Comparative Study and Implementation of JPEG and JPEG2000 Standards for Satellite Meteorological Imaging Controller using HDL

Vineeth Mohan, Ajay Mohanan, Paul Leons, Rizwin Shooja
Amrita Vishwa Vidyapeetham, Amritapuri, Clappana P.O, Kollam-690525, Kerala, India
vineethmohan@ieee.org, ajay-m@ieee.org, paulleons@ieee.org, rizwin@ieee.org

Abstract- **This paper presents a pipelined approach towards the implementation of the JPEG baseline encoder for grayscale image compression using HDL. The complete baseline encoder with DCT compressor model, quantization, and entropy encoding has been realized in HDL. The paper also delves into the implementation of the DWT core for a JPEG2000 image compression standard. A comprehensive testing and verification of the same in MATLAB environment is also briefly described.**

Keywords:* *JPEG; JPEG2000; HDL; DWT; Image compression.

I. Introduction

Meteorological images obtained from satellite have high resolution and take up a lot of disk space and bandwidth. Therefore, it is imperative to use an image compression algorithm that would do away with these problems associated with the transmission of such images. However, it is highly recommended that the implemented compression algorithm is supported by widely available image decoders. This instigates the need for implementing an industry standard like JPEG (Joint Photographic Experts Group) or JPEG2000. Most of the currently available solutions are software based and they can be uncomfortably slow and consume more computing power.

This calls for a dedicated hardware unit so that fast compression and transmission is possible. JPEG is a time tested and reliable standard for lossy image compression while JPEG2000 is a comparatively newer standard that supports lossless image compression. The hardware implementation of these two standards requires careful design strategy and compatibility with Field Programmable Gate Arrays (FPGA) for extensive testing and verification.

II. JPEG Standard

In 1994, the Joint Photographic Experts Group (JPEG) committee finalized the JPEG standard for still image compression as ISO/IEC 10918-1. The main goal behind such standardization was to allow high quality data compression with a global compatibility. The JPEG standard provided much higher compression rates than its predecessors while maintaining a very good image quality. It is still the most widely used scheme used for still image compression. The main parts of JPEG compressor engine are the DCT block, the quantizer block and the encoder block.

A. *DCT Engine*

The DCT engine is responsible for transforming the image data from time domain to spectral domain so that the inherent spatial redundancy could be reduced. JPEG is a lossy algorithm and typically achieves a compression ratio of around 10. The equations for calculating forward as well as the reverse DCT are as shown in equations (1) and (2). The hardware and software implementation of the same are explained in detail in the subsequent sections.

$$H(u,v) = \frac{2}{\sqrt{MN}} C(u)C(v) \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} h(x,y) \cos\frac{(2x+1)u\pi}{2M} \cos\frac{(2y+1)v\pi}{2N} \quad (1)$$

$$h(x,y) = \frac{2}{\sqrt{MN}} C(u)C(v) \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} H(u,v) \cos\frac{(2x+1)u\pi}{2M} \cos\frac{(2y+1)v\pi}{2N} \quad (2)$$

B. *Quantizer*

The next step in the JPEG compressor is the quantizer block, which performs an element-by-element division of coefficients with a standard matrix. There are different standards of quantization matrices available namely Q_{10}, Q_{50}, Q_{90} etc. We can choose any of these matrices depending on the demand of the application and in this paper we use Q_{50} matrix for quantization. High frequency coefficients in the lower right of the 8x8 DCT matrixes will be rounded to zero and this property of image is used in the compression of image by the encoder block.

C. *Entropy Encoder*

The 8x8 blocks of image data from the quantizer are given into the entropy encoder block. The main goal of entropy encoding is to reduce the statistical redundancy inherent in image data by encoding it with a suitable prefix-free code. The baseline encoder makes use of the standard Huffman code tables given in [7].

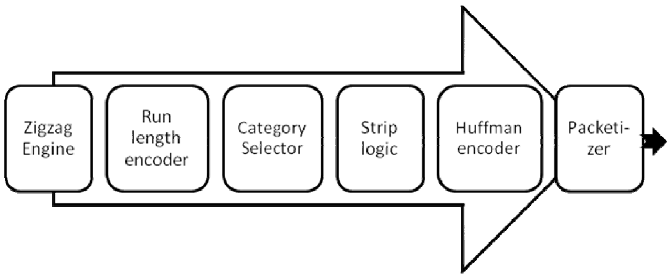

Fig.1. Entropy Encoder- overview

III. JPEG2000 STANDARD

JPEG2000 is a new standard put up by some major organizations like ISO, IEC, ITU etc. JPEG2000 makes use of Discrete Wavelet Transform (DWT) filling all the gaps made by the drawbacks of the JPEG standard, thus gaining wide spread acceptance.

The JPEG2000 compression includes the three main parts: Discrete Wavelet Transform (DWT), Quantization and Entropy encoding. DWT decomposes each component into a number of sub bands at various levels of resolution which can be independently quantized by the quantizer in case of lossy compression. This paper mainly focuses on the compression section of JPEG2000 and its hardware implementation.

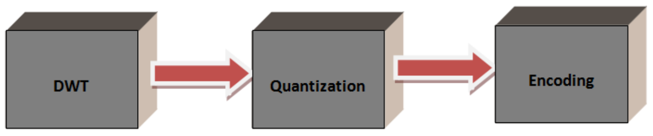

Fig.2. Overview of JPEG2000

A. *Discrete Wavelet Transform*

The basic structure of the first stage is shown in the figure.

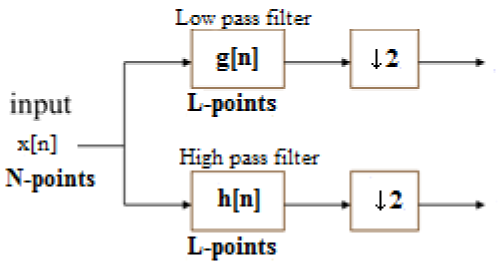

Fig.3. Block diagram of filter analysis

The relation equations are represented as follows:

$$x_{1,L}[n] = \sum_{k=0}^{K-1} x[2n-k]\, g[k]$$

$$x_{1,H}[n] = \sum_{k=0}^{K-1} x[2n-k]\, h[k] \qquad (3)$$

where g[n] is a low-pass filter like scaling function, and h[n] is a high-pass filter like mother wavelet function.

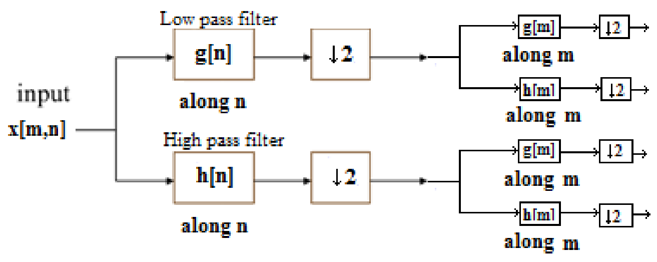

Fig.4. 2-D DWT

Compared with 1-D DWT, the main difference is that the 2-D DWT decomposes the low frequency and high frequency component in two dimensions separately, and the two-stage 1-D DWT decomposes twice in the same dimension.

B. *VLSI Architecture for the Convolutional Approach*

A semi-systolic architecture is used for the implementation of the convolution-based discrete wavelet transform. Although the basic principle of the architecture can be applied to implement any symmetric filter, we use the (9, 7) wavelet filter here as an example. The (9, 7) filter has been recommended for implementation of DWT in the JPEG2000 standard for its lossy mode of image compression. The design for the implementation of the DWT using the filter bank was implemented in Verilog HDL and the results are analyzed using by synthesized using Xilinx ISE. The simulator used for Verilog HDL is ModelSim 6.2c.

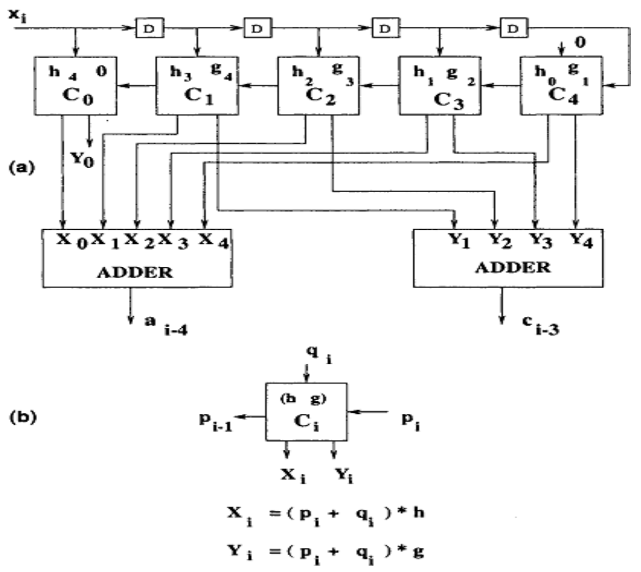

Fig.5. Semi Systolic architecture of DWT

IV. SOFTWARE IMPLEMENTATION

This section describes the software implementation of the DCT Engine of the JPEG standard and the DWT Engine of the JPEG2000 standard.

A. *DCT engine*

The captured image data, say of size 512x512, is split into sub-blocks of 8x8 and DCT is calculated as per the equation (4).

$$\mathrm{DCT} = T*(\text{image\_data})*T^{t} \tag{4}$$

where *T* is the standard transformation matrix and image data is the obtained by subtracting 128 from each pixel data [11]. The resulting 8x8 matrix has decimal as well as fractional parts.

$$T = \begin{bmatrix} .3536 & .3536 & .3536 & .3536 & .3536 & .3536 & .3536 & .3536 \\ .4904 & .4157 & .2778 & .0975 & -.0975 & -.2778 & -.4157 & -.4904 \\ .4619 & .1913 & -.1913 & -.4619 & -.4619 & -.1913 & .1913 & .4619 \\ .4157 & -.0975 & -.4904 & -.2778 & .2778 & .4904 & .0975 & -.4157 \\ .3536 & -.3536 & -.3536 & .3536 & .3536 & -.3536 & -.3536 & .3536 \\ .2778 & -.4904 & .0975 & .4157 & -.4157 & -.0975 & .4904 & -.2778 \\ .1913 & -.4619 & .4619 & -.1913 & -.1913 & .4619 & -.4619 & .1913 \\ .0975 & -.2778 & .4157 & -.4904 & .4904 & -.4157 & .2778 & -.0975 \end{bmatrix}$$

DCT module is implemented using the above transformation matrix and is compared with the inbuilt MATLAB command *dct2 ()* and upon comparison we obtained a tolerable error in the order of 10^{-5}. The efficiency of algorithm is computed with Peak Signal to Noise Ratio (PSNR) value as per the given equations.

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2 \tag{5}$$

$$PSNR = 10.\log_{10}\left(\frac{MAX^2}{MSE}\right) \tag{6}$$

Images, which are usually of larger size, are subdivided into 8x8 blocks and this must go as input to the DCT module. So there are 64 coefficients that are input to DCT module each coefficient taking one clock cycle. Testing is not easy by taking into consideration the various combinations of input. Hence, we use MATLAB to write the image data into a text file using commands like *fopen()* and *fwrite()*.In Verilog test bench, we use *$readmemb()* function that reads an image and store it as a 2D vector. Finally, the quantization results in MATLAB are compared to the output waveform and hence design is verified.

B. *DWT engine*

The main parts of software implementation include the implementation of JPEG2000 Discrete Wavelet Transform in MATLAB. The testing and simulation were performed on various test images and they were reconstructed for comparison purpose to determine the efficiency of the compression. Since the hardware implantation algorithm was based on sub band coding using the filter realizations, the same was again implemented in MATLAB for verifying the results. The MATLAB environmental provided a perfect platform for the software implementation by exactly realizing the filters so that there is a solid layout for the hardware implementation to start with.

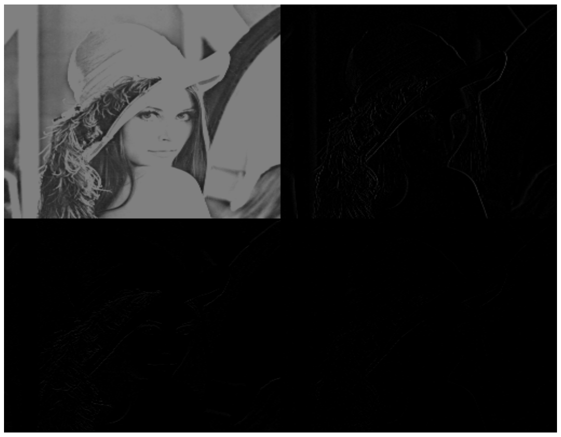

Fig.6. 512x512 image after both row wise and column wise filtering (Haar wavelet)

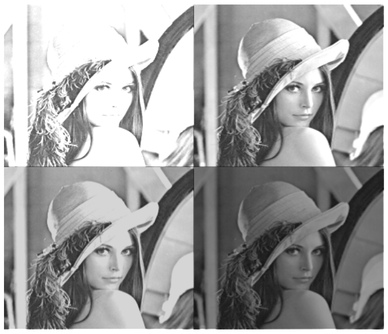

Fig.7. Lena 512x512 image after both row-wise and column-wise filtering (Daubechies wavelet)

V. HARDWARE IMPLEMENTATION

This section delves into the hardware implementation of the DCT Engine of the JPEG standard and the DWT Engine of the JPEG2000 standard.

A. *JPEG standard*

Most important part of JPEG compressor is the DCT engine. Direct implementation of DCT engine in hardware requires a large number of adders and multipliers. For the computation of DCT values, image data is represented in signed 2's complement form in the range -128 to 127. Since images of larger size involve complex arithmetic for finding DCT, the images are sub-divided into blocks of 8x8. Complex computations are further reduced by decomposing the 8x8 two-dimensional DCT into row wise DCT and then column wise DCT.

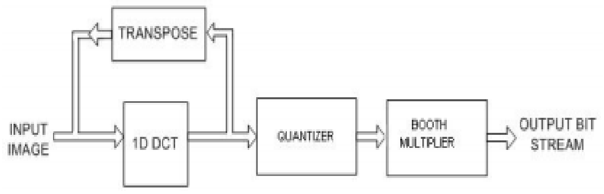

Fig.8. Iterative architecture of DCT, quantizer block

The iterative architecture of the 2D DCT engine is as show in the above figure. In this paper, an iterative architecture is preferred over pipelined architecture for two-dimensional DCT so that the one-dimensional DCT module can be reused saving lot of power as well as area. The computations of DCT are done as in [10] but instead of using Distributed Arithmetic Multiply-and-Accumulate (MAC) operation, we are using Booth multipliers and adders.

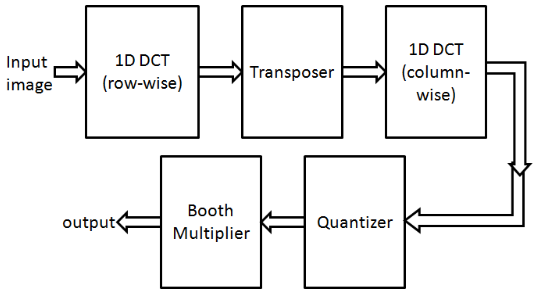

Fig.9. Pipelined architecture of DCT, quantizer block

In the DCT module, inputs are of 8 bits and hence the output after 7 additions can go to a maximum value of 15 bits. This approach takes 16 bits to represent it so that occurrence of overflow is avoided. We have to implement many additions and multiplications for computing forward DCT. Hence Booth multiplier is preferred as it gives better performance by reducing the number of multiplications from N to N/r where r is the radix of multiplication. Also, there is no need to find the two's complement of the negative numbers. While multiplying two 16 bit numbers, the answer generally becomes double the number of bits and hence, here it becomes 32 bits.

The transposer block accepts 32 bit image data as input from the DCT module that performs row-wise one-dimensional DCT and stores in an internal buffer. After receiving all the elements of the 8x8 matrix, this module transposes the matrix and gives as output i.e. it forwards the output row-wise which was written column-wise into the matrix. To achieve this functionality, we used eight shift registers which shift out the data into the memory as it is received from the input port. When the memory is full, the 32 bit data is then shifted out and is fed to the DCT block to perform one-dimensional DCT again to finally get a two-dimensional DCT matrix for further processing. We perform a bit reduction by ignoring LSB so as to avoid further increase in bits and ensuring that the error is minimal.

The quantizer module quantizes the value i.e. performs an element by element division which is multiplication of inverse of the numbers. So a buffer stores the incoming values and transfers it to booth multiplier module which performs the actual quantization. Fractional part is approximated to nearest decimal place and this approximation is the only lossy part in JPEG.

We have implemented both iterative and pipelined architecture for DCT engine along with quantizer. A comparative study is done between the two architectures. We see that iterative architecture occupies much lesser area compared to the pipelined architecture. Synthesis is done in Virtex2P FPGA and selected device is 2vp30ff1152-6.

TABLE I
DCT ENGINE- TIMING SUMMARY

Architectures	**Iterative Architecture**	**Pipelined Architecture**
Maximum Frequency	66.146MHz	66.146MHz
Minimum input arrival time before clock	6.448ns	6.697ns
Maximum output required time after clock	4.658ns	4.631ns
Maximum combinational path delay	5.809ns	7.448ns

TABLE II
DCT ENGINE- DEVICE UTILIZATION SUMMARY

Architectures	**Iterative Architecture**	**Pipelined Architecture**
Number of Slices	6523	8701
Number of Slice Flip Flops	7572	10062
Number of 4 input LUTs	8909	12375
Number of IOs	20	20
Number of bonded IOBs	20	20
IOB Flip Flops	8	8
Number of BRAMs	2	2
Number of GCLKs	1	1

B. JPEG2000

The key difference between current JPEG and JPEG2000 standards start with the adoption of discrete wavelet transform (DWT) instead of the 8x8 block based discrete cosine transform (DCT). DWT essentially analyzes a tile (image) component to decompose it into a number of sub-bands at different levels of resolution. Similar to what was done with DCT, the two-dimensional DWT is performed by applying the one-dimensional DWT row-wise and then column- wise in each component. In the first level of decomposition, four sub bands LL1, HL1, LH1, and HH1 are created.

For lossy compression using DWT, the default wavelet filter used in the JPEG2000 standard is the Daubechies (9, 7) biorthogonal spline filter. By (9, 7) we indicate that the analysis filter is formed by a 9-tap low-pass FIR filter and a 7-tap high-pass FIR filter where both filters are symmetric.

For lossless compression, the default wavelet filter used in the JPEG2000 standard is the Le Gall (5, 3) spline filter. Although this is the default filter for lossless transformation, it can be applied for lossy compression as well.

The Implementation of DWT core of JPEG2000 was done by the filter realization of DWT (Daubechies filters). A comparative study of the two filters: 9/7 wavelet filter and 5/3 wavelet filter was performed (see Table III and IV). The 5/3 filter was found to be better in terms of hardware and image quality was found to be almost same in both the filters. The 5/3 filter requires only less hardware as it has less number of coefficient and the coefficients are simple.

TABLE III
COMPARISON OF FILTER COEFFICIENTS

	9/7 wavelet filter			**5/3 wavelet filter**	
n	Low pass Filter	High pass Filter	n	Low pass	High pass
0	0.602949	1.115087	0	6/8	1
±1	0.266864	-0.591271	±1	2/8	-1/2
±2	-0.078223	-0.057543	±2	-1/8	
±3	-0.016864	0.0912717			
±4	0.0267487				

TABLE IV
COMPARISON OF SYNTHESIS REPORT OF FILTER BANKS s

Category	**9/7 filter**	**5/3 filter**
Flip-Flops	90	36
Number of Slices:	116	35
Number of 4 input LUTs:	214	60
Number of IOs:	178	66
Number of bonded IOBs:	178	66
IOB Flip Flops:	96	38
Number of MULT18X18SIOs:	4	2
Number of GCLKs:	1	1
Minimum input arrival time before clock	17.237ns	13.263ns
Maximum output required time after clock	4.283ns	4.283ns
Maximum combinational path delay	12.954ns	8.80ns

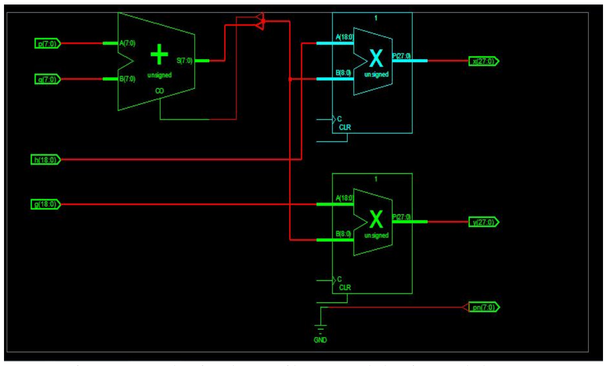

Fig.10. Synthesized 9/7 Filter -Bank basic module

VII. ENTROPY ENCODER

In the JPEG baseline encoder, the main function of the entropy encoder is to code the quantized coefficients from the encoder model using variable length encoding. The output of the DCT block is fed to the Entropy Encoder for source coding. The aim of source coding is to take the source data and make it smaller which is essential in a data compressor design. Entropy of a source is the measure of information or rather the randomness of information. At the ground level, source coding aims at reducing the data redundancy that is inherently present in the source, and then represent the source with lesser bits while maintaining the same information. Image compression makes extensive use of entropy encoding.

Our implementation tries to implement the entropy encoder in a pipelined manner where we have divided the entire entropy encoder engine into smaller independent blocks. Such a pipelined architecture will consume input data at the same rate as the DCT Engine. The lack of design specific literature for the entropy encoder with respect to the JPEG standard led us to come up with this design. Our implementation of the entropy encoder is based on the design suggested in [8].

A. Overview

The entropy encoder process flow consists of 1) ZigZag Engine, 2) Run Length Encoder, 3) Category Selector, 4) Strip Logic, 5) Huffman Encoder, and 6) Packetizer. Except the run length encoder, all other modules are extensively redesigned to attain reduced complexity. The main aim behind this approach to the implementation of the entropy encoder is to achieve a linear pipe with a small clock period for each stage.

B. ZigZag Engine

Each block of data that is output by the quantization module from the encoder model needs to be reordered in a zigzag fashion before being forwarded to the entropy encoder. This reordering is achieved using two 64x8 buffers that are organized in such a fashion that while one buffer is used to rearrange the incoming pixel values the already rearranged image data of previous image is shifted to the next block for run length encoding. This arrangement is shown in figure [11]. The reordering is done in a dynamic way, i.e. the system does not have to wait till all the 64 pixel values are obtained. The selection of buffers is done using a 2x1 MUX.

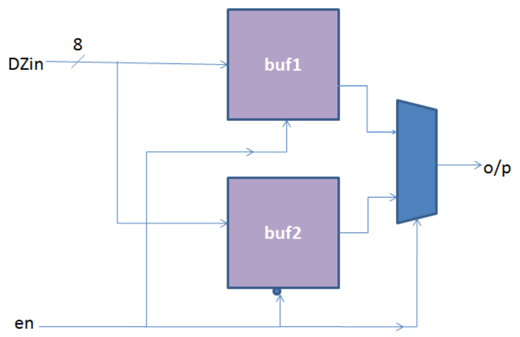

Fig.11. Block level diagram for the ZigZag Engine

C. *Run Length Encoder*

The zero-run length coder module performs the function of producing the run length count of null values and generates other status symbols. The coefficient count is maintained using a counter which is incremented every time a valid input is given. The very first pixel data of an image would be its DC component which is recognized when the counter is first incremented. The rest 63 coefficients are AC coefficients. The various stages of the zero run length coder are shown in figure [12].

The control logic block is primarily responsible for two things: differentiating between DC and AC coefficients and maintaining the run length counter. The control logic is also responsible for generating signals that indicate conditions such as the occurrence of 16 consecutive zeros, end-of-block, and whether the output data value represents a DC or an AC coefficient.

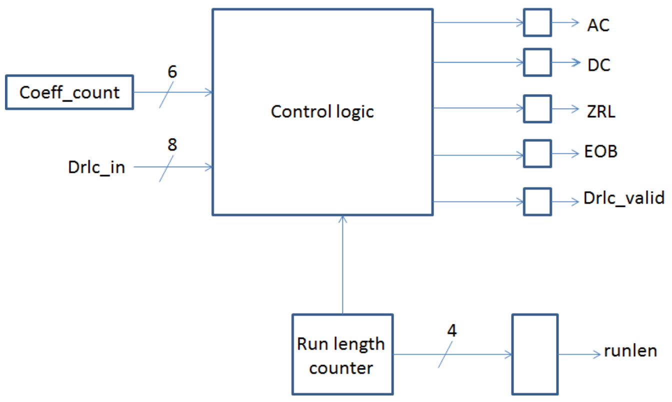

Fig.12. Run Length Encoder

D. *Category Selector*

This stage of the JPEG process is implemented using our original design. Category selection is defined in the JPEG compression standard [7]. It is important to note that for the sake of simplicity, we have considered only up to category 7 which holds good for most of the practical applications.

A straightforward implementation of category selection would require storing the ranges in memory and comparing the input data with those pre-stored values which requires complex address decoding and control logic. However, the table memory can be avoided and the entire category selection can be achieved with a simple combinational circuit. This circuit operates like an encoder that converts the given coefficient into the corresponding category in a single clock cycle. The circuit is given in figure [13].

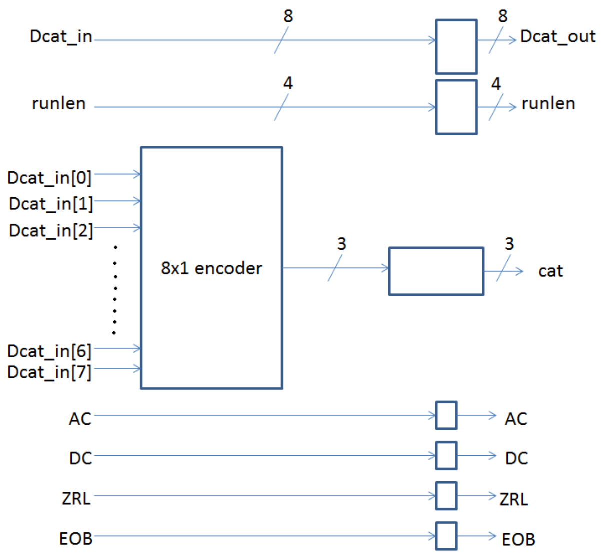

Fig.13. Category Selector

E. *Strip Logic*

The strip logic shown in Fig. is a slightly modified version of that presented in [7] and consists of five stages instead of four. The two main aims of this stage are to discard the zero-valued coefficients, as well as the redundant ZRL symbols occurring before an EOB symbol. Each stage has three registers to hold the coefficient, run length count and category fields corresponding to a data element output by the category selection unit and a set of 1-bit registers to hold the corresponding status. The status bits are decoded and used to strip the zero-valued coefficients and also to strip off the ZRL symbols that precede an EOB symbol. It should be noted that there could be a maximum of three ZRL symbols preceding an EOB symbol. The strip logic acts as a five-stage buffer through which the compressed data elements, after the removal of zero coefficients travel, before being forwarded to the Huffman encoder. The valid bit signal is set to high whenever valid data is being output by the strip logic for Huffman encoding. It must be noted that the ZRL bit needs to be reset whenever a ZRL symbol has been deleted from the data stream.

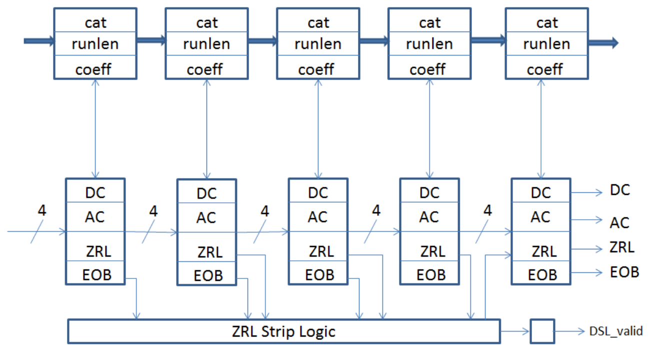

Fig.14. Strip Logic

F. *Huffman Encoder*

The Huffman encoder module that we have implemented is a heavily modified version of the design suggested in [7]. It consists of Huffman code tables stored in random access memory modules and logic for replacing the category, run length count pairs with the corresponding Huffman codes. The table is accessed by using the {runlen, cat} pair for addressing. The input data passes through each of the two stages, and depending on the address, the corresponding Huffman code and the code length are output. The hardware organization is shown in figure [15].

G. *Packetizer*

The Data Packer unit shown in Figure [16] is a heavily modified version of the one suggested in [7]. It is used to convert variable length compressed data into fixed length compressed data stream. The logic consists of registers A and B, two left-shift units, a masking logic unit, two ORing logic units, and control logic, which includes the two registers LENGTH and ENDP.

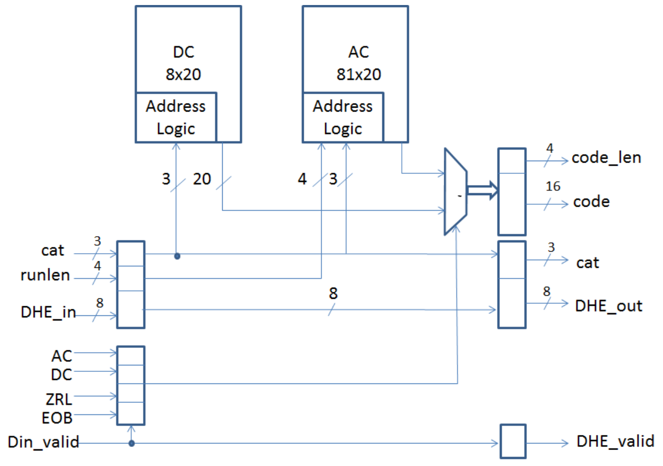

Fig.15. Huffman Encoder

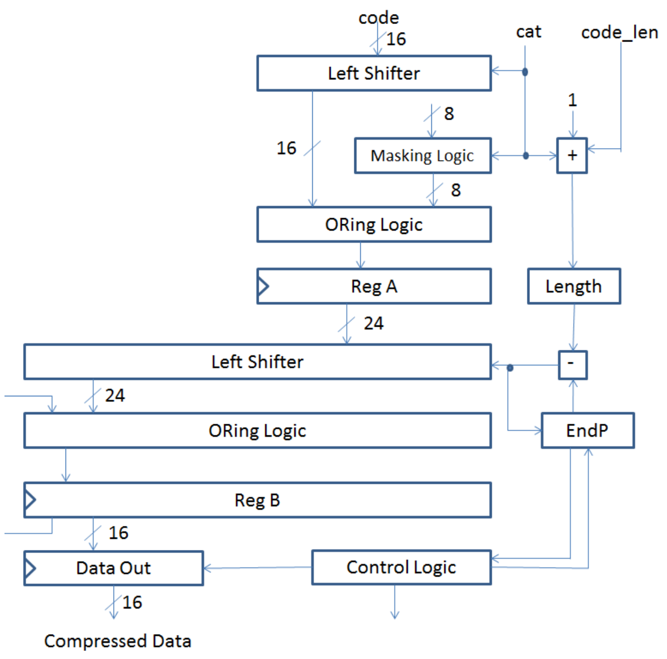

Figure.16. Packetizer

VI. CONCLUSION

In this paper we have described the implementation of a fully pipelined architecture for JPEG baseline image compression standard. The architectures for the various stages are based on efficient and high performance designs suited for VLSI implementation. The implementation was tested for functional correctness using Verilog with Mentor Graphics ModelSim 6.5 SE and Xilinx ISE 10.1. There were several reasons for our choice of implementing the compression algorithms like JPEG and JPEG2000. Firstly, we wanted to work on the implementation of compute intensive algorithms in hardware using HDLs. Secondly, the JPEG and JPEG2000 compression standard was a good candidate as it does not specify any particular architecture for its implementation and in this way permits the implementers to try various innovations. Our project provided us the opportunity to experiment with various design trade-offs on numerous occasions. Also, during our implementation we found that we needed to try different innovations to suit our requirements. Our architecture has been completely synthesized and work for its actual verification in a practical setup is in progress.

ACKNOWLEDGMENT

We gratefully acknowledge the Almighty GOD who gave us strength and health to successfully complete this venture. The authors wish to thank Amrita Vishwa Vidyapeetham, particularly the HUT Labs for kind support and guidance. We would also like to thank VLSI lab for access to their research facilities.

REFERENCES

[2] Gautam and Bhawna, “Image compression using discrete cosine transform and discrete wavelet transform,” in National Institute of Technology, Rourkela, 2010.

[3] K. K. M. Vijay Kumar Sharma1 and U. C, “An efficient distributed arithmetic based vlsi architecture for dct,” in National Institute of Technology, Rourkela, 2010.

[4] R. K. Megalingam, V. Krishnan, V. Sarma, M. M, and R. Srikumar, “Hardware implementation of low power, high speed dct/idct based digital image watermarking,” in Proceedings of the IEEE, Vol. 83, No. 2, 1995.

[5] T. Acharya and P.-S. Tsai, JPEG 2000 Standard for Image Compression: Concepts, Algorithms and VLSI Architectures. John Wiley, 2005.

[6] D. A. Huffman, “A Method for the Construction of Minimum-Redundancy Codes,” in Proceedings of the IRE, Vol. 40, No. 9, 1952, pp. 1098–1101.

[7] JPEG-Committee, “Information Technology - Digital Compression And Coding Of Continuous-Tone Still Images - Requirements And Guidelines,” in Recommendation T.81, 1995.

[8] M. Kovac and N. Ranganathan, “JAGUAR: A Fully Pipelined VLSI Architecture for JPEG Image Compression Standard,” in Proceedings of the IEEE, Vol. 83, No. 2, 1995.

[9] A. G. Claude Berrou and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo codes,” in Proceedings of the 1993 International Conference on Communications, 1993, pp. 1064 1070.

[10] Vijay Kumar Sharma, K. K. Mahapatra and Umesh C. Pati, “An Efficient Distributed Arithmetic based VLSI Architecture for DCT” at the Computing, Communication and Applications (ICCCA), 2012 International Conference.

[11] Image Compression and Discrete Cosine Transform by Ken Cabeen and Peter Gent, Math 45, College of Redwood.