IACSIT International Journal of Engineering and Technology, Vol.2, No.3, June 2010
ISSN: 1793-8236

JPEG Image Compression using FPGA with Artificial Neural Networks

Dr. Saudagar Abdul Khader Jilani, *Member, IEEE, IACSIT* and Dr. Syed Abdul Sattar

***Abstract*— Image and video compression is one of the major components used in video-telephony, videoconferencing and multimedia-related applications where digital pixel information can comprise considerably large amounts of data. Management of such data can involve significant overhead in computational complexity and data processing. Compression allows efficient utilization of channel bandwidth and storage size. Typical access speeds for storage mediums are inversely proportional to capacity. Through data compression, such tasks can be optimized. One of the commonly used methods for image and video compression is JPEG (an image compression standard).Image and video compressors and decompressors (codecs) are implemented mainly in software as digital signal processors. Hardware-specific codecs can be integrated into digital systems fairly easily, requiring work only in the areas of interface and overall integration. Using an FPGA (Field Programmable Gate Array) to implement a codec combines the best of two worlds. The implementation of this work is carried out with JPEG algorithm with Artificial Neural Networks (ANN). The core compression design was created using the Verilog hardware description language. The supporting software was written in matlab, developed for a DSP and the PC. The implementation of this work was successful on achieving significant compression ratios. The sample images chosen showed different degrees of contrast and fine detail to show how the compression affected high frequency components within the images.**

***Index Terms*—ANN, Bandwidth, Compression, FPGA, JPEG.**

I. Introduction

Data compression is the technique to reduce the redundancies in data representation in order to decrease data storage requirements and hence communication costs. Reducing the storage requirement is equivalent to increasing the capacity of the storage medium and hence communication bandwidth. Thus the development of efficient compression techniques will continue to be a design challenge for future communication systems and advanced multimedia applications.

Typically image and video compressors and decompressors (CODECS) are performed mainly in software as signal processors can manage these operations without incurring too much overhead in computation. However, the complexity of these operations can be efficiently implemented in hardware. Hardware specific CODECS can be integrated into digital systems fairly easily. Improvements in speed occur primarily because the hardware is tailored to the compression algorithm rather than to handle a broad range of operations like a digital signal processor.

For improving the ANN efficiency for image compression various works were suggested in past. In [1] the author expounds the principle of BP neural network with applications to image compression and the neural network models. Then an image compressing algorithm based on improved BP network is developed. Feedforward networks using backpropagation algorithm [2] adopting the method of steepest descent for error minimization is popular and widely adopted and is directly applied to image compression. Image data compression using Vector Quantization (VQ) has received a lot of attention because of its simplicity and adaptability. VQ requires the input image to be processed as vectors or blocks of image pixels. The Finite-state vector quantization (FSVQ) is known to give better performance than the memory less vector quantization (VQ). The author in [3] presents a novel combining technique for image compression based on the Hierarchical Finite State Vector Quantization (HFSVQ).

The fully functional JPEG decoder prototype [4] is capable of decompressing compressed data to achieve a frame rate of 30 frames per second. In [5] the architecture and the VHDL design of a Two Dimensional Discrete Cosine Transform (2-D DCT) for JPEG image compression. This architecture is used as the core of a JPEG compressor and is the critical path in JPEG compression hardware. An efficient architecture to implement a JPEG encoder on FPGA trying to use as less resources as possible without compromising much speed or quality is discussed by author in [6]. When compared to earlier JPEG this work with JPEG with ANN shows better performances (in terms of PSNR) for compression rates higher than 20.

II. Image Compression

Image compression is the application of Data compression on digital images. The objective of image compression is to reduce redundancy of the image data in order to be able to store or transmit data in an efficient form. Image compression can be lossy or lossless. Lossy compression though gives more compression compared to lossless compression; the accuracy in retrievation is less in case of lossy compression as compared to lossless compression.

JPEG compression is defined as a lossy coding system

Manuscript received April 8, 2010.

Dr. Saudagar Abdul Khader Jilani is with the University of Tabuk, Tabuk, The Kingdom of Saudi Arabia. (phone (M): 00 966 564545829; (R): 00 966 4 4227483; e-mail: saudagar_jilani@hotmail.com).

Dr. Syed Abdul Sattar is with Jawaharlal Nehru Technological University, Hyderabad, Andhra Pradesh, India. (e-mail: syed49in@yahoo.com)

which is based on the discrete cosine transform. However, there are several extensions to the JPEG algorithm to provide greater compression, higher precision, and can be tailored to specific applications.

A. Discrete Cosine Transform

The discrete cosine transform is the basis for the JPEG compression standard. Ahmed, Natarajan, and Rao originally proposed use of the DCT in 1974, and it has become the most popular transform for image and video coding. [7] Many compression techniques take advantage of transform coding as it decorrelates adjacent pixels from the source image. For JPEG, this allows for efficient compression by allowing quantization on elements that are less sensitive. The DCT algorithm is completely reversible making this useful for both lossless and lossy compression techniques.

The DCT is a special case of the well known Fourier transform. Essentially the Fourier transform in theory can represent a given input signal with a series of sine and cosine terms. The discrete cosine transform is a special case of the Fourier transform in which the sine components are eliminated. [8] For JPEG, a two-dimensional DCT algorithm is used, which is essentially the one-dimensional version evaluated twice. By this property there are numerous ways to efficiently implement a software or hardware based DCT module. [7] The DCT is operated two dimensionally taking into account an 8 by 8 block of pixels. The resulting data set is an 8 by 8 block of frequency space components, the coefficients scaling the series cosine terms, known as basis functions. The first element at row 0 and column 0, is known as the DC term, the average frequency value of the entire block. The other 63 terms are AC components which represent the spatial frequencies that compose the input pixel block, by scaling the cosine terms within the series.

There are two useful products of the DCT algorithm. First it has the ability to concentrate image energy into a small number of coefficients. Second, it minimizes the interdependencies between coefficients. [7] These two points essentially state why this form of transform is used for the standard JPEG compression technique. By compacting the energy within an image, more coefficients are left to be quantized coarsely, impacting compression positively, but not losing quality in the resulting image after decompression. Taking away inter-pixel relations allows quantization to be non-linear, also affecting quantization positively. DCT has been effective in producing great pictures at low bit rates and is fairly easy to implement with fast hardware based algorithms [9].

B. Quantization

Quantization is an extremely important step in the JPEG compression algorithm, as it removes a considerable amount of information, thus reducing the entropy in the input data stream. [10] Quantization alone, is essentially a lossy compression technique. However, quantization does benefit the compression process, regardless of the lossy artifacts that are produced. The high frequency AC coefficients typically will become zero, which aids in entropy coding. Quantization is intended to remove the less important components to the visual reproduction of the image. However, quantization is most effective when less important elements are quantized more coarsely. Larger quantization values will result in visual artifacts. Nonzero AC coefficients after quantization are worse for compression, but will suppress blocking artifacts in the reconstructed image. Blocking artifacts indicate high spatial frequencies caused by an absence of AC coefficients. In [10] Essentially, quantization, when used effectively, will result in high compression, with minimal loss in quality.

C. Entropy Encoding

Entropy is a measure of disorder and unpredictability. The degree of entropy can be used as a measure of the information carried by the message. In [8] Up to this point the input image has run through the DCT process, followed by zigzag ging to approximately organize the data stream with more entropy toward the beginning of the stream. Then this stream was quantized, and run through the DC differencing in an effect to reduce entropy in the data stream. All of these processes both add to compression, and more importantly, make the data stream ready for entropy coding. Various coding techniques like Run Length Coding, Huffman Coding are used for entropy encoding.

III. System Overview

The system architecture for this work has been divided into three main areas, which can be seen in the high level block diagram in Fig. 1. The major portions of this design are easily divided into three sections, PC, DSP, and FPGA.

The DSP and FPGA both reside on the PLogic PCI card, whereas the PC interfaces to the PLogic PCI card over the PCI bus. Although these sections are fairly separable, they interact heavily and are dependent on each other to fulfill the JPEG compression algorithm. The FPGA, is the heart, encoding input image data into a compressed output, used to create a JPEG image file. The DSP handles transactions between the PC and FPGA core, providing the interface to take raw pixel inputs from an input file, and command the FPGA to encode the input data. The DSP is also responsible to collect the output and assemble an entropy encoded bit stream for use in the JPEG file structure. The PC is responsible for collecting input data, grabbing the output data stream, and assembling the output file.

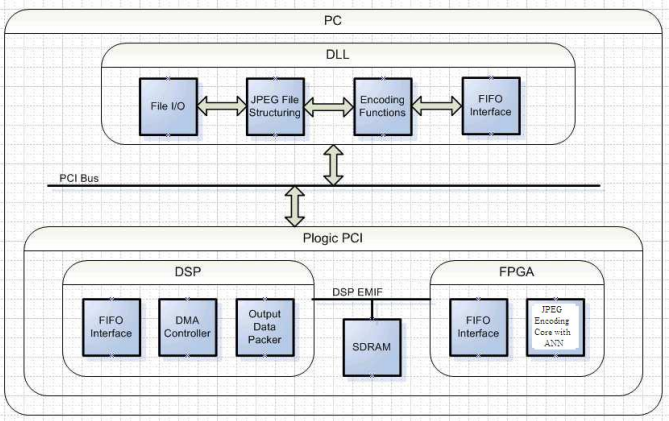


Figure 1. High Level System Overview

A. FPGA Overview

The FPGA is composed of two main systems as shown in Fig. 2. First is the FIFO interface to the DSP. The FIFO interface is designed to allow for high speed data transactions in an asynchronous environment. Additionally an interrupt source state machine is associated with the FIFO interface to assist in a sort of handshaking between the DSP and FPGA. The second portion of the design is the JPEG encoding core. The JPEG encoder core runs on a 25 MHz clock and is fully synchronous, pipelined in stages, and employs parallel computation to help increase throughput.

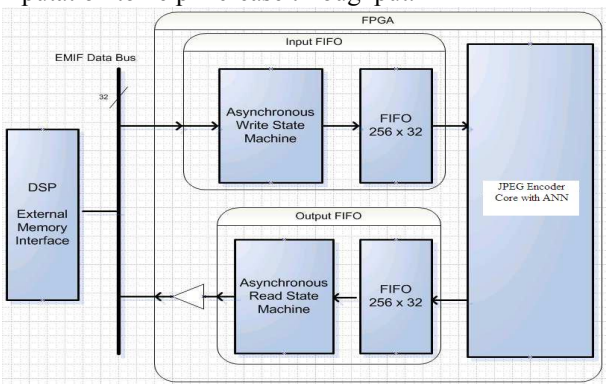


Figure 2. FPGA Core Overview

B. JPEG Encoder Core

The JPEG encoder core consists of four main modules which, although separable, are all necessary to work together to comply with the JPEG encoding specification. The units employed are for discrete cosine transform, quantization and entropy coding. Some of these modules are pipelined and some are comprised of parallel data paths. Each module is fully synchronous, and is designed to have a common interface. At the output of the final stage of JPEG encoding, is a data stream assembler, which takes two variable length outputs from the entropy coder and combines them in a common format understood by the DSP.

The JPEG compression algorithm requires an 8-bit pixel resolution for the data input. The other requirement is that the DCT based algorithm must produce an entropy coded output consisting of two variable length words. This data output must fit the JPEG file stream requirement. Now, it is necessary to alter this stream into a form ready for use with the ANN. A file is prepared where each symbol is written as two identical vectors to form a training pair. In other words, the first vector would be the input vector and the second vector would be the desired output. As many of the images tested were extremely large, there were many similar training pairs. To eliminate this redundant information, a program is used to search the training file for similar training pairs and delete them. This not only reduces the number of redundant training pairs, but reduces training time of the ANN. The Backpropagation Learning algorithm is then applied as shown in section 4 for image compression. Finally, the output produced by the ANN in the form of decompressed vectors or windows, is reconstructed and then given as input to FIFO interface.

C. FIFO Interface

The FIFO interface was designed for numerous reasons. First, the 100 MHz clock on the external memory interface of the DSP, is out of phase from the FPGA 25 MHz clock. This causes issues of meta stability when latching data into registers. If a setup or hold time is violated, there is a potential for incorrect data being input to the encoder. The second reason for a FIFO interface, is to allow for streaming data as fast as possible into the encoder. As the FPGA clock is slower than the DSP EMIF clock, there is the ability to write faster than the FPGA JPEG encoder can accept data. Similarly, the DSP must read from the FPGA JPEG encoder, while servicing writes. Keeping the writes ahead of the reads can help performance by taking advantage of the pipelining in the design. Thus, the valid data may build up in the FIFO, so when reads commence they can be read in sequential clock cycles.

D. DSP Overview

The DSP program uses a real time operating system supported by the DSP called DSP BIOS. This realtime operating system has configurable interrupt channels with interrupt service routines, a configurable 4 channel DMA structure, as well as several other features. Some structures were defined to store status and control for the imaging functions that interact with the FPGA for configuration and for running the compression sequence.

E. DMA Interface

The DSP has a DMA controller in parallel with its core. The DMA unit can operate on the external memory interface bus, to transfer data elements to and from the DSP internal memory, or between external memory devices. Each of the four DMA channels have nine level FIFOs associated to buffer memory transfers. These channels can be configured to operate on a wide number of interrupts. This works well for this design, allowing the external memory interrupts driven by the FPGA to trigger memory transfers between the DSP and FPGA. Each of these DMA channels labeled zero through three. These channels operate where channel zero is highest priority for transfers, and channel three is lowest priority. The DMA channels can be configured to increment the source and or destination address, within its configuration registers. These addresses can be configured to increment based upon an element size for 8-bits to 32-bits. Otherwise the address can be configured to remain static.

The DMA unit also has a transfer count register which keeps track of the number of elements, and frames of elements transferred. The transfer register gives 16-bits for frames, and 16-bits for elements. Once the element count reaches 0, the frame count is decremented, and the element count is reloaded by one of the global reload registers, which specifies how many elements are within a frame. Each individual DMA channel can be setup to transfer an element or a frame on a synchronizing event. These elements can be an interrupt generated by another DMA channel, an external interrupt, a timer interrupt, as well as other sources. This is particularly helpful in transferring elements based on interrupts generated on the external interrupt line from the FPGA. These synchronization events can be tied to the DMA transfer operating between the FPGA and SDRAM. The

DMA channels can also be configured to generate an interrupt based upon different events. For instance, the last transfer in a frame, defined in the transfer counter register, can generate a DMA channel interrupt. There can be an interrupt generated on the completion of the second to last frame. The ability to use different conditions to trigger interrupts gives a higher level of configurability for these DMA channels.

The general approach taken for this DMA strategy is to give the reads from the FPGA higher priority than writes, as writes will happen much more frequently, and we do not want to risk overfilling the output FIFO. The interrupt state machines associated with the input and output FIFOs generate external interrupts that trigger the DMA channels for data transfers.

IV. ARTIFICIAL NEURAL NETWORKS

There are four reasons to use an artificial neural network (ANN): i) Weights representing the solution are found by iteratively training, ii) ANN has a simple structure for physical implementation, iii) ANN can easily map complex distributions, and iv) generalization property of the ANN produces appropriate results for the input vectors that are not present in the training set.

A. Backpropagation Learning Algorithm

Step 1: Normalize the inputs and outputs with respect to their maximum values. It is proved that the neural networks work better if input and outputs lie between 0-1. For each training pair, assume there are 'l' inputs given by $\underset{l\times 1}{\{I\}_I}$ and 'n' outputs $\underset{n\times 1}{\{O\}_O}$ in a normalized form.

Step 2: Assume the number of neurons in the hidden layer to lie between l < m <2l

Step 3: [V] Represents the weights of synapses connecting input neurons and hidden neurons and [W] represents weights of synapses connecting hidden neurons and output neurons. Initialize the neurons to small random values usually from -1 to 1. For general problems, λ can be assumed as 1 and the threshold values can be taken as zero.

$$[V]^0 = \text{[random weights]}$$

$$[W]^0 = \text{[random weights]}$$

$$[\Delta V]^0 = [\Delta W]^0 = [0]$$

Step 4: For the training data, present one set of inputs and outputs. Present the pattern to the input layer {I}I as function, the output of the input layer may be evaluated as

$$\underset{1\times 1}{\{O\}_I} = \underset{1\times 1}{\{I\}_I} \quad (1)$$

Step 5: Compute the inputs to the hidden layer by multiplying corresponding weights of synapses as

$$\underset{m\times 1}{\{I\}_H} = \underset{m\times 1}{[V]^T}\underset{1\times 1}{\{O\}_I} \quad (2)$$

Step 6: Let the hidden layer units evaluate the output using the sigmoidal function as

$$\{O\}_H = \left\{ \begin{array}{c} \vdots \\ \dfrac{1}{(1+e^{-I_m})} \\ \vdots \end{array} \right\}_{m\times 1}$$

Step 7: Compute the inputs to the output layer by multiplying corresponding weights of synapses as

$$\underset{n\times 1}{\{I\}_O} = \underset{n\times m}{[W]^T}\underset{m\times 1}{\{O\}_H} \quad (3)$$

Step 8: Let the output layer units evaluate the output using sigmoidal function as

$$\{O\}_O = \left\{ \begin{array}{c} \vdots \\ \dfrac{1}{(1+e^{-I_o})} \\ \vdots \end{array} \right\}$$

The above is the network output.

Step 9: Calculate the error and the difference between the network output and the desired output as for the i^{th} training set as

$$E^P = \frac{\sqrt{\Sigma(T_j - O_{Oj})^2}}{n} \quad (4)$$

Step 10: Find $\{d\}$ as

$$\{d\} = \left\{ \begin{array}{c} \vdots \\ (T_k - O_{Ok})O_{Ok}(1 - O_{Ok}) \\ \vdots \end{array} \right\}_{n\times 1}$$

Step 11: Find $[Y]$ matrix as

$$\underset{m\times n}{[Y]} = \underset{m\times 1}{\{O\}_H}\underset{1\times n}{\langle d \rangle} \quad (5)$$

Step 12: Find

$$\underset{m\times n}{[\Delta W]^{t+1}} = \alpha\underset{m\times n}{[\Delta W]^t} + \eta\underset{m\times n}{[Y]} \quad (6)$$

Step 13: Find $\underset{m\times 1}{\{e\}} = \underset{m\times n}{[W]}\underset{n\times 1}{\{d\}}$ (7)

$$\{d^*\} = \begin{Bmatrix} \cdot \\ \cdot \\ e_i(O_{Hi})(1 - O_{Hi}) \\ \cdot \\ \cdot \end{Bmatrix}_{m \times 1 \quad m \times 1}$$

Find $[X]$ matrix as

$$[X]_{1\times m} = \{O\}_{I\,1\times 1}\langle d^*\rangle_{1\times m} = \{I\}_{I\,1\times 1}\langle d^*\rangle_{1\times m} \quad (8)$$

Step 14 : Find $[\Delta V]^{t+1}_{1\times m} = \alpha[\Delta V]^{t}_{1\times m} + \eta[X]_{1\times m}$ (9)

Step 15 : Find $[V]^{t+1} = [V]^{t} + [\Delta V]^{t+1}$ (10)

$$[W]^{t+1} = [W]^{t} + [\Delta W]^{t+1} \quad (11)$$

Step 16 : Find error rate as

$$errorrate = \frac{\sum E_P}{nset}$$

Step 17: Repeat steps 4-16 until the convergence in the error rate is less than the tolerance value.

V. PC Overview

The PC level of this design can be separated into three sections as shown in Fig. 3, all of which interact with each other. The lowest level section interacts with the DSP via a FIFO interface, as well as direct addressing to DSP registers, and the DSP external memory interface. The level above this deals with constructing the output JPEG image file, and similarly the input file. TCL was chosen for its ability to create a simple command driven, or graphical interface through buttons, and response windows.

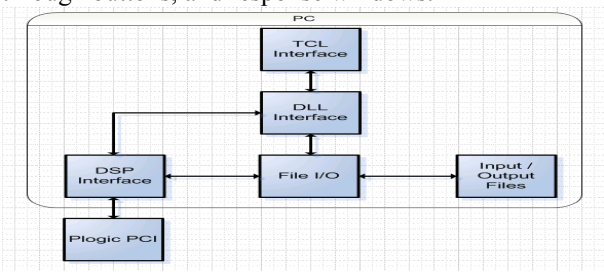


Figure 3. PC Overview

A. Imaging

There were several functions implemented to handle the understanding of the input image, sending it to the encoder, calling the encode process, retrieving the output, assembling the JPEG headers, and producing the output file. All of these functions were fairly simple to put together as they are high level functions within this design. The tricky part really has to do with the JPEG headers. Getting the JPEG headers correct is an important part of the design, as the entropy coded bit stream could be correct, but incorrect headers will result in an erroneous image output stream read by a decoder.

B. Input Image

The first step for the PC interface is to open a known type of input file, parse it, and send it to SDRAM. The supported input image format for this design was chosen to be the PPM format, which stands for portable pixel map. The values used for JPEG compression in this design are 8-bit, so the value in the input file here is 255, to represent 256 values starting from 0. The last two values specify the horizontal and vertical pixel resolutions. The values for horizontal and vertical pixel ratios are stored to the DSP control register.

The grayscale images, will have one value per pixel, to specify the luminance intensity, where 0 is white, and 255 is black. When parsing the input file, a temporary output file is created to store a list of hex pixel values.

C. Encoding Process

The encoding process will read 64 pixel components in a loop, from one of the files created during the parsing operation. Using one of the block transfer functions, these values are stored to SDRAM. This process goes on until the file is read completely, and transferred into SDRAM. At this point, the DSP control register is updated with the pixel aspect ratios and the image component type.

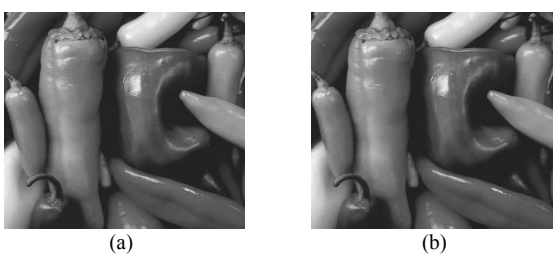
Prior to running the encoder on the FPGA the image headers are assembled. These headers are stored to an output file. After the header portion is completed, the actual encoding cycle is run, and the output bitstream is stored into the output file. The actual encode cycle on the FPGA is run once the command is given with the action code to encode the image. This is followed with a command, which will wait for a response from the DSP. The DSP response contains an integer specifying how many words must be read from SDRAM to obtain the assembled entropy encoded data stream. This integer will be used to run a loop and read SDRAM in blocks of 64 elements, until completed. After this data is written to the output file, the end of image marker is written and the file is closed.

VI. Results

A. Figures and Tables

(a) (b)

Figure 4. (a) Lena Source, (b) Lena Result



(a) (b)

Figure 5. (a) Peppers Source, (b) Peppers Result

TABLE I. COMPARISON FOR COMPRESSION RATIO TABLE

Image	Aspect Ratio	Compression
Sample1 (Lena)	512 x 512	23.17
Sample2 (Peppers)	256 x 256	11.37

VII. CONCLUSION

The core FPGA design had impressive results as described throughout the system overview section. In simulation on the FPGA, assuming it has streaming inputs and outputs, the encoder takes 162 cycles for the first block, and as long as the data stream is constant, it will take an additional 64 cycles per block after the first. This means that a 640x480 grayscale image would take 307,298 cycles, which at 25MHz is 12.29ms. This is roughly one fourth of the time that is necessary to make motion JPEG at 24 frames per second. As this core design is fully synchronous, it should give the same result in implementation.

The results of this design in terms of compression were quite impressive. Two different test images were used to get a good range of image types from those that would expect less compression and those that would expect greater compression based upon the image details. It can be seen that JPEG image compression using FPGA with Artificial Neural Networks gives more compression ratio when compared to existing system.

REFERENCES

[1] Tang Xianghong Liu Yang "An Image Compressing Algorithm Based on Classified Blocks with BP Neural Networks" *International Conference on Computer Science and Software Engineering,* Date: 12-14 Dec. 2008 Volume: 4, On page(s): 819-822.

[2] S. Anna Durai, and E. Anna Saro "Image Compression with Back-Propagation Neural Network using Cumulative Distribution Function" *World Academy of Science, Engineering and Technology* 2006.

[3] Karlik, Bekir "Medical Image Compression by using Vector Quantization Neural Network (VQNN) " *Neural Network World,* January 1, 2006.

[4] Yusof Z.M, Aspar Z, Suleiman I, "Field programmable gate array (FPGA) based baseline JPEG decoder ," *TENCON 2000. Proceedings* , vol.3, no., pp.218-220 vol.3, 2000.

[5] Agostini L.V, Silva I.S, Bampi S, "Pipelined fast 2D DCT architecture for JPEG image compression," *Integrated Circuits and Systems Design, 2001, 14th Symposium on.* , vol., no., pp.226-231, 2001.

[6] Quazi H, Qader F, Rasheed M.J, Mansoor H, "An Optimized Architecture Implementing the Standard JPEG Algorithm on FPGA," *Engineering Sciences and Technology, 2005. SCONEST 2005. Student Conference on* , vol., no., pp.1-5, 27-27 Aug. 2005.

[7] Iain Richardson, Video Codec Design – Developing Image and Video Compression Systems. Copyright by John wiley & Sons Ltd. 2002 (ISBN0471485535)

[8] John Watkinson, The Art of Digital Video. Copyright by John Watkinson, 2000 (ISBN 0240512871)

[9] Gonzalez Woods. Digital Image Processing. Copyright by Prentice Hall, Inc. 2002 (ISBN 0201508036)

[10] Weidong Kou. Digital Image Compression – Algorithms and Standards. Copyright by Kluwer Academic Publishers, 1995 (ISBN 079239626X).