An Efficient Low Area Implementation of 2-D DCT on FPGA

Atakan Doğan

Anadolu University, Electrical and Electronics Engineering, Eskişehir, Turkey
atdogan@anadolu.edu.tr

Abstract

This paper presents the design and implementation for 2-D discrete cosine transform (DCT) with the goal of achieving low area utilization and high-speed operation on FPGAs. The design is based on the row-column decomposition technique, which requires two successive 1-D DCT transforms and a transpose memory between them for storing and transposing the results of the first 1-D DCT. The proposed implementation of 2-D DCT is capable of compressing at least 70 images per second in 720x480 resolution on Xilinx Spartan 3E and 30 images per second in 1920x1080 resolution on Xilinx Virtex 7 FPGA. Consequently, the proposed 2-D DCT design and implementation can be very useful in various image and video compressing applications.

1. Introduction

The discrete cosine transform (DCT) plays a key role in JPEG for still picture compression [1], ITU H.261 [2] for teleconferencing, and MPEG for multimedia applications [3]. For example, in JPEG baseline encoder, an input image is split into non-overlapping blocks of 8×8 pixels, the pixel values are level shifted from unsigned integer to signed integer, and then 2-D DCT computation is performed on each block.

Among the various architectures and algorithms proposed for the computation of 2-D DCT, a popular approach is the *row-column decomposition method* [4-10]. Its popularity can be attributed to the following facts: (i) It is based on the separability property of 2-D DCT and enables the computation of 2-D DCT by using two successive 1-D DCT transforms. (ii) It requires a control logic with lower complexity due to its regularity and modularity. (iii) It reduces the computational complexity of 2-D DCT by a factor of four. For an 8x8 input matrix, 2-D DCT algorithm requires 4096 multiplication and 4096 addition operations. The row-column decomposition method, on the other hand, only needs 1024 multiplication and 1024 addition operations.

In this study, the 2-D DCT architecture introduced by [4] for an ASIC implementation is adopted, and is modified for a low area implementation on FPGA. There are of course several reasons why [4] is chosen to be implemented on FPGA: (i) It exploits the row-column decomposition. Thus, using single 1-D DCT core in a time-shared manner is expected to result in low area utilization. (ii) It uses a shift-register based transpose buffer that saves block RAM resources. (iii) The control logic can be distributed among its components, which results in simpler finite state machines.

The rest of the paper is organized as follows. Section 2 presents details of the proposed 2-D DCT architecture. Section 3 gives the implementation results and compares against other cores from the literature. Finally, Section 4 concludes the paper.

2. 2-D DCTArchitecture

The overview of the proposed FPGA implementation of 2-D DCT architecture is shown Fig. 1, which is inspired by [4] . With respect to Fig. 1, the main components include ping buffer, pong buffer, 1-D DCT, transpose buffer, and output buffer. It should be noted here that the modifications to [4] include pong buffer operation, inclusion of a pipeline register and rounding logic in 1-D DCT, output buffer, and a stoppable pipeline. Common to these components are their input and output interfaces, which are similar to writing into or reading from a FIFO buffer and described as follows:

- Input interface: A component consumes a new data word on *writeData* bus in the next rising edge of clock signal if *writeEn* is asserted while *full* is not asserted during the current clock period. Thus, the asserted *full* signal indicates that the component cannot currently accept a new data word.
  - *writeData*, input, 8-, 12-, or 96-bit
  - *writeEn*, input, 1-bit
  - *full*, output, 1-bit
- Output interface: A component produces a new data word on *readData* bus in the next rising edge of clock signal if *readEn* is asserted while *empty* is not asserted during the current clock period. Note that if *empty* is not asserted, there is a valid data word available on *readData* bus; otherwise, component cannot provide a new data word in the current clock cycle.
  - *readData*, output, 12-, or 96-bit
  - *readEn*, input, 1-bit
  - *empty*, output, 1-bit

As a result of their aforementioned input and output interfaces, four main components are seamlessly connected in Fig. 1 as follows:

- *readData* (Output)→*writeData* (Input): Data bus between producer and consumer components is established.
- *empty* (Output)→*inverter*→*writeEn* (Input): Producer component can write into consumer component.
- *readEn* (Input)←*inverter*←*full* (Output): Consumer component can read from producer component.

After the detailed description of input and output interfaces, how each of five components contributes to the computation of 2-D DCT will be explained in the following sections.

2.1. Ping Buffer

Ping buffer is basically a 96-bit shift-register whose operation is controlled by a two-state {*empty*, *full*} finite state machine (FSM) as follows:

- *empty* (serial-in): If *writeEn* is asserted, a new 12-bit word is serially shifted into ping buffer. Once the eighth

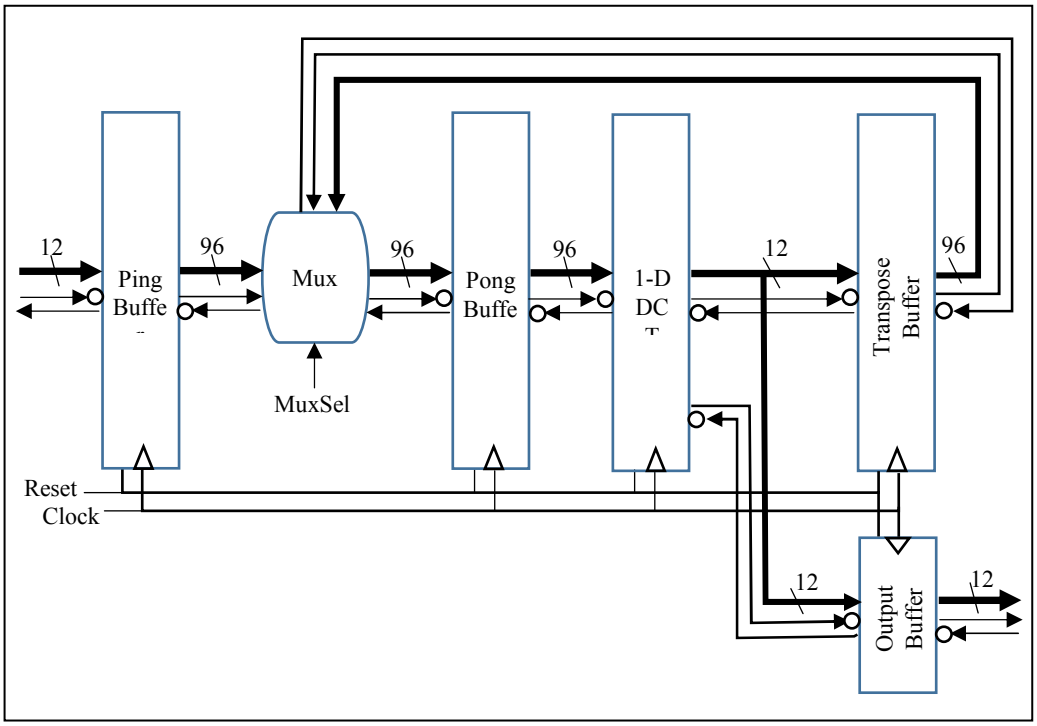

Fig. 1. Architecture of 2-D DCT hardware

12-bit word is inserted into the buffer, FSM goes to the other state. As a result, ping buffer requires at least eight clock cycles to become full. In *empty* state, *full* and *empty* output signals are deasserted and asserted, respectively.

- *full* (parallel-out): If *readEn* is asserted, 96-bit current state of ping buffer is shifted out and FSM goes to the other state. Thus, ping buffer becomes in *empty* state again. In *full* state, *full* and *empty* output signals are asserted and deasserted, respectively.

Consequently, it takes a total of 64+8=72 clock cycles for an 8x8 matrix of pixels, where 64 cycles are spent for loading all elements of 8x8 matrix into the buffer, and 8 cycles are needed for transferring 8x8 matrix row by row to pong buffer.

2.2. Pong Buffer

Pong buffer is simply a 96-bit register whose operation is managed by a four-state {*oned_dct_empty*, *oned_dct_full*, *twod_dct_empty*, *twod_dct_full*} finite state machine as follows:

- *oned_dct_empty* (parallel-in): If *writeEn* is asserted, a new 96-bit word (a row of eight pixels) is loaded and FSM goes to *oned_dct_full* state. *MuxSel* signal is not asserted so as to load from ping buffer. In this state, *full* and *empty* output signals are deasserted and asserted, respectively.
- *oned_dct_full* (coefficient computation): A new 1-D DCT coefficient is computed based on the current state of pong buffer in every clock cycle. Since there are eight pixels per row, the machine stays here only for eight clock cycle. At the end of the eighth clock cycle, during which the last coefficient for a row is being computed, it goes to *twod_dct_empty* state if the eighth row is being processed; otherwise, it makes a transition to *oned_dct_empty* state. In this state, *full* and *empty* output signals are asserted and deasserted, respectively.
- *twod_dct_empty* (parallel-in): This state is similar to *oned_dct_empty* state except that *MuxSel* signal is asserted in order to load from transpose buffer (a column of eight 1-D DCT coefficients) instead of ping buffer.
- *twod_dct_full* (coefficient computation): This state is similar to *oned_dct_full* state except that it goes to either *oned_dct_empty* state if the eighth column is being processed, or *twod_dct_empty* state at the end of the eighth clock cycle.

With respect to the pong buffer operation, 64+8=72 clock cycles are required for the computation of 1-D and 2-D DCT coefficients. As a result, pong buffer completes the processing of 8x8 matrix of pixels in 144 clock cycles. Furthermore, ping and pong buffers together introduces (72+144)-8=208 clock cycles of latency.

2.3. 1-D DCT

According to [9], eight-point 1-D DCT can be computed based on the row-column decomposition technique as follows:

$$z_0 = d(x_0 + x_7) + d(x_1 + x_6) + d(x_2 + x_5) + d(x_3 + x_4)$$
$$z_2 = b(x_0 + x_7) + f(x_1 + x_6) - f(x_2 + x_5) - b(x_3 + x_4)$$
$$z_4 = d(x_0 + x_7) - d(x_1 + x_6) - d(x_2 + x_5) + d(x_3 + x_4)$$
$$z_6 = f(x_0 + x_7) - b(x_1 + x_6) + b(x_2 + x_5) - f(x_3 + x_4)$$

$$z_1 = a(x_0 - x_7) + c(x_1 - x_6) + e(x_2 - x_5) + g(x_3 - x_4)$$
$$z_3 = c(x_0 - x_7) - g(x_1 - x_6) - a(x_2 - x_5) - e(x_3 - x_4)$$
$$z_5 = e(x_0 - x_7) - a(x_1 - x_6) + g(x_2 - x_5) + c(x_3 - x_4)$$
$$z_7 = g(x_0 - x_7) - e(x_1 - x_6) + c(x_2 - x_5) - a(x_3 - x_4)$$

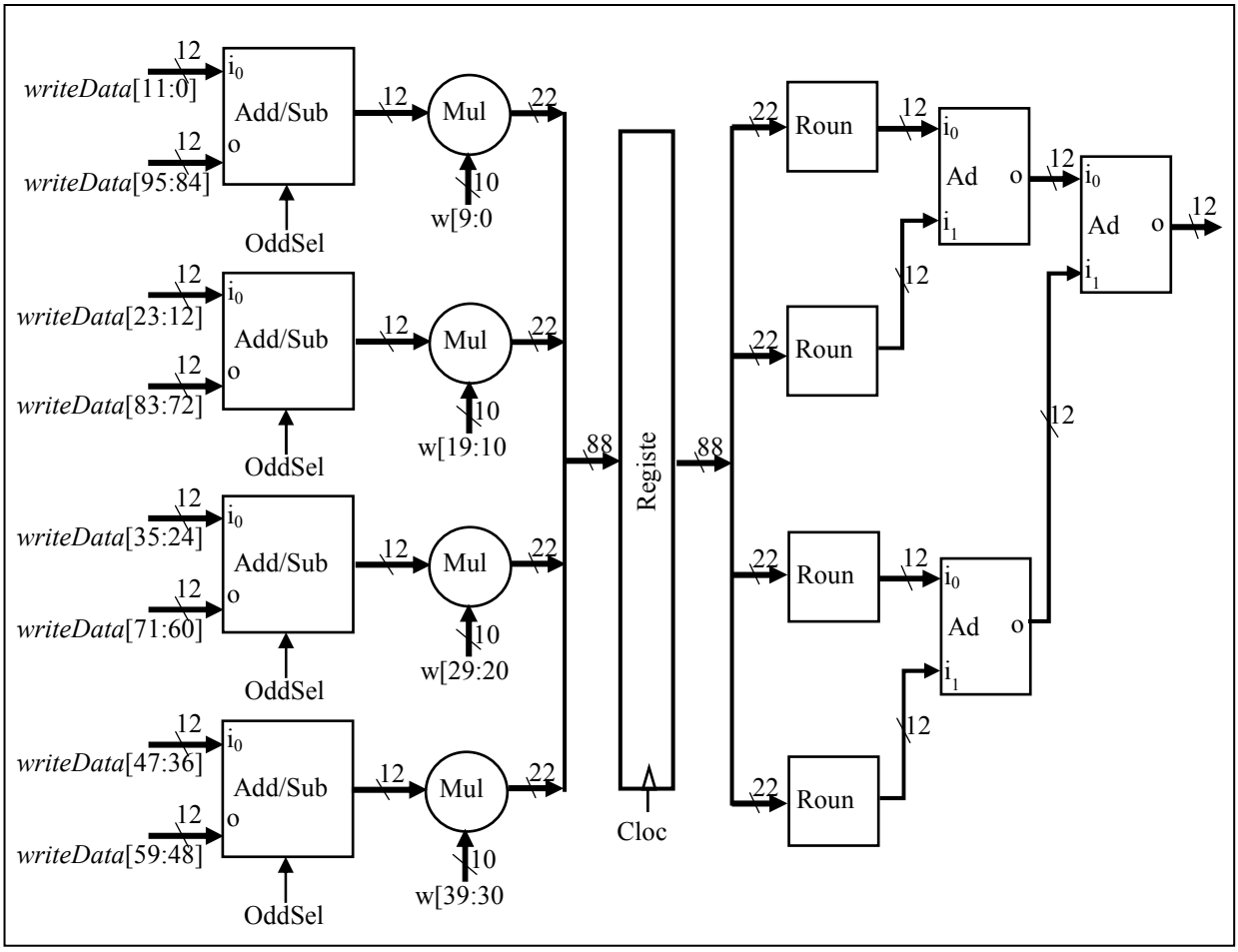

Fig. 2. Architecture of 1-D DCT hardware

where z_i is the transformed coefficient, x_i is the pixel data, $a=C_1$, $b=C_2$, $c=C_3$, $d=C_4$, $e=C_5$, $f=C_6$, $g=C_7$, $C_i=0{,}5\cos(k\pi/16)$, i=0,1,..7, and k=1,2,..7. Let $X_0 = x_0+x_7$, $X_2 = x_1+x_6$, $X_4 = x_2+x_5$, $X_6 = x_3+x_4$, $X_1 = x_0-x_7$, $X_3 = x_1-x_6$, $X_5 = x_2-x_5$, and $X_7 = x_3-x_4$.

In order to calculate the coefficients by means of these equations, 1-D DCT architecture in Fig. 2 is designed. In Fig. 2, the topmost Add/Sub component is used to compute either $X_0=(x_0+x_7)$ when *OddSel*=0 or $X_1=(x_0-x_7)$ when *OddSel*=1, and so on. As a result, *OddSel* is asserted only if an odd-indexed coefficient $\{z_1, z_3, z_5, z_7\}$ is calculated; Add/Sub components find out either $\{X_0, X_2, X_4, X_6\}$ or $\{X_1, X_3, X_5, X_7\}$ in parallel.

After add/sub operations, four integer multiplication operations are performed in parallel for every coefficient. In Fig. 2, w[39:0] denotes the set of weights used during a multiplication. For example, $z_2 = b^*\times X_0 + f^*\times X_2 \square f^*\times X_4 \square b^*\times X_6$ and w[39:0] $=\{b^*, f^*, -f^*, -b^*\}$, where * is used to indicate 10-bit 2's complement representations of the related weight values. Even though it is not shown in Fig. 2, there is single 8×40-bit look-up table where it is addressed by 3-bit index value of the coefficient being computed and its each 40-bit row stores four weights in 2's complement representation per coefficient.

After multiplication operations, there is 88-bit register whose operation is controlled by a two state {*empty*, *full*} FSM as follows:

- *empty*: If *writeEn* is asserted, a new 88-bit word is loaded into register and FSM goes to the other state. In *empty* state, *full* signal is deasserted, and *empty_transpose* (*empty* signal for transpose buffer) and *empty_outbuff* (*empty* signal for output buffer) signals are asserted.
- *full*: While either 1-D DCT or 2-D DCT coefficient is computed, if either *readEn_transpose* or *readEn_outbuff* is asserted, 88-bit current state of register is provided with either transpose buffer or output buffer, respectively. Furthermore, FSM stays in this state if *writeEn* is found to be asserted; otherwise, it goes to the other state. In this state, *full* signal is asserted only if *readEn_transpose* or *readEn_outbuff* is not asserted, and *empty_transpose* and *empty_outbuff* signals are deasserted, respectively.

22-bit result $\{r_{21}, r_{20}, \ldots, r_0\}$ of signed multiplication is rounded to 12-bit 2's complement number by a combinational logic circuit based on the sign of result as follows:

- *positive*: There are three cases:
  - If $\{r_{21}, r_{20}, \ldots, r_{10}\}$ is the maximum positive number, the rounded result is equal to $\{r_{21}, r_{20}, \ldots, r_{10}\}$.
  - If $\{r_{21}, r_{20}, \ldots, r_{10}\}$ is not the maximum positive number and r_9=0, the rounded result is equal to $\{r_{21}, r_{20}, \ldots, r_{10}\}$.

- If $\{r_{21}, r_{20}, \ldots, r_{10}\}$ is not the maximum positive number and r_9=1, the rounded result is equal to $\{r_{21}, r_{20}, \ldots, r_{10}\} + 1$.

- *negative:* If r_9=0, it is equal to $\{r_{21}, r_{20}, \ldots, r_{10}\}$; otherwise, $\{r_{21}, r_{20}, \ldots, r_{10}\} + 1$.

After rounding, a four-input, 12-bit adder tree is utilized to find out either 1-D or 2-D coefficient value in 2's complement representation. It should be noted that 1-D DCT component adds only one clock cycle of latency due to its register, which results in 208+1=209 clock cycles of latency in total.

2.4. Transpose Buffer

Transpose buffer is a 63×12=756-bit shift-register whose operation is similar to the one in [4] . There are two related scenarios:

- *serial-in*: If *writeEn* is asserted due to deasserting *empty_transpose* during 1-D DCT computation, a new 12-bit coefficient is serially shifted in transpose buffer. For both scenarios, *full* and *empty* signals are never asserted and deasserted, respectively.
- *parallel-out:* Consider that *shift_register*=$\{reg_{62}, reg_{62}, \ldots, reg_0\}$ is composed of 63 12-bit registers. When the shift register becomes full, a set of eight registers *column*=$\{reg_{56}, reg_{48}, reg_{40}, \text{reg}_{32}, reg_{24}, reg_{16}, reg_8, reg_0\}$ holds the first column of 1-D DCT coefficients. In the next clock cycle, the 64th 1-D DCT coefficient is shifted in while the first column is loaded into pong buffer, in which both *writeEn* and *readEn* are asserted. After the right-shift, *column* will hold the second column of 1-D DCT coefficients. In a similar manner, whenever *readEn* is asserted by pong buffer, the buffer is shifted to the right and *column* stores the next column coefficients.

2.5. Output Buffer

Output buffer holds 2-D DCT coefficients and provides isolation between 2-D DCT hardware in Fig. 1 and another component that will be connected its output. Output buffer has two registers, namely reg_0 and reg_1, and their operation is controlled by a three-state {*empty*, *almost-full*, *full*} FSM as follows:

- *empty*: Both registers are empty. If *writeEn* is asserted, a new 12-bit word is loaded into reg_0 and FSM goes to *almost-full* state. In *empty* state, *full* and *empty* output signals are deasserted and asserted, respectively.
- *almost-full:* If both *writeEn* and *readEn* are asserted or deasserted, it stays in this state. Furthermore, if they are asserted, a new word is loaded into reg_0. If *writeEn* is asserted, but *readEn* is deasserted, a new word is loaded into reg_1 and it goes to *full* state. If *writeEn* is not asserted, but *readEn* is asserted, it goes to *empty* state since reg_0 has been read. In *almost-full* state, *full* and *empty* output signals are deasserted and asserted, respectively.

  full: If *readEn* is asserted, it goes to *almost-full* state while old reg_1 is copied into reg_0. In *full* state, *full* and *empty* output signals are asserted and deasserted, respectively.

It should be emphasized that output buffer component introduces only one clock cycle of latency, which results in 209+1=210 clock cycles of total latency for the proposed 2-D DCT hardware.

3. Implementation Results and Comparisons

The 2-D DCT design presented in this study is described as a device independent fashion in Verilog HDL, simulated and verified by a test-bench using Xilinx ISim, and synthesized using Xilinx ISE 14.7 for Xilinx Spartan 3E (XC3S500E-5VQ100), Virtex IV (XC4VSX35-12FF668), and Virtex 7 (XC7VX330T-3FFG1157) FPGA devices.

The proposed 2-D DCT architecture is compared against two other competitive designs implemented for Virtex IV and Spartan 3E by [6] and [8] in Table 1 and Table 2, respectively. It should be noted here that both [6] and [8] are based on the row-column decomposition method and use two 1-D DCT cores with a transpose buffer between them. Their main difference is due to their implementations of 1-D DCT core, which will also be evident in the following tables.

Table 1. Device utilizations using Xilinx Virtex IV

Logic Utilization	Used [6]	Used	Available
Number of Slices	376	434	15360
Number of Slice Flip Flops	388	403	30720
Number of 4 input LUTs	751	701	30720
Number of bonded IOBs	33	30	448
Number of DSP48s	0	4	192
Block RAMs	42	0	192

Table 2. Device utilizations using Xilinx Spartan 3E

Logic Utilization	Used [8]	Used	Available
Number of Slices	1235	444	4656
Number of Slice Flip Flops	1551	400	9312
Number of 4 input LUTs	1239	720	9312
Number of bonded IOBs	23	30	66
Number of MULT18X18SIOs	8	4	20

According to Table 1, the proposed design and [6] result in similar device utilizations except for DSP48 and Block RAM resources. The design in [6] does not use any multiplier and relies heavily on Block RAMs for the implementation of a 2-D DCT architecture based on distributed arithmetic. In terms of the frequency of these designs, on the other hand, [6] achieves around 118.0 MHz as compared to 136.6 MHz by the presented design.

According to Table 2, the proposed design is clearly superior to [8] that implements a scaled 1-D DCT algorithm. However, [8] with the running frequency of 101.355 MHz seems to be faster than the 2-D DCT core that achieves 80.5 MHz on Spartan 3E.

The proposed 2-D DCT hardware with a latency of 210 clock cycles per 8x8 pixel block, when mapped to a Spartan 3E and Virtex 7 FPGA, takes about 2.6 µs and 1.02 µs per block, respectively. These processing rates are enough for achieving at least 70 fps for images with 720x480 pixels on Spartan 3E, and 30 fps for images with 1920x1080 pixels. Consequently, the presented 2-D DCT can be used as a core of an M-JPEG video compressor directed to SDTV or HDTV applications.

4. Conclusions

In this paper, a low area 2-D 8×8 DCT architecture is presented, and its implementation results are compared against two other competitive design from the literature. In order to

keep the FPGA device utilization as low as possible, the proposed design is based on the row-column decomposition method, which results in a time-shared use of single 1-D DCT core for computing 1-D and 2-D DCT coefficients and a transpose buffer for keeping and transposing the 1-D coefficients. The proposed 2-D DCT core is described in synthesizable Verilog HDL. The synthesis results show that it requires low area and achieves high processing rates that may be useful in M-JPEG video compressor directed to SDTV or HDTV applications.

5. References

[1] G. K. Wallace, "The JPEG still picture compression standard", *Communications of the ACM*, vol.34, no. 4, pp.30-44, April 1991.

[2] M. L. Liou, "Overview of the p x 64 kbit/s video coding standard," *Communications of the ACM*, vol. 34, No. 4, pp. 59-63, April 1991.

[3] D. L. Gall, "MPEG: a video compression standard for multimedia applications", *Communications of the ACM*, vol. 34, no. 4, pp. 46-58, April 1991.

[4] S.-C. Hsia, S.-H. Wang, "Shift-register-based data transposition for cost-effective discrete cosine transform", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 15, no. 6, pp. 725-728, June 2007.

[5] L. V. Agostini, I. S. Silva, S. Bampi, "Multiplierless and fully pipelined JPEG compression soft IP targeting FPGAs", *Microprocessors and Microsystems*, vol. 31, no. 8, pp. 487-497, December 2007.

[6] R. E. Atani, M. Baboli, S. Mirzakuchaki, S. E. Atani, B. Zamanlooy, "Design and implementation of a 118 MHz 2D DCT processor", *IEEE International Symposium on Industrial Electronics,* 2008, pp. 1076-1081.

[7] E. D. Kusuma, T. S. Widodo, "FPGA implementation of pipelined 2D-DCT and quantization architecture for JPEG image compression", *International Symposium in Information Technology*, 2010, pp. 1-6.

[8] T. Pradeepthi, A. P. Ramesh, "Pipelined architecture of 2D-DCT, quantization and zigzag process for JPEG image compression using VHDL", *International Journal of VLSI Design & Communication Systems*, vol. 2, no. 3, pp. 99-110, September 2011.

[9] S. Sanjeevannanavar, N. Nagamani, "Efficient design and FPGA implementation of JPEG encoder using Verilog HDL", *International Conference on Nanoscience, Engineering and Technology*, 2011, pp. 584-588.

[10] A. Madisetti, A. N. Willson, Jr., "A 100 MHz 2-D 8x8 DCT/IDCT processor for HDTV applications", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 5, no. 2, pp. 158-165, April 1995.