Implementation of Multiplierless Ramanujan Ordered Number DCT on FPGA

Geetha K. S. and Uttara Kumari M.

Department of E & CE, R. V. College of Engineering
geethakomandur@gmail.com, dr.uttarakumari@gmail.com

Abstract

An efficient implementation of discrete cosine transform (DCT) computations is presented based on the Ramanujan ordered number DCT (RDCT), a fast multiplierless DCT algorithm. Due to the simple form of the factorized matrices, the derived architecture can be easily constructed from the cascade of only two types of parameterized hardware modules: shifters and adders. The proposed implementations have many features and advantages, including low complexity, high-throughput and regularity. The regularity of RDCT algorithm and careful operation scheduling has resulted in a very efficient implementation of a multiplierless RDCT in Xilinx Spartan3 FPGA in the terms of logic requirements.

Keywords: *Multiplierless, RDCT, hardware implementation*

1. Introduction

The DCT algorithms are often implemented with either direct methods or indirect methods. The indirect methods implement the DCT algorithm indirectly by embedding it in an efficient implementation of the discrete Fourier transform (DFT) algorithm [1-3] or the discrete Hartley transform (DHT) algorithms [4, 5]. The CORDIC architecture [6] has been adopted as an alternative to the conventional arithmetic unit for realizing the multiplication- addition of sine and cosine functions. For example, in [3], an 8-point DCT processor is realized based on the SFG of an 8-point IFFT algorithm, using twenty adders and two multipliers, followed by three CORDIC rotators. In [5], the DCT is first converted into a DHT formulation through the process of data folding. Then the DHT is computed using a CORDIC based systolic array using CORDIC units and adders. In [7], the time-recursive 1-D -point DCT/discrete sine transform (DST) parallel lattice architecture consists of CORDIC modules; and the 2-D DCT/DST parallel architecture consists of CORDIC modules plus some circular shift matrices. Note that in these approaches, the number of arithmetic units used is proportional to the transform length N. Direct methods include various fast DCT algorithms that reduce the computation complexity by factorizing the coefficient matrix into products of simpler matrices [8-13]. For the implementation of 8-point DCT, dedicated data path architectures can be deduced directly from the signal flow graphs (SFG) of the fast DCT algorithms [14-20]. The arithmetic operations involved in these DCT processors are usually realized using either the conventional arithmetic units (multipliers and adders) [14-16] or the ROM-based distributed arithmetic (DA) [17-20]. The DA-based DCT processors require ROM size exponentially increased with transform length, making them only feasible for short-length DCT. In the implementations of 2-D DCT, the row-column method is usually adopted where a sequence of one-dimensional (1-D) DCT computations are first performed row wise, followed by a sequence of 1-D DCT computations performed column wise. If the 8X 8 data block is stored on chip, a transpose memory would be required. Although there are also more regular DCT architectures [21, 22], in general, they would require large amount of arithmetic

processing elements (PEs) and input–output (I/O) channels (in units of words) that are proportional to the transform length N.

A special class of recursive multiplierless transforms for computing Discrete Cosine Transform is introduced in [23, 24]. RDCT computation requires evaluation of cosine angles which are multiples of $2\pi/N$. The algorithm uses Ramanujan ordered Number of degree-2 which is represented as $2^{-l}+2^{-m}$. Thus the cosine functions can be computed by shifts and adds employing Chebyshev type of recursion. With this algorithm, the floating-point multiplication is completely eliminated and hence the multiplierless algorithm can be implemented using shifts and additions only. The factorization of the RDCT transformation matrix maintains the orthogonality and gives a recursive structure. Thus the recursive RDCT reduce the computational complexity. The structure of RDCT shows that the algorithm possesses good regularity.

In this article, we propose a novel direct approach to implement RDCT. We have developed custom-designed hardware implementation for the shifter and adder modules which are the basic building blocks in the hardware implementation of the fast RDCT algorithms. This proposed novel approach exhibits several distinct advantages over existing DCT architectures, the architectures proposed are modular, regular, and admit efficient pipelined implementation. Extensive comparison indicated that this proposed DCT algorithm is better compared with the other DCT implementations.

2. Ramanujan ordered DCT (RDCT)

Computation of DCT requires evaluation of cosine angles which are multiples of $2\pi/N$. If N, the transform length is chosen such that it can be represented as $2^{-l}+2^{-m}$, then the cosine functions which are the DCT kernel can be computed by shifts and adds employing Chebyshev type of recursion. Such integers are called Ramanujan ordered numbers [23]. With this algorithm, the floating-point multiplication is completely eliminated and hence the multiplierless algorithm can be implemented using shifts and additions only. The orthogonality of the recursive DCT kernel is well maintained through matrix factorization to reduce the computational complexity. The inherent parallel structure yields simpler programming and hardware implementation and provides $\frac{3}{2}N\log_2 N - N + 1$ additions and $\frac{N}{2}\log_2 N$ shifts which is very much less complex when compared to other recent multiplierless algorithms.

2.1 2-D RDCT: [23, 24]

The 2-D RDCT is defined based on the 1-D RDCT as follows:

$$r(k_1,k_2) = \frac{4}{N_1,N_2}\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} x(n_1,n_2)w(n_1,k_1)w(n_2,k_2) \qquad (1)$$

Neglecting the scaling factors and using the property of Seperability, we could write the RDCT equation as:

$$r(k_1,k_2) = \sum_{n_1=0}^{N_1-1}\left(\sum_{n_2=0}^{N_2-1} x(n_1,n_2)w(n_2,k_2)\right)w(n_1,k_1) \qquad (2)$$

where $w(n,k)$ represents the RDCT kernel.

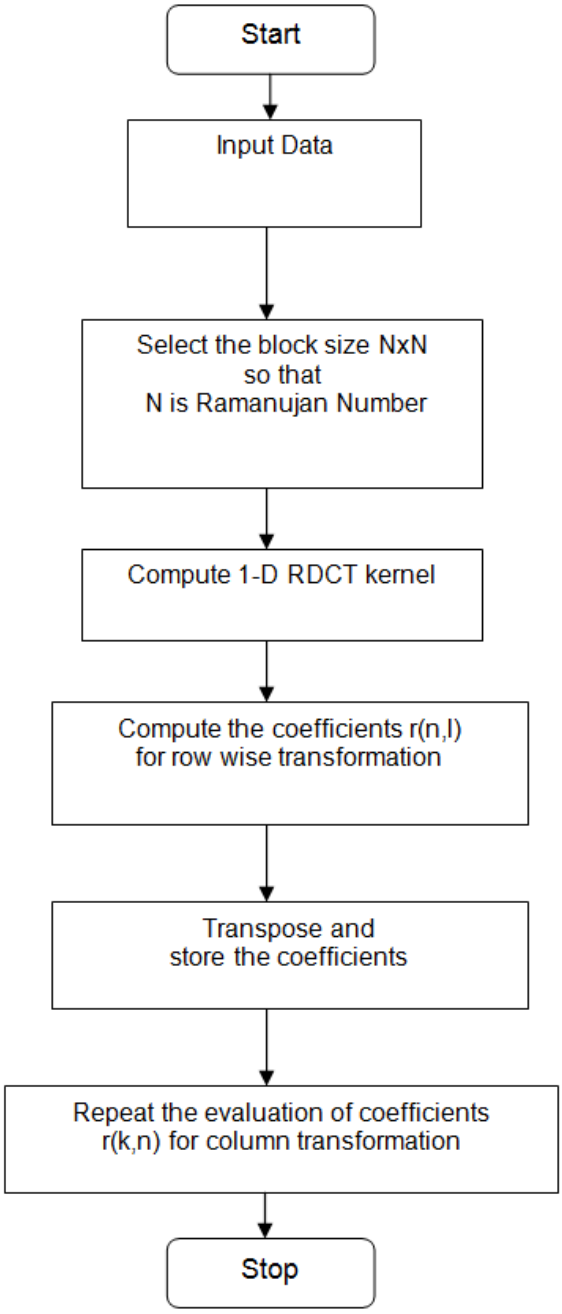

Figure 1. Computational Flow of DCT Coefficients using RDCT

According to the definition of RDCT from we could represent the RDCT kernel as

$$\hat{A}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \qquad (3)$$

Where $a = 1/\sqrt{2}$, $b = \cos(\pi/8)$ and $c = \cos(3\pi/8)$.

In general, we could define the matrix of the recursive kernel as C_N , so that we have the RDCT Coefficient matrix defined as

$$A_N = [P_L].[C_N] \tag{4}$$

and

$$[C_N] = [P_1].\begin{bmatrix} C_{\frac{N}{2}} & 0 \\ 0 & C_{\frac{N}{2}} \end{bmatrix}.\begin{bmatrix} I & 0 \\ 0 & T \end{bmatrix}.[P_2] \tag{5}$$

Where

(1) $[P_1]$ is the permutation matrix to interlace the two halves of the input data sequence as $\tilde{x} = [P_1].x$ where

$$\tilde{x} = \left[x_1, x_{\frac{N}{2}+1}, x_2, x_{\frac{N}{2}+2}, \ldots x_{\frac{N}{2}}, x_N \right] \tag{6}$$

and $x = [x_1, , x_2, \ldots . x_{N-1}, x_N]$.

(2) $[\boldsymbol{T}]$ is a diagonal trigonometric matrix. For a RDCT matrix of length N, $[\boldsymbol{T}]$ is a $M \times M$ matrix, where $M=N/2$.

$$[T] = diag\left[\cos(\theta_m)\right],\ \theta_m = \frac{2\pi(2m+1)}{N}2^{-2} \tag{7}$$
$$m = 0,1,\ldots . M-1$$

N being a Ramanujan Number, the matrix $[\boldsymbol{T}]$ could then be represented as

$$[T] = diag\left(2^{-l_{m1}} + 2^{-l_{m2}}\right)$$

where l_{m1} and l_{m2} are non-negative integers.

(3) $[P_2]$ is an integer coefficient matrix to perform additions and subtractions of the input sequence. For $\tilde{y} = [P_2].y$

Where $$\tilde{y} = \left[y_1 + y_N, y_2 + y_{N-1}, \ldots . y_{\frac{N}{2}-1} + y_{\frac{N}{2}+1}, y_1 - y_N, y_2 - y_{N-1}, \ldots . y_{\frac{N}{2}-1} - y_{\frac{N}{2}+1} \right] \tag{8}$$

(4) $[P_L]_N$ is the product of $\log_2 N - 1$ sparse factor matrices, if the length of the input sequence is N. Each factor matrix is sparse and the non-zero elements are only $\pm 1, 2$.

$$[P_L]_N = \left[P_{L(\log_2 N - 1)} \right]_N \ldots . [P_{L2}]_N . [P_{L1}]_N$$

For length 2N sequence, we have

$$\left[P_L\right]_{2N} = \left[P_{L(\log_2 N-1)}\right]_{2N} \cdot \begin{bmatrix} \left(P_{L(\log_2 N-1)}\right)_N & 0 \\ 0 & \left(P_{L(\log_2 N-1)}\right)_N \end{bmatrix}_{2N} \cdots \begin{bmatrix} \left(P_{L1}\right)_4 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \left(P_{L1}\right)_4 \end{bmatrix} \quad (9)$$

where

$$\left[P_{L(\log_2 N-1)}\right]_N = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \ddots & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 2 & 0 & 0 \\ 0 & 0 & \iddots & 0 & 0 & 0 & \ddots & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}_N \quad (10)$$

for ex.

$$\left[P_L\right]_4 = \left[P_{L1}\right]_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 2 \end{bmatrix}$$

$$\left[P_L\right]_8 = \left[P_{L2}\right]_8 \left[P_{L1}\right]_8 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 2 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 2 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 2 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 2 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix} \quad (11)$$

3. DCT Hardware's Platforms

There are a number of different alternatives for hardware realization of a DCT. The possible selections for digital signal processing system design are, software tools such as the PC Digital Signal Processing Programs (MATLAB, IDL), hardware tools such as Application Specific Integrated Circuits (ASICs), Dedicated Digital Signal Processors DSPs, and the Field Programmable Gate Arrays FPGA e.g., Xilinx, Altera.

3.1 FPGA's

Field Programmable Gate Arrays are newer, more efficient than DSPs system-on-chip configurable design devices that belong to the Programmable Logic Devices (PLDs) family. The first FPGA chip produced to the world was by Xilinx in 1986 (XC2000 family). FPGA devices developed because the PLDs chips could not support the rapid increasing demands for the greater on-chip logic capacity. The drawback of the CPLD chips was that the ratio of

sequential logic resources (flip-flops) compared to combinational logic (logic gates) was small and therefore insufficient to implement many tasks. The basic outline architecture of FPGA devices consists of a number of arrays of logic blocks connected with interconnection bus lines as shown in Figure 2. Sea-of-gate FPGAs consist of a system of logic blocks (flip-flops, gates, look up tables) together with some amount of RAM. FPGAs have embedded processor as well as Giga bit I/O. The configuration of each of the functions of each logic block and its connections to other blocks are given by the configuration bit stream loaded from outside the FPGA device. FPGAs give system designers a broad scale and flexibility for implementing different algorithms.

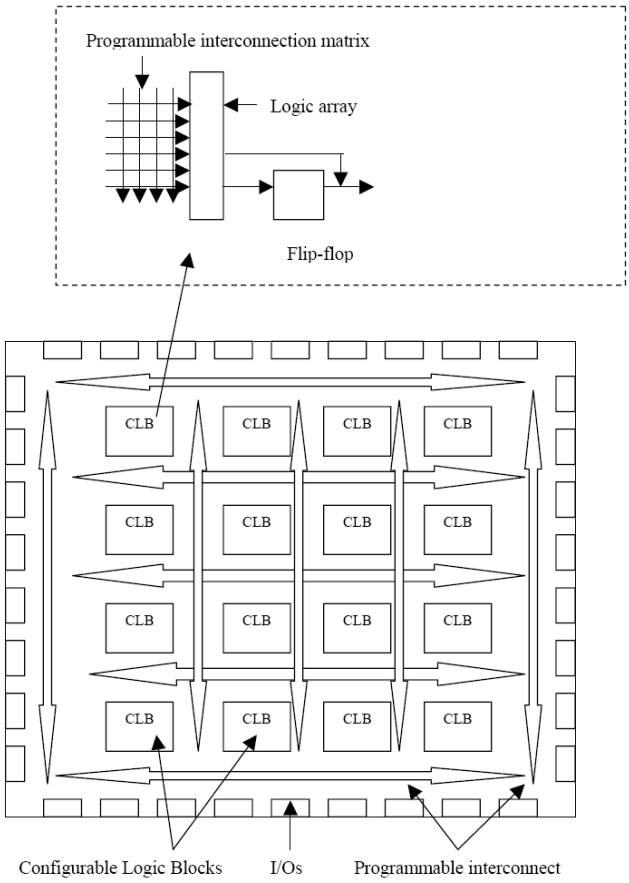

Figure 2. Field Programmable Gate Array (FPGA) Internal Basic Structure

FPGAs have advantages over DSPs, since FPGAs permits parallelism, floating-point operation, and local memory. The parallel reconfigurable technology would have benefits for problems with a parallel nature and when a speed is a requirement for other approaches. FPGAs provides a level of both functional and data specialization. They are also extremely useful in quickly permitting generic prototyping. The ability to keep up-to-date and follow the constantly changing standards in todays advanced technology for example, the latest wireless, multimedia and image processing algorithms require a new system-on-chip technology, such as state of the art re-configurable FPGA hardware. In actual fact, the hardware description languages HDL allows the existing architecture to track the changing standards, removing necessitates to run brand new algorithms on yesterday's dedicated hardware architectures.

4. Design and Development of Implementation

4.1 Basic Block Diagram

The algorithm is implemented for 8 point RDCT. The elements of the DCT kernel are in the form of a sum of multiples of two. This is achieved by using Ramanujan numbers for approximating $2\pi/N$ and using Chebyshev recursion for computing the cosine values (which are now a sum of multiples of two). Since 32 bit data needs to be given eight times as input and also taken as output, Matlab is used to convert input data from real/float to hexadecimal and output data from hexadecimal to real/float to simplify the testing. Xilinx simulator is used for simulation. The flow of the algorithm is as shown in the Figure 3.

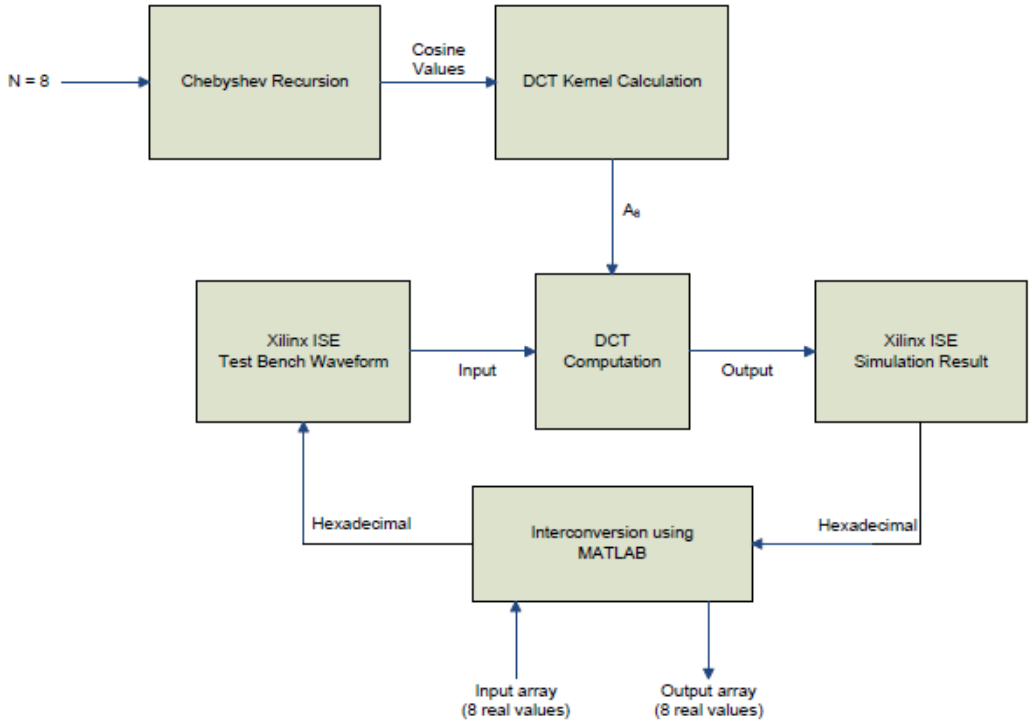

Figure 3. Basic Block Diagram

4.1.1 Recursive Adder with Shifter

The RDCT kernel evaluation handles numbers in the form of power of two and hence there a combination of shift/add operations. We use the 32-bit adder/sub block for the implementation of the addition and subtraction operations. This block is a two input adder with option for subtraction also. A dedicated 32-bit shifter block is used for the shifting operations. The output of the shifter is saturated if there is an overflow or underflow. The shift/add operations have to be computed repeatedly, hence a separate customized block is designed which performs recursive additions of appropriately shifted values of input, to obtain the result. Figure 4 shows the design structure of the recursive adder with shifter.

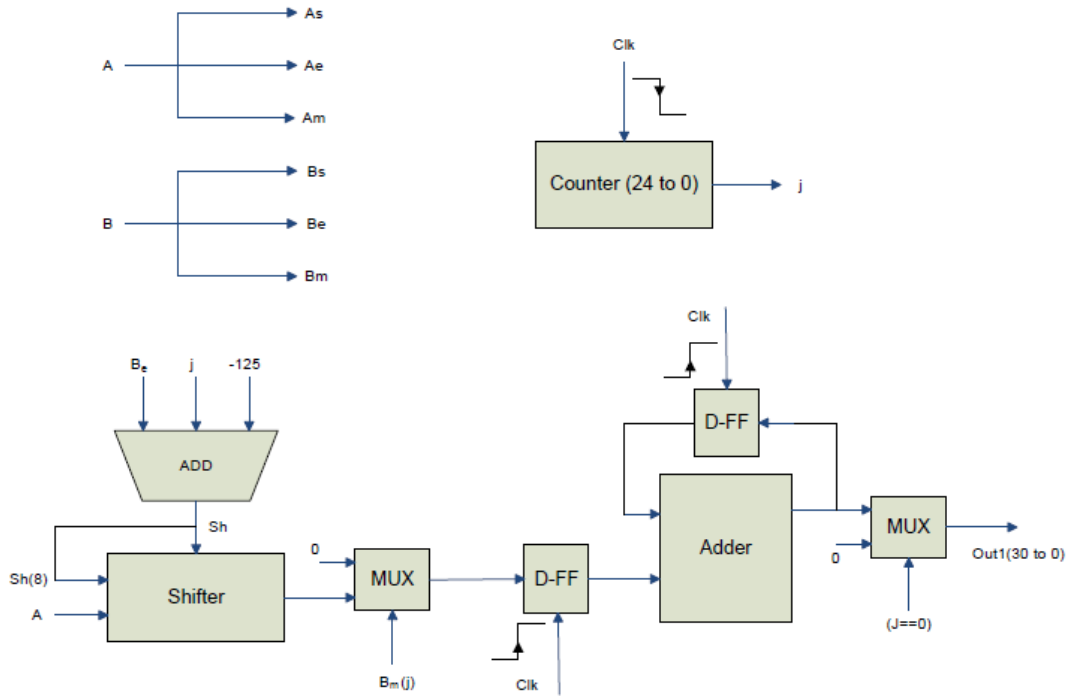

Figure 4. Recursive Adder with Shifter

The simulation waveform is as shown in Figure 5. Simulation results show that one recursive adder with shifter is implemented successfully and it takes 25 clk cycles.

Figure 5. Simulation Results of Recursive Adder with Shifter

A = 0x4023D70A = 2.56, B = 0x3BA3D70A = 0.005 = $2^{-8} + 2^{-10} + 2^{-14} + \ldots$
C = 0x3C51B712 (after 25 clk cycles) = 0.0128 = (A>>8)+(A>>10)+(A>>14)+...

4.1.2 Chebyshev Recursion for Calculation of Cosine Values

According to the flow of the RDCT algorithm explained in Figure 1, the value of x is approximated as $x = (2\pi/N)2^{-2}$ using Ramanujan ordered numbers. The values of x for various values of N are stored in LUT. $\alpha = \frac{x^2}{2!} - \frac{x^4}{4!}$ is then computed and the recursion process is continued. The flow diagram of the Chebyshev recursion is as shown in Figure 6.

i. Approximate value of x is determined by using the concept of Ramanujan numbers.

$$x = \left[\frac{2\pi}{N}\right]2^{-2}$$

x is stored in LUT (Look Up Table) for different values of *N* (here *N*=8). The accuracy of *x* depends on order of Ramanujan number used.

ii. *(1-α)* is calculated where $\alpha = \frac{x^2}{2!} - \frac{x^4}{4!}$ It requires 78 clk cycles to complete the calculation.

iii. After 78 *clk* cycles, initial values are assigned before starting Chebyshev recursion. After every 26 *clk* cycles, a recursion is done to obtain one cosine value. Hence, after 156(6*26) *clk* cycles the 8 cosine values are obtained. Each recursion is given by

$$t_n = 2(1-\alpha)t_{n-1} - t_{n-2}$$

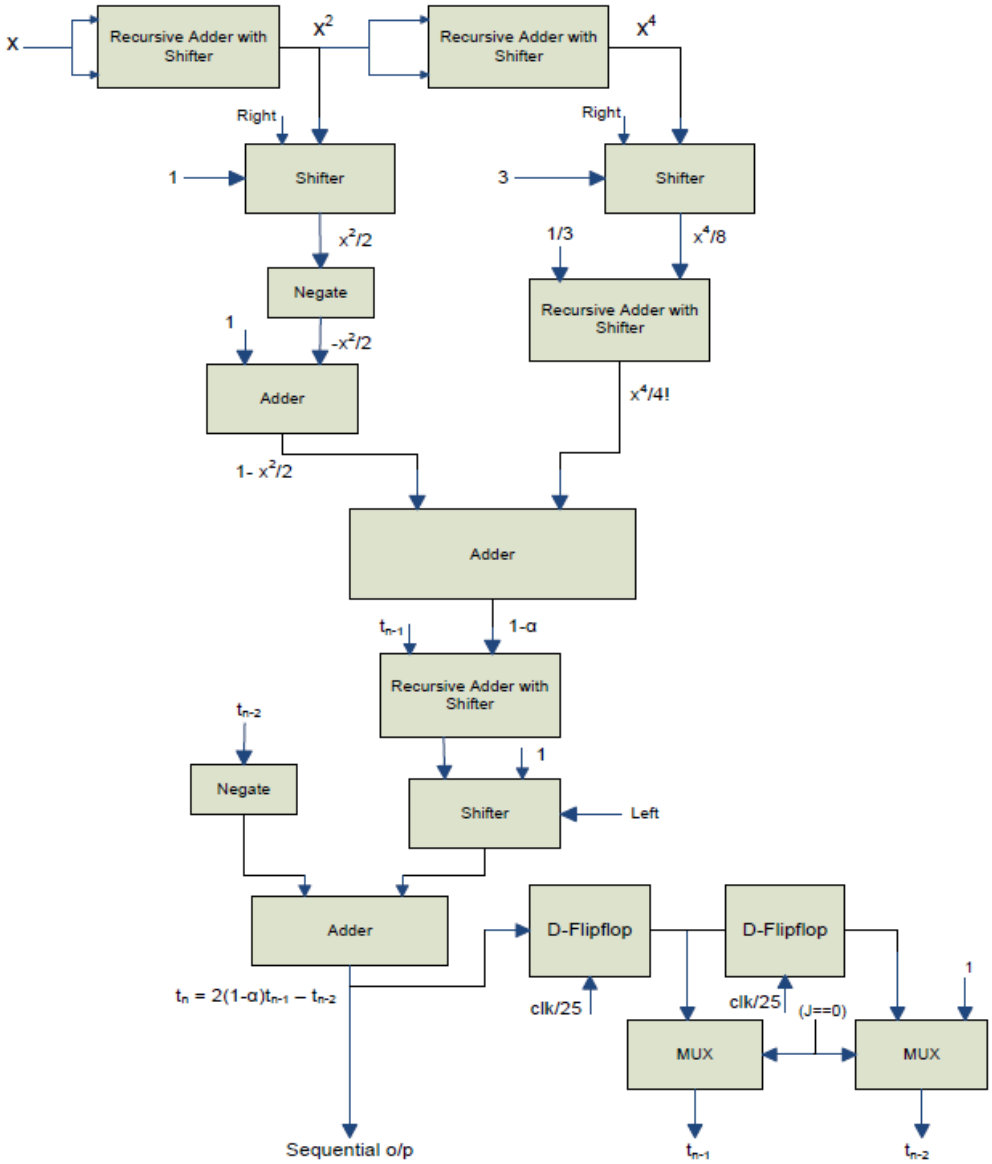

Figure 6. Chebyshev Recursion

The simulation results are as shown in Figure 7.

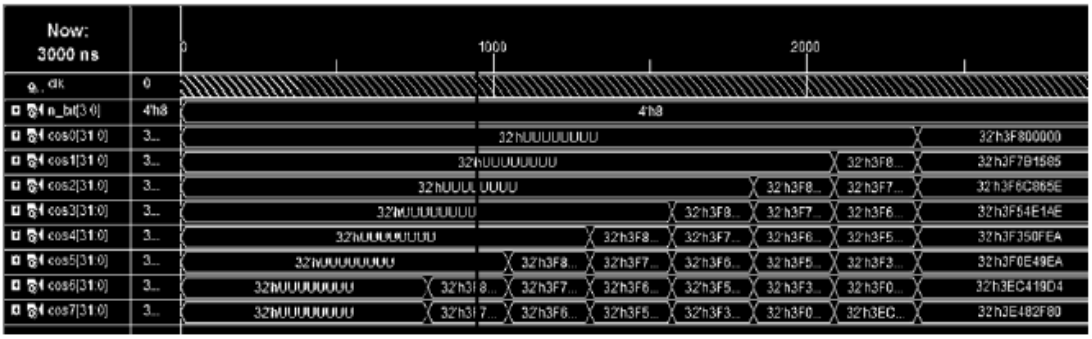

Figure 7. Simulation Results of Chebyshev Recursion

cos0 = 0x3F800000 = 1 = $\cos(0)$

cos1 = 0x3F7B1585 = 0.9808 ≈ $\cos\left(\frac{\pi}{16}\right)$

cos2 = 0x3F6C865E = 0.9239 ≈ $\cos\left(\frac{\pi}{8}\right)$

cos3 = 0x3F54E1AE = 0.8316 ≈ $\cos\left(\frac{3\pi}{16}\right)$

cos4 = 0x3F350FEA = 0.7073 ≈ $\cos\left(\frac{\pi}{4}\right)$

cos5 = 0x3F0E49EA = 0.5558 ≈ $\cos\left(\frac{5\pi}{16}\right)$

cos6 = 0x3EC419D4 = 0.3830 ≈ $\cos\left(\frac{3\pi}{8}\right)$

cos7 = 0x3E482F80 = 0.1955≈ $\cos\left(\frac{7\pi}{16}\right)$

4.1.2 RDCT Kernel Calculation

RDCT kernel is calculated using the orthogonal property of RDCT. According to the Equation 4, we calculate the RDCT kernel using the recursive structure of $[C]_8$. In accordance with Equation 11, the $[P_L]_8$ is computed and the $[A]_8$ can be calculated as

$$[A]_8 = [P_L]_8[C]_8$$

$$= \begin{bmatrix} 1 & 1 & 1 & 1 \\ t_1 & t_3 & t_5 & t_7 \\ t_2 & t_6 & -t_6 & -t_2 \\ (2t_1t_2 - t_1) & (2t_3t_6 - t_3) & (-2t_5t_6 - t_5) & (-2t_2t_7 - t_7) \\ t_4 & -t_4 & -t_4 & t_4 \\ (t_1 - 2t_1t_2 + 2t_1t_4) & (t_3 - 2t_3t_6 - 2t_3t_4) & (t_5 + 2t_5t_6 - 2t_5t_4) & (t_7 + 2t_7t_2 + 2t_7t_4) \\ (2t_2t_4 - t_2) & (-2t_6t_4 - t_6) & (2t_6t_4 + t_6) & (-2t_2t_4 + t_2) \\ (-t_1 - 2t_1t_4 + 4t_1t_2t_4) & (-t_3 + 2t_3t_4 - 4t_3t_4t_6) & (-t_5 + 2t_4t_5 + 4t_4t_5t_6) & (-t_7 - 2t_4t_7 - 4t_2t_4t_7) \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ -A_8(1,3) & -A_8(1,2) & -A_8(1,1) & -A_8(1,0) \\ A_8(2,3) & A_8(2,2) & A_8(2,1) & A_8(2,0) \\ -A_8(3,3) & -A_8(3,2) & -A_8(3,1) & -A_8(3,0) \\ A_8(4,3) & A_8(4,2) & A_8(4,1) & A_8(4,0) \\ -A_8(5,3) & -A_8(5,2) & -A_8(5,1) & -A_8(5,0) \\ A_8(6,3) & A_8(6,2) & A_8(6,1) & A_8(6,0) \\ -A_8(7,3) & -A_8(7,2) & -A_8(7,1) & -A_8(7,0) \end{bmatrix}_{8X8}$$

iv. The cosine values obtained using Chebyshev recursion are represented as t_0 , t_1 ,t_2 ,t_3 ,t_4 ,t_5 ,t_6 and t_7 .

v. Each element is calculated concurrently, using above matrix equation.

4.1.3 RDCT Calculation for the Given Input

The RDCT kernel is now available with its elements being sums of powers of two. Eight inputs (8 real numbers) are now taken whose RDCT is required. The RDCT is calculated using the input elements and the kernel using only shifting and addition operations (because of the form of the elements in the kernel). 512*52 clock cycles are required for the calculations. After the specified number of cycles, eight outputs are available in the hexadecimal format. These values are then ported onto Matlab to view the results in the decimal format.

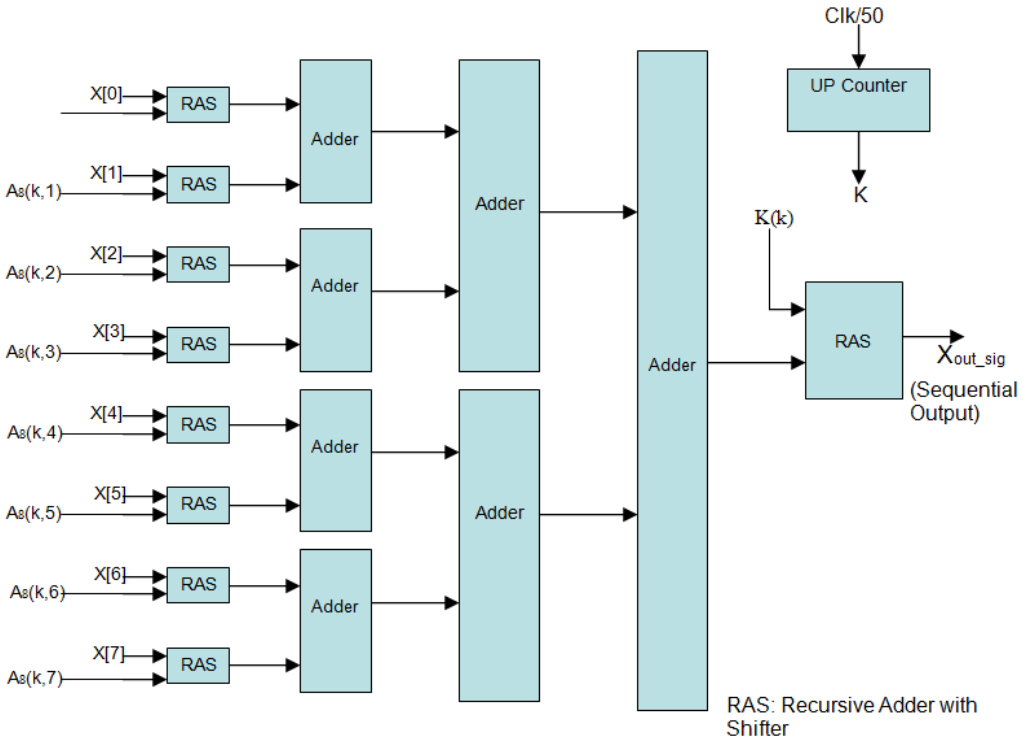

Figure 8. RDCT Calculation for a Given Input

4.2 Implementation Results

The design was implemented on Spartan-3 XC3S400 FPGA board. The device has 8064 logic blocks and a Speed grade of 5 with 4 million gates on board with a maximum operating frequency max of 280MHz.

The simulation results are verified with the matlab results at different stages. The error in the cosine values obtained by Chebyshev recursion is given in Table 1. From the Table 1, it is clear that the error of approximation is validated in the implementation also.

Table 1. Error in Cosine Values

Matlab output	VHDL output	Error (Actual – VHDL)
Cos (π/16) = 0.9808	0.9808	-1.1832 e-05
Cos(3 π /16) = 0.8315	0.8317	-9.8986 e-05
Cos(5π/16) = 0.5556	0.5558	-0.00024510
Cos(7π/16) = 0.1951	0.1955	-0.00040337

The DCT kernel matrices obtained theoretically, by VHDL for RDCT and error between them are given in the Figures 9-10 below.

	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	1
2	0.9808	0.8316	0.5558	0.1955	-0.1955	-0.5558	-0.8316	-0.9808
3	0.9239	0.3830	-0.3830	-0.9239	-0.9239	-0.3830	0.3830	0.9239
4	0.8316	-0.1946	-0.9816	-0.5567	0.5567	0.9816	0.1946	-0.8316
5	0.7073	-0.7073	-0.7073	0.7073	0.7073	-0.7073	-0.7073	0.7073
6	0.5558	-0.9817	0.1954	0.8333	-0.8333	-0.1954	0.9817	-0.5558
7	0.3830	-0.9248	0.9248	-0.3830	-0.3830	0.9248	-0.9248	0.3830
8	0.1955	-0.5563	0.8327	-0.9830	0.9830	-0.8327	0.5563	-0.1955

Figure 9. RDCT Kernel obtained through Xilinx Simulator

	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	1
2	0.9808	0.8315	0.5556	0.1951	-0.1951	-0.5556	-0.8315	-0.9808
3	0.9239	0.3827	-0.3827	-0.9239	-0.9239	-0.3827	0.3827	0.9239
4	0.8315	-0.1951	-0.9808	-0.5556	0.5556	0.9808	0.1951	-0.8315
5	0.7071	-0.7071	-0.7071	0.7071	0.7071	-0.7071	-0.7071	0.7071
6	0.5556	-0.9808	0.1951	0.8315	-0.8315	-0.1951	0.9808	-0.5556
7	0.3827	-0.9239	0.9239	-0.3827	-0.3827	0.9239	-0.9239	0.3827
8	0.1951	-0.5556	0.8315	-0.9808	0.9808	-0.8315	0.5556	-0.1951

Figure 10. DCT Kernel Obtained using Matlab in Built Function (floating -point multiplications)

The error in the values between the Matlab function and the implementation results are tabulated in Figure 11.

	1	2	3	4	5	6	7	8
1	0	0	0	0	0	0	0	0
2	-1.1832e-05	-9.8986e-05	-2.4511e-04	-4.0338e-04	4.0338e-04	2.3175e-04	9.8986e-05	1.1832e-05
3	-4.5748e-05	-3.2612e-04	3.2612e-04	4.5748e-05	4.5748e-05	3.2612e-04	-3.2612e-04	-4.5748e-05
4	-9.8986e-05	-5.1895e-04	7.9510e-04	0.0012	-0.0012	-7.9510e-04	5.1895e-04	9.8986e-05
5	-1.6730e-04	1.6730e-04	1.6730e-04	-1.6730e-04	-1.6730e-04	1.6730e-04	1.6730e-04	-1.6730e-04
6	-2.4523e-04	9.3624e-04	-2.6283e-04	-0.0018	0.0018	2.6283e-04	-9.3624e-04	2.4523e-04
7	-3.2743e-04	9.1520e-04	-9.1520e-04	3.2743e-04	3.2743e-04	-9.1520e-04	9.1520e-04	-3.2743e-04
8	-4.0576e-04	7.6814e-04	-0.0012	0.0022	-0.0022	0.0012	-7.6814e-04	4.0576e-04

Figure 11. Difference between the RDCT Kernel Values obtained by Xilinx Simulator and using Matlab Function

The RDCT kernel is calculated for N=8. The implementation results are analyzed according to the flow of the algorithm where the complexity of Chebyshev Recursion and RDCT Kernel calculation are evaluated for N=8 only.

For Chebyshev Recursion,

- Since (1-α) is equal to $1 - \hat{x}^2/2$, and it needs 2 Adders, 2 Shifters and 3 Recursive Adder with Shifter.

- For calculation of t_0 to t_7, according to Equation 8 of [23], we need 1 Adder, 1 Shifter and 1 Recursive Adder with Shifter.

The number of adders, shifters and the recursive adders with shifter specified are the hardware blocks required for the Chebyshev recursion and one shouldn't interpret these as operations.

For DCT kernel calculation,

- To calculate C4, according to Equation 5, two Recursive Adder with Shifter are needed.
- To calculate C8, according to Equation 5, (after thorough optimization) 04 Recursive Adder with Shifter are required.
- To calculate A8, which is the final RDCT kernel, according to Equation 4, five Adders and 1 Shifter are required.

Table 2 shows clock requirements of each block, and number of sub-blocks (Adders, Shifters and Recursive Adder with Shifter) used for implementation. The operating frequency is 280MHz, and hence it takes 1μs from the time of entering the input to the time of getting the final result. This is considerably fast although the simulation needs to be more optimized. After decomposing Recursive Adder with Shifter, totally 29 Adders and 12 Shifters are required for complete RDCT kernel calculation. This complies with the computational complexity inferred in Chapter 4.

Table 2. Timing and Hardware Requirements

Block	No. of Clock Cycles	No. of Adders	No. of Shifters	No. of Recursive Adder with Shifter
Recursive Adder with Shifter	25	1	1	-
Chebyshev Recursion	235	03	03	04
RDCT Kernel Calculation	299	18	01	04
RDCT Calculation of Input	325	7	0	9

Table 3. Device Utilization Report

Blocks		Recursive Adder with shifter	RDCT kernel	Final RDCT output
No. of Slices	Used	697	896	1147
	Available	3584	3584	3584
	Utilization	19%	25%	32%
No. of 4 input LUT's	Used	1315	3505	3964
	Available	7168	7168	7168
	Utilization	18%	49%	55%
No. of logic cells	Used	325	1961	2678
	Available	8064	8064	8064
	Utilization	4%	24%	33%
No. of Slice Flip-Flops	Used	74	1405	1720
	Available	7168	7168	7168
	Utilization	1%	19%	24%

Table 3 gives the complete implementation results in terms of the utilization of the individual blocks.

5. Conclusions

Ramanujan ordered number DCT (RDCT) was implemented on a target device namely Spartan-3 XC3S400 FPGA. The device is of CMOS technology and has 8064 logic blocks with gate capacity of 4 million. RDCT algorithm consisted of evaluation of coefficients using only shifts and adders, hence existing architecture techniques like the Distributed arithmetic or the CORDIC architecture could not be used. Hence, a dedicated block of Recursive adder with shifter was designed. Synthesis results shows that this block consumed 10.4% area consisting of different blocks on an average. The simulation results show that the block evaluated the output in 25 clock cycles. RDCT kernel calculation involves evaluation of values using Chebyshev recursion which consumed around 29% area on an average including all the blocks. This block takes requires 299 clock cycles. The final output takes 325clock cycles and occupies 38% utilization area. Finally the RDCT coefficients are evaluated at 1 μs (from the time of giving the input to the time of getting the output). The number of shifters and adders required for implementation is same as that discussed in theoretical evaluation of 1-D RDCT. These results clearly show that the RDCT algorithm has a simple structure and exhibits regularity.

References

[1] R. M. Haralick, "Astorage efficientway to implement the discrete cosine transform", IEEE Trans. Comp., vol. 25, **(1976)** July, pp. 764–765.
[2] M. J. Narasimha and A. M. Peterson, "On the computation of the discrete cosine transform", IEEE Trans. Commun., vol. 26, no. 6, **(1978)**, June, pp. 934–936.
[3] S. Yu and E. E. Swartzlander Jr., "A scaled DCT architecture with the CORDIC algorithm", IEEE Trans. Signal Process., vol. 50, no. 1, **(2002)** January, pp. 160–167.
[4] H. Malvar, "Fast computation of discrete cosine transform through fast Hartley transform", Electron. Lett., vol. 22, **(1986)**, pp. 352–353.

[5] J. H. Hsiao, L. G. Chen, T. D. Chiueh and C. T. Chen, "High throughput CORDIC-based systolic array design for the discrete cosine transform", IEEE Trans. Circuits Syst. Video Technol., vol. 5, no. 3, **(1995)** June, pp. 218–225.
[6] J. E. Volder, "The CORDIC trigonometric computing technique", IRE Trans. Eletron. Comput., vol. 8, **(1959)**, pp. 330–334.
[7] J. Chen and K. J. Ray Liu, "A complete pipelined parallel CORDIC architecture for motion estimation", IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process., vol. 45, no. 6, **(1998)** June, pp. 653–660.
[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, no. COM-9, **(1977)** September, pp. 1004–1009.
[9] B. G. Lee, "A new algorithm to compute the discrete cosine transform", IEEE Trans. Acoust., Speech, Signal Process., vol. ASSP-32, no. 6, **(1984)** December, pp. 1243–1245.
[10] H. S. Hou, "A fast recursive algorithm for computing the discrete cosine transform", IEEE Trans. Acoust., Speech, Signal Process., vol. ASSP–35, no. 10, **(1987)** October, pp. 1455–1461.
[11] C. Loeffler, A. Lightenberg and G. Moschytz, "Practical fast 1-D DCT algorithms with 11 multiplications", Proc. IEEE ICASSP, vol. 2, **(1989)** February, pp. 988–991.
[12] E. Feig and S. Winograd, "Fast algorithms for the discrete cosine transform", IEEE Trans. Signal Process., vol. 40, no. 9, **(1992)** September, pp. 2174–2193.
[13] V. Britanak and K. R. Rao, "Two-dimensional DCT/DST universal computational structure for block sizes 2 _ 2", IEEE Trans. Signal Process., vol. 45, no. 11, **(2000)** November, pp. 3250–3255.
[14] Madisetti and A. N. Willson Jr., "A 100 MHz 2-D 8_8 DCT/IDCT processor for HDTV applications", IEEE Trans. Circuits Syst. Video Technol., vol. 5, no. 2, **(1995)** April, pp. 158–165.
[15] Y.-P. Lee, T.-H. Chen, L.-G. Chen, M.-J. Chen and C.-W. Ku, "A costeffective architecture for 8_8 two-dimensional DCT/IDCT using direct method", IEEE Trans. Circuits Syst. Video Technol., vol. 7, no. 3, **(1997)** June, pp. 459–467.
[16] T. Xanthopoulos and A. P. Chandrakasan, "A low-power IDCT macrocell for MPEG2MP@MLexploring data distribution properties for minimal activity," IEEE J. Solid-State Circuits, vol. 34, no. 5, **(1999)** May, pp. 693–703.
[17] M. T. Sun, T. C. Chen and A. M. Gottlieb, "VLSI implementation of a 16X16 discrete cosine transform", IEEE Trans. Circuits Syst., vol. 36, no. 4, **(1989)**, pp. 610–617.
[18] S. Uramoto,Y. Inoue, A. Takabatake, J. Takeda,Y. Yamashita, H. Terane and M.Yoshimoto, "A 100-MHz 2-D discrete cosine transform core processor", IEEE J. Solid-State Circuits, vol. 27, no. 4, **(1992)** April, pp. 492–498.
[19] T. Xanthopoulos and A. P. Chandrakasan, "A low-power DCT core using adaptive bitwidth and arithmetic activity exploring signal correlations and quantization", IEEE J. Solid-State Circuits, vol. 35, no. 5, **(2000)** May, pp. 492–498, 740–750.
[20] S. Yu and E. E. Swartzlander Jr., "DCT implementation with distributed arithmetic", IEEE Trans. Comp., vol. 50, no. 9, **(2001)** September, pp. 985–991.
[21] Y.-T. Chang and C.-L.Wang, "New systolic array implementation of the 2-D discrete cosine transform and its inverse", IEEE Trans. Circuits Syst.Video Technol., vol. 5, no. 2, **(1995)** April, pp. 150–157.
[22] S. B. Pan and R. -H. Park, "Unified systolic arrays for computation of the DCT/DST/DHT", IEEE Trans. Circuits Syst. Video Technol., vol. 7, no. 2, **(1997)** April, pp. 413–419.
[23] K. S. Geetha and M. Uttarakumari, "A new multiplierless discrete cosine transform based on the Ramanujan ordered numbers for image coding", in International Journal of Signal Processing, Image Processing and Pattern Recognition, ISSN: 2005-4254, vol. 3, no. 4, **(2010)** December, pp. 1-14, published by Science & Engineering Research Support Center, Republic of Korea.
[24] V. K. Ananthashayana and K. S. Geetha, "Multiplierless Recursive algorithm using Ramanujan ordered Numbers", IETE Journal of Research, ISSN: 0377-2063, vol. 56, no. 4, **(2010)** July-August, pp. 182-188.