x}. \qquad (4.66)$$

Here is a short procedure for computing the inverse. All equations are defined modulo arbitrary polynomial $M(z)$.

$$
\begin{aligned}
P(z) &\equiv Y(z^{-1})z^x\\
z^x &\equiv R(z)\\
P(z) &\equiv Y(z^{-1})R(z)
\end{aligned}
$$

$$\begin{aligned} Q(z) &\equiv P(z) \\ Y(z^{-1}) &\equiv \frac{Q(z)}{R(z)} \\ R_{inv}(z) * R(z) &\equiv 1 \\ Y(z^{-1}) &\equiv Q(z)R_{inv}(z) \end{aligned}$$

The inverse of $R(z)$ can be computed only if $(R(z), M(z)) = 1$, i.e. if those two polynomials are relatively prime.

5 Reference DCT Implementation

5.1 Distributed Arithmetic

Distributed arithmetic is an approach for computing inner products with look-up tables and accumulators based on the bit-level reformulation of the algorithm. This approach is particularly suitable for FPGA devices because of their regular architecture based on configurable logic blocks (CLB). Every CLB of Virtex-II FPGA device is composed of 2 slices and each of them has two look-up tables (LUT), storage elements and some other special-purpose logic. Thus, a distributed arithmetic DCT algorithm is simple and relatively efficient choice for the hardware implementation and it will be used as a reference point for the evaluation of polynomial transform algorithm.

The operation made of combined multiplication and additions is often used in digital signal processing and it is called MAC (multiply-accumulate). Usually one of the inputs is a constant.

If serial input is acceptable, a parallel MAC can be realized in relatively small amount of resources, as shown in Fig. 5.1. Input data (A, B, C and D) are serialized and shifted bit by bit (starting with the least significant bit) into scaling accumulator. Obtained products are summed with an adder tree.

As MAC is just a sum of vectors, which is a linear operation, it is clear that the circuit in Fig. 5.1 can be reorganized as shown in Fig. 5.2. Instead of individual accumulation of every single partial product and summation of results, we can postpone the accumulation until all N partial products are summed together. Such a reorganization eliminates $N-1$ scaling accumulators. If $C_x,\ x = 0, 1, \cdots, N-1$ are constants, the adder tree becomes a Boolean logic function of four input variables. The function can be implemented using one LUT in Xilinx family of FPGA devices[1]. Sign-extended LUT outputs are further added to the contents of accumulator. An example of the function table is given in Fig. 5.3.

Although just explained arithmetic manipulations seem intuitive and logical, it is necessary to verify their correctness mathematically. A simple proof

[1]Each LUT has 4 inputs and can be used as ROM, RAM, one-bit shift register or as an arbitrary 4-input boolean function circuit.

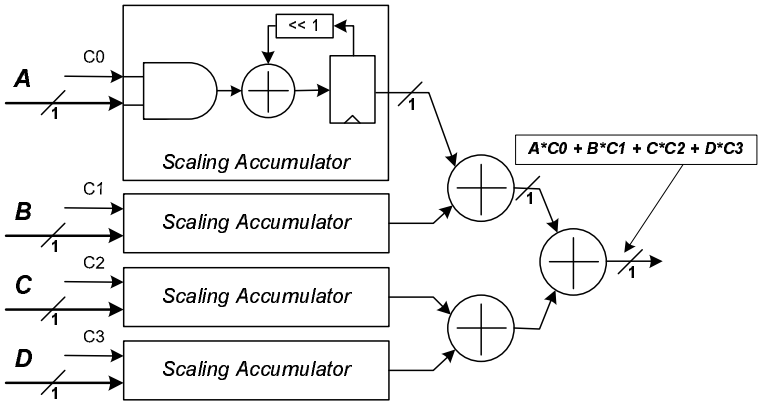


Figure 5.1: Final products summation

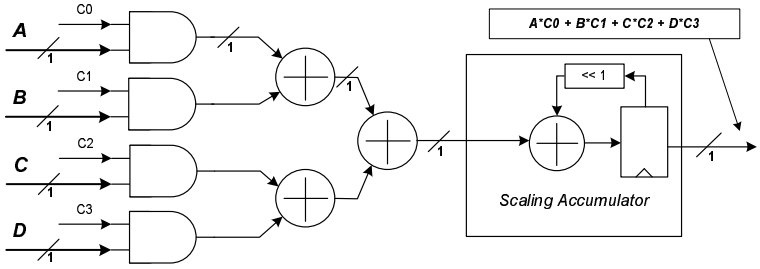


Figure 5.2: Summation of partial products

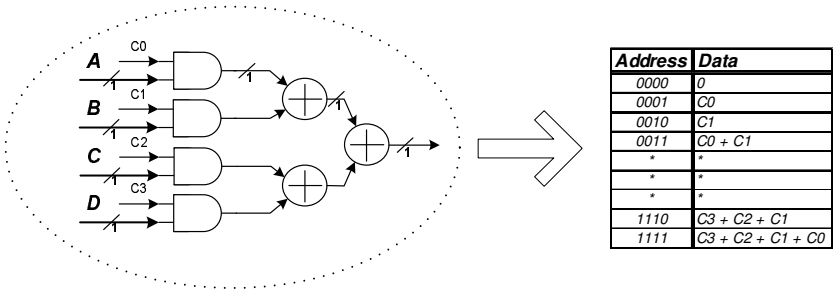


Figure 5.3: Implementation of partial product addition table

will be given. Equation 5.1 is an example of inner product with fixed coefficients A_k and input data x_k.

$$y = \sum_{k=1}^{K} A_k x_k \tag{5.1}$$

If every x_k is a binary number in two's complement representation scaled so that $|x_k| < 1$ (this is not a necessary condition, but it does simplify the proof), then x_k can be expressed at the bit level:

$$x_k = -b_{k_0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n}, \tag{5.2}$$

where b_{k_n} are bits (0 or 1). Bit b_{k_0} is a sign bit. Previous two equations can be combined.

$$y = \sum_{k=1}^{K} A_k \left[-b_{k_0} + \sum_{n=1}^{N-1} b_{k_n} 2^{-n} \right] \tag{5.3}$$

The equation for computing inner product in distributed arithmetic is obtained by changing the order of summation:

$$y = \sum_{n=1}^{N-1} \left[\sum_{k=1}^{K} A_k b_{k_n} \right] 2^{-n} + \sum_{k=1}^{K} A_k (-b_{k_0}). \tag{5.4}$$

As b_{k_0} can have only values of 0 and 1, expression $\sum_{k=1}^{K} A_k b_{k_n}$ can have only 2^K possible values. Those values can be precomputed and stored in read-only memory (ROM). Input data are shifted out bit by bit. All bits that are shifted out in one cycle are concatenated together and used as ROM address vectors. Depending on the implementation, we can shift the last significant or the most significant bit first. The result is then stored in the accumulator, which contains the final result after N cycles. It should be noted that in the second part of equation 5.4 the sign has to be changed in the last cycle (if the most significant sign bit is shifted-out as the last one). This last cycle is called sign-bit time (SBT).

Another way to implement the sign change in the last cycle is to fill ROM with both positive and negative values. Negative values are then used in the last cycle. This approach means that 2^{K+1} words have to be stored in the ROM. The sign change in SBT cycle is used for the simulation and implementation of DADCT in the thesis.

5.2 Algorithm Realization

Two-dimensional DCT equation:

$$X_{k_1,k_2} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n_1,n_2} cos\left[\frac{2\pi(2n_1+1)k_1}{4N_1}\right] cos\left[\frac{2\pi(2n_2+1)k_2}{4N_2}\right] \quad (5.5)$$

is isomorphic to the product of coefficient matrix and input signal vector. Coefficients are $A = cos(\frac{\pi}{4}), B = cos(\frac{\pi}{8}), C = cos(\frac{3\pi}{8}), D = cos(\frac{\pi}{16}), E = cos(\frac{3\pi}{16}), F = cos(\frac{5\pi}{16}), G = cos(\frac{7\pi}{16})$.

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{bmatrix} = \begin{bmatrix} A & A & A & A & A & A & A & A \\ D & E & F & G & -G & -F & -E & -D \\ B & C & -C & -B & -B & -C & C & B \\ E & -G & -D & -F & F & D & G & -E \\ A & -A & -A & A & A & -A & -A & A \\ F & -D & G & E & -E & -G & D & -F \\ C & -B & B & -C & -C & B & -B & C \\ G & -F & E & -D & D & -E & F & -G \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} \quad (5.6)$$

If we compute 1-D DCT directly as a product of a matrix and a vector as shown in Eq. 5.6, it would take 8 MAC circuits similar to the one shown in Fig. 5.2. Every circuit would compute 8 products (one matrix row dotted with an input data vector) and 7 additions. Such a circuit can be implemented with 8-bit input Boolean function that can be implemented in 3 FPGA LUTs. A better solution is to factorize DCT matrix [15] into two 4x4 matrices.

$$\begin{bmatrix} X_0 \\ X_2 \\ X_4 \\ X_6 \end{bmatrix} = \begin{bmatrix} A & A & A & A \\ B & C & -C & -B \\ A & -A & -A & A \\ C & -B & B & -C \end{bmatrix} \begin{bmatrix} x_0 + x_7 \\ x_1 + x_6 \\ x_2 + x_5 \\ x_3 + x_4 \end{bmatrix} \quad (5.7)$$

$$\begin{bmatrix} X_1 \\ X_3 \\ X_5 \\ X_7 \end{bmatrix} = \begin{bmatrix} D & E & F & G \\ E & -G & -D & -F \\ F & -D & G & E \\ G & -F & E & -D \end{bmatrix} \begin{bmatrix} x_0 - x_7 \\ x_1 - x_6 \\ x_2 - x_5 \\ x_3 - x_4 \end{bmatrix} \quad (5.8)$$

The algorithm then requires an addition butterfly and a number of 4-input MACs that can be realized with only one LUT per MAC. The data flow

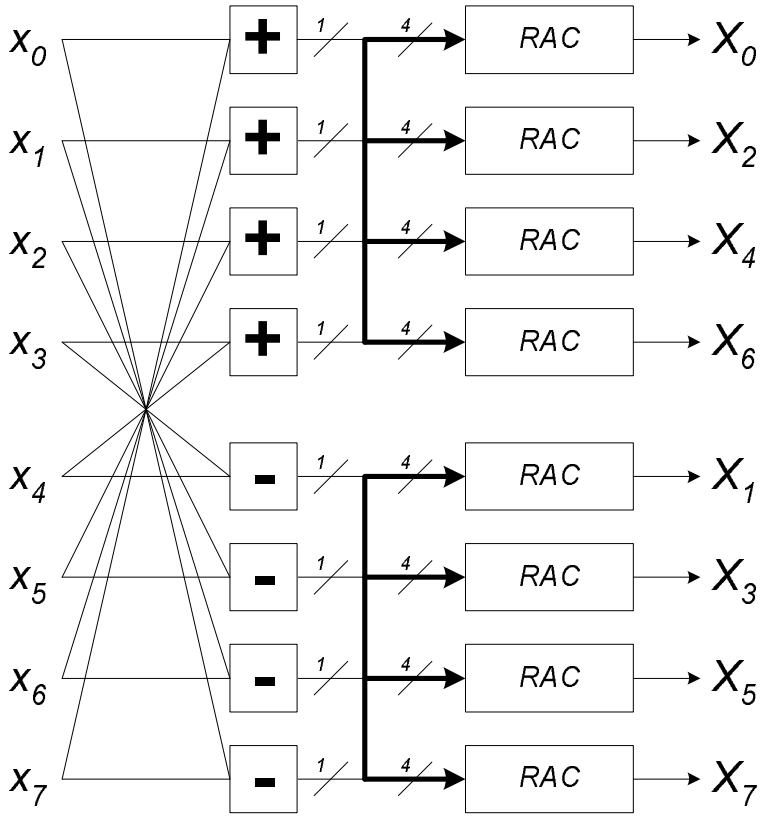


Figure 5.4: Data flow diagram of 8-point DADCT algorithm

diagram in Fiq. 5.4 is the implementation of 1-D DCT using partially factorized DCT matrix. Combined ROM-accumulate blocks (RAC) compute the result in nine cycles for eight-bit input data. Upper half of the circuit can be factorized further, but it might not be worth the effort for 8-point DCT since only marginal savings can be achieved.

ROM coefficients are fractional numbers in two's complement representation. Correction factors can also be easily computed into ROM coefficients and it can be shown that these corrected coefficients are in the range from -1.73145 to 2.82843. Hence, 3 bits are needed for the representation of the coefficient in front of the virtual floating point and an arbitrary number of precision bits.

The reference DADCT implementation for comparison with MPTDCT algorithm will be implemented by using row-column decomposition. Every dimension requires 8 1-D DADCT circuits, like the one shown in 5.4.

5.3 Accuracy Analysis

As mentioned before, accuracy of the forward-inverse DCT transformation process influences significantly the quality of multimedia content. The recommendations for the accuracy of 2-D NxN IDCT implementations are standardized by IEEE [1] and are given in Table 5.1. These recommendations will be considered as the highest error allowed for DCT implementations.

Metric	Recommended maximum value
Pixel Peak Error (PPE)	1
Peak Mean Square Error (PMSE)	0.06
Overall Mean Square Error (OMSE)	0.02
Peak Mean Error (PME)	0.015
Overall Mean Error (OME)	0.0015

Table 5.1: Maximal allowed errors for 2-D DCT implementations

Another important metric for the measurement of accuracy is peak signal to noise ratio (PSNR). Assume we are given a source image $f(i,j)$ that contains NxN pixels and a reconstructed image $F(i,j)$ where F is reconstructed by IDCT of the discrete cosine transform of the source image. OMSE is computed as the summation over all pixels of squared error. And PSNR is computed as shown in Eq. 5.10.

$$OMSE = \frac{\sum\sum[f(i,j) - F(i,j)]}{N^2} \tag{5.9}$$

$$PSNR = 10log_{10}\left(\frac{255}{\sqrt{OMSE}}\right) \tag{5.10}$$

A block diagram of accuracy measurement is depicted in Fig. 5.5. Tested implementation of DCT is used to transform original image to DCT frequency domain. IDCT is computed by an ideal IDCT (64 bit floating point) and the error is represented by the difference between IDCT output and the original image.

Both DADCT and MPTDCT algorithms have been simulated in SystemC[2], a C++ library made for system level design and verification. It provides an abundance of fixed point types for the simulation of finite register length effects. This approach seems to be very practical for functional

[2] **http://www.systemc.org/**

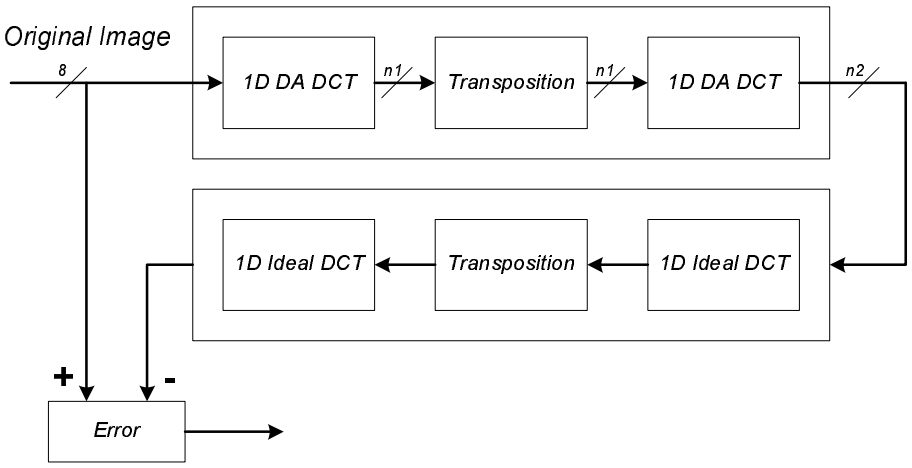


Figure 5.5: DCT accuracy measurement

verification and accuracy measurement. It makes register pruning simple because register widths can be changed easily and simulation rerun to analyze accuracy of the new circuit with pruned register.

For DADCT algorithm simulation three parameters have been varied and simulation was rerun each time to obtain new accuracy measurement data:

- $ROM_{width} \sim$ bit-width of coefficient word in ROM,
- $1DIM_{prec} \sim$ number of precision bits of the first dimension DCT result,
- $2DIM_{prec} \sim$ number of precision bits of the final result.

Other parameters can be computed from those three. It can be formally proved that the output from the first dimension (n_1 in Fig. 5.5) cannot have more than 12 accurate bits in front of the virtual floating point for an eight bit input. Alternatively, ROM words need to have 3 integer bits for the representation of DCT coefficients and since 9 addition cycles are needed (two 8-bit values are added in the butterfly prior to MACs), we can get at most 12 accurate bits. Therefore, the width of ROM words is:

$$ROM_{width} = 3 + ROM_{prec}, \tag{5.11}$$

where ROM_{prec} is an arbitrary number of precision bits. The result in accumulator can be truncated in order to decrease the number of n_1 bits. In the simulation and implementation another kind of quantization was used - rounding towards infinity, implemented by adding the most significant deleted bit to the left bits. Accordingly, n_1 output can be expressed as:

$$n_1 = 12 + 1DIM_{prec}. \tag{5.12}$$

It has been empirically established that increasing the number of necessary output integer bits from the second dimension above n_1 bits does not result in any significant accuracy improvement, so equation 5.13 has been used for the width of output from the second dimension, where $2DIM_{prec}$ is an arbitrary number of precision bits. The same ROM_{width} parameter has been used for both dimensions. This simplification, together with parameter range constraints and simplifications introduced by equations 5.11, 5.12 and 5.13, was necessary to limit the number of combinations that have to be simulated.

$$n_2 = n_1 + 2DIM_{prec} \tag{5.13}$$

Three input stimulus files have been used; Lena 5.6(a) and boats 5.6(b) pictures and a random picture generated by the C program listed the in appendix of IEEE standard [1].

(a) Lena (b) Boats

Figure 5.6: Simulation stimulus pictures

According to simulation results in Fig. 5.7, ROM_{width} has the largest impact on accuracy. Parameters $1DIM_{prec}$ and $2DIM_{prec}$ have been fixed to 2. The results for both pictures are almost the same, while the accuracy of noise stimulus file is significantly worse because of higher frequencies in the picture. If ROM_{width} is increased above certain number, error drops to zero and PSNR is infinite. For example, for both test pictures, error drops to zero when 11 or more bits are used for ROM words.

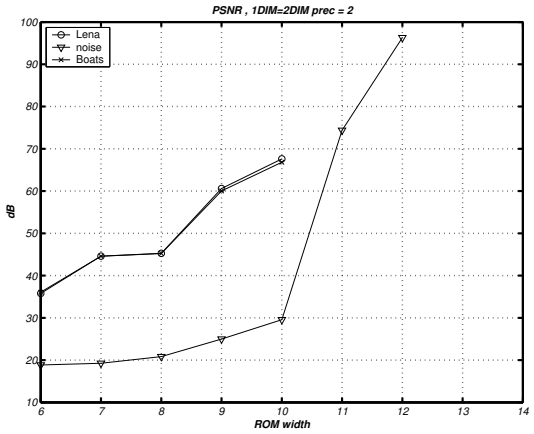
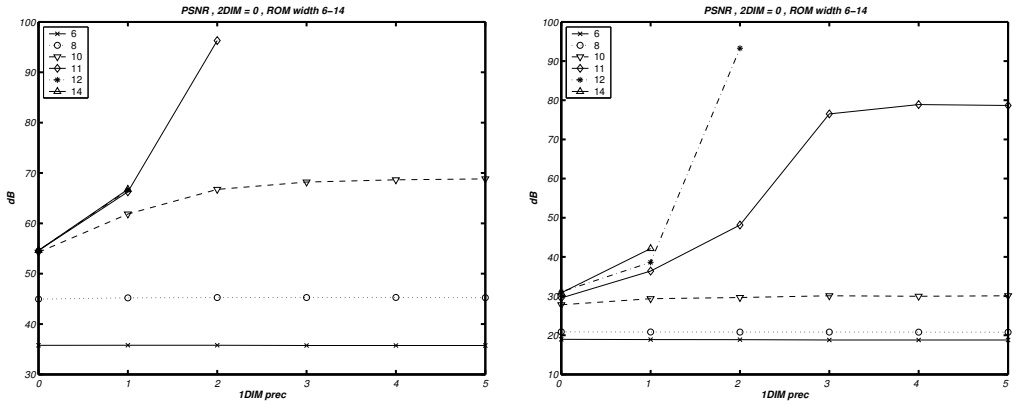


Figure 5.7: DADCT simulation results for various ROM word-lengths

Figure 5.8(a) supports the claim that ROM word length is the most influential paramter. The same claim is also valid for noise stimulus, according to 5.8(b).



(a) Lena (b) Noise

Figure 5.8: DADCT simulation results for different ROM_{width} and $1DIM_{prec}$ values

Much better insight in interdependencies of different parameters can be gained by considering PSNR while changing two variables. Simulation results, shown in graphs 5.9, clearly mean that the ROM word length is the most important factor, followed by the precision of the first dimension output. The precision of the output from second dimension has almost no influence

on PSNR.

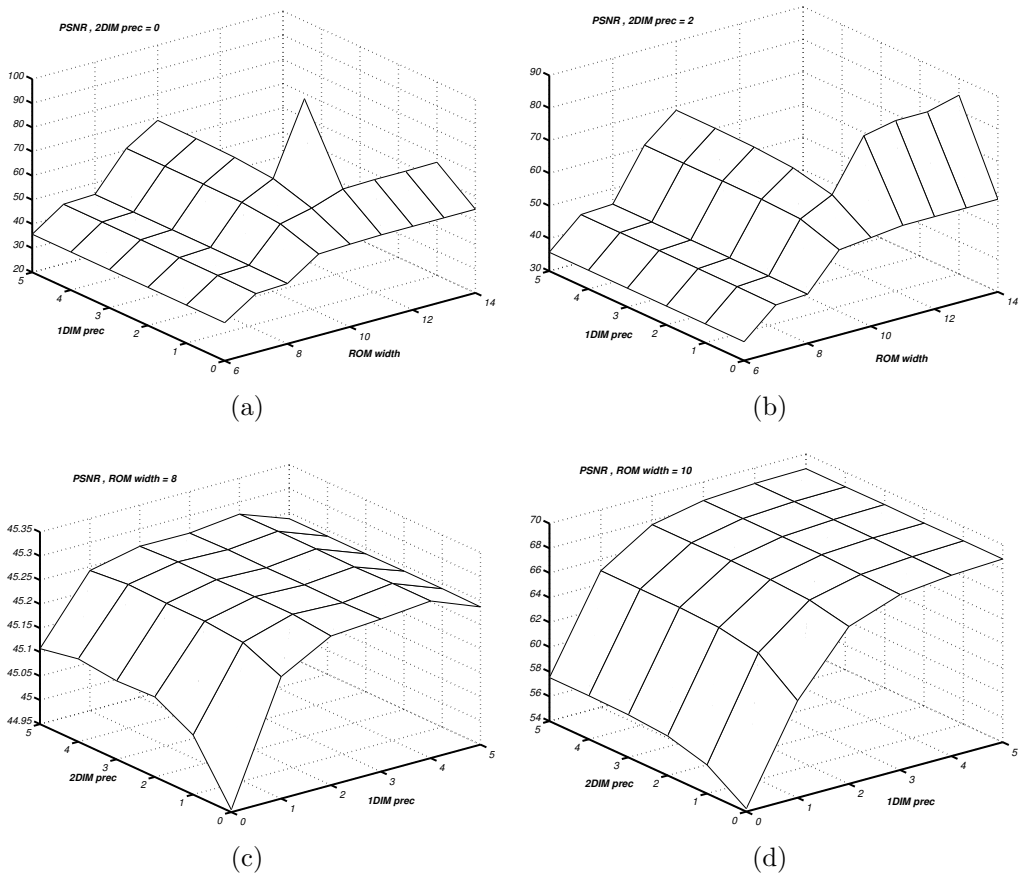
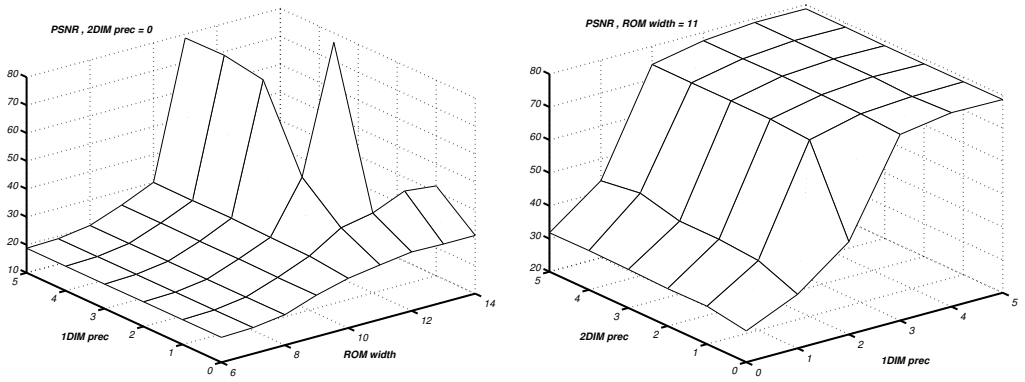


Figure 5.9: Simulation results for picture Lena

Noise stimulus yields similar results with somewhat steeper surfaces (Fig. 5.10 on the next page).

5.4 FPGA Implementation

The following parameter values have been chosen for the implementation, according to the simulation results in previous section: $ROM_{width} = 10$, $1DIM_{prec} = 2$ and $2DIM_{prec} = 2$. The implementation completely satisfies the standard as shown in Table 5.2, except for the PME metric which is slightly above the recommended value. This has no visible impact on the



(a) (b)

Figure 5.10: Simulation results for noise stimulus

picture quality. If deviations from the standard are allowed, significant hardware resources can be saved by register truncation and shorter ROM words.

Metric	Value obtained by simulation
PPE	1
PMSE	0.03256
OMSE	0.01661
PME	0.01637
OME	0.0
PSNR	65.9255 dB

Table 5.2: DADCT implementation accuracy

The design has been synthesized in Xilinx Virtex-II FPGA device, speed grade -6 by using reentrant route with the guide file from the multi-pass place and route process. Input-output pin insertion was disabled during the synthesis. This implementation requires 6401 Virtex-II LUTs, what is less then reported by Dick in [9] for the same algorithm. His implementation requires 3328 Xilinx 4000 CLBs. Each 4000 series CLB has 2 LUTs, so it totals to 6656 LUTs. But it is hard to make direct comparison because he hasn't reported the accuracy of his implementation. In addition, our Virtex-II implementation has three-state buffers instead of relatively large multiplexor that would be needed at the output and three-state buffers are not available in 4000 series.

Implementation area group summary:

```
AREA_GROUP AG_DADCT
RANGE: SLICE_X0Y111:SLICE_X95Y72
Number of Slices:           3,840 out of  3,840  100%
Number of Slices containing
   unrelated logic:           526 out of  3,840   13%
Number of Slice Flip Flops: 4,941 out of  7,680   64%
Total Number 4 input LUTs:  6,401 out of  7,680   83%
   Number used as 4 input LUTs:           5,185
   Number used as route-thru:                16
   Number used as 16x1 ROMs:              1,200
Number of Tbufs:              960 out of      0    0%
```

According to the chosen implementation parameters, first dimension output is 14 bits wide, while the output from second dimension is 15-bit wide. Nine cycles would do for computing the first dimension as its input is 8-bit, but for computing second dimension we need 15 cycles, hence the complete core can produce a new results every 15 cycles. The part of core for computing second dimension is somewhat larger and it limits the maximum achievable working frequency because of longer carry chains.

Important property of hard macros[3] is porosity. It is the degree to which the macro forces the signals of other macros to be routed around it rather than through it. As DADCT core is tightly packed (fills 83% of available LUTs of designated area group), porosity might be a problem, but it can be simply solved by relaxing placement constraints.

Frame size	Frame rate
1600x1200	277
1920x1080	257
1024x1024	508
2048x2048	127
4096x4096	31
6000x6000	14

Table 5.3: Parallel DADCT processor frame rates

Input data 8x8 block of 8-bit data is shifted in parallel DADCT core in 8 cycles, latency is 30 cycles and new results are available every 15 cycles. Minimum period is 7.998ns (∼125MHz).

Let us denote the number of cycles needed to transform data matrix as C, operating frequency as f, frame size as F_ixF_j, and DCT size as NxN,

[3]Compiled, mapped, placed and routed digital design block.

then the time needed to compute the DCT transform of the frame can be expressed as:

$$t_{frame} = \frac{F_i F_j}{N^2} \, C \, \frac{1}{f}, \tag{5.14}$$

and frame rate is equal to $R = 1/t_{frame}$. Frame rates for some frame sizes are given in Table 5.3.

6 MPTDCT Implementation

The first dimension in the MPTDCT algorithm is equivalent to the first dimension of DADCT implementation, while the second one is implemented via a number of adders and shifters. Implementation parameters for the first dimension were $ROM_{width} = 11$, $1DIM_{prec} = 2$ and for the second the only parameter is $2DIM_{prec} = 0$. The extension of the ROM word length for one bit will be explained later. Result in the accumulator in the first dimension was simply truncated before proceeding with computation. This simple round-off scheme produced better results in combination with polynomial transform than rounding towards infinity when second dimension is implemented using distributed arithmetic. In addition, less hardware resources were used for accumulator implementation.

6.1 Accuracy Analysis

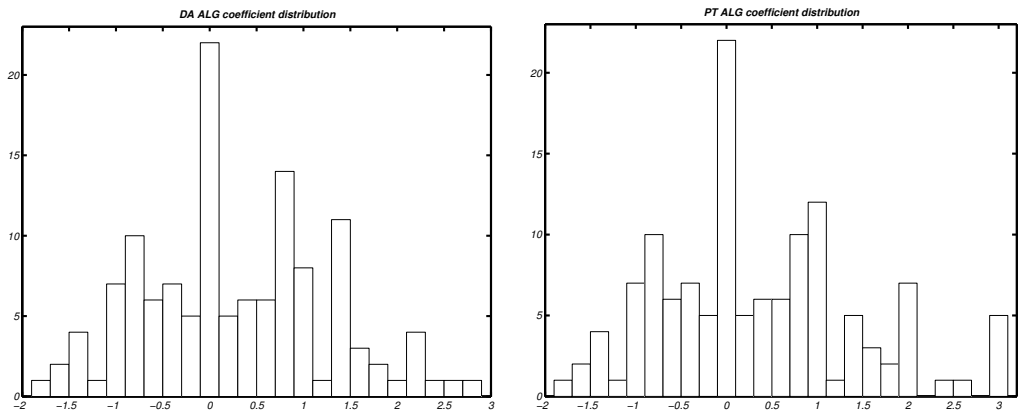
In DADCT implementation there are two major sources of errors, namely the finite register length of ROM memory for storing DCT coefficients and accumulator truncation. As explained before, for DADCT it was necessary to implement rounding towards infinity to reduce the error in order to comply with the IEEE standard. Simple rounding scheme was more efficient, in the terms of usage of the FPGA resources, than increasing ROM word length. The computation of both dimensions introduced noise in transformed data.

Polynomial transform implementation of DCT, on the other hand, does not introduce any error in second dimension because all operations are simple additions or subtractions. From accuracy measurement results in Table 6.1, it follows that much better results have been achieved.

Metric	Value obtained by simulation
PPE	1
PMSE	0.00138889
OMSE	0.000477431
PME	0
OME	0.0000385802
PSNR	81.3417 dB

Table 6.1: MPTDCT implementation accuracy

The correction factors for DADCT were incorporated into DCT coefficients in ROM in both dimensions. Polynomial transform algorithm is defined without these factors, and the result has to be scaled after computing the second dimension. This can be implemented with simple truncations and bit serial multiplications with correction factors in the last stage of computation with relatively insignificant hardware resources. As DADCT coefficients have been scaled, their range and distribution is somewhat different from those of MPTDCT, as shown in histograms 6.1. As correction factors are smaller then one, DADCT has narrower ROM word range. According to the given histograms, 3 bits for the integer part is not enough for MPTDCT, and 4 bits have to be used instead. As only a small percentage of words has extreme values, HDL (Hardware Description Language) compilers tend to optimize ROMs pretty well, and this single bit extension has not resulted in significantly larger circuit. Hence, it seems to be fair to compare DADCT with 10 bit ROM word length against 11-bit MPTDCT, especially because the precision is the same.



(a) With correction factors (b) Without correction factors

Figure 6.1: Coefficients distribution histograms

The most influential parameter is ROM word length, the same as was the case for distributed arithmetic implementation. Even if we compare the two implementations with the same ROM word length (equivalent to decreasing MPTDCT precision for one bit), polynomial transform based approach is still slightly better in the terms of accuracy.

The precision of output from second dimension had almost no influence on the accuracy. Hence, it was interesting only to simulate the influence of variations of $1DIM_{prec}$ and ROM_{width} parameters. Results are shown in

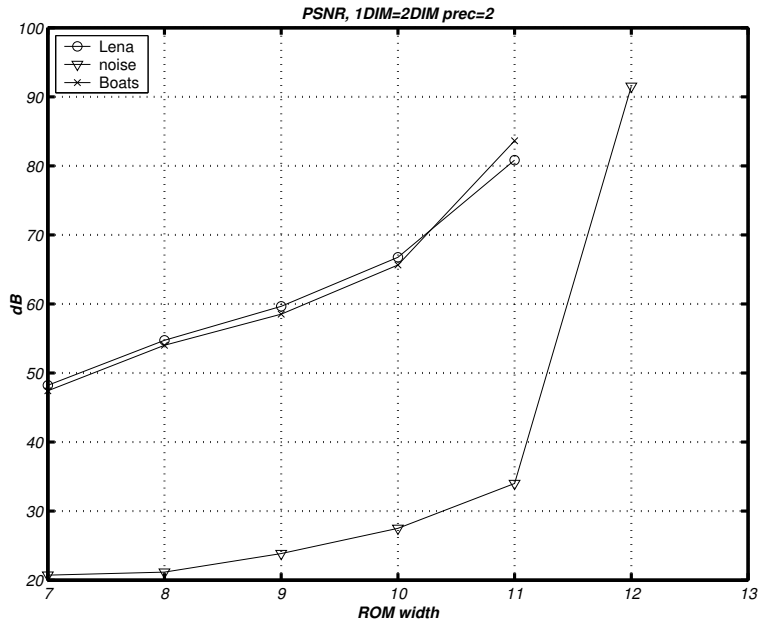


Figure 6.2: MPTDCT simulation results for various ROM word lengths

Fig. 6.3.

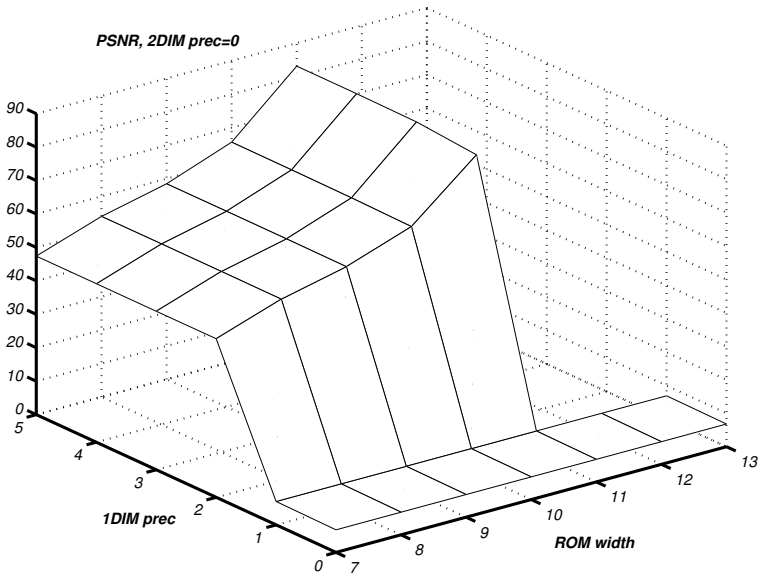


Figure 6.3: MPTDCT accuracy simulation results

6.2 FPGA Implementation

Input data matrix has to be permuted first, according to Fig. 6.4. The shaded elements in the first matrix represent the first row, the second matrix the second row, and so on. Elements remain in the same column as shown. Hence, the first row of permuted matrix would contain elements $E_{0,0}$, $E_{1,1}$, $E_{2,2}$ and so on. Input data are shifted in the circuit in 8 cycles (64-bit bus) and sorted according to permutation matrix to proper registers. Wide multiplexors have been implemented by using three-state buffers.

Second problem in the implementation was to analyze algorithm symmetries. *Mathematica* code listing in Appendix A.1 was used for that purpose. The code first constructs symbolic functions V that are used in the analysis. Second, basic functions needed for the construction of polynomial transform are defined and the function for computation of $X(k,l)$ in equation 4.26 on page 45 is given.

The rest of the listing constructs polynomial transform matrix that is used for the analysis of possible introduction of correction factors in the first dimension ROMs. If it were possible to efficiently incorporate this correction coefficient into ROM, final scaling would be avoided cheaply. So it is interesting to investigate this possibility.

Assume that the one-dimensional DCT without correction factors is computed on the result of permuted 8x8 input data matrix, and that the result can be represented as 64x1 input vector X_i. Polynomial transform is in that case 64x64 linear operator matrix P_t. Let us denote the result of 2D DCT (without correction) as 64x1 vector X_0 and correction factor matrix as 64x64 matrix K. Hence, the computation of MPTDCT can be expressed as:

$$P_t X_i = X_0. \tag{6.1}$$

DCT matrix can be orthonormalized by multiplication with K. Now, an interesting question is whether there exists a simple regular matrix C that satisfies:

$$P_t C X_i = K X_0. \tag{6.2}$$

Simple calculation yields that matrix C can be computed as:

$$C = P_t^{-1} K P_t. \tag{6.3}$$

Ideal C matrix would have all rows the same, meaning that every element of input data matrix can be scaled by certain value in the first dimension

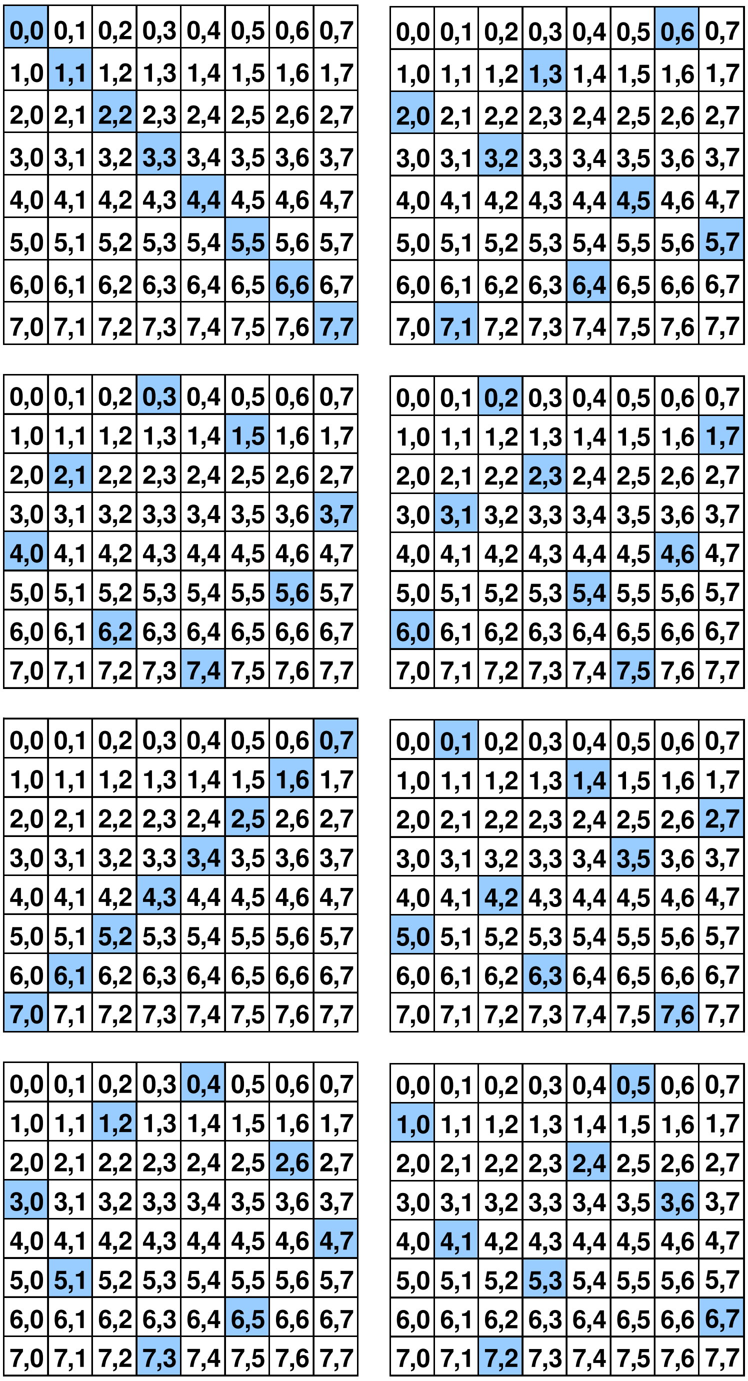


Figure 6.4: Input matrix permutation

stage and the result would be a complete DCT. But, as polynomial transform is doing computation on the complete 8x8 matrix (that's why it has to be represented as 64x64 operator), C matrix is irregular and therefore not practical. Thus, the only solution is to scale the result.

After profound symmetry analysis, it became clear that computation stages can be reorganized in such a way that only a very simple output reordering has to be done. Therefore it can be said that this implementation does not need output permutation processor, as the one in [8]. But some reordering also has to be done after first dimension in order to obtain more regular butterfly stage. The first stage of butterfly is a regular addition and subtraction stage, where values can be processed in any order that is most suitable for later computation. Second stage, denoted simply as S, is more interesting and it is given in the Appendix A.2. The code for other stages can also be found in the Appendix.

Although the code for second stage seems quite irregular, it is possible to organize the computation in an efficient way by reordering the computation of first stage. The most problematic are stages S5 and S7. Their implementation is based on the fact that 3 out of 8 values are mirrored in the sequence, so they can be precomputed during computation of first four values, stored in a buffer and simply multiplexed into later stages a few cycles later. Although it might seem a good idea to start the organization of computation from the first stage, every later stage is more irregular, so we have used the scheduling of computation from the last and the most irregular stage in the algorithm. Stages are denoted as follows; F-first, S-second, C-third (according to the symbol for C polynomial in the algorithm equations) and X is the final result. The cycle of polynomial is denoted by the number next to the stage symbol, so for example, the fifth cycle in the final stage would be X5.

The design has been synthesized in Virtex-II FPGA device, speed grade -6. Input-output pin insertion was disabled. Although not yet fully optimized, MPTDCT implementation of 8x8 DCT required 3949 LUTs, what is less then reported by Dick in [8] for Duhamel's algorithm. His algorithm requires 4496 XC4000 LUTs. The direct comparison is hard because Virtex-II devices have some features that are not available in XC4000 family. In addition, our MPTDCT algorithm implementation is not yet fully optimized and control logic is implemented in a very naive manner. Therefore we have implemented classical 2D DADCT algorithm in Virtex-II in 6400 LUTs in order to obtain a referent implementation for a comparison. Our MPTDCT requires 37,5% less logic resources than highly optimized DADCT while maintaining the same throughput and working frequency. Even better results can be achieved by optimizing the implementation and redesigning the con-

trol logic. An implementation of Duhamel's 8x8 DCT algorithm in XC4000 saved 33% of resources comparing to DADCT implementation in XC4000. Main reasons for lower resource requirements of MPTDCT algorithm that are simpler butterfly addition stage and the absence of complex output permutation processor. Additionally, some savings have been achieved by using three-state buses as a part of input permutation processor.

According to static timing analysis, design should be working on the frequency of 201.7 Mhz (4.957ns period). A new result can be obtained every 10 cycles, while the total latency is 60 cycles. For HDTV resolution frame (1920x1080) it means a throughput of 622 frames per second. Although latency is larger then in DADCT implementation, this is usually not an issue for multimedia applications and throughput is considered more important.

The significant effort to realize an implementation of such a complex algorithm seems to be justified for high-performance multimedia applications that require block processing in the real time. As DCT is separable transform, higher-dimensional DCT could be computed by using two-dimensional implementation presented here. Therefore this very high throughput design could be especially interesting for emerging applications like mentioned in [7].

Summary

Polynomial transform was presented as an efficient way to map multidimensional transforms (like DCT and DFT) and convolutions to one-dimensional ones. Resulting algorithms have significantly lower computational complexity, especially for convolutions and DCT. If optimal one-dimensional DCT would be available, it would be possible to obtain optimal multidimensional algorithms by using polynomial transforms.

As demonstrated in the thesis, polynomial transform based DCT can be implemented in significantly less logic resources than row-column distributed arithmetic algorithm. Polynomial transform computation is performed as the series of additions and subtractions, so the finite-register length effects can be avoided resulting in higher accuracy.

Second dimension of distributed arithmetic DCT implementation was more problematic due to larger input word length. The resulting longer carry chain, especially the one in the large accumulators at the output, was the major obstacle to achieving higher operating frequency. The implementation of polynomial transform DCT avoids this obstacle by using a butterfly of adders and no accumulators are needed at the output.

The accuracy of both implementations has been extensively measured and detailed results were reported, more detailed then available in the previous work. Once the strong foundations for comparison were built, other factors, like resource usage, maximum achievable frequency, latency and throughput have been compared.

Another contribution of the thesis is the proposal of designing the butterfly stage of such highly complex algorithms by scheduling the computation from the last stage, where the designer has no flexibility. This approach has yielded a very efficient FPGA implementation, superior to others reported.

Keywords: discrete cosine transform, polynomial transform, FPGA

Sažetak

Polinomijalna transformacija je predstavljena kao efikasan način mapiranja višedimenzionalnih transformacija (kao što su DCT i DFT) i konvolucija u jedno-dimenzionalne. Algoritmi koji nastaju takvim mapiranjem zahtjevaju znatno manji broj računskih operacija, posebno u slučaju konvolucija i DCT-a. Ako se koristi optimalan DCT algoritam za prvu dimenziju, korištenjem polinomijalnih transformacija se mogu konstruirati višedimenzionalni algoritmi koji dostižu teoretski minimum broja operacija.

Kao što je pokazano u magistarskom radu, DCT baziran na polinomijalnim transformacijama se može implementirati tako da zauzima znatno manje logičkih resursa nego algoritam baziran na distribuiranoj aritmetici. Polinomijalne transformacije se računaju nizom zbrajanja i oduzimanja, tako da se mogu izbjeći efekti konačne dužine registara i prema tome dobiti viša preciznost.

Druga dimenzija implementacije bazirane na distribuiranoj aritmetici je znatno problematičnija od prve zbog veće širine ulazne riječi. Glavna prepreka većoj brzini rada je dugački lanac prijenosa u akumulatoru na izlazu druge dimenzije. Implementacija bazirana na polinomijalnim transformacijama nema takvih nedostataka jer se izvodi nizom zbrajala i oduzimanja bez akumulatora na izlazu.

Izmjerena je preciznost obje implementacije i prikazani su rezultati koji su detaljniji od predhodno objavljenih. Jednom kad su položeni jaki temelji za usporedbu, obje implementacije su uspoređene i po drugim faktorima kao što su zauzeće logičkih resursa, maksimalna radna frekvencija, latencija i količina podataka koja se može obraditi u jedinici vremena.

Dodatni doprinos rada je prijedlog za dizajniranje izlaznog stupnja koji je baziran na organizaciji operacija počevši od zadnjeg stupnja, gdje dizajner ima najmanje slobode. Takav pristup je rezultirao u vrlo efikasnoj FPGA implementaciji koja je superiorna drugim implementacijama koje su objašnjene u literaturi.

Ključne riječi: diskretna kosinusna transformacija, polinomijalna transformacija, FPGA

Resume

Domagoj Babić was born 24^{th} September 1977. in Sisak, Croatia. He has received Dipl.ing. title form the Faculty of Electrical Engineering and Computing in Zagreb 2001. His major was industrial electronics. From 2001. he is employed as a research assistant on the Faculty of Electrical Engineering and Computing where he has enrolled in the postgraduate programme of Core Computer Science. His research interests include design and verification of digital circuits and digital signal processing algorithms.

Životopis

Domagoj Babić je rođen 24. rujna 1977. u Sisku. Studirao je na Fakultetu elektrotehnike i računarstva u Zagrebu i diplomirao 2001. godine na smjeru Industrijska elektronika. Od 2001. godine je zaposlen kao znanstveni novak na Fakultetu elektrotehnike i računarstva kada je upisao i poslijediplomski studij, smjer Jezgra računarskih znanosti. Bavi se dizajnom i verifikacijom digitalnih sklopova i algoritmima za obradu digitalnih signala.

Appendix A

```
(* Transform size *)
DN = 8;

(* V_p symbolic function definition *)
Ve[p_Integer, j_Integer] := V[p, j] /; 0 = j < DN /; 0 = p < DN
Ve[p_Integer, DN] := V[p, DN];
Ve[p_Integer, j_Integer] := -V[p, 2*DN - j] /; DN = j < 2*DN /;
  0 = p < DN
V[p_Integer, DN] := 0;

(* U_p symbolic polynomial definition *)
U[p_Integer] := Collect[Sum[(z^j)*Ve[p, j], {j, 0, 2*DN - 1}], z];

(* Converts list to polynomial of variable var *)
List2Poly[lst_List, var_] := Module[{res = 0, len = Length[lst]},
  For[idx = 1, idx = len, idx++, res = res + var^(idx - 1)*
  lst[[idx]]];
  res
];

(* C_k symbolic polynomial definition *)
Cfun[k_Integer] := Collect[
  PolynomialRemainder[
    Sum[(z^(4*k*p))*U[p], {p, 0, DN - 1}], z^(2*DN) + 1, z
  ], z
]
(* B_k symbolic polynomial definition *)
Bk[k_Integer] := Collect[
  PolynomialRemainder[(z^k)*Cfun[k], z^(2*DN) + 1, z], z
]


(* X_k polynomial reconstruction - symbolic solution
   of polynomial transform *)
bkHelper[bkl_List, idx_Integer] := -bkl[[1]] /; idx == 17
bkHelper[bkl_List, idx_Integer] := bkl[[idx]] /; 1 = idx < 17
Xk[k_Integer] := Module[{
    lst = List{}, bk = CoefficientList[Bk[k], z]},{
    For[i = 0, i < DN, i++, {
      AppendTo[lst, (bkHelper[bk, i + 1] -
      bkHelper[bk, 17 - i])/2]
    }],
    List2Poly[lst, z]
  }
]
```

```
(* Convert X_k to list *)
XkList[k_Integer] := CoefficientList[Xk[k][[2]], z]
(* Create a list of lists, y contains
   all symbolical solutions *)
y = {
  XkList[0], XkList[1], XkList[2], XkList[3],
  XkList[4], XkList[5], XkList[6], XkList[7]
};

(* Convert symbolic solutions to matrix *)
ymat = Partition[Sort[y], 8];

(* Initialize an empty 64x64 table *)
TableForm[PT = Table[0, {64}, {64}]];

(* Create PT operator matrix, dimension N^2 x N^2 *)
For[i = 1, i = DN, i++, For[j = 1, j = DN, j++,
  For[k = 1, k <= Length[Level[Expand[ymat[[1]][[i]][[j]]],
    {1}]], k++,
    Module[{dpt = Depth[tmp = Part[Expand[ymat[[1]][[i]][[j]]],
      k]]},{
      lvl = Level[tmp, {1}],
      elem = 0,
      coef = 0,
      If[
        dpt == 3, {elem = lvl[[2]],
        coef = lvl[[1]]}, {elem = tmp, coef = 1}
      ],
      tmpl = {ToExpression[ToBoxes[elem][[1]][[3]][[1]][[1]]],
        ToExpression[ToBoxes[elem][[1]][[3]][[1]][[3]]]
      },
      inr = (i - 1)*8 + j, inc = tmpl[[1]]*8 + tmpl[[2]] + 1,
      PT[[inr, inc]] = coef
    }
  ]
]]]

(* Compute inverse PT operator *)
IPT = Inverse[PT];
```

Listing A.1: Mathematica code for symmetry analysis and computing PT transform matrix

```
for (c=0; c < 8; c++) {
    S[0][c] = F[0][c] + F[2][c];
    if (c==0) {
        S[1][c] = F[1][c];
    } else {
        S[1][c] = F[1][c] + F[3][8-c];
    }
    S[2][c] = F[0][c] - F[2][c];
    if (c < 7) {
        S[3][c] = F[1][c+1] - F[3][7-c];
    } else {
        S[3][c] = -F[3][0];
    }
    S[4][c] = F[4][c] + F[6][c];
    if (c==0) {
        S[5][c] = F[5][c];
    } else {
        S[5][c] = F[5][c] + F[7][8-c];
    }
    S[6][c] = F[4][c] - F[6][c];

    if (c < 7) {
        S[7][c] = F[5][c+1] - F[7][7-c];
    } else {
        S[7][c] = -F[7][0];
    }
}
```

Listing A.2: Second stage of MPTDCT algorithm

```
for (c=0; c < 8; c++) {
    C[0][c] = S[0][c] + S[4][c];
    if (c < 4) {
        C[1][c] = S[1][c] + S[7][3-c];
    } else {
        C[1][c] = S[1][c] + S[5][c-4];
    }
    if (c==0) {
        C[2][c] = S[2][c].dbl();
    } else {
        C[2][c] = S[2][c] + S[6][8-c];
    }
    if (c == 0) {
        C[3][c] = S[1][0] + (-S[7][3]);
    } else if (c > 0 && c < 5) {
        C[3][c] = S[3][c-1] + (-S[7][3+c]);
    } else {
        C[3][c] = S[3][c-1] + S[5][12-c];
    }
    C[4][c] = S[0][c] - S[4][c];
    if (c < 3) {
        C[5][c] = S[1][c+1] - S[7][2-c];
    } else if (c >= 3 && c < 7) {
        C[5][c] = S[1][c+1] - S[5][c-3];
    } else {
        C[5][c] = -S[3][7] - S[5][c-3];
    }
    if (c < 7) {
        C[6][c] = S[2][c+1] - S[6][7-c];
    } else {
        C[6][c] = -S[6][0].dbl();
    }
    if (c < 4) {
        C[7][c] = S[3][c] - (-S[7][4+c]);
    } else {
        C[7][c] = S[3][c] - S[5][11-c];
    }
}
```

Listing A.3: Third stage of MPTDCT algorithm

```
X[0][0] = C[0][0];
X[1][0] = C[7][0];
X[2][0] = C[6][1];
X[3][0] = C[5][2];
X[4][0] = C[4][4];
X[5][0] = C[3][5];
X[6][0] = C[2][6];
X[7][0] = C[1][7];

for (c=1; c < 8; c++) {
    X[0][c] = C[0][c];
    X[1][c] = (C[1][c-1] - (-C[7][c]));
    if (c == 1) {
        X[2][c] = (C[6][0] - (-C[6][2]));
    } else if (c > 1 && c < 7) {
        X[2][c] = (C[2][c-2] - (-C[6][c+1]));
    } else {
        X[2][c] = (C[2][c-2] - C[2][7]);
    }
    if (c < 3) {
        X[3][c] = (C[5][2-c] - (-C[5][2+c]));
    } else if (c >= 3 && c < 6) {
        X[3][c] = (C[3][c-3] - (-C[5][c+2]));
    } else {
        X[3][c] = (C[3][c-3] - C[3][13-c]);
    }
    if (c < 4) {
        X[4][c] = (C[4][4-c] - (-C[4][4+c]));
    } else if (c == 4) {
        X[4][c] = C[4][c-4] ;
    } else {
        X[4][c] = (C[4][c-4] - C[4][12-c]);
    }
    if (c < 3) {
        X[5][c] = (C[3][5-c] - (-C[3][c+5]));
    } else if (c >= 3 && c < 6) {
        X[5][c] = (C[3][5-c] - C[5][10-c]);
    } else {
        X[5][c] = (C[5][c-6] - C[5][10-c]);
    }
    if (c == 1) {
```

```
            X[6][c] = (C[2][6-c] - (-C[2][7]));
        } else if (c > 1 && c < 7) {
            X[6][c] = (C[2][6-c] - C[6][9-c]);
        } else {
            X[6][c] = (C[6][0] - C[6][2]);
        }
        X[7][c] = (C[1][7-c] - C[7][8-c]);
    }
```

Listing A.4: Fourth stage of MPTDCT algorithm

Bibliography

[1] IEEE Standard Specifications for the Implementations of 8x8 Inverse Discrete Cosine Transform, December 1990.

[2] N. Ahmed, T. Natarajan, and K. R. Rao. Discrete Cosine Transform. *IEEE Trans. Computer*, 23(1):90–93, January 1974.

[3] James F. Blinn. What's the Deal with the DCT ? *IEEE Computer Graphics and Applications*, 13(4):78–83, July/August 1993.

[4] Wen-Hsiung Chen, C. Harrison Smith, and S. C. Fralick. A Fast Computational Algorithm for the Discrete Cosine Transform. *IEEE Transactions On Communications*, COM-25(9):1004–1009, September 1977.

[5] Nam Ik Cho and Sang Uk Lee. Fast algorithm and implementation of 2-D discrete cosine transform. *IEEE Trans. on Circuits and Systems*, 38(3):297–306, March 1991.

[6] C.M.Rader. Discrete Fourier transforms when the number of data samples is prime. In *Proceedings of the IEEE*, volume 56, June 1968.

[7] Forman Aggoun De. Quantisation Strategies For 3d-Dct-Based Compression Of Full Parallax 3d Images.

[8] Chris Dick. Computing Multidimensional DFTs Using Xilinx FPGAs. In *The 8th International Conference on Signal Processing Applications and Technology, Toronto Canada*, September 1998.

[9] Chris Dick. Minimum multiplicative complexity implementation of the 2-D DCT using Xilinx FPGAs. In *Configurable Computing: Technology and Applications, Proc. SPIE 3526, Bellingham, WA*, pages 190–201, November 1998.

[10] James R. Driscoll, Dennis M. Healy Jr., and Daniel N. Rockmore. Fast discrete polynomial transforms with applications to data analysis for distance transitive graphs. *SIAM J. Comput.*, 26(4):1066–1099, 1997.

[11] Dan E. Dudgeon, Russell M. Mersereau, and Russell M. Merser. *Multidimensional Digital Signal Processing.* Prentice Hall, 1995.

[12] Pierre Duhamel and C. Guillemot. Polynomial Transform Computation of the 2-D DCT. In *Proceedings IEEE International Conference Acoustics, Speech and Signal Processing*, pages 1515–1518, April 1990.

[13] Pierre Duhamel and H. H'Mida. New 2^n DCT Algorithms suitable for VLSI Implementation. In *Proceedings IEEE International Conference Acoustics, Speech and Signal Processing ICASSP-87, Dallas*, page 1805, April 1987.

[14] E. Feig and S. Winograd. On the multiplicative complexity of discrete cosine transform. *IEEE Trans. Inf. Theory*, 38(4):1387–1391, July 1992.

[15] Ephraim Feig and Shmuel Winograd. Fast algorithms for the discrete cosine transform. *IEEE Trans. on Signal Processing*, 40(9).

[16] Alan Goluban. Prikaz volumnih objekata korištenjem njihovog opisa u frekvencijskoj domeni. Master's thesis, Faculty of Electrical Engineering and Computing, Zagreb, 1998.

[17] James H. and C. McClellan Rader. *Number Theory in Digital Signal Processing*. Prentice Hall, 1979.

[18] Israel Koren. *Computer Arithmetic Algorithms*. A K Peters, Ltd., second edition, 2002.

[19] Christoph Loeffler, Adriaan Ligtenberg, and George S. Moschytz. Practical Fast 1-D DCT Algorithms with 11 Multiplications. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages 988–991, 1989.

[20] Yoshitaka Morikawa, Hiroshi Hamada, and Nobumoto Yamane. A Fast Algorithm for the Cosine Transform Based on Successive Order Reduction of the Chebyshev Polynomial. *Electronics and Communications in Japan, Part 1*, 69(3):45–54, 1986.

[21] Madihally J. Narasimha and Allen M. Peterson. On the Computation of the Discrete Cosine Transform. *IEEE Transaction on Communication*, COM-26(6):934–936, June 1978.

[22] Henri J. Nussbaumer. Digital filtering using polynomial transforms. *Electron. Lett.*, 13:386–387, June 1977.

[23] Henri J. Nussbaumer. Fast polynomial transform algorithms for digital convolution. *IEEE Trans. on ASSP*, 28(2):205–215, April 1980.

[24] Henri J. Nussbaumer. Fast polynomial transform computation of the 2-D DCT. In *International Conference on Signal Processing*, pages 276–283, 1981.

[25] Henri J. Nussbaumer. *Fast Fourier Transform and Convolution Algorithms*. Springer Verlag, second edition, 1982.

[26] Henri J. Nussbaumer and Philippe Quandalle. Fast computation of discrete Fourier transforms. *IEEE Trans. on ASSP*, 27(2):169–181, April 1979.

[27] Alexander D. Poularikas. *The Transforms and Applications Handbook*. CRC Press, second edition, 2000.

[28] Jacques Prado and Pierre Duhamel. A polynomial-transform based computation of the 2-D DCT with minimum multiplicative complexity. In *Proceedings IEEE International Conference Acoustics, Speech and Signal Processing*, pages 1347–1350, April 1996.

[29] John G. Proakis and Dimitris G. Manolakis. *Digital Signal Processing: Principles, Algorithms and Applications*. Prentice Hall, third edition, 1996.

[30] Ming-Ting Sun, Ting-Chung Chen, and Albert M. Gottlieb. VLSI Implementation of a 16 X 16 Discrete Cosine Transform. *IEEE Transactions on Circuits and Systems*, 36(4):610–617, April 1989.

[31] S. Uramoto, Y. Inoue, A. Takabatke, J. Takeda, Y. Yamashita, and M. Toshimoto. A 100 MHz 2-D Discrete Cosine Transform Core Processor. *IEEE Journal of Solid State Circuits*, 27(4):492–498, April 1992.

[32] Martin Vetterli and Henri J. Nussbaumer. Simple FFT and DCT algorithms with reduced number of operations. *Signal Processing*, 6:267–278, 1984.

[33] S. White. Applications of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review. *IEEE ASSP Magazine*, pages 4–19, July 1989.

[34] Hong Ren Wu and Zhihong Man. Comments on "Fast algorithms and implementation of 2-D discrete cosine transform". *Circuits and Systems for Video Technology, IEEE Transactions on Volume*, 8(2):128–129, April 1998.

[35] Yinghui Wu, Xin Guan, Mohan S. Kankanhalli, and Zhiyong Huang. Robust Invisible Watermarking of Volume Data Using the 3D DCT. In *Computer Graphics International 2001 (CGI'01), Hong Kong, China, Proceedings*. IEEE Computer Society, July 2001.

[36] Xilinx. *The Programmable Logic Data Book*, 1999.

[37] P. P. N. Yang, M. J. Narasimha, and B. G. Lee. A prime factor decomposition algorithm for the computation of discrete cosine transform. *International Conference on Computers, Systems & Signal Processing, Bangalore, India*, December 1984.

[38] P. Yip and Kamisetty Ramamohan Rao. *Discrete Cosine Transform: Algorithms, Advantages, and Applications*. Academic Press, 1990.

[39] Sungwook Yu and Earl E. Swartzlander. DCT Implementation with Distributed Arithmetic. *IEEE Transactions on Computers*, 50(9):985–991, September 2001.

[40] Yonghong Zeng, Guoan Bi, and A. R. Leyman. New polynomial transform algorithm for multidimensional DCT. *IEEE Trans. Signal Processing*, 48(10):2814–2821, 2000.

[41] Feng Zhou and P. Kornerup. A High Speed DCT/IDCT Using a Pipelined CORDIC Algorithm. In *Proc. 12th IEEE Symposium on Computer Arithmetic*. IEEE Press, July 1995.