Design and Implementation of JPEG Image Compression and Decompression

K. Deepthi
M.Tech (VLSI System Design)
CVSR College of Engineering, Hyderabad, Andhra Pradesh, India

R. Ramprakash
Assistant Professor, Department of ECE
CVSR College of Engineering, Hyderabad, Andhra Pradesh, India

Abstract - Image compression is very important for efficient transmission and storage of images. Demand for communication of multimedia data through the telecommunications network and accessing the multimedia data through Internet is growing explosively. With the use of digital cameras, requirements for storage, manipulation, and transfer of digital images, has grown explosively. These image files can be very large and can occupy a lot of memory. A gray scale image that is 256 x 256 pixels have 65, 536 elements to store and a typical 640 x 480 color image have nearly a million. The basic objective of image compression is to find an image representation in which pixels are less correlated. Architecture and VHDL design of 2-D DCT, combined with quantization and zig-zag arrangement, is described in this paper. The output of DCT module needs to be multiplied with post-scalar value to get the real DCT coefficients. Post-scaling process is done together with quantization process. The decompression has to revert the transformations applied by the compression to the image data. The decoder therefore takes the compressed image data as its input. It then subsequently applies a Run length decoding [RLD], Inverse zigzag scan [ZZ], dequantization [DQ], inverse discrete cosine transform [IDCT], a color conversion and reordering to it. It then obtains the reconstructed image.

Key words: JPEG, DCT, IDCT.

I .INTRODUCTION

1.1 JPEG Image Compression

Data compression method is different depending on the type of data. For information in the form of images, one of the most popular compression method is JPEG. JPEG stands for Joint Photographic Expert Group. Accordingly widely used in JPEG image included on the internet web pages. Use JPEG create a web page with a picture can be accessed faster than a web page with an image without compression. Color image JPEG compression consists of five steps [1]. This is shown in figure 1. The steps are: color space conversion, down sampling, 2-D DCT, quantization and entropy coding. Grayscale image compression uses only last three steps

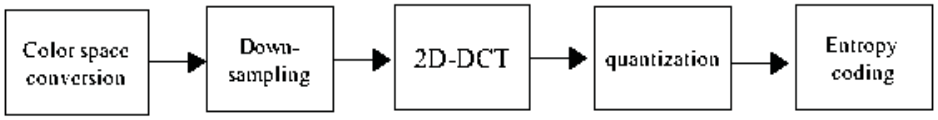


Figure 1. JPEG compression steps of color images

However, this paper's main interest is only on hardware implementation of 2-D DCT combined with quantization and zig-zag process. To achieve high throughput, this paper uses pipelined architecture, rather than single clock architecture designed.

II. DISCRETE COSINE TRANSFORM (DCT)

Using DCT-2D, pixel values of a spatial image in the region will be transformed into a set of DCT coefficients in the frequency region. Before compression, image data in memory is divided into several blocks MCU (minimum code units). Each block consists of 8x8 pixels. Compression operations including DCT-2D in it will be done on each block [5].

Two dimensional DCT, because of its advantage in image compression, is an interesting research subject that invite many researcher [1] ,[2], [3], [4], [7] and others to participate in. That makes many algorithms of DCT is developed.

2.1. 1-D Discrete Cosine Transform (DCT)

There are several ways to compute 1-D DCT. It can be computed with straightforward computation – just multiply input vector by raw DCT coefficients without any algorithm. This method is fast but need large logic utilization, especially multiplier. The other ways is computing DCT with multiplier reduction algorithm [1],[2],[4]. Reduced multiplier means reduced complexity. FPGA chip usually has only a few multipliers. In this case, Spartan-3E XCS500E has only 20 multipliers. This paper adopts the work of Agostini [1] that implemented Arai scaled 1-D DCT algorithm [2]. It means the DCT coefficients produced by the algorithm are not the real coefficients. To get the real coefficients, the scaled ones must be multiplied with post-scaler value. Equation (1) is showing scaled 1-D DCT process.

$$\mathbf{y' = C\ x} \quad (1)$$

Variable **x** is 8 point vector. **C** is Arai's DCT matrix and **y'** is vector of scaled DCT coefficients. To get the real DCT coefficients, y' must be element by element multiplicated with post-scaling factor. It is shown in(2).

$$\mathbf{y = s\ .*\ y'} \quad (2)$$

Constant **s** is vector of post-scaling factor. Element by element multiplication is shown with ".*' operator adopted from MATLAB. Output vector **y** is the real DCT coefficients. The DCT matrix **C** will not be discussed in this paper.

The complete 1D-DCT algorithm is presented in Table. 1, where

- $m1 = \cos(4\pi/16)$
- $m2 = \cos(6\pi/16)$
- $m3 = \cos(2\pi/16) - \cos(6\pi/16)$
- $m4 = \cos(2\pi/16) + \cos(6\pi/16)$

Table 1. 1D – DCT Algorithm from Agostini et.al.[1]

Step 1:	$a_0 = x_0 + x_7$ $a_1 = x_1 + x_6$ $a_2 = x_3 - x_4$ $a_3 = x_1 - x_6$ $a_4 = x_2 + x_5$ $a_5 = x_3 + x_4$ $a_6 = x_2 - x_5$ $a_7 = x_0 - x_7$
ep 2:	$b_0 = a_0 + a_5$ $b_1 = a_1 - a_4$ $b_2 = a_2 + a_6$ $b_3 = a_1 + a_4$ $b_4 = a_0 - a_5$ $b_5 = a_3 + a_7$ $b_6 = a_3 + a_6$ $b_7 = a_7$
Step 3:	$d_0 = b_0 + b_3;$ $d_1 = b_0 - b_3;$ $d_2 = b_2;$ $d_3 = b_1 + b_4;$ $d_4 = b_2 - b_5;$ $d_5 = b_4;$ $d_6 = b_5;$ $d_7 = b_6;$ $d_8 = b_7;$
Step 4:	$e_0 = d_0;$ $e_1 = d_1;$ $e_2 = m_3 * d_2;$ $e_3 = m_1 * d_7;$ $e_4 = m_4 * d_6;$ $e_5 = d_5;$ $e_6 = m_1 * d_3;$ $e_7 = m_2 * d_4;$ $e_8 = d_8;$

Step 5:	f0 = e0;
	f1 = e1;
	f2 = e5 + e6;
	f3 = e5 - e6; f4 = e3 + e8; f5 =

$$s = \begin{pmatrix} c7/c6 \\ c6/c4 \\ c5/c2 \\ c4 \\ c3/c2 \\ c2/c4 \\ c1/c6 \end{pmatrix} \quad (3)$$

2.2. Two Dimensional Discrete Cosine Transform (2D- DCT)

2D-DCT is made from two 1D-DCT. Using DCT matrix, scaled 2D-DCT operation was shown in (4).

$$\mathbf{Y' = [C]\,[X]\,[C]^T} \quad (4)$$

X = 8x8 input matrix
Y' =Scaled 2D-DCT 8x8 output matrix
C = DCT matrix, same as matrix in equ. 1.

To get 2D-DCT real value, Y' must be element by element multiplied by post-scaler as shown in (5). Now, the post-scaler is in matrix form.

$$\mathbf{Y = [S].*[Y']} \quad (5)$$

with S = post scaler matrix and Y = real DCT coefficients. Post scaler matrix **S** is computed in (6).

$$\mathbf{S = s\ s^T} \quad (6)$$

2.3. Quantization

Quantized output is made by divide each DCT coefficient by quantization value. It is done by doing element by element matrix multiplication between DCT output and Quantization matrix as shown in (7) and (8).

$$\mathbf{Yq = [Q]\ .*[Y]} \quad (7)$$
$$\mathbf{Yq = Q\ .*\ [S].*[Y']} \quad (8)$$

Since the image used in this paper is only grayscale picture, the quantization matrix applied is only for the luminance. The matrix was modified from [6] and post- scaled by matrix S then multiplied with 2^{12} to produce integer number in order to be applied in VHDL.

2.4. Zig-zag process

Quantized output is sent sequentially byte-by-byte in zig-zag pattern. Zig-zag operation is done for every 8X8 block. The pattern is shown in figure 2 [6].

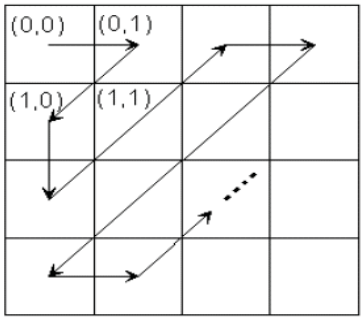


Figure 2. Zig-zag process

2.5. Run Length Encoding

Run-length encoding for a data sequence having frequent runs of *zeros*. Each time a zero is encountered in the input data, *two* values are written to the output file. The first of these values is a zero, a flag to indicate that run-length compression is beginning. The second value is the number of zeros in the run. If the average run-length is longer than two, compression will take place. On the other hand, many single zeros in the data can make the encoded file larger than the original.

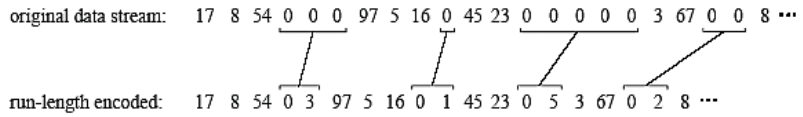


Figure 3. Run Length Encoding process

III. DECOMPRESSION

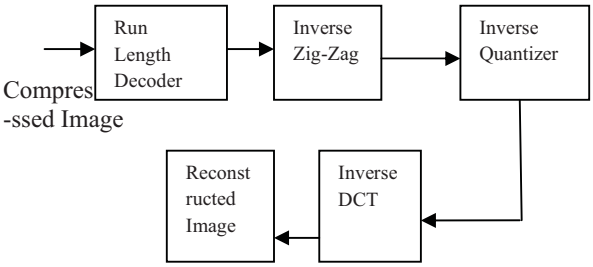


Figure 4. Decompression process

The decoder therefore takes the compressed image data as its input. It then subsequently applies a Run length decoding [RLD], Inverse zigzag [ZZ], dequantization [DQ], inverse discrete cosine transform [IDCT], then obtains the reconstructed image.

1.2 Run Length Decoding

Run Length Decoding will perform the reverse process of run length encoding, which takes the run length encoded data as its input and produces the original data stream as its output.

For example, the input: '1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16' will yield the output:
'1,1,2,1,3,1,4,1,5,1,6,1,7,1,8,1,9,1,10,1,11,1,12,1,13,1,14,1,15,1,16,1'.

1.3 Inverse Zig-Zag Order

This will perform the Inverse zig- zag operation . Inverse zig-zag operation is done for every 8X8 block.

1.4 Inverse Quantization

Dequantization (inverse quantization) can be referred as,

$$\mathbf{Y = QO \times QS}$$

where QO is the quantized value, QS is the step size, and Y is the dequantized value.

3.4. Inverse Discrete Cosine Transform

The IDCT unit in the image processor should realize the inverse discrete cosine transform. It therefore takes one block as its input. It then applies an inverse discrete transformation with an 8-bit precision to it. The mathematical definition of the IDCT is given in appendix B.1. After computation of the IDCT, the signed output samples are level-shifted. This level shifting converts the output to an unsigned representation. For 8-bit precision, the level shift is performed by adding 128 to every element of the block that came out of the IDCT.

$$\mathbf{N = round(T' Y T) + 128}$$

IV.FPGA IMPLEMENTATION

4.1. Compression and Decompression

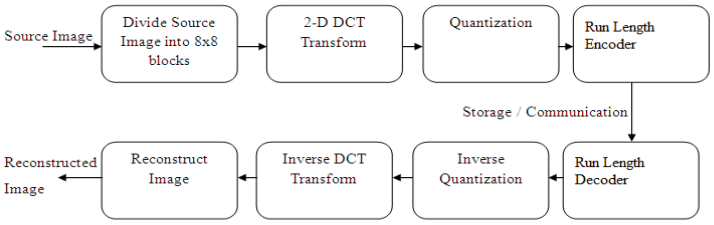


Figure 5. Block representation of entire system

Block diagram of entire system to be implemented in FPGA is shown in figure 3. Input data is inserted into the system every 8 bit sequentially. Actually, many DCT designs insert the input to the DCT in parallel. For example is 8 x 8 bit [1],[3],[4]. This is ideal for DCT computing because it only consumes a clock cycle to insert data to 1D-DCT unit. With sequential manner, it takes 8 clock cycles to insert a set of data (8 points) to the DCT unit. The sequential architecture is chosen to save I/O port in FPGA chip. Some 2D-DCT intellectual property designs from Xilinx also use 8-bit input.

4.2. System Architecture

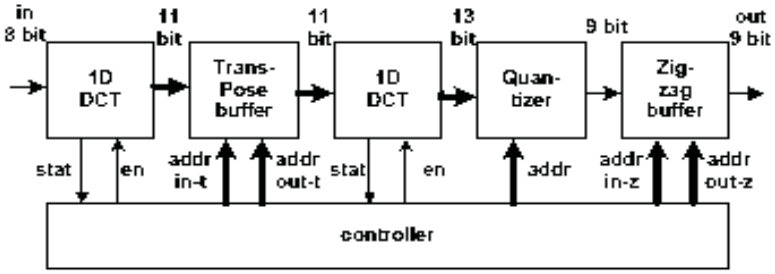


Figure 6. System Architecture

The 2D-DCT architecture, combined with zigzag and quantization used in this paper is shown in Fig. 4. The 2D-DCT module construction is modified from [7], that also put the data sequentially into the module. Thus, the architecture of 2D-DCT was divided into two 1D DCT modules and one transpose buffer. The two 1D DCT modules are similar but the bit widths at each module are different. The transpose buffer operates like a temporal barrier between the first and the second 1D DCT. It made from static RAM with two sets of data and address bus. One for read process and the other for write.

4.3. 1D- DCT pipeline process

Algorithm defined in table 1 from [1] is used to compute 1D-DCT. The algorithm itself has 6 steps. Since the DCT input/output has 8 points and data has to be entered and released in sequential manner, it takes 8 clock cycles for each input and output process. Totally, 8 points 1D-DCT computation needs 22 clock cycles. Design for data input and output in this paper is inspired by design from [7]. The input and output process visualization is shown in figure 5. The difference from system [1] is that it computes every single operation in a clock cycle, so every step needs 8 – 9 clock cycles. In this paper, system computes every step in a clock cycle, so DCT computation can be done faster.

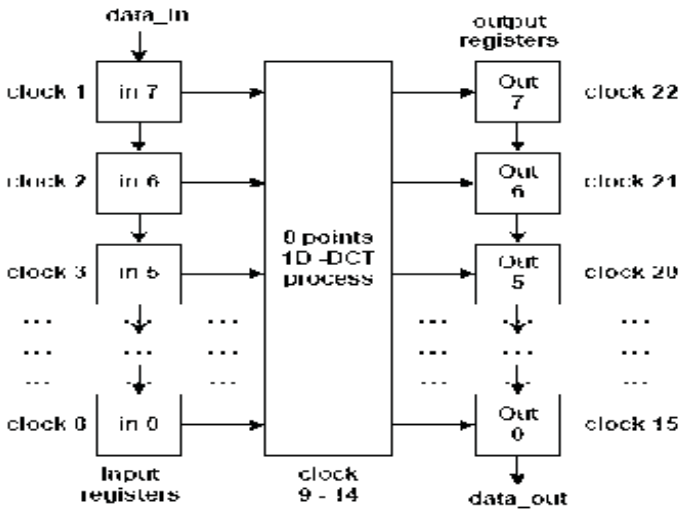


Figure 7. DCT pipelining process

1D-DCT computation is done in pipeline process. The pipeline process is made in three stages. Thus, one pipeline stage is done in 8 clock cycles

4.4. Transpose Buffer

Transpose buffer is static RAM, designed with two set of data and address bus. It has input and output data-address buses. The module block symbol is shown in fig. Data input come from output of first 1D-DCT. Address in, out, and WE (write enable) are generated from controller module. Input address is generated in normal sequence (0,1,2,3,4,5,6, …, 63) but output address is generated in transposed sequence (0,8,16,24,32,40,48,56,1,9,17,..,55, 63).

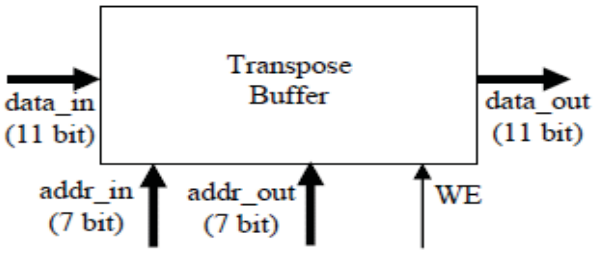


Figure 8. Transpose Buffer

4.5. Quantizer

Originally, quantization process in JPEG compression is done by divide every 2D-DCT coefficient by quantizing value from quantization table. To apply the operation in VHDL, division is converted into multiplication. The modified quantization table will be post-scaled with some post-scaling table. The post-scaled quantization table, written in matrix form in (9), is the table that will be applied to the VHDL code.

To implement the quantization process, the output of 2D-DCT is multiplied by quantization value. The value is generated by quantization ROM that made for store the quantizing-post scaling value. Block Diagram of the implementation is shown.

F. Zig - Zag Buffer

Like transpose buffer, zigzag buffer is made from static RAM. Its construction is like transpose buffer. It has two sets of data – address bus. Input address bus is accessed by normal sequence, but output address is given some zigzag sequence described in figure 2. Zigzag address is generated by a zigzag ROM. The sequence is

stored or preprogrammed in the ROM. When the ROM address bus is accessed by normal address sequence, ROM data bus will emit zigzag value.

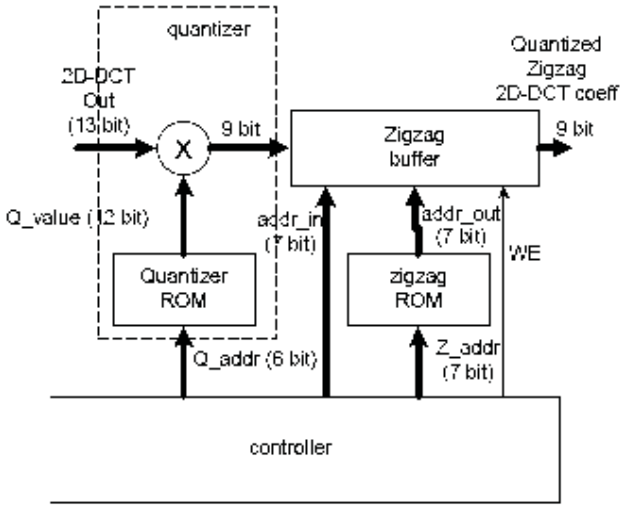


Figure 9. describes zigzag buffer and ROM construction in the system.

is accessed by normal address sequence, ROM data bus will emit zigzag value. Figure 9 describe zigzag buffer and ROM construction in the system.

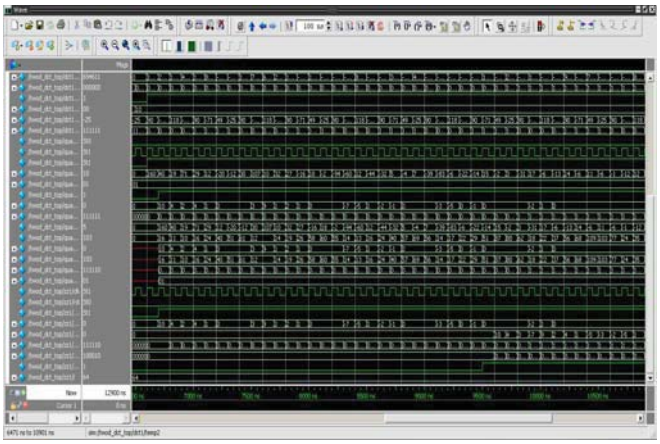
Implementation Result

The 2-D DCT architecture was described in VHDL. This VHDL was synthesized into an Xilinx Spartan 3E family FPGA [8]. Nevertheless, there is a requirement to use this architecture. The FPGA must have at least 11 multipliers, because the system utilizes only internal multiplier. System is tested with real photographic image. Simulation results give the quantized 2D DCT coefficients. To verify the numerical result, the system is given by data from grayscale picture. Data from the picture was converted to VHDL test bench. The result then compared to the result from MATLAB computation.

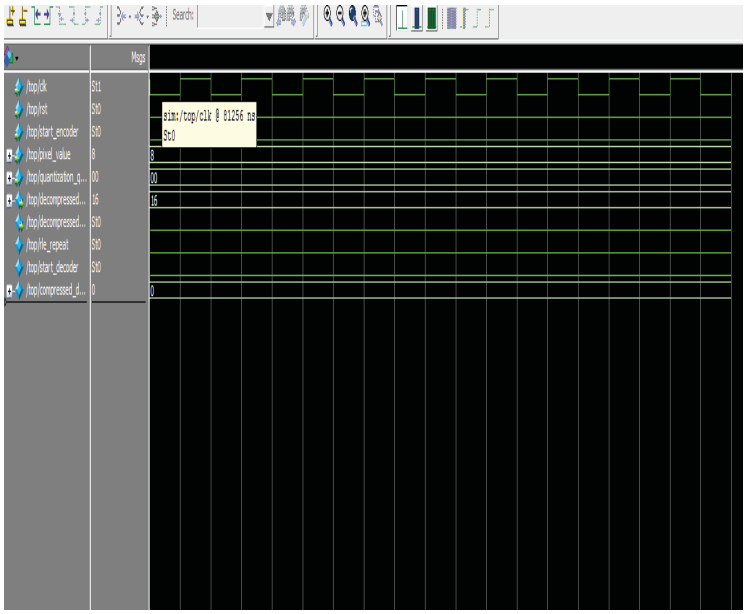
Generally, mean squared error between VHDL and non- rounded MATLAB result is 0.060552 for 64 data. The complete synthesis results to Spartan-3E FPGA are presented, whose hardware was fit in an XCS500E device. System designed in [3] uses Virtex FPGA and straightforward multiplication without special algorithm.

V. SIMULATION RESULTS

1. Encoder



2. Decoder



VI. SYNTHESIS RESULTS

dct Project Status			
Project File:	twodct..xise	Parser Errors:	No Errors
Module Name:	dct	Implementation State:	Synthesized
Target Device:	xc3s100e-5vq100	• Errors:	No Errors
Product Version:	ISE 12.4	• Warnings:	60 Warnings (18 new)
Design Goal:	Balanced	• Routing Results:	
Design Strategy:	Xilinx Default (unlocked)	• Timing Constraints:	
Environment:	System Settings	• Final Timing Score:	

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	920	960	95%
Number of Slice Flip Flops	1540	1920	80%
Number of 4 input LUTs	1857	1920	96%
Number of bonded IOBs	24	66	36%
Number of MULT18X18SIOs	4	4	100%
Number of GCLKs	1	24	4%

Detailed Reports					
Report Name	Status	Generated	Errors	Warnings	Infos
Synthesis Report	Current	Tue Oct 4 19:58:51 2011	0	60 Warnings (18 new)	3 Infos (0 new)

VII. CONCLUSION AND FUTURE SCOPE

2D-DCT combined with quantization and zigzag buffer is designed using VHDL. System is tested with real grayscale image. In this paper, a new fully parallel architecture based on row-column decomposition has been proposed for the computation of the 2D DCT. The system involves no memory transposition, and is highly modular and utilizes a highly parallel structure to achieve high-speed performance. Due to its widely identical units, it will be relatively easy to implement and very suited to VLSI implementation. It uses two identical units for the computation of the row and column transforms and arrays of shift registers to perform the transposition operation. As compared to a pipelined regular architecture, the proposed architecture achieves the same throughput rate at much lower hardware cost and communication complexities. It is also worth mentioning that in the proposed design, the same architecture can be used for the computation of both the forward and the inverse 2D DCT.

The aforementioned attributes of the DCT have led to its widespread deployment in virtually every image/video processing standard of the last decade, for example, JPEG (classical), MPEG-1, MPEG-2, MPEG-4, MPEG-4 FGS, H.261, and H.263. Nevertheless, the DCT still offers new research directions that are being explored in the current and upcoming image/video coding standards.

REFERENCES

[1] L. Agostini, S. Bampi, “Pipelined Fast 2-D DCT Architecture for JPEG Image Compression” Proceedings of the 14th Annual Symposium on Integrated Circuits and Systems Design, Pirenopolis, Brazil. IEEE Computer Society 2001. pp 226-231.

[2] Y. Arai, T. Agui, M. Nakajima. “A Fast DCT-SQ Scheme for Images”. Transactions of IEICE, vol. E71, nÂ. 11, 1988, pp. 1095-1097.

[3] D. Trang, N. Bihn, “A High-Accuracy and High-Speed 2-D 8x8 Discrete Cosine Transform Design”. Proceedings of ICGC-RCICT 2010, vol. 1, 2010, pp. 135-138

[4] I. Basri, B. Sutopo, “Implementasi 1D-DCT Algoritma Feig- Winograd di FPGA Spartan-3E (Indonesian)”. Proceedings of CITEE 2009, vol. 1, 2009, pp. 198-203

[5] E. Magli, “The JPEG Family of Coding Standard,” Part of “Document and Image Compression”, New York: Taylor and Francis, 2004.

[6] Wallace, G. K. ,"The JPEG Still Picture Compression Standard", Communications of the ACM, Vol. 34, Issue 4, pp.30-44. 1991.

[7] Sun, M., Ting C., and Albert M., ‘‘VLSI Implementation of a 16 X 16 Discrete Cosine Transform’’, IEEE Transactions on Circuits and Systems, Vol. 36, No. 4, April 1989.

[8] Xilinx, Inc., “Spartan-3E FPGA Family : Data Sheet ”, Xilinx Corporation, 2009.

[9] Omnivision, Inc., “OV9620/9120 Camera Chip Data Sheet ”, Xilinx Corporation, 2002.

[10] Xilinx, Inc., “2D Discrete Cosine Transform (DCT) V2.0 ”, Logicore Product Specification, Xilinx Corporation, 2002.