The JPEG standard is most widely used method of lossy compression for digital photography. In this paper, we will discuss the implementation of JPEG Encoder for FPGAs. The target device is a Stratix IV FPGA. The JPEG Encoder was synthesized and simulated using the Quartus FPGA design software at the clock frequency of 122.5 MHz. The design was tested on an image for different values of the Quality factor. We will also discuss the hardware utilization and the performance results of the implemented design on the FPGA. We will start with some background about the FPGA encoding technique in the next section.
**Table of Contents**

1. Background: JPEG Encoder ................................................................. 2
2. Implementation......................................................................................... 7
3. Simulation & Synthesis Setup ................................................................. 8
4. Synthesis Results ................................................................................... 9
   4.1 Hardware Resources Utilized.............................................................. 9
   4.2 Performance ....................................................................................... 10
5. Simulation Results ................................................................................ 10
6. Conclusion & Future Work .................................................................... 12
References .................................................................................................... 13
1. Background: JPEG Encoder

The JPEG (Joint Photographic Experts Group) is the most commonly used method of lossy compression for digital photography. The block diagram of the JPEG Encoder is shown in the Figure 1. The main steps in the JPEG Encoding are: RGB to YCbCr conversion, 2-D DCT of 8X8 blocks, Quantization, zig-zag scan of 8X8 blocks, Differential DC Encoding and Run Length Encoding (RLE) for AC components and Entropy Encoding. In this section, all the steps of JPEG Encoding will be discussed in detail.

Figure 1. JPEG Encoder Block Diagram
RGB to YCbCr conversion

The JPEG compression requires conversion of the RGB input values to the corresponding Y, Cb & Cr values. This conversion is performed using the formula given below.

\[
Y = 0.299 \times \text{Red} + 0.587 \times \text{Green} + 0.114 \times \text{Blue}
\]

\[
Cb = -0.1687 \times \text{Red} - 0.3313 \times \text{Green} + 0.5 \times \text{Blue} + 128
\]

\[
Cr = 0.5 \times \text{Red} - 0.4187 \times \text{Green} - 0.0813 \times \text{Blue} + 128
\]

8X8 2-Dimensional DCT

The key to the JPEG baseline compression process is a mathematical transformation known as the Discrete Cosine Transform (DCT). The basic purpose of the DCT operations is to take a signal and transform it from one type of representation to another. The DCT in JPEG Encoding is used to convert the signal (spatial information) into numeric data (“frequency” or “spectral” information) so that the image’s information exists in a quantitative form that can be manipulated for compression.

The top, left element called the DC value of a two-dimensional DCT matrix contains a value that is almost always of a very great magnitude. The farther away an AC term is from the DC term, the higher the frequency its corresponding waveform will have and the smaller its magnitude will be.
The formula for 8X8 2D DCT is given as:

\[ G_{u,v} = \sum_{x=0}^{7} \sum_{y=0}^{7} \alpha(u)\alpha(v)g_{x,y} \cos \left[ \frac{\pi}{8} \left( x + \frac{1}{2} \right) u \right] \cos \left[ \frac{\pi}{8} \left( y + \frac{1}{2} \right) v \right] \]

where \( u \) is the horizontal spatial frequency, \( u \in [0,7] \)

And, \( v \) is the vertical spatial frequency, \( v \in [0,7] \)

\[ \alpha(u) = \begin{cases} \sqrt{\frac{1}{8}}, & \text{if } u = 0 \\ \sqrt{\frac{2}{8}}, & \text{otherwise} \end{cases} \]

**Quantization**

The quantization part is the main lossy component of the JPEG Encoding. Each DCT component is divided by a separate quantization coefficient, and rounded to the nearest integer step in the JPEG compression algorithm. This process removes a considerable amount of information and thus compresses the data enormously. The amount of compression done depends on the selection of the quantization matrix. The quantization matrix is not fixed and it can be even selected based on the image. The quantization matrix as specified in the original JPEG standard is given by:
The Quantized DCT values are calculated using the formula:

\[ B_{j,k} = \text{round} \left( \frac{G_{j,k}}{Q_{j,k}} \right) \text{ for } j = 0, 1, 2, \ldots, 7; \ k = 0, 1, 2, \ldots, 7 \]

Where, \( G \) is the unquantized 8X8 DCT block

And, \( Q \) is the quantization matrix.

**Zigzag Sequencing**

The zigzag process shown in the Figure 2 is used to arrange the quantized matrix from low to high spatial frequencies. This process gives more compression by putting more order in the entropy. Essentially our lower-frequency components, which describe the gradual luminance changes, are more important to the human visual system than the high frequency changes since human eye cannot find the changes in higher frequency components. By ordering the more important coefficients in the beginning of the 8x8 block, we can expect more runs of zeros later after quantization, toward the end of the 8x8 block [1].
**Differential DC Encoding**

The Differential DC encoding as shown in Figure 2 is performed on the DC components from each of the 8X8 blocks. Differential DC is calculated as:

\[
\text{Diff DC} = \text{DC}(\text{block}_i) - \text{DC}(\text{block}_{i-1})
\]

**Figure 2: Differential DC encoding and Zig-zag sequence**

**Entropy Encoding (RLE)**

The AC components in the zigzag sequence for each of the 8X8 blocks normally contains run of zero elements followed by non-zero values. This property is utilized to further compress the AC elements using RLE. Run length encoding codes the coefficients in the quantized block into a run length (or number of occurrences) and a level or amplitude. Each nonzero AC coefficient is represented in combination with the run length (consecutive number) of zero-valued AC coefficients which precede it in the zigzag sequence. Huffman coding is used which is an entropy encoding algorithm used...
for lossless data compression. The standard Huffman tables are defined for the JPEG
implementation and can be used for the standard implementation of entropy encoding.
The information is represented as RUNLENGTH, SIZE, and AMPLITUDE, which is
simply the amplitude of the nonzero AC coefficient. RUNLENGTH is the number of
consecutive zero-valued AC coefficients in the zigzag sequence preceding the nonzero
AC coefficient being represented. SIZE is the number of bits used to encode
AMPLITUDE.

2. Implementation

Matlab is used to generate the testbench. It creates the 24 bit input stream from the .tif
image where each 24 bit represents a single pixel: 8 bits each for R, G & B. The input
image is subdivided into 8X8 blocks. In each clock cycle, one pixel input is given to the
module which converts the RGB values to the corresponding YCbCr values. The input
enable signal is kept high while the data is being input to the encoder.

The next step is the 2D DCT module for each of the Y, Cb & Cr 8X8 blocks. The 2D
DCT values are calculated using the formula :

\[ Y = C \cdot X \cdot C^T \]

where C is matrix with the DCT coefficient and X is the data matrix

The output from the DCT modules is 11 bits long. The outputs from the DCT modules
for Y, Cb & Cr components are given to the inputs of respective quantization module.
The quantized values of Y, Cb and Cr coefficients are calculated separately by dividing
them by the respective quantization matrices. The values in the Quantization matrix can
be changed in the quantizer module. The final quantized values are rounded based on the value in the 11th LSB.

The output from the quantization modules are zigzag scanned and DPCM is calculated for the DC components and RLE for the AC components and then it goes to the input of the Huffman encoder module. There are three Huffman modules, one for each of the Y, Cb & Cr components. The standard Huffman table is used by the Huffman modules. The output from the three Huffman modules goes to the output_synchronisation module which holds the outputs from each of the Y, Cb & Cr Huffman modules. The output JPEG stream combines the Y, Cb, and Cr Huffman codes together, and it starts with the Y Huffman codes, followed by the Cb Huffman codes, and finally the Cr Huffman codes for each 8x8 block of the image. Then the Huffman codes from the next 8x8 block of the image are put into the bit-stream.

The final output stream generated by the encoder is converted to the JPEG file with the help of MATLAB which creates the header file required for the JPEG image file.

### 3. Simulation & Synthesis Setup

Altera Quartus FPGA design software was used for simulation and synthesis of the proposed encoder design. The hardware descriptor language used for the design was Verilog HDL. The design was targeted for Stratix IV FPGA. Matlab was used to create the RGB input stream from the .tif image file for the testbenches and to generate the final JPEG file from the output stream.
## 4. Synthesis Results

### 4.1 Hardware Resources Utilized

The summary for hardware resources utilized in the FPGA is shown in the Table 1 and 2. The logic utilized is 20% of the total available logic in the FPGA. The amount of DSP blocks utilized is very large (95 %). The design requires 67 pins and less than1% of memory blocks present in the FPGA.

<table>
<thead>
<tr>
<th>Logic Utilization</th>
<th>20%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Registers</td>
<td>31061</td>
</tr>
<tr>
<td>Total Pins</td>
<td>67 (14%)</td>
</tr>
<tr>
<td>Total block memory bits</td>
<td>4016 (&lt;1%)</td>
</tr>
<tr>
<td>DSP block 18-bit elements</td>
<td>1224 (95%)</td>
</tr>
</tbody>
</table>

**Table 1: Hardware Resources Utilized on Stratix IV FPGA**

<table>
<thead>
<tr>
<th>Simple Multipliers (18-bit)</th>
<th>168</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple Multipliers (36-bit)</td>
<td>222</td>
</tr>
<tr>
<td>DSP block 18-bit elements</td>
<td>1224</td>
</tr>
<tr>
<td>Signed Multipliers</td>
<td>168</td>
</tr>
<tr>
<td>Unsigned Multipliers</td>
<td>222</td>
</tr>
</tbody>
</table>

**Table 2. DSP Block Usage Summary**
4.2 Performance

The performance of the JPEG encoder is calculated for the image size 1024X768 pixels. The number of cycles required to process each 8X8 block is 98 cycles. The number of blocks in 1024X768 image is 12288. The encoder is running at 122.5 MHz clock frequency on the FPGA. The total cycles required to process the image is 1204224 cycles. This means that encoder requires 9.83 milliseconds for the encoding the image of size 1024X768 pixels. The performance results are shown in the Table 3.

<table>
<thead>
<tr>
<th>Clock frequency</th>
<th>122.5 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycle for processing each 8X8 block</td>
<td>98 cycles</td>
</tr>
<tr>
<td>Cycles Required for 1024*768 image</td>
<td>1204224 cycles</td>
</tr>
<tr>
<td>Time(in msec) required</td>
<td>9.83 ms</td>
</tr>
</tbody>
</table>

Table 3: Performance Results

5. Simulation Results

The quality of a JPEG encoded image can be tuned using a parameter called quality or a Q factor. There are many different implementations having varying scales of Q Factors. The Independent JPEG Group(IJG) quality factor values ranges from 1 to 99. A Q factor value of 1 produces the worst image quality and a Q factor of 99 produces the best one but with the largest image size. The IJG implementation uses the standard quantization tables, computes the Scaling Factor(S) from the Quality Factor(Qf), and
then uses the Scaling Factor to scale the standard quantization tables. The formulas used by the IJG implementation are shown below [2].

**Scaling factor, S = \frac{5000}{Q}, 1<Qf<50**

\[
S = 200 - 2\times Q, \quad 50<Qf<100
\]

**Quantization Matrix, QMatrix[i] = \frac{(stdQMatrix[i] \times S + 50)}{100}**

The code was simulated after synthesis for a test image. The implementation was tested for different compression levels (Q factors) as shown in the Figure 3.

![Testing Image](image1.png)

(a) Qf = 100  
(b) Qf = 50  
(c) Qf = 25  
(d) Qf = 10

Figure 3. Testing Image for various Quality factors: 100, 50, 25, 10
As the value of the Q factor decreases, the size of the image also decreases. The variation size of the image size for various quality factor is shown in the Figure 4. It is clear from the graph that image size is significantly large for Q factor = 100 as compared to size of the image with Q factor = 10. This is because for low quality image the value of elements in the quantization matrix are large which makes most of the final AC quantized values to be zero and we lose the finer details in the image (as seen in Figure 1(d) for Qf =10).

Figure 4: Variation of Image Size with change in Q factor

**6. Conclusion & Future Work**

The JPEG encoding module was successfully designed and synthesized for Startix IV FPGA running at clock frequency of 122.5 MHz. The design was tested on an image for
different values of Q factor using IJG scale for Quality factor. It was observed that the image size reduces significantly when the value of Q factor is taken as small but the image quality is also degraded.

In future work, the design can be further optimized for amount of resources used and performance by exploring techniques like Fast DCT, better Huffman encoding design etc. The compression of the image can also be improved by adding a block in the design which finds out the optimal Huffman table for entropy encoding based on the nature of the image instead of using the standard JPEG Huffman table. The same thing can be done to find the optimal quantization matrix based on the image. The addition of these blocks can significantly improve the JPEG compression at the cost of latency.

References


[2] Chandra, Surendra; Ellis, Carla Schlatter, “JPEG Compression metric as a quality aware image transcoding”