JPEG Compression through VLSI Design

Antonio Anzaldua, *Member, IEEE*, Franklin Sinche, Member, IEEE

*Abstract*— A JPEG utilizes a lossy form of compression based on a discrete cosine transform (DCT) which expresses a finite sequence of data points in terms of a sum of cosine functions constantly oscillating at various frequencies. A lossy form of compression is referred to as irreversible compression is a method of data encoding which uses inexact approximations and partial data discarding to represent the actual content. JPEG compression can be divided into the following phases: splitting image (or downsample), forward DCT, quantization, entropy encoding, retrieve encoded image, entropy decoding, dequantization, reverse DCT, and decoding image into blocks. In this paper we will be implementing each phase of JPEG compression in the coding language Verilog. That will allow us to create the chip layout via the CAD Tool Encounter.

# INTRODUCTION

I

mage compression is a highly discussed topic in commercial, industrial, and academic applications. Issues with management of data in complexity, storage, and data processing can be optimized through data compression. This is why the JPEG is a commonly used method of compressing digital images. There are two techniques that could be used for JPEG compression: Lossless and Lossy schemes. Using a lossy form of compression based on a DCT allows the designer to access a finite sequence of data points in terms of a sum of cosine functions through an oscillating frequency. The issue that comes with a lossy form of compression is the fact that some original data could be lost and cannot be restored. For this is why the method of lossless compression is favored. Lossless compression uses a predictive scheme based on the nearest (causal) neighbors and entropy coding which is for finding the prediction error. The lossless compression of an image will not lose original data information however this scheme is approach is not widely supported in products. A lossless compression also leads to small compression ratios that fail to meet the needs of many applications. For these reasons we will be implementing a lossy form of JPEG compression through the CAD tool, Encounter.

# Design Approach

In order to properly implement a lossy from of image compression we will need to abide to the following steps: split the image (or downsample), forward DCT, quantization, entropy encoding, retrieve encoded image, entropy decoding, dequantization, reverse DCT, and decoding image into blocks (Figure 1). All steps can be established in three phases; Discrete Cosine Transform, Quantization, and Entropy Encoding. Each phase will need to be completely understood in order to create a Verilog code that will allow us to implement this design on a chip. Once each phase is tested the simulation should yield a correct waveform that displays the appropriate storage of all pixel values and an accurate DCT.

# The Process

There are three main steps for JPEG compression, each step will have an inverse or mirrored sequence after one completion of the initial set of steps. The first step will be to obtain image blocks that can go through a Discrete Cosine Transform. The following step will be to quantize the values in each block after the DCT. The third step will be to begin entropy encoding; this step can be intuitively completed by creating an encoder. The remaining three steps will be the entropy decoding, dequantization, and inverse DCT.

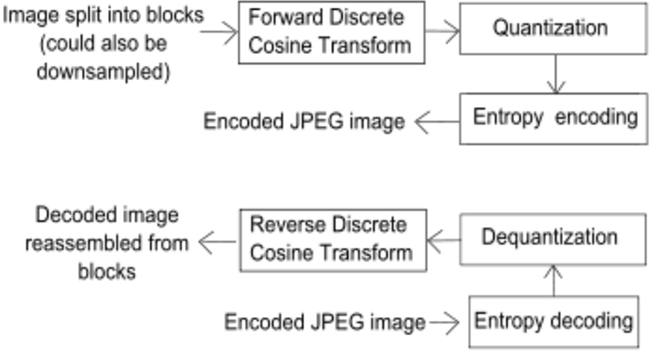


Fig.1. Baseline sequential JPEG encoding and decoding processes

## Discrete Cosine Transform (DCT):

A discrete cosine transform expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies [1]. This is a technique for converting a signal into frequency components used to compress images. Usage of cosine over sine functions is a very crucial element for compression due to the lesser cosine functions needed to approximate a signal. For analysis of two-dimensional (2D) signals such as images we need a 2D version of the DCT. The equation that is commonly used for this compression can be viewed in Figure 2.

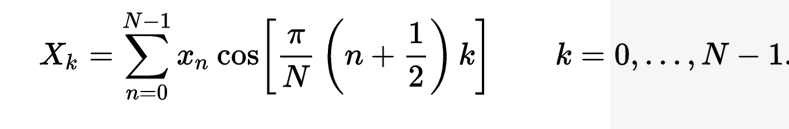


Fig.2. DCT-II (1D) Equation used for image compression

Referring back to Figure 1, there is a step where we will need to calculate the reverse DCT or inverse. The inverse DCT can be computed by multiplication with the inverse DCT matrix or by following the equation in Figure 3.

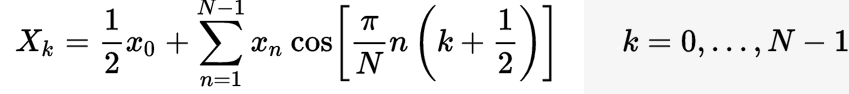


Fig.3. DCT-III Inverse DCT Equation used for image compression

Both forms of DCT will be necessary to compute to begin to encode and decode an image for compression. Since we will be using a two-dimensional signal, the equation for finding the DCT will vary slightly. The algorithm needed to compute a two-dimensional DCT of an image or matrix is the DCT-II (1D) equation performed along the rows and columns of the DCT matrix, (Figure 4).

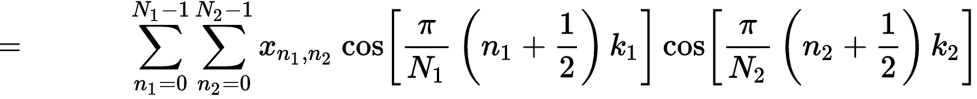


Fig.4. DCT-II (2D) Equation used for image compression

## Quantization

This a tool used throughout the world of mathematics for processing a digital signal and mapping a large set of input values to a countable set [2]. Quantization is involved to some degree in nearly all digital processing of signals. This is a lossy compression technique achieved by compressing a range of values to a single quantum value. As soon as the number of discrete symbols in a given stream is reduced, the stream becomes more compressible. As humans, we are able to detect small differences in brightness over a large area, but the same cannot be said about distinguishing the exact strength of a high frequency brightness variation. The analysis of pixels on the image will be divided into blocks and subjected to DCT calculations to determine the frequency components. The resulting block or matrix will be pre-multiplied by the quantization scale code and divided element-wise by the matrix followed by the rounding of each element. The result should yield a matrix with values primarily in the upper left corner (low frequency) [2]. The quantized matrix can be efficiently stored compared to its’ non-quantized form. In order for us to create the equivalent process through Verilog we will need to establish a 8x8 block of DCT coefficients. Next we will need to allow the user to give an input value ranging from 1 to 100. Inputting a 1 will represent poor quality image with high compression and 100 represent high quality and low compression.

## Entropy Encoding

In information theory, an entropy encoding is a lossless data compression scheme that is independent of the specific characteristics of the medium [3]. Since this step in image compression is lossless, the original data will not be lost after the quantization process. There are two common entropy encoding techniques called the Huffman Coding and the Arithmetic Coding [3]. Huffman coding is a particular type of optimal prefix code that is a form of lossless compression. At this point of the image compression process we seek all opportunities that allow us to complete the process without losing any data. For this reason, Huffman coding is what we are striving to utilize in this project. This step aims to reduce the number of bits used to represent the image. The entire quantized matrix’s coefficients are coded into the binary format by a basic encoder.

# Matlab Process & Results

Since the input of our analysis will be an image, it makes sense to begin with MATLAB to establish algorithms for each step in the JPEG compression process. The algorithms used will need to be well thought-out so Verilog transposition will be possible.

## Discrete Cosine Transform

The great thing about MATLAB is that you can actually give a signal input and use the command ‘dct ( )’ and actually get the DCT of the signal. However if the endgame is to have a chip layout via Encounter, the algorithm needs to be clear for Verilog interpretation.

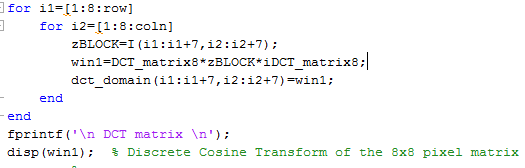


Fig.5. DCT performed in MATLAB

The code shown in figure 5 is a method of approximating the DCT of an input by a direct 2-D matrix factorization method that involves the use of an Identity matrix [4]. This will make the computational process a lot easier however it will only be an approximation and will not be as accurate as completing the normal DCT. Both for loops located in figure 5 are to focus on each row and column in the 8x8 matrix. ‘*DCT\_matrix8’* is the identity matrix of 8 which is being multiplied to corresponding rows and columns in the pixel matrix retrieved from the image, ‘*zBlock;’* this will be multiplied one more time by ‘*iDCT\_matrix8*’ which is another identity matrix except with ones instead of eight. This will display the DCT representation of the images’ matrix on the command window. The advantage of this approach is by coding this algorithm with Verilog Hardware Description Language (HDL), the simulation and verification of circuit functionalities and timing characters are easily determined using Cadence tools [4]. Figure 6 shows the original photo before it passes through the DCT algorithm without using the approximation method. The image is shown as pixel cubes; these are the 8x8 matrices that represent the pixel intensities of the image.

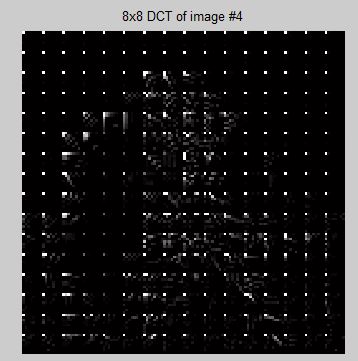


Fig.6. Original Input image Fig.7. DCT of image; 8x8 matrices

## Quantization

The next step for JPEG compression is to take the DCT matrix and perform quantization of each entry of the matrix. This is the main source of the Lossy compression; the values in figure 8. These values are known as the quantization table, they are chosen to preserve low-frequency information and discard high-frequency, noise, since human perception seems to be less critical to the loss of information in this aspect.

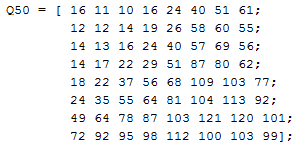


Fig.8. Quantization table shown in MATLAB

The code written in MATLAB script begins with allowing the user to give a desired level of compression they want to be performed before executing the script. There is a couple of ‘if statements’ shown in figure 9 that check if the user selected values less than 50, greater than 50, and equal to 50 for compression quality. Depending on the desired quality of compression the quantization table found in figure 8 will be adjusted accordingly.

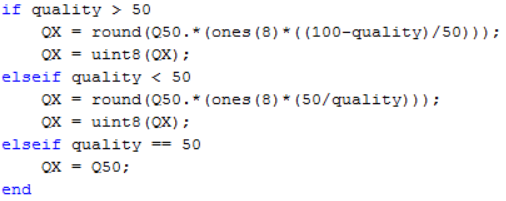


Fig.9. Conditions to find compression quality matrix performed in MATLAB

The output of the conditions above will be used to quantize the DCT matrix based off the selected quality of compression.

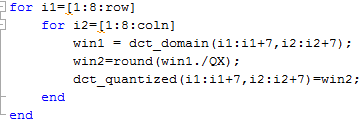


Fig.10. Quantization performed in MATLAB

Figure 10 demonstrates how each row and column of the DCT matrix will be divided by the ‘*QX’* which is the compression quality matrix. The next step will be to dequantize the matrix. This is simply the inverse operation of what was executed in quantizing the matrix. To achieve the dequantized matrix simply multiply the output value ‘*win2’* found in figure 10 by the compression quality matrix.

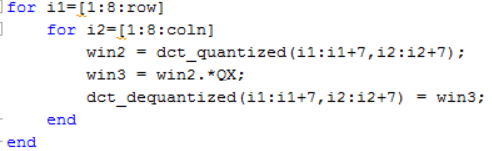


Fig.11. Dequantization performed in MATLAB

## Inverse DCT

The final step to achieve the new compressed pixel intensity matrix is to find the inverse discrete cosine transform (IDCT). Obtaining the IDCT is very similar to finding the DCT; we will take the Identity matrix with diagonal values equal to eight and multiply it with the deqauntized matrix that we obtained in figure 11, ‘*win3*.’ Then the product of the Identity matrix by dequantized matrix will be multiplied with another identity matrix which is demonstrated in figure 12.

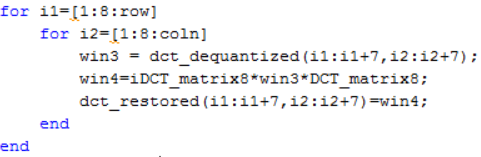


Fig.12. Quantization table shown in MATLAB

## Results

After running a few tests of the code the results obtained showed if the user chose 75% quality compression of the original image to be performed it necessarily did not indicate the image would be the best quality. This correlates to the theory that low compression gives better quality of the image and high compression will deliver low quality resolution. Viewing the images compared to the original in figures 13, 14, and 15 the user is able to notice these claims of quality resolution being altered with desired compression.

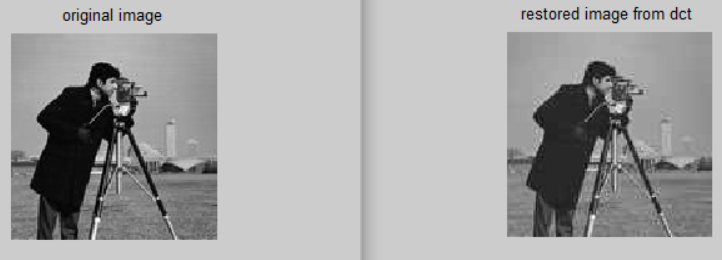


Fig.13. MATLAB results of JPEG compression, 30% quality of compression

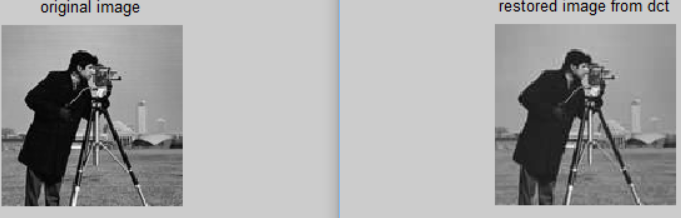


Fig.14. MATLAB results of JPEG compression, 50% quality of compression

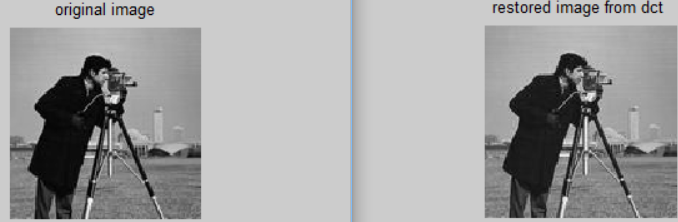


Fig.15. MATLAB results of JPEG compression, 75% quality of compression

What next? Well based off the factorization method provided by Hsiao, Shen-Fu, Yu Hen Hu, T-B Juang, and Chung-Han Lee, the results allow the design to get an approximation DCT and IDCT. This is a very crucial method since it will allow this design to be created into a VLSI design. Through the coding language Verilog, the design will be a challenging transposition but will be more feasible to go from MATLAB to Verilog. At that point the design can be processed through the CADENCE tool, Encounter. Using this tool will allow the design to be given a chip layout to be made ready for fabrication.

# Verilog Implementation

Implementing the MATLAB data analysis was very challenging and provided us several obstacles that we had to overcome. Through trial and error, we produced a very well laid out Verilog code which incorporated many key steps to JPEG compression.

We began setting the quality of compression to 50 which provided best quality compression; any higher would have provided blurred and pixeled image. Also in turn, provided us a 8x8 matrix pixel intensity that will be used to divide into the DCT matrix gathered from MATLAB. Every value of the 8x8 matrix of DCT was divided by the Pixel intensity matrix. This process was the quantization and was achieved by creating a binary division module that was instantiated in the top module and was called 64 times. Following Quantization, Entropy process began. In this case, two modules were instantiated, the 10-4 Encoder and the 4-10 Decoder. Both simulating the Entropy encoding process that is done for the JPEG compression.

In the Dequantization process, the pixel intensity matrix was multiplied to the Entropy matrix. This was done by utilizing behavioral modeling. Finally, Inverse DCT was done by multiplying the Identity matrix to the Pixel intensity matric and having that value multiplied to the Dequantization process.

# Results and Chip Layout

Upon synthesizing the top module, no errors were found except one warning. It stated that all resources were utilized, we moved forward giving our quality compression matrix and this lead to no simulation due to engine fuse error. We believe since a lot of storage was used and the software capabilities were exceeded that the error was given.

# Conclusion

The JPEG continuous-tone image compression guideline is not a remedy for all myriad issues that will come up before digital images are fully integrated within all applications that in the end will benefit from them. If two applications, for example, cannot exchange uncompressed images because they use incompatible color spaces, aspect ratios, dimensions, technical specifications, etc. then a common compression method will not help. These applications will go through a thorough technical evaluation, testing, selection, validations, and documentation work which JPEG committee members have performed is expected to soon yield an approved international standard that will withstand the tests of quality and time. As different imaging applications become increasingly common and are implemented on open networked computing systems, the ultimate measure of the committee’s success will be when JPEG-compressed digital images come to be regarded and even taken for granted as “just another data type”, as text and graphics are today.

References

1. *"Discrete Cosine Transform." Wikipedia. Wikimedia Foundation, n.d. Web. 06 Nov. 2016.*
2. *"Quantization (signal Processing)." Wikipedia. Wikimedia Foundation, n.d. Web. 06 Nov. 2016.*
3. *"Entropy Encoding." Wikipedia. Wikimedia Foundation, n.d. Web. 06 Nov. 2016.*
4. Hsiao, Shen-Fu, Yu Hen Hu, T.-B. Juang, and Chung-Han Lee. "Efficient VLSI Implementations of Fast Multiplierless Approximated DCT Using Parameterized Hardware Modules for Silicon Intellectual Property Design." *IEEE Transactions on Circuits and Systems I: Regular Papers* 52.8 (2005): 1568-579. Web.