High Speed DCT/IDCT Using a Pipelined CORDIC Algorithm

Feng Zhou* and Peter Kornerup†
Dept. of Mathematics and Computer Science
Odense University
Campusvej 55, DK−5230 Odense M, Denmark

Abstract

This paper describes DCT (IDCT) computations using the CORDIC algorithm. By rewriting the DCT, for a 1×8 DCT only 6 CORDIC computations are needed, whereas a 1×16 DCT requires 22 CORDIC computations. But these can all be pipelined through a single CORDIC unit, so 16×16 DCT's becomes feasible for HDTV compression. Only some simple adders, registers and a more complicated carry look-ahead adder are needed, and the computing speed can be very high. Limited only by the delay of a carry look-ahead adder, the delay time of the pipelined structure is 2−10ns and the data rate is 100−500MHz for an 8×8 DCT/IDCT and 72.2−366.6MHz for a 16×16 DCT/IDCT when using two units.

1 Introduction

The discrete cosine transform (DCT) is being widely used in image compression. It has already been introduced into international standards, such as recommendation H.261[1], JPEG[2], MPEG[3]. Recently the DCT has become one of the main components of the compression techniques in HDTV video coding[4]. Many DCT chips have been produced [5]-[9]. Because the sampling rate is very high in HDTV, it is necessary to solve the computing speed problem of the DCT and IDCT[10] , [11]. For HDTV pictures, the correlation among the picture elements is far stronger than that of common video pictures, so 16×16 DCT/IDCT is more suitable for HDTV video coding.

Ahmed et al.[12] first introduced DCT and computed the N point DCT/IDCT using a $2N$ point FFT in 1974. Since then many fast algorithms for DCT/IDCT have been proposed[13]-[17]. In order to accelerate the DCT/IDCT computation, some algorithm used the FFT, some used FHT (fast Hartley transform), some directly decomposed the DCT itself. Their common aim is to reduce the number of multipliers and adders using a butterfly structure and to obtain $O(NlogN)$ running time. Another approach[16],[17] is to convert the DCT/IDCT to a skew-circle convolution. The DCT/IDCT can be computed using distributed arithmetics[18],[21] which uses memories to replace multipliers which are required in any butterfly type of implementation of the DCT/IDCT. Several other attempts regarding VLSI realisation for 2-D DCT for HDTV coding have been reported. Miyasaki et al.[11] presented an 8×8 DCT/IDCT processor for HDTV using an extended version of DSP Silicon compiler to which a matrix-vector product module composed of multiply-accumulations is added. The processor worked at a 50MHz data rate.

This paper presents a high speed DCT using the CORDIC algorithm. The CORDIC (Coordinate Rotation Digital Computer) is an arithmetic technique developed by Volder[19] to solve trigonometric problems that arise in navigation applications. Walther[20] later developed it into a unified algorithm to compute a variety of transcendental functions. The reason that the CORDIC algorithm now appears in many applications is that it uses only primitive operations such as shifts and additions to implement relatively complex functions. Using the CORDIC algorithm, we can simultaneously complete four coefficient multiplications including the computation of the trigonometric coefficients. In 1974 Despain[22] suggested the use of CORDIC rotation for the Fourier transform. In 1990 Duh and Wu[23],[24] reported architectures for DCT based on index partition and the use of multiple, iterated CORDIC structures, requiring 108 CORDIC computations for the 16-point DCT. In 1995 Hu and Wu[25] proposed a systolic structure of $N(N+2)/4$ cells, each performing a complex multiplication using a pipelined CORDIC unit. Here we use a fairly straightforward approach where the complexity of a $1\times N$-DCT is still $O(N^2)$, but with small constants. Exploiting some symmetries in the DCT/IDCT definitions, we find that only 6 CORDIC computations are needed for a 1×8 DCT/IDCT, and 22 CORDIC computations for a 1×16 DCT/IDCT. A CORDIC computation can be realised using a pipelined structure, which only uses some simple constant time adders and registers. We rewrite the DCT/IDCT computation into a structure where butterfly computation are either very simple add/subtract structures (no multiplications), or more complex butterflies corresponding to rotations which can be realised by the CORDIC algorithm. However, the structure is such that the result of a complex butterfly is never used as input for another complex butterfly operation. This implies that the latency of the complex butterfly only affects the

*On leave from Dept. of Information and Electronic Engineering at Zhejiang university in P.R.CHINA, supported by the Bao Yu-kong and Pao Zhao-long Scholarship.

†Supported by the Danish Research Councils, grant number 5.21.08.02

total latency of the DCT/IDCT. A simple pipeline implementing this operation can then be kept busy irrespective of the granularity of the pipe, and can thus reach a high throughput.
Since the CORDIC pipeline performs the work of four multipliers a comparison between such alternative implementations is carried out. It is found that the combinational circuitry needed is about the same, but the multiplier approach requires more space for constants, and more latches when deeply pipelined.
Section 2 briefly introduces the CORDIC algorithm. Then Section 3 and 4 explains how to perform DCT (IDCT) computations using the CORDIC algorithm. Section 5 discusses the pipelined DCT/IDCT using the pipelined CORDIC structure. Finally Section 6 concludes on 2D DCT/IDCT.

2 The CORDIC Algorithm

Consider the problem of rotating a vector (R, β) through an angle θ. The original vector is expressed in terms of its rectangular components X and Y. After rotating, the components X' and Y' of the rotated vector are

$$X' = Xcos\theta - Ysin\theta = (X - Ytan\theta)cos\theta$$
$$Y' = Ycos\theta + Xsin\theta = (Y + Xtan\theta)cos\theta.$$

The CORDIC algorithm is an iterative procedure in which each step rotates the vector in one or the other direction through an angle of magnitude $\alpha_j = atan2^{-j}$, that is:

$$\begin{aligned} X_{j+1} &= X_j - Y_j \tan\alpha_j = X_j - \delta_j Y_j \times 2^{-j} \quad (1)\\ Y_{j+1} &= Y_j + X_j \tan\alpha_j = Y_j + \delta_j X_j \times 2^{-j} \quad (2)\\ \theta_{j+1} &= \theta_j + \delta_j \alpha_j, \qquad j = 0, 1, 2, \cdots, \end{aligned}$$

where X_0, Y_0 are the original components X and Y, $\theta_0 = 0$, and $\delta_j \in \{-1, +1\}$. The direction δ_j for each α_j is chosen as $+1$ if $\theta - \theta_{j-1} > 0$ and -1 if $\theta - \theta_{j-1} < 0$. Continuation of the above iterations until θ_j is suitably close to θ therefore produces X and Y components that are K_c times as large as the true X' and Y', where K_c is given by $K_c \approx 1.646760255\cdots$ which is Volder's value through j=24. According to the above iterative algorithm, a block diagram of the logic structure for CORDIC is shown in Figure 1. Note that when using the CORDIC algorithm, every computation can use the same number of iterations, so that a common value of the factor K_c can be used.
In many situations, we want to compute X' and Y'

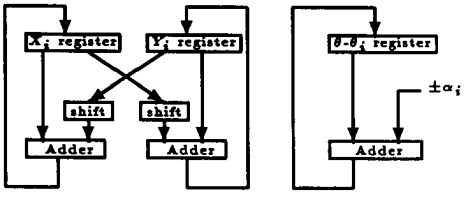


Figure 1: A block diagram of the logic structure for CORDIC

for some given angle. When the angle θ is known in advance, the signs δ_j can be computed in advance, since $\theta = \sum_j (-1)^{\delta_j}\alpha_j$. Using the above iteration structure, the computation time is determined by the cycle time and the the number of iteration steps. To accelerate the data rate, we can use a pipelined CORDIC structure as shown in Figure 2.

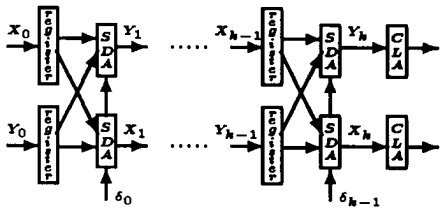


Figure 2: A pipelined CORDIC structure

Observe that in each stage of the pipeline the amount of shift needed is now constant and can be implemented by a wiring. Assuming the delay time between two steps is τ and the number of iteration steps is p, then $p\tau$ is the latency. It is sufficient to use simple (redundant) carry-save or borrow-save adders (SDA) internally in the pipeline, only at the end of the computation the results then have to be converted to non-redundant using more complex carry-look-ahead adders (CLA). The speed of these CLA adders will then determine the delay time τ. Using suitable and fast technology for these adders, τ is 2−10 ns, and data can be processed at the rate $1/\tau = 100 - 500MHz$.

3 1-D DCT Using CORDIC

To rewrite the discrete cosine transform we need to observe some trigonometric identities. For $cos\frac{2i+1}{2N}u\pi$ and $sin\frac{2i+1}{2N}u\pi$, let $u = N - u'$, we have

$$cos\frac{2i+1}{2N}(N-u')\pi = (-1)^i sin\frac{2i+1}{2N}u'\pi \qquad (3)$$

$$sin\frac{2i+1}{2N}(N-u')\pi = (-1)^i cos\frac{2i+1}{2N}u'\pi. \qquad (4)$$

If $i = N/K - 1 - i', K = 1, 2, 4, 8, 16, \cdots$, when $u = 0, K, 2K, 3K, \cdots$, we have

$$cos\frac{2(\frac{N}{K}-1-i')+1}{2N}u\pi = (-1)^{\frac{u}{K}} cos\frac{2i'+1}{2N}u\pi \qquad (5)$$

and when $u = K/2, 3K/2, 5K/2, \cdots, K \geq 2$ we have

$$cos\frac{2(\frac{K}{N}-1-i')+1}{2N}u\pi = (-1)^{\frac{u-K/2}{K}} sin\frac{2i'+1}{2N}u\pi \quad (6)$$

$$sin\frac{2(\frac{K}{N}-1-i')+1}{2N}u\pi = (-1)^{\frac{u-K/2}{K}} cos\frac{2i'+1}{2N}u\pi. \quad (7)$$

Now we analyze the one-dimensional DCT. The cosine transform is defined as follows

$$F(u) = \sqrt{\frac{2}{N}} \sum_{i=0}^{N-1} C(u) cos(\frac{2i+1}{2N}u\pi) f(i),$$
$$u = 0, 1, 2, \cdots, N-1,$$

where $C(0) = 1/\sqrt{2}, C(u) = 1, u = 1, 2, \cdots, N-1$. In the following we will assume that N is a power of 2, $N = 2^n$. Because $\sqrt{2/N}$ is a constant, we presently do not consider it for simplicity. After removing $\sqrt{2/N}$ the above formula may be divided into two parts, using formula (5) for the second part ($K = 1$), then

$$\begin{aligned} F(u) &= \sum_{i=0}^{N/2-1} C(u)cos(\frac{2i+1}{2N}u\pi)f(i) \\ &\quad + \sum_{i=N/2}^{N-1} C(u)cos(\frac{2i+1}{2N}u\pi)f(i) \\ &= \sum_{i=0}^{N/2-1} C(u)cos(\frac{2i+1}{2N}u\pi) \\ &\quad \times [f(i) + (-1)^u f(N-1-i)]. \end{aligned} \quad (8)$$

Now we define two sets of functions $f_K^-(i)$ and $f_K^+(i)$ to express the above function. For $K = 1$, let

$$f_1^+(i) = f_1^-(i) = f(i), \qquad i = 0, 1, 2, \cdots, N-1. \quad (9)$$

and for $K = 2, 4, 8, 16, \cdots, N/2$, $i = 0, 1, 2, 3, \cdots, N - K/2$, let

$$f_K^-(i) = f_{K/2}^+(i) - f_{K/2}^+(2N/K - 1 - i) \quad (10)$$

$$f_K^+(i) = f_{K/2}^+(i) + f_{K/2}^+(2N/K - 1 - i) \quad (11)$$

so formula (8) may repeatedly be split into two parts. In general, let $K = 2, 4, 8, \cdots, N/2$, when $u = 0, K/2, K, 3K/2, 2K, \cdots, N - K/2$, we obtain

$$\begin{aligned} F(u) &= \sum_{i=0}^{\frac{N}{2K}-1} C(u)cos(\frac{2i+1}{2N}u\pi) \\ &\quad \times \{f_{K/2}^+(i) + (-1)^{\frac{2u}{K}} f_{K/2}^+(\frac{2N}{K} - 1 - i)\} \\ &\quad + \sum_{i=\frac{N}{2K}}^{\frac{N}{K}-1} C(u)cos(\frac{2i+1}{2N}u\pi) \\ &\quad \times \{[f_{\frac{K}{2}}^+(i) + (-1)^{\frac{2u}{K}} f_{\frac{K}{2}}^+(\frac{2N}{K} - 1 - i)\} \end{aligned} \quad (12)$$

Observe that the value of the right hand side does not depend on K, but it can only be used to compute the value of $F(u)$ for a set of values of u which depends on the value of K. For $u = K/2, 3K/2, 5K/2, \cdots, N - K/2, C(u) = 1$, using formula (6) and (10), the above expression reduces to

$$\begin{aligned} F(u) &= \sum_{i=0}^{\frac{N}{2K}-1} \{cos(\frac{2i+1}{2N}u\pi)f_K^-(i) + (-1)^{\frac{u-K/2}{K}} \\ &\quad \times sin(\frac{2i+1}{2N}u\pi)f_K^-(\frac{N}{K} - 1 - i)\} \end{aligned} \quad (13)$$

For $u = 0, K, 2K, \cdots, N - K$, using formulas (5) and (11), then (12) becomes

$$\begin{aligned} F(u) &= \sum_{i=0}^{\frac{N}{2K}-1} C(u)cos(\frac{2i+1}{2N}u\pi) \\ &\quad \times \{f_K^+(i) + (-1)^{\frac{u}{K}}[f_K^+(\frac{N}{K} - 1 - i)]\} \end{aligned} \quad (14)$$

which again can be split as in (12) until there is only one term left in the sum (i,e. for $K = N/2$), allowing the computation of $F(0)$ and $F(N/2)$. Note that $C(0) = 1/\sqrt{2} = cos(\pi/4) = sin(\pi/4)$, therefore

$$F(0) = cos(\tfrac{\pi}{4})f_{\frac{N}{2}}^+(0) + sin(\tfrac{\pi}{4})f_{\frac{N}{2}}^+(1). \quad (15)$$

For $u = K = N/2$ we get from (14)

$$F(\tfrac{N}{2}) = -\{cos(\tfrac{\pi}{4})f_{\frac{N}{2}}^+(1) - sin(\tfrac{\pi}{4})f_{\frac{N}{2}}^+(0)\}. \quad (16)$$

We can replace u in formula (13) with $N - u'$, using formulas (3) and (4), when $K = 2, 4, 8, \cdots, N/2$, so for $u' = K/2, 3K/2, 5K/2, \cdots, N - K/2$, we have an alternative expression:

$$\begin{aligned} F(N-u') &= \sum_{i=0}^{\frac{N}{2K}-1} (-1)^i \{sin(\frac{2i+1}{2N}u'\pi)f_K^-(i) \\ &\quad -(-1)^{\frac{u'-K/2}{K}} cos(\frac{2i+1}{2N}u'\pi) \\ &\quad \times f_K^-(\frac{N}{K} - 1 - i)\}. \end{aligned} \quad (17)$$

Because $-sin\theta = sin(-\theta)$, $cos\theta = cos(-\theta)$, the sign term $(-1)^{(u-K/2)/K}$ in formulas (13) and (17) may also be moved into the *sin* function. The angles $(-1)^{\frac{u-K/2}{K}}(2i+1)u\pi/2N$ may be thus expressed as follows

$$\begin{aligned} (-1)^{\frac{u-K/2}{K}}(2i+1)u\pi/2N &= m(i,u)\pi + \theta(i,u), \\ m(i,u) &= 0, \pm 1, \pm 2, \cdots, -\tfrac{\pi}{2} < \theta(i,u) < \tfrac{\pi}{2}, \end{aligned} \quad (18)$$

so we have

$$\begin{aligned} (-1)^{\frac{u-K/2}{K}} sin\frac{2i+1}{2N}u\pi &= sin(-1)^{\frac{u-K/2}{K}}\frac{2i+1}{2N}u\pi \\ &= (-1)^{m(i,u)}sin\theta(i,u) \\ cos\frac{2i+1}{2N}u\pi &= cos(-1)^{\frac{u-K/2}{K}}\frac{2i+1}{2N}u\pi \\ &= (-1)^{m(i,u)}cos\theta(i,u). \end{aligned}$$

For a given value of K the two expressions (13) and (17) can compute the same set of values $F(u)$ when u ranges over $u = K/2, 3K/2, 5K/2, \cdots, N - K/2$. So with u from half the range, $u =$

$K/2, 3K/2, 5K/2, \cdots, (N-K/2)$, $F(u)$ and $F(N-u)$ together then provide the complete set of values:

$$F(u) = \sum_{i=0}^{\frac{N}{2K}-1} (-1)^{m(i,u)} \{f_K^-(i) cos\theta(i,u) + f_K^-(\frac{N}{K} - 1 - i) sin\theta(i,u)\}$$

$$F(N-u) = \sum_{i=0}^{\frac{N}{2K}-1} (-1)^{i+\frac{u-K/2}{K}+m(i,u)} * \{f_K^-(\frac{N}{K} - 1 - i) cos\theta(i,u) - f_K^-(i) sin\theta(i,u)\},$$

when $u = K/2, 3K/2, \cdots, (N-K)/2$ and $K = 2, 4, 8, \cdots, N/2$, $F(0)$ and $F(N/2)$ are calculated according to the formulas (15), (16). Comparing the above formulas with the CORDIC algorithm, it is found that the $N = 2^n$ DCT $F(u)$ ($u = 0, 1, 2, 3, \cdots, N-1$) can be computed using the CORDIC algorithm,

$$F(u) = \sqrt{\frac{2}{N}} \frac{1}{K_c} \sum_{i=0}^{N/2K-1} Y(i,u)$$

$$F(N-u) = \sqrt{\frac{2}{N}} \frac{1}{K_c} \sum_{i=0}^{N/2K-1} X(i,u)$$

for $u = K/2, 3K/2, 5K/2, \cdots, (N-K)/2, K = 2, 4, 8, \cdots, N/2$, and

$$F(0) = \sqrt{\frac{2}{N}} \frac{1}{K_c} Y(0,0), \qquad F(\frac{N}{2}) = \sqrt{\frac{2}{N}} \frac{1}{K_c} X(0,0).$$

Here the values

$$X(i,u) = (-1)^{i+\frac{u-K/2}{K}+m(i,u)} X_q, \qquad X(0,0) = -X_q,$$
$$Y(i,u) = (-1)^{m(i,u)} Y_q, \qquad Y(0,0) = Y_q$$

are computed from results of CORDIC computations yielding the values X_q and Y_q after q steps, using suitable initial values (for $K = 0$, $X_0 = f_{N/2}^+(1)$, $Y_0 = f_{N/2}^+(0)$, and for $K = 2, 4, 8, \cdots, N/2$, $X_0 = f_K^-(N/K - 1 - i)$, $Y_0 = f_K^-(i)$), and rotation through angles $\theta(i,u)$.

Now let us see how many CORDIC computations are necessary in this algorithm for the DCT. For some K, $u = K/2, 3K/2, 5K/2, \cdots, (N-K)/2$, i changes from 0 to $N/2K-1$, hence there are $N \times N/(2K \times 2K)$ CORDIC computations. When K changes from 2 to $N/2$, the total number of CORDIC computations (including computing $F(0), F(N/2)$) is $(N^2+8)/12$. For $N = 8$, only 6 CORDIC computations are needed, whereas there are 22 CORDIC computations for $N = 16$. The functions $f_K^-(i)$ and $f_K^+(i)$ can easily be obtained from formulas (9–11). A block diagram of the DCT computation using the CORDIC algorithm for $N = 16$ is shown in Figure 3.

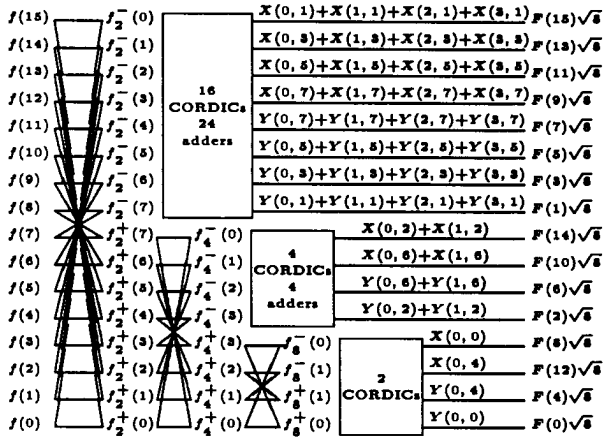


Figure 3: A block diagram of the DCT using CORDIC

4 1-D IDCT Using CORDIC

The one-dimensional even inverse discrete cosine transform (IDCT) is defined by

$$f(i) = \sqrt{\frac{2}{N}} \sum_{u=0}^{N-1} C(u) cos(\frac{2i+1}{2N} u\pi) F(u), \qquad i = 0, 1, 2, 3, \cdots, N-1$$

$$C(0) = 1/\sqrt{2}, C(u) = 1, u = 1, 2, \cdots, N-1.$$

We will initially not consider the constant $\sqrt{2/N}$ for simplicity. The above formula is also divided into two parts, then replacing u with $N-u'$ for the second term, using formula (3), $f(i)$ becomes

$$\begin{aligned} f(i) &= \sum_{u=0}^{N/2-1} C(u) cos(\frac{2i+1}{2N} u\pi) F(u) + \sum_{u=N/2}^{N-1} C(u) cos(\frac{2i+1}{2N} u\pi) F(u) \\ &= \frac{F(0)}{\sqrt{2}} + (-1)^{\lceil \frac{i+1}{2} \rceil} \frac{F(N/2)}{\sqrt{2}} + \sum_{u=1}^{N/2-1} \{cos(\frac{2i+1}{2N} u\pi) F(u) + (-1)^i sin(\frac{2i+1}{2N} u\pi) F(N-u)\} \\ &= \sum_{u=0}^{N/2-1} g(i,u) \end{aligned} \tag{19}$$

where for $u = 1, 2, \cdots, N/2$ and all i

$$g(i,0) = cos(\frac{\pi}{4}) F(0) + (-1)^{\lceil \frac{i+1}{2} \rceil} sin(\frac{\pi}{4}) F(\frac{N}{2})$$

i	u=1	3	5	7	2	6	4	0
0	$g(0,1)$	$g(0,3)$	$g(0,5)$	$g(0,7)$	$g(0,2)$	$g(0,6)$	$g(0,4)$	$g(0,0)$
1	$g(1,1)$	$g(1,3)$	$g(1,5)$	$g(1,7)$	$g(1,2)$	$g(1,6)$	$g'(0,4)$	$-g'(0,0)$
2	$g(2,1)$	$g(2,3)$	$g(2,5)$	$g(2,7)$	$g'(1,2)$	$g'(1,6)$	$-g'(0,4)$	$-g'(0,0)$
3	$g(3,1)$	$g(3,3)$	$g(3,5)$	$g(3,7)$	$g'(0,2)$	$g'(0,6)$	$-g(0,4)$	$g(0,0)$
4	$g'(3,1)$	$g'(3,3)$	$g'(3,5)$	$g'(3,7)$	$-g'(0,2)$	$-g'(0,6)$	$-g(0,4)$	$g(0,0)$
5	$g'(2,1)$	$g'(2,3)$	$g'(2,5)$	$g'(2,7)$	$-g'(1,2)$	$-g'(1,6)$	$-g'(0,4)$	$-g'(0,0)$
6	$g'(1,1)$	$g'(1,3)$	$g'(1,5)$	$g'(1,7)$	$-g(1,2)$	$-g(1,6)$	$g'(0,4)$	$-g'(0,0)$
7	$g'(0,1)$	$g'(0,3)$	$g'(0,5)$	$g'(0,7)$	$-g(0,2)$	$-g(0,6)$	$g(0,4)$	$g(0,0)$
8	$-g'(0,1)$	$-g'(0,3)$	$-g'(0,5)$	$-g_2(0,7)$	$-g(0,2)$	$-g(0,6)$	$g(0,4)$	$g(0,0)$
9	$-g'(1,1)$	$-g'(1,3)$	$-g'(1,5)$	$-g'(1,7)$	$-g(1,2)$	$-g(1,6)$	$g'(0,4)$	$-g'(0,0)$
10	$-g'(2,1)$	$-g'(2,3)$	$-g'(2,5)$	$-g'(2,7)$	$-g'(1,2)$	$-g'(1,6)$	$-g'(0,4)$	$-g'(0,0)$
11	$-g'(3,1)$	$-g'(3,3)$	$-g'(3,5)$	$-g'(3,7)$	$-g'(0,2)$	$-g'(0,6)$	$-g(0,4)$	$g(0,0)$
12	$-g(3,1)$	$-g(3,3)$	$-g(3,5)$	$-g(3,7)$	$g'(0,2)$	$g'(0,6)$	$-g(0,4)$	$g(0,0)$
13	$-g(2,1)$	$-g(2,3)$	$-g(2,5)$	$-g(2,7)$	$g'(1,2)$	$g'(1,6)$	$-g'(0,4)$	$-g'(0,0)$
14	$-g(1,1)$	$-g(1,3)$	$-g(1,5)$	$-g(1,7)$	$g(1,2)$	$g(1,6)$	$g'(0,4)$	$-g'(0,0)$
15	$-g(0,1)$	$-g(0,3)$	$-g(0,5)$	$-g(0,7)$	$g(0,2)$	$g(0,6)$	$g(0,4)$	$g(0,0)$

Table 1: Computation Scheme for values of $g(i,u)$

$$g(i,u) = cos(\frac{2i+1}{2N}u\pi)F(u) + (-1)^i sin(\frac{2i+1}{2N}u\pi)F(N-u).$$

Similar to the DCT, let $K = 1,2,4,8,\cdots,N/2$ using formulas (6), (7), when $u = K, 2K, 3K, \cdots$, we get:

$$\begin{aligned} g(\frac{N}{K}-1-i,u) &= (-1)^{u/K}\{cos(\frac{2i+1}{2N}u\pi)F(u) \\ &\quad +(-1)^i sin(\frac{2i+1}{2N}u\pi)F(N-u)\} \\ &= (-1)^{u/K}g(i,u). \end{aligned} \quad (20)$$

When $u = K/2, 3K/2, 5K/2, \cdots$, let $K = 2,4,8,\cdots,N/2$, we get:

$$\begin{aligned} g(\frac{N}{K}-1-i,u) &= (-1)^{\frac{u-K/2}{K}}\{sin(\frac{2i+1}{2N}u\pi)F(u) \\ &\quad -(-1)^i cos(\frac{2i+1}{2N}u\pi)F(N-u)\} \\ &= g'(i,u). \end{aligned} \quad (21)$$

where we define $g'(i,u)$ as follows (note that there is a relation between u and K):

$$\begin{aligned} g'(0,0) &= sin(\frac{\pi}{4})F(0) - cos(\frac{\pi}{4})F(\frac{N}{2}) = -g(1,0) \\ g'(i,u) &= (-1)^{u-K/2)/K}\{sin(\frac{2i+1}{2N}u\pi)F(u) \\ &\quad -(-1)^i cos(\frac{2i+1}{2N}u\pi)F(N-u)\} \end{aligned}$$

Exploiting the symmetries a complete set of values $g(i,u)$ needed for the computation of $f(i)$ according to (19), using (20) and (21), can be calculated from $(N^2+8)/12$ $(g(i,u), g'(i,u))$ pairs of values. Table 1 shows the results of all $g(i,u)$ for $N = 16, u = 0,1,2,\cdots,7, i = 0,1,2,\cdots,15$. Note from the table that only 22 different pairs (g,g') are needed for the whole table.

Similar to calculating the DCT using CORDIC, we move the sign $(-1)^i$ to the *sin* function. The angle $(-1)^i(2i+1)u\pi/2N$ can be expressed by

$$\begin{aligned} (-1)^i(2i+1)u\pi/2N &= h(i,u)\pi + \beta(i,u), \\ h(i,u) &= 0, \pm1, \pm2, \cdots; -\pi/2 < \beta(i,u) < \pi/2. \end{aligned} \quad (22)$$

So we have

$$\begin{aligned} g(i,u) &= (-1)^{h(i,u)} \\ &\quad *\{cos\beta(i,u)F(u) + sin\beta(i,u)F(N-u)\} \\ g'(i,u) &= (-1)^{i+\frac{u-K/2}{K}+h(i,u)} \\ &\quad *\{cos\beta(i,u)F(N-u) - sin\beta(i,u)F(u)\} \\ g(0,0) &= cos(\pi/4)F(0) + sin(\pi/4)F(N/2) \\ g'(0,0) &= -\{cos(\pi/4)F(N/2) - sin(\pi/4)F(0)\}. \end{aligned}$$

$(g(i,u), g'(i,u))$ and $(g(0,0), g'(0,0))$ can be computed using q steps of the CORDIC algorithm,

$$\begin{aligned} g'(i,u) = (-1)^{i+\frac{u-K/2}{K}+h(i,u)}X'_q, &\quad g'(0,0) = -X'_q, \\ g(i,u) = (-1)^{h(i,u)}Y'_q, &\quad g(0,0) = Y'_q, \\ X'_{l+1} = X'_l - \delta_l Y'_l \times 2^{-l}, & \\ Y'_{l+1} = Y'_l + \delta_l X'_l \times 2^{-l}, & \\ \beta_{l+1} = \beta_l + \delta_l \alpha_l, &\quad l = 0,1,\cdots,q-1, \end{aligned}$$

where δ_l is chosen such that β_l converge to the angle $\beta(i,u)$, using initial values $X'_0 = F(N-u)$, $Y'_0 = F(u)$, for $u = 0$ $X'_0 = F(N/2)$, $Y'_0 = F(0)$.
The number of CORDIC computations for IDCT is the same as for DCT, i.e $(N^2+8)/12$. A block diagram of the even IDCT computation using the CORDIC algorithm for $N = 16$ is shown in Figure 4. Comparing the DCT and IDCT structures, we can see that the IDCT computations are the inverses of the DCT computations. The CORDIC units are the same for DCT and IDCT computations, with exception of some sign differences of the rotation angles between DCT and IDCT, which will be discussed in next Section.

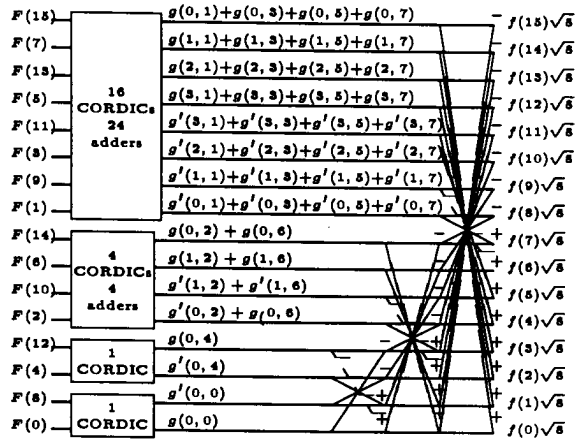


Figure 4: A block diagram of the IDCT using the CORDIC

5 DCT/IDCT Structure Using a Pipelined CORDIC

The above block diagrams of the DCT and IDCT using CORDIC would be suitable for parallel computations to speed up computations. So 22 CORDIC units would be needed for $N = 16$. However, if latency is not of particular concern, using a pipeline structure we actually only need one pipelined CORDIC unit.
The DCT or IDCT structure using CORDIC can be split into two parts: One is the butterfly additions/subtractions and another is the CORDIC computations and accumulation. For the DCT the butterfly computations are first performed, then the CORDIC and accumulations. For the IDCT the order is reversed. The CORDIC and accumulations are almost the same for the DCT and IDCT. The only difference is in the signs δ_l. These are determined by the angles $\theta(i,u)$ in DCT and $\beta(j,u)$ in IDCT. $\theta(i,u)$ and $\beta(j,u)$ can be calculated by formula (18), (22), and for $N = 16$, $\theta(i,u)/\beta(j,u)$ are tabulated as the following multiplies of $\pi/32$

i	$u=1$	$u=3$	$u=5$	$u=7$
0	1/1	-3/3	5/5	-7/7
1	3/-3	-9/-9	15/-15	11/11
2	5/5	-15/15	-7/-7	-3/3
3	7/-7	11/11	3/-3	15/15
i	$u=2$	$u=6$	$u=4$	$u=0$
0	2/2	-6/6	4/4	8/8
1	6/-6	14/14		

We can see from the above that there are some sign differences between $\theta_K(i,u)$ and $\beta(i,u)$ for some i,u. For each angle ϕ we can calculate the corresponding signs $\delta_l, l = 0,1,2,3,\cdots,q-1$ in advance such that $\phi = \sum_0^{q-1}(-1)^{\delta_l}\alpha_l$, where q is the total number of CORDIC steps. $q = 16$ is enough for 12 bit accuracy. In the circuits we make $T_l(i,u) = 0$ correspond to $\delta_l(i,u) = +1$, and $T_l(i,u) = 1$ correspond to $\delta_i(i,u) = -1$, so each angle is represented by a bit vector $T_0T_1\cdots T_{15}$. For example, $T_0T_1T_2\cdots T_{15}$=0110111010011000 corresponds to the angle $\pi/32$.
According to (18) and (22) we also need to compute "offsets" $m(i,u)$ and $h(i,u)$, which are the appropriate multiples of π to be added to the angles, and thus results in changes in the signs of the computed results. It is hence sufficient to compute the remainder modulo 2 of $m(i,u)$ and $h(i,u)$. After computing $m(i,u)$ and $h(i,u)$ for $N = 8, 16$, we found that $m(i,u) \equiv h(i,u) (mod\ 2)$. We will term the two signs of outputs of the CORDIC pipeline for the DCT/IDCT computations respectively $s1(i,u), s2(i,u)$. For $N = 16$, $s1(i,u)/s2(i,u)$ can be tabulated as follows

i	$u=1$	3	5	7	2	6	4	0
0	0/0	0/1	0/0	0/1	0/0	1/1	0/0	0/0
1	0/1	0/0	0/1	1/1	0/1	1/1		
2	0/0	0/1	1/1	1/0				
3	0/1	1/1	1/0	0/0				

The pipelined CORDIC unit for an $N = 16$ DCT and IDCT structure is shown in Figure 5. A counter counts from 0 to 21 and an encoder is used to generate 16 values of $T_0T_1T_2\cdots T_{15}$. Then T_l must be delayed by $(l-1)\tau$ to control the l'th SDA's operation (add/subtract). $s1(i,u), s2(i,u)$ also produced by the encoder controls the signs of the outputs. According to the DCT and IDCT computation, some CORDIC outputs must be accumulated. So at the end of the pipeline we use two SDA's and two registers to perform accumulation. For no accumulation or at the beginning of accumulations two registers must be cleared.

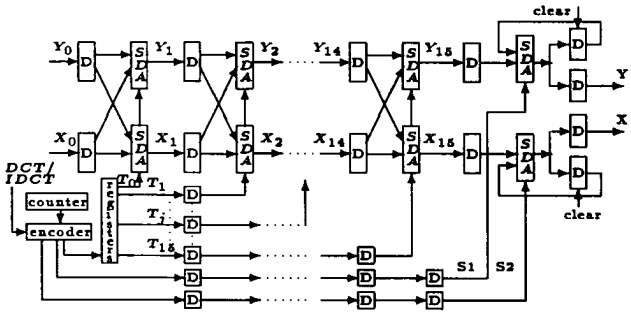
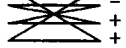


Figure 5: A pipelined CORDIC unit for the DCT/IDCT

The simple butterfly operations of the DCT and IDCT can all be performed by iterating using feedback through the smallest (simplest) butterfly operation:



as found in Figures 3 and 4. For the DCT the results of the two additions continue performing the butterfly computations, the results of the two subtractions are sent to the CORDIC unit which outputs the DCT results. For the IDCT two of 16 inputs are first sent to the CORDIC unit, then the outputs of the CORDIC unit are sent to the butterfly computation unit.
Combining the pipelined DCT and IDCT, we can design a chip which performs the 1-D DCT or the 1-D IDCT computation using only one CORDIC unit, one smallest butterfly computation, one multiplier (multiplying by the coefficient $\sqrt{2/N}/K_c$), some registers and MUXs. The whole structure for 16 DCT/IDCT is shown in Figure 6. For DCT computations $f(i)$ are first input to $D0$-$D15$ at the rate R. After receiving

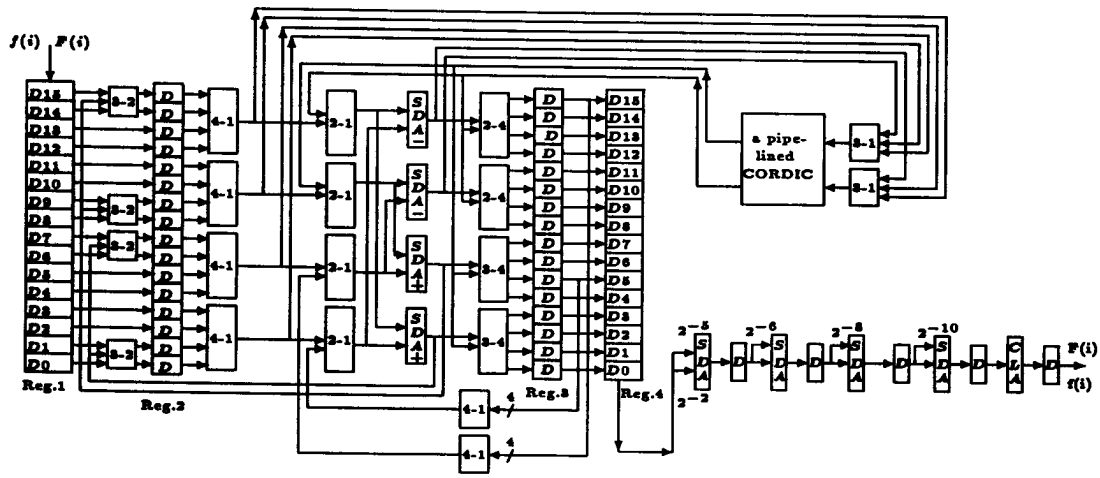


Figure 6: A DCT/IDCT pipeline structure using CORDIC

all 16 $f(i)$, the values are sent simultaneously to Reg. 2. When a DCT computation is being performed, outputs of the CORDIC unit are sent to Reg.3. When all $F'(u)$ have been obtained, all $F'(u)$ in Reg. 3. are simultaneously sent to Reg. 4. IDCT computations are similar to the DCT computations, the difference is the order of the CORDIC and butterfly computations. The feedback for DCT is between inputs of Reg. 2 and outputs of a butterfly computation, the feedback for IDCT is between inputs of a butterfly computation and outputs of Reg. 3. Finally the output has to be multiplied with the constant $\sqrt{2/N}/K_c$ and converted into non redundant form. This multiplication can be realized by a simple structure as shown in Figure 8, by noting that for $N = 16$:

$$\sqrt{\frac{2}{N}}\frac{1}{K_c} \approx \frac{1}{4}(1-\frac{1}{8})(1-\frac{1}{64})(1-\frac{1}{256})(1+\frac{1}{1024})$$

is a very good approximation. When used in 2D application only a single final multiplication is needed, and here another approximation can similarly be used:

$$\frac{2}{N}\frac{1}{K_c^2} \approx \frac{1}{32}(1+\frac{1}{2})(1-\frac{1}{64})(1-\frac{1}{512})(1+\frac{1}{1024}).$$

The whole structure is a pipeline. Data is input and output at a rate R. The time that it takes to input 16 data to Reg. 1 is $16/R$. After data in Reg. 1 simultaneously has been sent to Reg. 2, the DCT/IDCT computations are performed. Assuming that the step delay time of the pipelined structure is τ, the time that data has to stay in Reg. 2 for one $N = 16$ DCT/IDCT computations is 22τ (because there are 22 pairs of data to be sent through the CORDIC unit). To match the above, $16/R$ must be equal to or larger than 22τ. When τ is $2-10ns$, the maximal rate R is 363.36MHz to 72.72MHz. For 1-D $N = 8$ DCT, there are only 6 CORDIC computations to perform, so the maximal rate R is 500MHz–100MHz. For $N = 16$, the total latency for the entire DCT/IDCT is (16+1+4)/R+(22+p+3)τ (16 for inputing data to Reg. 1, one for outputting data from Reg. 3, four for multiplication, $22+p$ for the CORDIC pipeline and 3 for accumulation, butterfly and transfer to Reg. 3.

Finally let us compare this CORDIC-based implementation with a traditional structure based on a multiply-accumulate approach, where the trigonometric coefficients are found by table look-up. First notice that the CORDIC pipeline of Figure 6 in this context performs the work of 4 multipliers, together with suitable ROM's for the tables. For an equivalent implementation we will assume that the multipliers are also implemented as pipelined arrays. Since the ROM values are in non-redundant form (and can be assumed to be positive), and the other operand is in redundant (signed-digit) representation, it is simplest to use the ROM values as the multiplicand. Partial products are thus formed using digits -1,0,1 from the redundant multiplier, and accumulated using rows of full adders (3-to-2 reduction).

To produce the high-order 16 digits of a 16×16 multiplication in an accuracy comparable to the CORDIC pipeline, it is sufficient with an array of adders where partial products are formed and accumulated, starting from the most significant end of the multiplier. Since the adders of the CORDIC pipeline are signed-digit (4-to-2, two levels of full adders), the number of adders needed for four multipliers is the same as in the CORDIC pipeline. The add/subtract control is slightly more complicated for the multiplier $(-1, 0, 1)$ compared to the $(-1, 1)$ control of the CORDIC structure. Also the four multipliers need 2 ROM's for storing the *sine* and *cosine* values, as opposed to a single ROM or PLA for the encoded angles used in the CORDIC unit.

But there is a significant difference in the amount of latches needed for buffering at the pipeline stages. If we assume that the CORDIC structure is pipelined in 16 stages, then the multipliers need only be pipelined in 8 stages to run at the same frequency. Buffering the accumulated values in four multipliers then requires the same number of latches as the CORDIC pipeline of Figure 5. But buffering the redundant multiplier

and the multiplicand at each stage in four multipliers requires about 6 times as many latches (approx. 900 versus 150 for $N = 16$) as the buffering of the encoded angles of the CORDIC pipeline. On the other hand the CORDIC approach requires a final multiplication by the constant $\sqrt{2/N}K_c$, whereas such constants can easily be incorporated in the ROM tables in the multiplier approach. However, the suggested simple constant multiplication requires less hardware than the extra latches needed in the latter approach, and it may thus be concluded that the CORDIC method seem to be marginally more economical than a more traditional multiplier based implementation.
An interesting alternative to explore might be to directly implement the DCT/IDCT computational scheme of Figure 3 and 4 using parallel on-line CORDIC unit, in particular for $N = 8$ where only 6 such unit are needed together with 16 on-line adders.

6 Summary and Conclusions

Exploiting some symmetries in the DCT/IDCT computation, the number of multiplications needed for DCT/IDCT is reasonably small for small N values, and the CORDIC algorithm (which simultaneously completes four multiplications) can be used for the rewritten expression. A pipeline can then be used to obtain high processing speed. For the 2-D DCT/IDCT, we can apply two one-dimensional pipeline DCT/IDCT structures to realize the two dimensional DCT/IDCT. The data rate of the DCT/IDCT depends on one carry look-ahead adder (CLA), defining the pipelined step delay time. At present integration technology, these pipeline steps can be realized in about $2-10ns$. If a 2-D DCT/IDCT chip uses two of the above 1-D DCT/IDCT structures, the rate R is the same as that of a 1-D DCT/IDCT. If it uses only a single 1-D DCT/IDCT structure, the rate R is halved.

References

[1] CCITT Recommendation H.261, 1990.

[2] ISO/IEC JTCI/SC29/WG10, JPEG Committee Draft CD10918, 1991.

[3] ISO/IEC JTCI/SC29/WG11, MPEG Committee Draft CD11172, 1991.

[4] Woo Paik, " Digicipher All Digital, Channel Compatible, HDTV Broadcast System," *IEEE Trans. on Broadcasting*, Vol.36, No.4, pp.245-254, Dec. 1990.

[5] M. Vetterli, et al., "A Discrete Fourier-Cosine Transform Chip," *IEEE Journal on Selected Areas in Communications*, Vol.SAC-4, pp.49-61,1986.

[6] Jc. Calach, et al., "TCD: a 27 MHz 8×8 Discrete Cosine Transform Chip," *Proc. ICASSP'89*, pp.2429-2432, 1989.

[7] IMS A121 2-D Discrete Cosine Transform Video Processor, INMOS, Mar. 1989.

[8] STV3208 8×8 Discrete Cosine Transform, *SGS-THOMSON Microelectronics*, May 1989.

[9] L64730 Discrete Cosine Transform Processor, LSI Logic, July 1990.

[10] S.Uramoto, et al., "A 100MHz 2-D Discrete Cosine Transform Core Processor," *Proc. Symposium on VISL Circuit*, pp.35-36, 1991.

[11] Miyazaki, et al., "DCT/IDCT Processor for HDTV Developed with DSP Silicon Compiler," *Journal of VISL Signal Processing*, 5, pp.151-158, 1993.

[12] N.Ahmed, et al., "Discrete Cosine Transform," *IEEE Trans. on Computers*, Vol.C-23, pp.90-93, 1974.

[13] M.J. Narasimha and A.M. Peterson, "On the Computation of the Discrete Cosine Transform," *IEEE Trans. on Comm.*, Vol.COM-26, No.6, pp.934-936, June 1978.

[14] R.N. Bracewell, "Fast Computation of Discrete Cosine Transform through Fast Hartley Transform," *Electron Lett.*, Vol.22, No.7, pp.352-353, Mar. 1986.

[15] H.V. Sorensen, et al., "Real-Valued Fast Fourier Transform Algorithms," *IEEE Trans. Acoust., Speech, Signal processing*, Vol.ASSP-35, No.6, pp.849-863, June 1987.

[16] J. Makhoul, " A Fast Cosine Transform in One and Two Dimension," *IEEE Trans. Acoust., Speech, Signal processing*, Vol.ASSP-28, No.1, pp.27-34, June 1980.

[17] A. Leger, et al., "Distributed Arithmetic Implementation of the DCT for Real Time Photovideo on ISDN," *Proc. SPIE Int. Soc. Opt. Eng.*, Vol.804, pp.364-370, 1987.

[18] Weiping Li, "A New Algorithm to Compute the DCT and its Inverse," *IEEE Trans. on Signal Processing*, Vol.39, No.6, pp.1305-1313, June 1991.

[19] J. Volder, "The CORDIC Trigonometric Computing technique," *IRE Trans. Electron. Computer*, Vol.EC-8, No.3, pp.330-334, Sept. 1959.

[20] J.S. Walther, "A Unified Algorithm for Elementary Functions," in *Proc. AFIPS Spring Joint Computer Conf.*, pp.379-385, 1971.

[21] A. Peled and B. Liu, " A New Hardware Realization of Digital Filters," *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-22, No. 6, Dec. 1974.

[22] A.M. Despain, "Fourier Transform Computers Using CORDIC Iterations," *IEEE Trans. on Computer*, Vol. C-23, pp.993-1001, 1974.

[23] Ja-Ling Wu and Wei-Jou Duh, "Novel Concurrent Architecture to Implement the Discrete Cosine Transform Based on Index Partitions," *INT.J.ELECTRONICS*, Vol.68, No.2, pp.165-174, 1990.

[24] Wei-Jou Duh and Ja-Ling Wu, "Constant-Rotation DCT Architecture Based on CORDIC Techniques," *INT.J.ELECTRONICS*, Vol.69, No.5, pp.583-593, 1990.

[25] Yu Wu Hen Hu and Zhenyang Wu, " An Efficient CORDIC Array Structure for the Implementation of Discrete Cosine and Transform," *IEEE Trans. Signal Processing*, Vol.43 No.1, pp.331-336, Jan. 1995.