International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Impact Factor (2012): 3.358

Algebraic Integer based 2-D DCT Computation using Single Channel Architecture

B. Uday[1], P. Apsar[2]

[1]Assistant Professor, Department of ECE, Intell Engineering College, Anantapur, India

[2]PG Student, Department of ECE, Intell Engineering College, Anantapur, India

Abstract: ***This paper presents an area efficient row-parallel architecture for the real-time implementation of bivariate algebraic integer (AI) encoded 2-D discrete cosine transform (DCT) for image and video processing. An improved fast algorithm for AI based 1-D DCT computation is proposed along with single channel 2-D DCT architecture. The proposed architecture computes 8x8 2-D DCT transform based on the Arai DCT algorithm. The design improves on the 4-channel AI DCT architecture that was published recently by reducing the number of integer channels to one and the number of 8-point 1-D DCT cores reduced from 5 down to 2. The architecture offers exact computation of 8x8 blocks of the 2-D DCT coefficients up to the FRS, which converts the coefficients from the AI representation to fixed-point format using the method of expansion factors.***

Keywords: DCT, Algebraic Integers, Expansion factors, Arai DCT algorithm

1. Introduction

In this paper, we propose an improved fast algorithm for the AI based 1-D DCT computation derived from the algorithm proposed in [9].The discrete cosine transform (DCT) is a widely used mathematical tool in image and video compression. It is a core component in contemporary media standards such as JPEG .Several algorithms have been proposed to reduce the complexity of DCT circuits by exploiting its mathematical properties[3],[4].Integer transforms are employed in video standards such as H.264. Indeed, the DCT is known for its properties of decorrelation, energy compaction, separability, symmetry, and orthogonality [2]. However, we emphasize that these methods are approximations, which are inherently inexact and may introduce a computational error floor to the DCT evaluation.

To wit, one of the main obstacles in performing accurate DCT computations is the implementation of the irrational coefficient multiplications needed to calculate the transform. Traditional DCT implementations adopt a compromise so-lution to this problem employing truncation or rounding off [6] to approximate these quantities. As a consequence, computational errors are systematically introduced into the Computation, leading to degradation of the signal-to-noise ratio (SNR).

To partially address this issue, algebraic integer (AI) encoding [7] has been employed [8], [9], [10], [11]. The main idea in this approach is to map required irrational numbers into an array of integers, which can be arithmetically manipulated in an error free manner. At the end of the computation, AI based algorithms require a final reconstruction step (FRS) in order to map the resulting encoded integer arrays back into the usual fixed-point representation. FRS can be implemented by means of individualized circuits, at in principle *any given precision* [4].

Architectures based on the low-complexity Arai DCT algorithm [3] has been proposed in [9], [12]. The Arai DCT algorithm is an algorithm for 8-point DCT computation in video and image processing applications because of its relatively low computational complexity. It is noted that this algorithm requires only five multiplications to generate the eight output coefficients. In the AI based architectures proposed in [9], [12], the algebraically encoded numbers are reconstructed and rep- resented in fixed-point format at the end of column-wise DCT calculation by means of an intermediate reconstruction step; then data is coded again before the row-wise DCT calculation. As a result, the intermediate reconstruction step introduces errors that propagate into subsequent sections. In a sense, the presence of such computational error in a intermediate stage of the calculation diminishes the point of employing an AI based structure. To address this issue, in [13] a doubly AI encoded architecture where the reconstruction is performed only once at the end of the entire computation was proposed. Such architecture could allow a completely error free computation throughout all algorithm stages until the reconstruction stage for the 2-D DCT computation. In fact, after the column-wise computation, data path is divided and the row-wise computation is performed separately for each AI base. We use the term channel to refer to these data paths.

Additionally, we present a 2-D DCT architecture based on the improved 1-D transform. This 2-D constitutes of a single channel which provides improvements in terms of area and power consumption when compared with the four channel architecture described in [13]. Detailed comparisons between the proposed architecture with some of the existing are also provided.

TABLE I
2-D AI ENCODING OF ARAI DCT CONSTANTS.

$\cos(4\pi/16)$	$\cos(2\pi/16) - \cos(6\pi/16)$
$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix}$
$\cos(6\pi/16)$	$\cos(2\pi/16) + \cos(6\pi/16)$
$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 2 \\ 0 & 0 \end{bmatrix}$

2. Review of AI Based DCT Computation

AI encoding provides an exact representation devoid of quantization noise in DCT computations. An AI is defined as a root of a monic polynomial whose coefficients are integers [14]. AI may constitute a basis and real numbers may be expressed as a integer linear combination of such basis elements. Thus, real numbers can be possibly represented without errors by an array of integers.

In [9] AI encoding is adapted into Arai DCT algorithm [3], and the corresponding encoding is shown in Table I. The corresponding AI basis is furnished by $\begin{pmatrix} 1 & Z_1 \\ Z_2 & Z_1 \end{pmatrix}$

Where $Z_1 = \sqrt{2+\sqrt{2}} + \sqrt{2-\sqrt{2}}$ and $Z_2 = \sqrt{2+\sqrt{2}} - \sqrt{2-\sqrt{2}}$

It should be noted that the hardware implementation of this representation requires only of adders/subtracters. Indeed, a given real number x is encoded into an integer-array

$$\begin{pmatrix} x^{(a)} & x^{(b)} \\ x^{(c)} & x^{(d)} \end{pmatrix}$$

Where $x^{(a)}$, $x^{(b)}$, $x^{(c)}$ and $x^{(d)}$ indicate the integers associated to basis elements 1, Z_1, Z_2 and Z_1 Z_2 respectively. We refer to these integers as the AI components of x.

Therefore, quantity x can be decoded (reconstructed) from its AI encoded form according to [9]:

$$x \equiv \mathrm{tr}\left(\begin{bmatrix} x^{(a)} & x^{(b)} \\ x^{(c)} & x^{(d)} \end{bmatrix} \cdot \begin{bmatrix} 1 & z_1 \\ z_2 & z_1 z_2 \end{bmatrix}^{\top} \right) = x^{(a)} + x^{(b)} \cdot z_1 + x^{(c)} \cdot z_2 + x^{(d)} \cdot z_1 z_2, \quad (1)$$

Where tr(.) returns the trace of its argument and superscript ⊤ corresponds to the transposition operation. The decoding can be done in a tailored FRS where the AI basis is represented at the desired precision. Several FRS structures have been proposed, including schemes that employ Booth encoding [9] and the expansion factor method [13], [15].

3. Improved AI- Based 1-D DCT Algorithm

The proposed 1-D DCT algorithm is derived from the1-D AI Arai DCT [9], [13], using algebraic manipulations. Its computational complexity consists of 20 additions; no multiplication or shift operations are required, as depicted in Fig. 1. This provides an improvement in terms of hardware resources when compared with the algorithm proposed in [13] and [9], which requires 21 additions and 2 shift operations. Although when only a single DCT is considered, the economy of one addition and two bit-shifting operations may seem modest, we note that in video processing systems, the 1-D DCT is performed many times (once per column, per row, per component, per 8x8 block, per frame). Thus, the overall computational savings is cumulative. Further, the proposed method as well as its competitors is highly optimized procedures; thus one may not expect enormous gains in terms of computational complexity. Indeed, we are approaching the asymptotic limits of the theoretical DCT complexity. Finally, it is noted that the savings in multiplications account for about 5% in total complexity for considering DCTs only, which can be a significant gain especially for low-power applications.

For the purpose of mathematical analysis, the transform is described in matrix notation. Let **A** denote the matrix transformation associated to the 1-D Arai DCT algorithm defined over the AI structure and **B** denote the matrix related to the operations in the FRS. Matrix **A** is represented in terms of signal flow diagram in the dashed block of Fig. 1. Matrix **B** is simply represents the AI presentation, ensuring that the resulting calculation furnishes AI as shown in (1).
Therefore, the complete 1-D AI DCT is given by

$$\mathbf{X}_{1D} = \mathbf{B} \quad \mathbf{A} \quad \mathbf{x}_{1D},$$

Where x_{1D} and X_{1D} are 8-point column vectors for the input and output sequences, respectively, and where

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & 0 & 0 & -1 & -1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 0 & -1 & -1 & 0 & 0 & 1 & 1 & 0 \\ -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix} \quad (2)$$

$$\mathbf{B} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & Z_1 Z_2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & -Z_1 Z_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -Z_2 & -Z_1 Z_2 & -Z_1 & 1 \\ 0 & 0 & 0 & 0 & Z_2 & -Z_1 Z_2 & Z_1 & 1 \\ 0 & 0 & 0 & 0 & -Z_1 & Z_1 Z_2 & Z_2 & 1 \\ 0 & 0 & 0 & 0 & Z_1 & Z_1 Z_2 & -Z_2 & 1 \end{pmatrix}$$

The multiplications present in **B** can be efficiently implemented by means of the expansion factor method as described in [15]. The expansion factor method employs a factor which scales the required multiplicands z_1, z_2, and z_1 z_2 Into quantities that are close to integers governed by the following relationship:

$$\alpha \cdot \approx \begin{pmatrix} Z_1 & Z_2 & Z_1 Z_2 \end{pmatrix} \begin{pmatrix} m1 & m2 & m3 \end{pmatrix}$$

Where m1, m2, and m3 are integers. This approach entails a reduction in the overall number of multiplications required by the FRS [15].

4. Row- Parallel 2-D DCT Architecture

Let x_{2D} and X_{2D} be 8x8 input and output data of the 2-D DCT, respectively. The 2-D DCT of X2D can be obtained after (i) applying the 1-D DCT to its columns; (ii) transposing the resulting matrix; and (iii) applying the 1-D DCT to the rows of the transposed matrix. Mathematically, above procedure is given by:

$$\mathbf{X_{2D} = B \cdot A \cdot x_{2D} \cdot (B \cdot A)^T = B \cdot A \cdot x_{2D} \cdot A^T \cdot B^T}$$

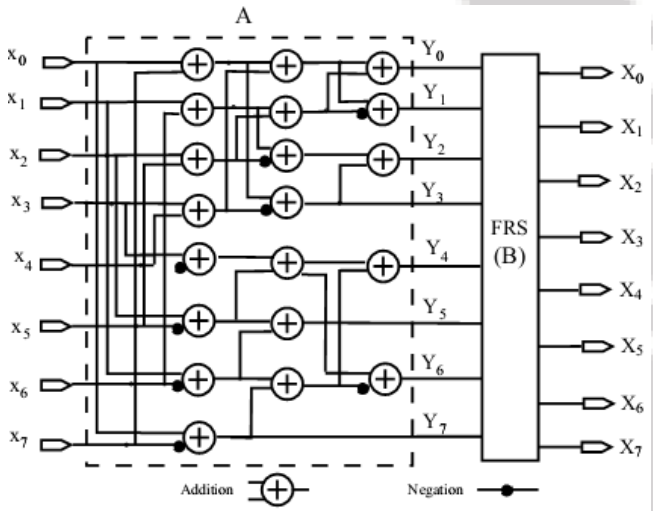

Figure 1: Fast Algorithm for AI based 1 D-DCCT

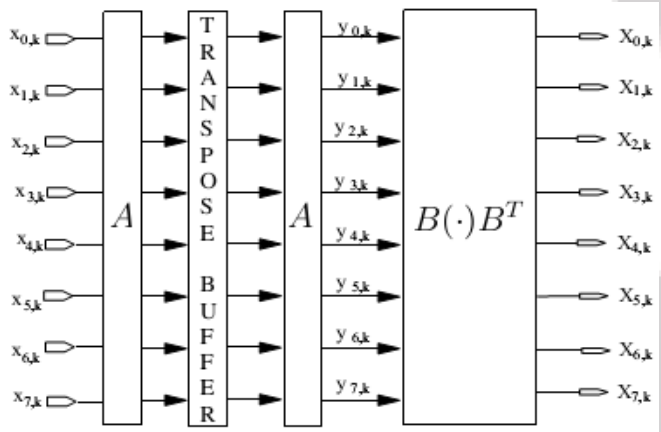

Figure 2: Block diagram of the proposed 2-D DCT architecture

Matrix multiplications by B (reconstruction step) should be the final computation stage. Otherwise, an earlier multiplication by B represents an intermediate reconstruction stage, which may reintroduce numerical representation errors. Such errors would then be propagating to subsequent 1-D DCT calls. This may render the purpose of AI encoding ineffective. In [13] an AI based architecture, where the reconstruction step occurs only at the end of the computation, was proposed. In other words, the reconstruction step was placed after both row- and column-wise transforms. In the current contribution, we propose a similar scheme. However, unlike the architecture described in [13], which requires a column-wise DCT call for each of the four AI components, we propose an architecture which requires a single column-wise DCT. As a result, the computational complexity is greatly decreased in this particular section of the algorithm. The complete block diagram of the 2-D DCT architecture is given in Fig. 2.

A. Derivation of the 1-Channel Architecture

In this subsection, we describe mathematically the derivation of the new architecture. Consider the following decomposition of the matrix B, which follows from (2):

$$\mathbf{B = B_0 + B_1 \cdot Z_1 + B_2 \cdot Z_2 + B_3 \cdot Z_1 Z_2}$$

$$\mathbf{B_0} = \begin{pmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&1 \end{pmatrix} \begin{pmatrix} 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&-1&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&-1&0&0&0\\ 0&0&0&0&1&0&0&1 \end{pmatrix}$$

$$\mathbf{B_1} = \quad (5)$$

$$\mathbf{B_2} = \begin{pmatrix} 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&-1&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&-1&0 \end{pmatrix} \begin{pmatrix} 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&-1&0&0&0&0\\ 0&0&0&0&0&-1&0&0\\ 0&0&0&0&0&-1&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&1&0&0 \end{pmatrix}$$

$$\mathbf{B_3} =$$

Matrices $\mathbf{B_0}$, $\mathbf{B_1}$, $\mathbf{B_2}$, and $\mathbf{B_3}$ are sparse, and contain only 0, 1, and 1, leading to low complexity. Therefore, applying (5) into (4) the 2-D DCT computation can be rewritten as follows. Let $\mathbf{Y_{2D}} = \mathbf{A} \cdot x_{2D} \cdot A^{\top}$. Notice that the calculation of $\mathbf{Y_{2D}}$ requires no multiplications. Then, we have

$$\mathbf{X}_{2D} = \mathbf{B} \cdot \mathbf{Y}_{2D} \cdot \mathbf{B}^{\top} = \sum_{i=0}^{3} \sum_{j=0}^{3} \mathbf{B}_i \cdot \mathbf{Y}_{2D} \cdot \mathbf{B}_j^{\top} \cdot z_1^m z_2^n \quad (6)$$

Where m; n 2 f0; 1; 2g. Therefore, the evaluation of $\mathbf{X_{2D}}$ is highly dependent on the computation of the structure $B_i \cdot (.) \cdot B^{\top}_j$.

Efficient Implemenation Of $B_i.(.).B^{\top}_j$ Block

Matrices $\mathbf{B_i}$, $i \in \{1, 2, 3, 4\}$ contain only a single non-zero element in each row. Therefore any matrix multiplication by Bi can be trivially performed. To maintain the row-parallel structure of the design, the block that computes $B_i \cdot Y_{2D} \cdot B^{\top}_j$ should output a single row of data per clock cycle. Hence the 64 elements in the 8X8 $\mathbf{Y_{2D}}$ matrix should be stored for 8 clock cycles.

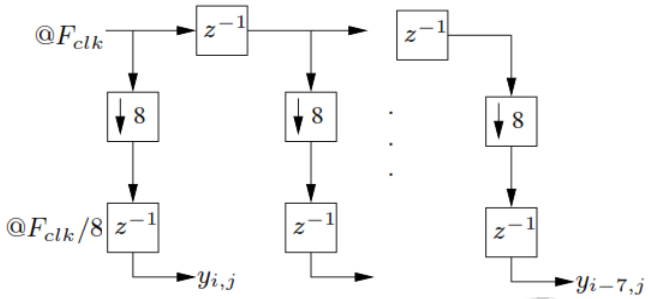

Figure 3: SPG depicting the direct implementation of the buffer section in $B_i.(.).Bj^T$ block for the jth Row

The signal flow graph (SFG) corresponding to the j th row of the storage structure is depicted in Fig 3. The left and right-multiplication of Y_{2D} by an arbitrary 8X8 matrix would require 56 storage elements operating at a clock rate of **F**clk and another 64 storage elements operating at a rate of **F**clk =8. However, this operation requires far less storage elements due to the sparse nature of the matrices under consideration. In the next section we investigate this possibility.

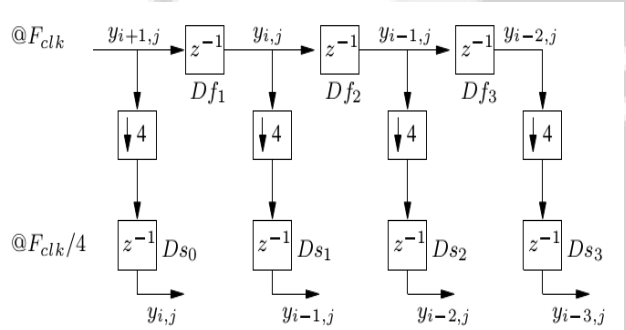

Fig. 4. SFG depicting the multiplexed implementation of the buffer section in $B_i \cdot (\cdot) \cdot B_j^T$ block for the j^{th} row.

A. Derivation of Half Column Independence

The 8X8 matrices $\mathbf{B_i}$, $i \in \{1, 2, 3, 4\}$ can be decomposed into a block matrix of 4X4 blocks in the following manner:

$$\mathbf{B_i} = \begin{pmatrix} \mathbf{B_{i,0}} & \mathbf{0_4} \\ \mathbf{0_4} & \mathbf{B_{i,1}} \end{pmatrix} \quad (7)$$

Where, $\mathbf{0_4}$ is the 4X4 null matrix. **Bi** is a block diagonal matrix.

Similarly,

$$\mathbf{Y_{2D}} = \begin{pmatrix} \mathbf{Y_{2D,0}} & \mathbf{Y_{2D,1}} \\ \mathbf{Y_{2D,2}} & \mathbf{Y_{2D,3}} \end{pmatrix}$$

Therefore, the product $\mathbf{B_i} \cdot \mathbf{Y_{2D}} \cdot \mathbf{B_j^T}$ is given by

$$\mathbf{B_i} \cdot \mathbf{Y_{2D}} \cdot \mathbf{B_j^T} = \begin{pmatrix} \mathbf{B_{i,0}} & \mathbf{0_4} \\ \mathbf{0_4} & \mathbf{B_{i,1}} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{Y_{2D,0}} & \mathbf{Y_{2D,1}} \\ \mathbf{Y_{2D,2}} & \mathbf{Y_{2D,3}} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{B^T_{i,0}} & \mathbf{0_4} \\ \mathbf{0_4} & \mathbf{B^T_{j,1}} \end{pmatrix}$$

$$= \begin{pmatrix} \mathbf{B_{i,0}} \cdot \mathbf{Y_{2D,0}} \cdot \mathbf{B^T_{j,0}} & \mathbf{B_{i,0}} \cdot \mathbf{Y_{2D,1}} \cdot \mathbf{B^T_{j,1}} \\ \mathbf{B_{i,1}} \cdot \mathbf{Y_{2D,2}} \cdot \mathbf{B^T_{i,0}} & \mathbf{B_{i,1}} \cdot \mathbf{Y_{2D,3}} \cdot \mathbf{B^T_{i,1}} \end{pmatrix} \quad (8)$$

Above relations show that the computation of $\mathbf{B_i} \cdot \mathbf{Y_{2D}} \cdot \mathbf{B_j^T}$, for a particular choice of i and j , can be performed in two sequential steps which can be computed independently of each other. The computation of (8) is achieved by the following steps:

1. Compute the first four rows of the resulting matrix using the first four rows of $\mathbf{Y_{2D}}$;
2. Compute the last four rows of the resulting matrix using the last four rows of $\mathbf{Y_{2D}}$.

The above sequence of computation can be realized using the SFG in Fig. 4. Eight consecutive rows are stored in clock FIFO registers. A total of 24 FIFO registers are needed for the section of the circuit operating at **F**clk , while 32 registers are required for the slower section operating at **F**clk/8.

B. Buffer

The buffer shown in Fig. 5 consists of a shift register section and a bank of parallel-load registers. At time instant ***k***, the shift registers hold four consecutive rows of $\mathbf{Y2D}$ denoted by $yk,0\ldots7$, $yk-1,0\ldots7$, $yk-2,0\ldots7$, and $yk-3,0\ldots7$. When $k \pmod 4 = 0$, a parallel load is executed, synchronously transferring the content of the shift registers (Fig. 5, block P) into the parallel load register bank (Fig. 5, block Q). The timing diagram portraying register transfers in this block is shown in Fig 6. Transfers occur at every positive clock edge of **F**clk .Next, the computation of $\mathbf{B_{i,q}} \cdot \mathbf{Y_{2D,r}} \cdot \mathbf{B_{j,s}^T}$ block, where $i \in \{0, 1, 2, 3\}$, $q \in \{0, 1\}$, $r \in \{0, 1, 2, 3\}$, and $s \in \{0, 1\}$ is considered.

C. Computing $\mathbf{B_{i,j}} . \mathbf{Y_{2D,r.}} \mathbf{B^T_{j,k}}$ Block

The computation of $\mathbf{B_{i,j}}\ \mathbf{Y_{2D,r}}\ \mathbf{B^T_{j,k}}$ is achieved by cross-wiring. This step of the computation is free of addition and subtraction operations. Eight-input multiplexers are used for cross-wiring. Notice that (i) $\mathbf{B_{i,j}}\ \mathbf{Y_{2D,0}}$ and $\mathbf{B_{i,0}}\ \mathbf{Y_{2D,2}}$ and (ii) $\mathbf{B_{i,0}} \cdot \mathbf{Y_{2D,1}}$ and $\mathbf{B_{i,0}}\cdot\ \mathbf{Y_{2D,3}}$ are to be computed in a time synchronous scheme. Thus, we need in total 32 multiplexers in total in order to compute all the 16 terms of (6). The multiplexers are commuted time-synchronously, leading to periodic connections being made to the appropriate outputs of Block Q in Fig. 5. The wiring of the multiplexer inputs is governed by (6) and (8).

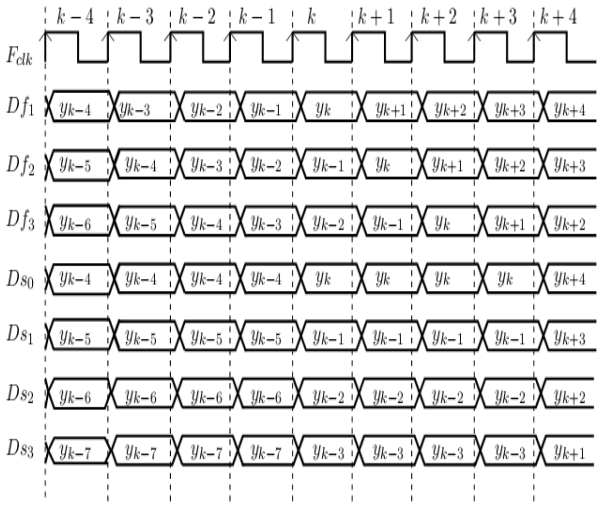

Fig. 6. Timing diagram depicting the register content at each clock cycle of F_{clk} with $k \pmod 4 = 0$. Delay elements Df_α, Ds_α, where $\alpha = 0, 1, 2, 3$ are defined in Fig 4 (Active high, edge triggered).

D. Final Reconstruction Step (FRS)

The cross-wiring form the $\mathbf{B_i \cdot Y_{2D} \cdot B^T}$ block are presented to the FRS stage in order to recover the fixed point coefficients. The computation is completely error free up to the FRS stage. The FRS implementation based on expansion factors α = 4.5958 and 167.2309 proposed in [13] was employed in the proposed hardware implementation. These expansion factors correspond to integer sets [m_1 m_2 m_3] = [12 5 13] and [437 181 473] respectively, with each set satisfying (3). The architecture using expansion factor 167.2309 offers a significant improvement in accuracy when compared to expansion factor 4.5958 [13]. In [13, Sec. IV] a comprehensive account of the FRS implementation is furnished. Such previously described FRS is applied in the proposed architecture.

5. Hardware Implementation and Results

Two designs corresponding to expansion factors 4.5958 and 167.2309 were physically implemented and tested on-chip using field programmable gate array (FPGA) technology and thereafter mapped to 45 nm CMOS technology. We employed a Xilinx ML605 evaluation kit which is populated with a Xilinx Virtex-6 XC6VLX240T FPGA device. The JTAG interface was used to input the test 8X8 2-D DCT arrays to the device from the MATLAB workspace. The measured outputs were returned to the MATLAB workspace.

A. On-chi p Verification using Success Rates

As a figure of merit, we considered the success rate defined as the percentage of coefficients which are within the error limit of $\pm e$ %. For the range $e = \{0.005, 0.01, 0.05, 0.1, 1, 5, 10\}$, the success rates were of precision as given in the Table II. Input word length L was set to 4 and 8 bits. The proposed AI architectures for the FPGA are designed to be overflow-free at each stage throughout the AI encoded structure. The accuracy results obtained are exactly same as the ones for the designs proposed in [13] based on expansion factor FRS. Notice however that up to the FRS all AI encoded 2-D DCT computation is totally error free.

B. Area, Critical Path Delay, and Power Metrics

The design was simulated up to place and route in 45 nm CMOS process using NCSU 45 nm PDK for ASICs and physically implemented using 40 nm CMOS Xilinx Virtex-6 XC6VLX240T FPGA. Then the area (A), critical path delay (T) and power consumption for each implementation were obtained, and are shown in Tables III and IV for FPGA- based physical implementation and ASIC simulation. The ASIC was simulated for power consumption and timing at an operating voltage (V_{DD}) of 1.1 V. Area utilization in mm^2 and the gate count in terms of 2-input NAND gates are also provided. The metrics were measured for different choices of finite precision using input word length $L \in \{4, 8\}$ bits. Similar metrics reported in [13] using the same FPGA device and tools are also given in Table III for comparison purposes.

The area utilization in FPGA implementation is given in Table III in terms of the number of elementary programmable logic blocks (slices) whereas the ASIC equivalent is given in terms of the chip area. The total power consumption of the hardware design is constituted of static (leakage) and dynamic components. Static power consumption in FPGAs is dominated by leakage power of the logic fabric and of the configuration memory. Thus this quantity is essentially independent of the proposed design. Hence, only the dynamic power consumption of the FPGA implementation is given. For the ASIC implementation both leakage and dynamic power components are given in Table IV. The area-time (AT) and area-time-squared (AT^2) metrics are also provided as a measurement of the overall performance. The AT performance is a suitable metric for area efficient designs while the AT^2 metric could be used for designs with the speed of operation as the optimization goal [16]. The proposed architecture leads to a reduction of 23% of AT and 22% of AT^2 (Table III) when compared with the architecture requiring five 1-D DCT cores [13], for designs using the set of integers [437 181 473] with 8-bit input word length. The CMOS design with clock frequency of 951 MHz amounts to a pixel rate of 7.608 G pixels/sec and an 8×8 block rate of 118.875 M blocks/sec.

C. Comparison with Other Architectures

Table V provides a comparison between the existing AI based 2-D DCT architectures and designs based on the pro- posed architecture. Eight-bit versions of designs using expansion factors 4.5958 and 167.2309 corresponding to integer sets[12 5 13] and [437 181 473], respectively, are used for the comparison.

References

[1] C.-J. Lian, Y.-W. Huang, H.-C. Fang, Y.-C. Chang, and L.-G. Chen, "JPEG, MPEG-4, and H.264 codec IP development," in *Proceedings of the Design, Automation and Test in Europe*, vol. 2, Mar. 2005, pp. 1118–1119.

[2] J. F. Blinn, "What's that deal with the DCT?" *Computer Graphics and Applications, IEEE*, vol. 13, no. 4, pp. 78–83, Jul. 1993.
[3] M. N. Yukihiro Arai, Takeshi Agui, "A fast DCT-SQ scheme for images," *The Transactions of the IEICE*, vol. e71, pp. 1095–1097, 1988.
[4] V. S. Dimitrov, G. A. Jullien, and W. C. Miller, "A new DCT algorithm.
[5] V. Oppenheim and C. J. Weinstein, "Effects of finite register length in digital filtering and the fast Fourier transform," *Proceedings of the IEEE*, vol. 60, no. 8, pp. 957–976, Aug. 1972.
[6] J. Cozzens and L. Finkelstein, "Computing the discrete fourier transform using residue number systems in a ring of algebraic integers," *Information Theory, IEEE Transactions on*, vol. 31, no. 5, pp. 580–588, Sep.1985.
[7] R. A. Games, D. Moulin, and J. J. Rushanan, "VLSI design of an algebraic-integer signal processor," in *Proceedings of the 32nd Midwest Symposium on Circuits and Systems*, vol. 2, Aug. 1989, pp. 808–812.
[8] V. Dimitrov, K. Wahid, and G. Jullien, "Multiplication-free 8_8 2D DCT architecture using algebraic integer encoding," *IEE Electronics Letters*, vol. 40, no. 20, pp. 1310–1311, 2004.
[9] S. Mohammadi and A. Javadi, "An efficient technique for error-free implementation of H.264 using algebraic integer encoding," in *International Conference on Signal Acquisition and Processing*, Feb. 2010, pp. 145–150.
[10] Pradini, T. M. Roffi, R. Dirza, and T. Adiono, "VLSI design