Proc. of SPIE's Photonics East'98 *Configurable Computing: Technology and Applications*, Boston, MA, USA, pp. 190-201, November 1-6 1998.

Minimum Multiplicative Complexity Implementation of the 2-D DCT using Xilinx FPGAs

Chris Dick
chrisd@xilinx.com

Xilinx Inc.
2100 Logic Drive
San Jose, CA 95124, USA

ABSTRACT

This paper investigates two options for the field programmable gate array (FPGA) implementation of a very high-performance 2-D discrete cosine transform (DCT) processor for real-time applications. The first architecture exploits the transform separability and uses a row-column decomposition. The row and column processors are realized using distributed arithmetic (DA) techniques. The second approach uses a naturally 2-D method based on polynomial transforms. The paper provides an overview of the DCT calculation using DA methods and describes the FPGA implementation. A tutorial overview of a computationally efficient method for computing 2-D DCTs using polynomial transforms is presented. A detailed analysis of the datapath for this approach using an 8×8 data-set is given. Comparisons are made that show the polynomial transform approach to require 67% of the logic resources of a DA processor for equal throughputs. The polynomial transform approach is also shown to scale better with increasing block size than the DA approach.

Keywords: 2-D DCT, FPGA, polynomial transforms, real-time video

1 INTRODUCTION

The 2-D DCT is at the heart of a vast majority of low-rate codecs for video compression. For example, the DCT is an integral part of MPEG[1] and H.261.[2] Its time efficient computation is therefore of great interest in the current technological climate where there is explosive growth in the communications and multimedia area. An integral aspect of this confluence of computing, communications and multimedia are images and real-time video. The efficient storage and transmission of video signals is a key component to the success of multimedia-based systems.

The efficient computation of both the 1-D and 2-D DCT has received much attention in the research community.[8–10] There are several strategies available to the DSP systems engineer for realizing a DCT-based transform coder. One option is to use a software programmable DSP processor like the TMS320C5x.[11] This brings high flexibility to a design while sacrificing performance. At the other end of the implementation spectrum is an ASIC solution, providing high performance at the expense of system flexibility. A third option is a field programmable gate array based approach. This alternative has the potential to bring high-performance as well as flexibility to a design.

Several investigators have explored approaches for implementing DCTs using field programmable logic. In[3] a 1-D DCT based on the algorithm in[4] is described for the Xilinx XC6200[5] fine-grain architecture FPGA. In[6] a distributed arithmetic (DA)[7] approach for computing the 1-D DCT is described.

The conventional technique for realizing a 2-D DCT is to exploit the transform separability and decompose the problem into a sequence of 1-D sub-problems - the row-column approach. For high-resolution $N \times N$-pixel, $N \geq 1024$, color images, a parallel architecture that incorporates row and column processors, as well as a matrix transposition engine must be used to accommodate real-time data rates. Using the 1-D DA DCT reported in,[6] the logic to realize each of the row and column processors alone is 264 *configurable logic blocks* CLBs),[12] and this is only for one of the three primary colors, and the matrix transposition has yet to be accounted for.

In this paper a new approach for implementing 2-D DCTs using Xilinx 4000 series FPGAs is described. The technique is based on a little known result based on arithmetic in polynomial fields.[16,13] This algorithm has the property of only requiring N 1-D DCTs of size N for performing an $N \times N$ 2-D transform. Thus, the multiplicative complexity is halved when compared to classical row-column algorithms. The resulting implementation requires only 67% of the FPGA logic resources compared to a DA 2-D DCT engine with equivalent throughput. In addition, as a result of the reduction in number of multiplications, computation noise introduced by round-off errors in the fixed-point multipliers, is less than the row-column approach.

The DA approach for computing 1-D DCTs is first described. An architecture for computing 2-D DCTs using FPGA-based 1-D DCT processors is presented. The logic requirements and performance data are reported. An overview of a polynomial transform (PT) DCT technique is then given. The mathematical framework of this elegant technique is provided, and then by way of an example ($N = 8$), the datapath implications are derived and implementation using Xilinx FPGA is reported. To illustrate the utility of the polynomial transform method, the logic requirements are compared to a DA DCT implementation.

2 COMPUTING DCTs USING DISTRIBUTED ARITHMETIC

To appreciate the utility of the polynomial transform based approach for computing DCTs presented later in the paper, it is useful to provide a context in which to draw comparisons. The DCT defining equation is

$$Y_k = \alpha_k \sum_{n=0}^{N-1} x_n \cos\left(\frac{2\pi}{4N}(2n+1)k\right) \qquad k = 0, \ldots, N-1 \tag{1}$$

where $\alpha_0 = 1\sqrt{N}$ and $\alpha_k = \sqrt{2/N}$ for $1 \leq k \leq N-1$. In,[6] distributed arithmetic[7] is used as the basis for a 2-D DCT Xilinx[12] FPGA implementation. The method is explained by first considering the matrix-vector form of the 1-D DCT equation defined in Eq. (2) for the case $N = 8$.

$$\begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \\ Y_5 \\ Y_6 \\ Y_7 \end{bmatrix} = \alpha_k \begin{bmatrix} C_0 & C_0 & C_0 & C_0 & C_0 & C_0 & C_0 & C_0 \\ C_1 & C_3 & C_5 & C_7 & -C_7 & -C_5 & -C_3 & -C_1 \\ C_2 & C_6 & -C_6 & -C_2 & -C_2 & -C_6 & C_6 & C_2 \\ C_3 & -C_7 & -C_1 & -C_5 & -C_5 & C_1 & C_7 & C_3 \\ C_4 & -C_4 & -C_4 & C_4 & C_4 & -C_4 & -C_4 & C_4 \\ C_5 & -C_1 & C_7 & C_3 & -C_3 & -C_7 & C_1 & -C_5 \\ C_6 & -C_2 & C_2 & -C_2 & -C_2 & C_2 & -C_2 & C_6 \\ C_7 & -C_5 & C_3 & -C_1 & C_1 & -C_3 & C_5 & -C_7 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} \tag{2}$$

This formulation clearly reveals that each DCT output sample $Y_k, k = 0, \ldots, 7$ can be computed as a vector dot-product of the basis function $C_k, k = 0, \ldots, 7$ and input samples $x_i, i = 0, \ldots, 7$ where $C_k = \cos(2\pi k/32)$. To compute all 8 DCT values, 8 FIR filters operating in parallel can be used. Each filter operates on the same set of input data, but uses different coefficient sets. Each filter's coefficient data is taken in a row-wise fashion from

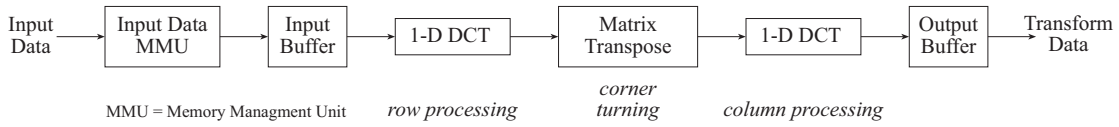

Figure 1: Row-column 2-D DCT processor employing 2 1-D DCT engines.

M	Γ (milli-seconds)	R
1024	23.6	42
2048	94.4	10
4096	377.5	2

Table 1: Frame rate R and 2-D DCT execution times Γ for several image sizes. DA DCT, $b = 8$, $f_c = 50$ MHz, gray-scale data.

the basis function matrix in Eq. (2). The symmetry embodied in the coefficient data means that each filter need only compute four product terms. This observation is used to implement very area efficient DA FIR filters for performing DCT calculations.

For 8-bit precision input samples, and following the resource accounting detailed in,[6] the FPGA logic breakdown for one filter is as follows. A 16-word deep 8-bit wide LUT occupying 4 CLBs is required for computing the sum-of-products. This unit is cascaded with an 8-bit parallel adder/subtracter requiring 9 CLBs. This gives a total of 13 CLBs for the basic inner-product engine. Input buffering occupies a further 64 CLBs, and 12 CLBs are used for producing the DA LUT addresses. The final components are an output buffer stage employing 64 CLBs, and 20 CLBs for the control-path. To support the concurrent calculation of all N DCT output values a total of 264 CLBs are required to implement a single 1-D DCT processor.

2.1 DA DCT Performance

A 2-D DCT processor can be implemented by exploiting the transform separability and using row-column processing as shown in Figure 1.

Data is loaded into the input buffer and N 1-D N-point DCTs are performed on the rows. N 1-D DCTs are then performed on the columns of this intermediate result. Depending on the DCT processor architecture, a matrix transposition may be required between the two transform stages.

For b-bit precision input samples, $b+1$ clock cycles are required to compute each N-point DCT. For an $M \times M$ image frame, and using $N \times N$ source blocks, the transform time Γ is

$$\Gamma = \frac{M \times M}{N \times N} \times N(b+1) \times \frac{1}{f_c} \tag{3}$$

where f_c is the system clock frequency. The frame rate R is $1/\Gamma$. The frame throughput and DCT execution times for several values of M, and using $b = 8$, are shown in Table 1.

From the tabulated performance data, the DA approach obviously has reasonable throughput, but there will always be applications with higher throughput demands than can be satisfied with this approach. High-performance would also be required if color images, or multiple video channels are to be coded.

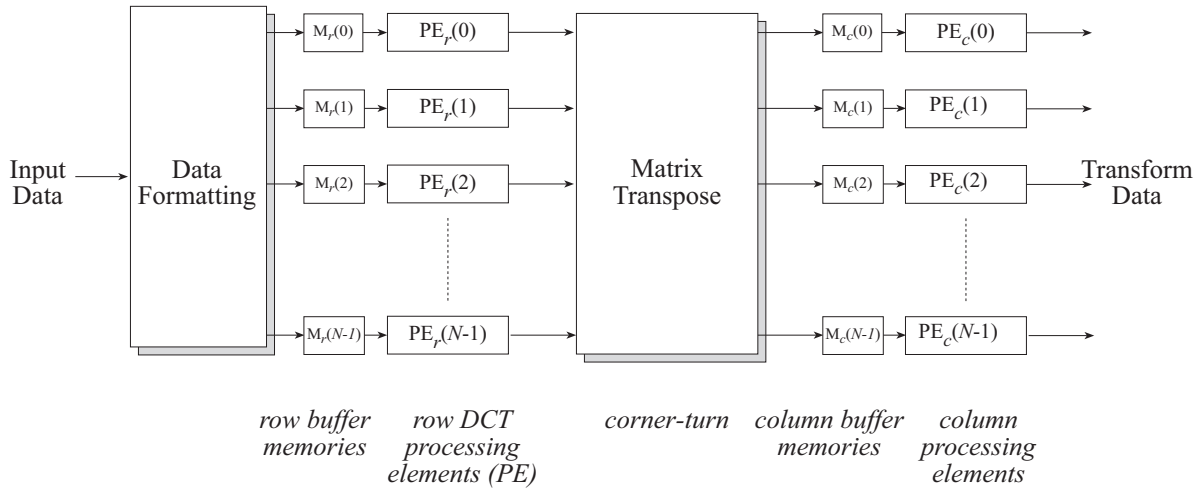

Figure 2: Parallel DCT processor for computing 2-D DCTs in real-time.

3 HIGH-PERFORMANCE 2-D DCT PROCESSOR

By parallelizing the row and column processing, a very high-performance DCT processor can be realized. This architecture is shown in Figure 2. Separate DCT processors are used for each row and column transform. The input data is distributed in a row-wise fashion to each of the row buffer memories $M_r(p)$, $p = 0, \ldots, N-1$. The row processing elements $PE_r(p)$ perform all of the N row transforms in parallel. The matrix transposer re-orders the data and presents it to the column processing elements $PE_c(p)$ via the column buffer memories $M_c(p)$. Suitable pipelining is employed so that the row processing, corner turning, and column DCT computation phases are overlapped.

Using this approach throughput is increased by a factor of N. However, this is achieved with a substantial increase in logic resources. Excluding the matrix transposer, this architecture requires approximately 3328 CLBs.

If the remaining arithmetic redundancies in the row-column approach can be eliminated, a more area efficient design can be produced. This goal is achieved by using an inherently 2-D approach to computing the transform, rather than decomposing the problem into a set of 1-D calculations. Instead of interpreting the input data as a 2-D array of real numbers, it is treated as an ordered column vector of polynomials. Each polynomial corresponds to one raster line, and can be thought of as an ordered N-tuple of data samples. The atomic unit of data that is now manipulated is a polynomial, rather than a real number as is the case with DCT methods that employ transform separability.

4 DCT COMPUTATION USING POLYNOMIAL TRANSFORMS

Polynomial transforms have been applied to computing 2-D DCTs by several researchers.[13–16] The derivation by Prado[16] will be used to formulate a datapath that results in a useful and efficient FPGA implementation. Before considering the datapath that has been developed by the author, the mathematical foundation of the technique is first described.

4.1 Polynomial Transform Calculation of 2-D DCTs

Since the inverse DCT is more easily expressible in terms of polynomial evaluation than the forward DCT, it will be used to define the formal framework of the method.

Consider the 1-D DCT of the length N sequence x_n

$$X_k = c_k \sum_{n=0}^{N-1} x_n \cos\left(\frac{2\pi}{4N}(2n+1)k\right) \qquad k = 0, \ldots, N-1 \tag{4}$$

where $c_0 = 1\sqrt{N}$ and $c_n = \sqrt{2/N}$ for $1 \leq k \leq N-1$. Without loss of generality the constant scaling term c_k will be excluded from the remainder of the analysis.

By defining the sequences

$$\begin{cases} y_n = x_{2n} \\ y_{N-1-n} = x_{2n+1} \qquad n = 0, \ldots, \frac{N}{2} - 1 \end{cases} \tag{5}$$

and

$$\begin{cases} Y_0 = 2X_0 \\ Y_k = X_k \qquad k = 1, \ldots, N-1 \end{cases} \tag{6}$$

it can be shown that

$$X_k + jX_{N-k} = \sum_{n=0}^{N-1} y_n W_{4N}^{-(4n+1)k} \tag{7}$$

and

$$y_n = \frac{1}{2} \sum_{k=0}^{N-1} (Y_k + jY_{N-k}) W_{4N}^{(4n+1)k} \tag{8}$$

Inverting Eq. (8) and applying standard results from polynomial algebra, results in the 1-D inverse DCT being expressed as a polynomial evaluation in Eq. (10) .

$$Y(z) = \sum_{k=0}^{N-1} (Y_k + jY_{N-k}) z^k \tag{9}$$

$$2y_n = Y(z)|_{z=W_{4N}^{(4n+1)k}} = Y(z) \mod \left(z - W_{4N}^{(4n+1)k}\right) \tag{10}$$

Eq. (10) has the following symmetry

$$jY_{N-k}^*(z) = z^N Y_k(z^{-1}) \tag{11}$$

4.1.1 The 2-D DCT as a Polynomial Evaluation

The 2-D DCT X_{n_1,n_2} of the sequence x_{n_1,n_2} is defined as

$$X_{k_1,k_2} = \sum_{k_1=0}^{N-1} \sum_{k_2=0}^{N-1} x_{n_1,n_2} \cos\left(\frac{2\pi}{4N}(2n_1+1)k_1\right) \cos\left(\frac{2\pi}{4N}(2n_2+1)k_2\right) \tag{12}$$

Define a permuted sequence as

$$\begin{aligned} y_{n_1,n_2} &= x_{2n_1,2n_2} \\ y_{N-n_1-1,n_2} &= x_{2n_1+1,2n_2} \\ y_{n_1,N-n_2-1} &= x_{2n_1,2n_2+1} \\ y_{N-n_1,N-n_2-1} &= x_{2n_1+1,2n_2+1} \quad n_1, n_2 = 0, \ldots \tfrac{N}{2} - 1 \end{aligned} \tag{13}$$

As noted by Prado[16] the 2-D equivalents of Eq. (7) and Eq. (8) can be obtained by some quite tedious computations as

$$\hat{X}_{k_1,k_2} = (X_{k_1,k_2} - X_{N-k_1,N-k_2}) + j(x_{N-k_1,k_2} + X_{k_1,N-k_2}) \tag{14}$$

$$= \sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} y_{n_1,n_2} W_{4N}^{-(4n_1+1)k_1} W_{4N}^{-(4n_2+1)k_2} \tag{15}$$

$$z_{n_1,n_2} = \sum_{k_1=0}^{N-1}\sum_{k_2=0}^{N-1} \hat{X}_{k_1,k_2} W_{4N}^{(4n_1+1)k_1} W_{4N}^{(4n_2+1)k_2} \tag{16}$$

$$= 4y_{n_1,n_2} - 2\sum_{k_1=0}^{N-1} \hat{X}_{k_1,0} \cos\left(\frac{2\pi}{4N}(4n_1+1)k_1\right) \tag{17}$$

$$-2\sum_{k_2=0}^{N-1} \hat{X}_{0,k_2} \cos\left(\frac{2\pi}{4N}(4n_2+1)k_2\right) \tag{18}$$

Eq. (16) is equivalent to an inverse 2-D DCT but for a few additive terms. Introducing an auxiliary sequence $\hat{Y}_{k_1,k_2}$

$$\begin{array}{ll} \hat{Y}_{k_1,0} = 2\hat{X}_{k_1,0} & k_1 = 0,\ldots,N-1 \\ \hat{Y}_{k_1,k_2} = \hat{X}_{k_1,k_2} & k_1,k_2 = 1,\ldots,N-1 \\ \hat{Y}_{0,k_2} = 2\hat{X}_{0,k_2} & k_2 = 0,\ldots,N-1 \end{array} \tag{19}$$

allows z_{n_1,n_2} to be written more compactly as

$$z_{n_1,n_2} = 4y_{n_1,n_2} = \sum_{k_1=0}^{N-1}\sum_{k_2=0}^{N-1} \hat{Y}_{k_1,k_2} W_{4N}^{(4n_1+1)k_1} W_{4N}^{(4n_2+1)k_2} \tag{20}$$

Using a standard result from polynomial algebra[13] allows Eq. 20 to be expressed as a polynomial evaluation along one dimension, n_2, while keeping the other one unchanged.

$$z_{n_1,n_2} = \sum_{k_1=0}^{N-1} \hat{Y}_{k_1}(z) W_{4N}^{(4n_1+1)k_1} \mod \left(z - W_{4N}^{(4n_2+1)}\right) \tag{21}$$

where

$$\hat{Y}_{k_1}(z) = \sum_{k_2=0}^{N-1} \hat{Y}_{k_1,k_2} z^{k_2} \tag{22}$$

Making use of the observation that $(z - W_{4N}^{(4n_2+1)})$ divides $(z^N + j)$, and replacing the index sequence $\{4n_1+1\}$ with a permutation $\{4n_1'+1\}$ defined as

$$\{4n_1'+1\} = \{(4n_1+1)\times(4n_2+1)\} \mod 4N \tag{23}$$

allows the 2-D IDCT to be expressed as

$$\hat{Y}(z) = \sum_{k_2=0}^{N-1} \hat{Y}_{k_1,k_2} z^{k_2} \tag{24}$$

$$\breve{Y}_{n_1} = \sum_{k_1=0}^{N-1} \hat{Y}_{k_1}(z) z^{(4n_1+1)k_1} \mod (z^N + j) \tag{25}$$

$$z_{n_1',n_2} = \breve{Y}(z) \mod \left(z - W^{(4n_2+1)4N}\right) \tag{26}$$

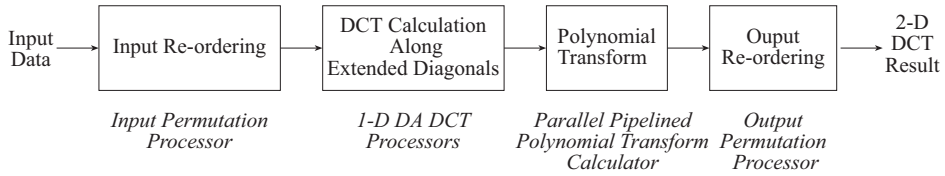

Figure 3: Polynomial transform DCT system architecture.

Eq. (25) is recognized as being a polynomial transform and is multiplier-free. Eq. (26) defines the calculation of N 1-D N-point IDCTs. The significant result to appreciate is that all of the multiplications from one dimensional of the problem have been removed. And further more, there has been no increase in the number of additions required.

A similar process can be used for calculating the forward transform, only now the order of the processing is changed and the IDCTs are replaced by DCTs.

5 FPGA IMPLEMENTATION OF 2-D POLYNOMIAL TRANSFORM BASED DCTs

From the formal mathematical explanation of the PT 2-D DCT in Section 4, the implications for the datapath architecture are not immediately obvious. To further expose the operation of the algorithm and its implications for FPGA implementation, a detailed example for the case $N = 8$ is considered in this section. This value has been chosen because of its widespread use in video coding standards such as MPEG[1] and H.261.[2]

A block diagram of the PT 2-D DCT system architecture is shown in Figure 3. Data is received from the input source and re-ordered. N 1-D DCTs are computed along extended diagonals of the permuted data, a polynomial transform is calculated, and some output re-ordering is performed to produce the final result.

Although formally defined above, for the purposes of clarifying the procedure and providing some insight to the requirements of the FPGA DCT processor, the data re-ordering process is examined here in more detail.

Figure 4 shows the first stage of the input permutation. The starting point is the naturally ordered input data in Figure 4(a). Applying the permutation defined in Eq. 13 for $N = 8$ results in the data-set shown in Figure 4(b). Only the indices for the two-dimensional input samples are indicated in the figures. Inspection of the re-ordered matrix reveals the symmetries in the permutation mapping. It is worth noting that the address sequencing required to effect the mapping is easily accommodated by an FPGA. The designer has the luxury of customizing the address generator to meet any arbitrary data access requirements without any impact in the time dimension. Of course area is traded in return for this benefit. However, a microprocessor implementation is likely to be inefficient. Either an arithmetic approach that requires use of the processor arithmetic logic unit (ALU) and considerable control overhead, or a table lookup scheme could be employed. Both methods will impinge on the reduction in the arithmetic workload that the PT algorithm provides.

Next, a sequence of N, in this case 8, 1-D DCTs are computed along extended diagonals of the permuted input data. The eight diagonals are shown in Figure 5. All of the 1-D DCTs are computed in parallel, so of course each 1-D DCT processor must have the appropriate source data available to it. This data organization task is performed by the *Input Permutation Processor* in Figure 3. This functional unit assembles the input data and formats it into eight independent memories, each containing data along one of the extended diagonals.

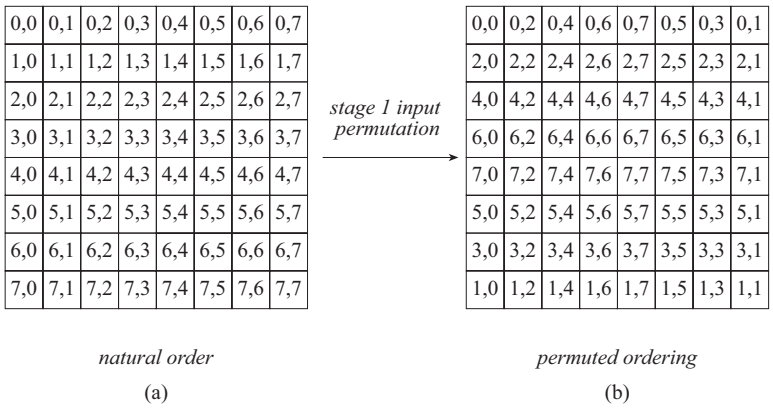

Figure 4: Polynomial transform 2-D DCT - first stage of input data re-ordering.

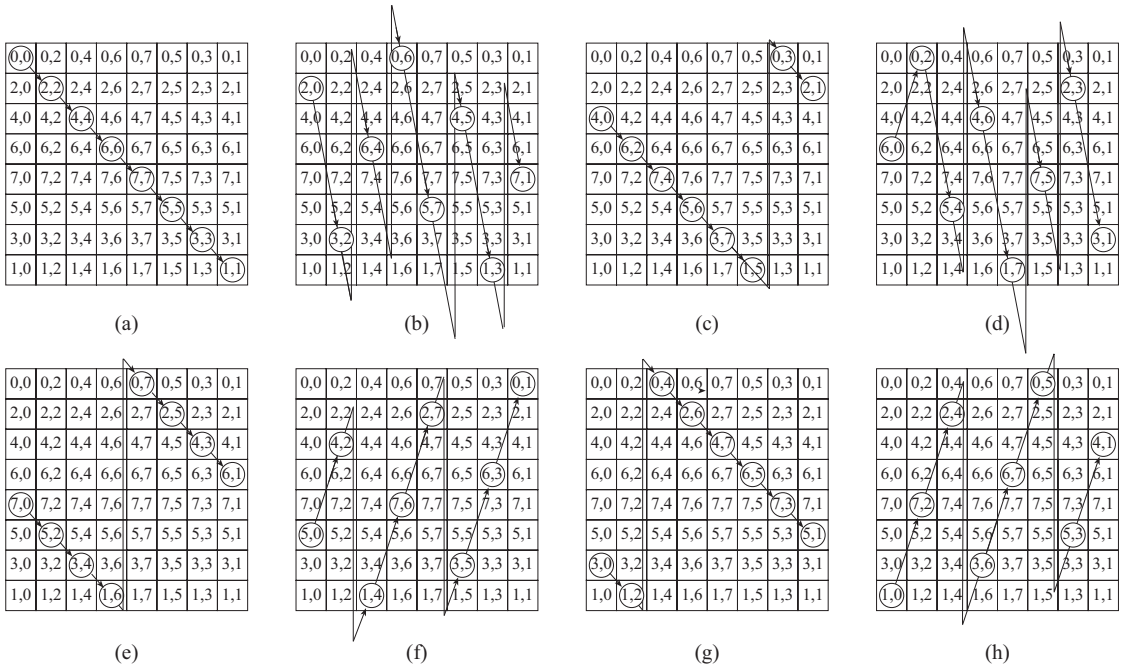

Figure 5: Computing 1-D DCTs along extended diagonals of the permuted input array.

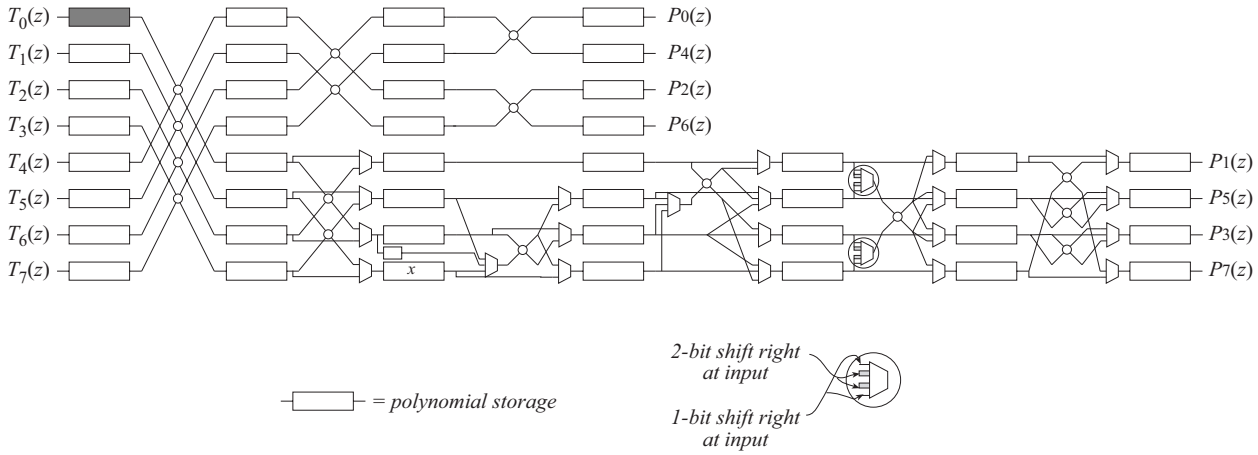

Figure 6: DCT polynomial transform signal flowgraph. $N = 8$.

The requirements of the first stage permutation and the extended diagonal indexing are contracted into a single process that is performed by this memory management unit.

Each DA processor requires $b + 1$ clock cycles to compute a complete 1-D N-point DCT. With the PT post-processing overlapping the first stage DCT calculation, the data re-ordering must be completed in $b + 1$ clock cycles so as not to stall the processor. In the available $b + 1$ clock cycles, N^2 numbers must be handled by the Permutation Processor. This can be achieved by presenting the data as $N - 1$ degree polynomials to the input stage. For the widely used case of $N = 8$ this is not an unreasonable requirement. The implication is that 8 b-bit data samples be available at each clock cycle. For $b = 8$, this amounts to the provision of a 64-bit wide input bus.

The key equations for constructing the datapath for the PT processor are embodied in Eq. (25). Several options are available for a hardware realization. The PT is computed with a minimum number of additions if a radix-2 partitioning, similar to the classical FFT one, is used. This processing stage will consist of several columns of polynomial butterflies. Unlike the butterflies that are used in many common FFT algorithms, the PT transform butterflies operate on polynomials rather than complex tuples. Furthermore, they are multiplier free, requiring only addition and subtraction. An initial inspection of Eq. (25) may indicate the use of complex arithmetic in the procedure. However, the complex writing of the inverse 2-D DCT is nothing more than a convenient option that allows a succinct mathematical specification of the calculations that are to be undertaken. On another level of observation, since the input data to the transform is real-valued, and because the DCT kernel does not involve complex arithmetic, the result will naturally be a sequence of real numbers. It follows that there must be a way to realize the system of equations Eq. (24-26) without requiring the use of complex variables. The problem is one of revealing this possibility and architecting a suitable datapath.

A detailed analysis of Eq. (25) for the case $N = 8$, resulted in the signal flowgraph of Figure 6. The input polynomials $T_k(z)$, $k = 0, \ldots, 7$, are the results from the extended diagonal DCT calculations. The polynomials $P_k(z)$, are the transform results in polynomial bit-reversed order. The calculation consists primarily of polynomial butterflies, but with suitable multiplexing at appropriate points to avoid the introduction of any complex quantities and to implicitly handle *multiplication* by the polynomial transform kernel. It should be noted that the multiplication referred to here is not an arithmetic product and is implemented by appropriate indexing of the data and possibly the introduction of a sign-change. A parallel pipelined structure is used in order to match the latency of the first stage DCT processing. Recall that the DCT processing requires 9 clock cycles to execute. The PT calculation must be performed in this same amount of time, or less, to avoid any stalls.

Each butterfly consists of an adder and subtracter. Most of the polynomial storage can employ FIFOs so

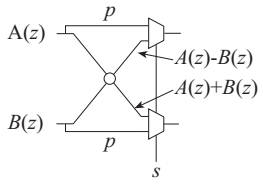

Figure 7: DCT rank-2 butterfly.

minimizing control overhead, some of the memory elements must be dual-port RAM. The signal flow graph does not posses the same degree of uniformity that the classical FFT algorithm has. The first processing stage is straightforward, the four butterflies accept 7'th-order polynomials and form polynomial sums and differences that are written to output FIFOs or dual-port memory - the processing is the same for the 4-pairs of rows. Subsequent processing stages are not as regular, but essentially perform *butterfly-like* processing. This point is highlighted by one example. In the second column of butterflies, rows 4 and 6 are combined using the butterfly circuit in Figure 7 to produce new sequences for these same rows but one iteration later in time. The computation is defined in Eq. (27) and Eq. (28). The primes indicate the new outputs and the notation $y(i,j)$ references the jth element in the i'th row of a two-dimensional data-set.

$$\begin{aligned} y'(4,i) &= \frac{1}{2}\left(y(4,i) - y(6,8-i)\right) \\ y'(6,i) &= \frac{1}{2}\left(y(4,i) + y(6,8-i)\right) \qquad i = 1,2,3 \end{aligned} \tag{27}$$

$$\begin{aligned} y'(4,8-i) &= \frac{1}{2}\left(y(4,8-i) - y(6,i)\right) \\ y'(6,8-i) &= \frac{1}{2}\left(y(4,8-i) + y(6,i)\right) \qquad i = 1,2,3 \end{aligned} \tag{28}$$

Observe that in these equations column elements 0 and 4 are not involved in the calculation, they must of course pass through this processing phase unaltered in order to be available to later processing stages. The modified butterfly that facilitates this is shown in Figure 7. The multiplexor select signal s allows the coefficients for terms z^0 and z^4 from $A(z)$ and $B(z)$ to use the bypass path p and progress to the next stage of processing unaltered. For all other terms, the polynomial sum and difference is presented at the multiplexor output. The scaling by $1/2$ in Eq. (27) and (28)is simply a right-shift by one bit and is accommodated by wiring between adjacent stages.

5.1 Logic Requirements and Comparison

A figure of 200 CLBs will be used to account for each 1-D DA DCT processor. This number is obtained from data in Section 2, where the total cost for a complete DA DCT engine was introduced as 264. The cost of the dual-port output buffer, 64 CLBs, has been removed since it is not required for the processing elements in the concurrent approach. Using 8-bit precision, the datapath defined in Figure 6 can be implemented with 584 Xilinx 4000 series CLBs. Add to this $8 \times 200 = 1600$ CLBs for the first stage DA-based DCT processing and a further 64 CLBs for the row buffer memories, and the net logic requirement is 2248 CLBs. From Section 3, the CLB requirements for a transform processor that solely employs DA DCT techniques was 3328. So the polynomial transform method uses $2248/3328 \times 100\% = 67\%$ of logic resources of the DA-only processor while providing the same processing throughput.

5.2 Scalability

Both the 2-D DA DCT processor and the 2-D PT DCT architecture are obviously scalable. It is instructive to consider the area implications for scaling both approaches. Aside from the dimension M of the input data-set, there are essentially two other variables to consider from a scalability perspective: the precision of the input samples and the size N of the transform data blocks. We will investigate the effect of manipulating N. What effect does a doubling of N produce? For a fixed frame rate requirement, a doubling of N implies approximately a four-fold increase in the processing load. In going from $N = 8$ to $N = 16$, 16 DA engines will be required in each of the row and column processors. To match the processing latency for the $N = 8$ case using the larger block size, requires the 16-bit processors to process two bits of each input sample in parallel. The logic requirements of a dual-bit processor are approximately twice that of a single-bit processor. So again using the figure of 200 CLBs for a single-bit DA DCT engine, the new CLB count for the 16×16 transform engine is $2 \times (16 + 16) \times 200 = 12,800$ CLBs. Now consider the logic impact for the PT architecture.

The first stage processing will grow by the same amount as the row processor requirements in the DA processor. However, the polynomial transform scales logarithmically. A good estimate of the impact can be obtained by considering how butterfly networks in general scale. Let A_N denote the logic resource requirements for an order N polynomial transform. The complexity of the polynomial transform is approximately $N/2 \log_2 N$. The impact on silicon requirements in going from an $O(N/2)$ problem to an $O(N)$ transform can be expressed as the ratio

$$\lambda(N) = \frac{A_N}{A_{N/2}} = \frac{\frac{N}{2}\log_2 N}{\frac{N}{4}\log_2 \frac{N}{2}} \tag{29}$$

Having the CLB count $\Xi(8)$ for $N = 8$, lets us compute a good area estimate for a 16-point polynomial transform as $\lambda(16) \times \Xi(8) = 2.666 \times 584 = 1557$. Including the first stage DCT processing gives a CLB count of $2 \times 16 \times 200 + 1557 = 7957$ CLBs. Comparing this figure with the DA CLB count, shows that the PT processor uses $7957/12800 \times 100 = 62\%$ of the logic requirements.

The conclusion is that the polynomial transform algorithm scales in a more desirable manner than the DA-only DCT based processor. This observation may warrant devoting some attention to this approach for demanding combinations of problem size and frame rate requirements.

6 CONCLUSION

This paper has explored a new architectural option for implementing high-performance 2-D DCT processors using FPGA technology. Most would agree that the mathematical framework of the polynomial transform DCT calculation is reasonably cumbersome. This investigation has attempted to reveal some details of the operation of the algorithm, and show how a suitable datapath can be constructed for performing the required calculations. By comparison with one of the few FPGA DCT implementations reported in the literature it has been demonstrated that the polynomial transform based DCT calculation has a contribution to make in the context of minimizing FPGA logic resources. For the case of 8×8 coding blocks, a polynomial transform DCT architecture uses 67% of the CLB resources of DA-only approach for the same throughput.

As a final note, it is interesting to observe that since all of the multiplications have been completely eliminated from one dimension of the problem, the computation noise introduced by rounding of product terms has been removed. The only operations required in the second stage of the polynomial transform DCT are addition and subtraction. When implemented using integer arithmetic, the calculations are performed exactly and with no introduction of noise.

7 REFERENCES

[1] ISO/IEC JTC1/SC29/WG11, MPEG Committee Draft CD11172, 1991.

[2] CCITT Recommendation H.261, 1990.

[3] D. W. Trainor and R. F. Woods, "Architectural Synthesis and Efficient Circuit Implementation for Field Programmable Gate Arrays", *Proc. Of the 6th International Workshop on Field-Programmable Logic and Applications, FPL'96*, Ed. R. W Hartenstein and Manfred Glesner, pp. 116-125, Darmstadt, Germany, Sept. 1996.

[4] F. A. McGovern, R. F. Woods and M. Yan, "Novel VLSI Implementation of the 8 8 point 2-D DCT, *IEE Electronics Letters*, Vol. 30, No. 4, pp. 624-626, April 1994.

[5] XC6200 Family Datasheet, Xilinx Inc. 1996.

[6] L. Mintzer, "The FPGA as a Multimedia Processor", *Xilinx Internal Report*, Sept. 1997. (5 pages)

[7] S. A. White, "Applications of Distributed Arithmetic to Digital Signal Processing", *IEEE ASSP Magazine*, Vol. 6(3), pp. 4-19, July 1989.

[8] W. H. Chen, C. H. Smith and S. C. Fralick, "A Fast Computational Algorithm for the Discrete Cosine Transform", *IEEE Trans. Commun.*, Vol. 25, pp. 1004-1009, Sept. 1977.

[9] N. I. Cho and S. U. Lee, "Fast Algorithm and Implementation of the 2-D Discrete Cosine Transform, *IEEE Trans. Cir. And Sys.*, pp. 297-305, March, 1991.

[10] A. Madisetti and A. N. Willson Jr., "A 100 MHz 2-D 8 8 DCT/IDCT Processor for HDTV Applications, *IEEE Trans. On Cir. And Sys. For Video Technology*, Vol. 5. No. 2, pp. 158-165, April 1995.

[11] Texas Instruments, TMS320C5x User's Guide, 1997.

[12] Xilinx Inc., *The Programmable Logic Data Book*, 1998.

[13] H. J. Nussbaumer, *Fast Fourier transform and convolution algorithms*, Springer-Verlag, New York, 1981.

[14] P. Duhamel and C. Guillemot, "Polynomial Transform Computation of the 2-D DCT", *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP*, Albuquerque, New Mexico, USA, pp. 1515-1518, April 3-6, 1990.

[15] S. Pei and E. Huang, "Improved 2-D Discrete Cosine Transforms Using Generalized Polynomial Transforms and DFTs", *Proc. of International Conference on Communications, Links for the Future: Science, Systems and Services for Communications*, Amsterdam, Eds. P. Dewilde and C. A. May, Elsevier Science, pp. 242-244, 1984.

[16] J. Prado and P. Duhamel, "A Polynomial Transform Based Computation of the 2-D DCT with Minimum Multiplicative Complexity", *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP*, pp. 1347-1350, 1996.