FPGA Implementation of Image Compression Algorithm using Angular Domain

Pravin B. Pokle
Research scholar, member IEEE B.D.C.E.Sewagram, Dist.Wardha.(India)

N.G. Bawane, PhD
Principal and senior member IEEE
S.B.J.I.T.M.R., Nagpur (India)

ABSTRACT
Image compression is the science of reducing the size of image file in bytes by reducing the redundancy between pixels in an image without degrading the quality of image so that it can store more images in a given amount of disk or memory space and also it tends to reduces the time required to send the images over the network. Many hardware efficient techniques exist, inspired from it, this paper, propose an image compression technique based on the pixel-wise fidelity and its FPGA implementation. The proposed method is used to reduce the bit rate of the pixels for better image compression by using angular transformation. Here propose an hardware efficient FPGA architecture using angular domain concept based on CORDIC algorithm is presented. In this paper, the architecture is first simulated in MATLAB for calculating PSNR, MMSE and compression ratio and then it simulated and synthesized using Xilinx ISE tool and verify the parameters such as area, power and delay required for compressing the image with visual appearance of the output compressed image.

Keywords
Image compression, Angular transformation, VHDL, FPGA, CORDIC, Bit plane slicing, MMSE, PSNR, RTL.

1. INTRODUCTION
In day to day life, image compression is very essential for transmission and storage of data. Recently, many researchers have been proposed different techniques and algorithms for image compression such as JPEG, MPEG and H.263 so as to improve the performance measures in last decades in order to achieve the better performance and quality. The key factor in image compression is to remove similar pixels by discarding the redundant pixels in an image [1]. Almost all the methods proposed are either define in frequency domain or in time domain. The major drawback of using frequency domain is that the low frequencies pixels are discarded whereas the time domain reduces the size of an image this will degrade the visual appearance of the image [2]. However every algorithm has its advantages and disadvantages due to nature of each scheme. Amongst all the schemes proposed till now are based on discrete cosine transform. Vector Quantization is the recent technique developed for image compression. The performance of VQ is directly proportional to the vector size and codebook size [3, 4]. This means with increase in vector size, the codebook size also increases which results in exponential increase in encoding complexity. On the other hand, microprocessor and microcontroller based systems can process image data very easily with less complexity, but these are not able to target onto actual hardware. Thus the efficient technique is come into existence using HDL [5]. This paper proposes hardware efficient FPGA architecture scheme based on angular transformation for the better image compression. This scheme considers angular position of each pixel under a sine wave by converting every pixel into angle by using CORDIC algorithm. For achieving further compression of image data, bit plane slicing is used. The main objective of this paper is to develop an efficient hardware on FPGA chip for compression of an image using angular domain in order to optimize area, power and delay and to achieve high compression ratio without degrading the original image quality.

2. PROPOSED METHOD
This paper presented new concept for image compression using angular transformation based on CORDIC (coordinate rotation digital computer) algorithm. The compression is achieved in three steps i.e. angular transformation using CORDIC, divider block and Bit plane slicing. Before processing an input image for HDL simulation, first step is to convert all pixels into text I/O file format and stored it into memory block in which all the pixels are properly arranged so that further it will process pixel by pixel. Figure 1 below shows the complete top level image compression system.

![Fig.1: Top level entity for proposed scheme.](image)

In proposed scheme, the input image is first stored in memory block in column and row matrix form because VHDL language cannot read image directly. It needs to convert first the image data into equivalent text I/O file format. In which if column matrix is equal to row matrix is greater than the size of image then go to the next row, thus the pixel wise image data in readable form is obtained[6]. Now this data is process pixel by pixel for converting into corresponding angle using one element processor. Once one element processor has finished processing all pixels then output writer could write the image into file after output. This will results in compression of image at the output. In this paper the top level block is further divided into smaller subsystem so that it can be easily implemented on FPGA.

2.1 Memory Block
In memory block the input image for the compression is passed. Image is created with the multiple pixels. For the angular transformation it requires single pixel by pixel input. In memory block the image is decomposed in the number of rows and columns according to the pixels resolution of the image. For every column and row pixel first check whether its size is greater than size of image, if it’s true then go to next row for the column and repeat the process and if the size of pixel is less then image size then save the value of pixel in memory and pass this pixel to the one element processor for the angular transformation process. The block diagram of the memory block shown in figure 2.
3. SYSTEM BLOCK DIAGRAM

Top level entity is further divided into subsystem as shown in figure 3 below: One element processor consists of sine coder, divider block and bit plane slicing.

3.1 Angular transform

According to sine transform, the sine transform has the property that for two different angles, it has same gray level value. i.e. Pixels at an angle $89^\circ$ and $91^\circ$ have the same gray level value:

$$\text{i.e. } \sin x \cdot \frac{\pi}{2} \leq x \leq \frac{\pi}{2} \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \}
\[ Q_2 = \cos(\Phi) \times \left[ P_1 \times \tan(\Phi) + Q_1 \right] \]

\[ \mathbf{...} \]

In rotation mode, the initial angle is \( \theta \) which can be rotated in steps \( \Phi_i \) to become 0. Now at \( i^{th} \) step, choose tan\( \Phi_i \) as a fractional power of 2 such that we can replace multiplication by tan\( \Phi \) by right shift. Thus we have,

\[ \Phi_i = \tan^{-1}(2^{-i}) \]

\[ \mathbf{...} \]

Here multiplication by \( \cos(\Phi) \) tends to a constant value \( K \) irrespective to the initial values of \( P, Q \) and \( Z \) thus we can ignore it. Where \( K \) is defined as:

\[ K = \cos(\tan^{-1}(2^{-0})) \times \cos(\tan^{-1}(2^{-1})) \times \cos(\tan^{-1}(2^{-2})) \times \mathbf{...} \]

\[ \mathbf{...} \]

\[ K = \frac{2^N}{\sqrt{1 + 2^{2N}}} = 0.607 \]

\[ \mathbf{...} \]

Thus if we consider cosine is equal to \( p_0 \), sine equal to \( q_0 \) and angle equals to \( z_{in} \) then iterations per pixels are as follows:

\[ Q = q = 0, \; P = p = 0.607 \quad \text{and} \quad Zin = 0 \]

\[ p_0 = pi + q \quad \text{when} \quad Zin < 0 \quad \text{else} \quad pi - q \]

\[ q_0 = qi - q \quad \text{when} \quad Zin < 0 \quad \text{else} \quad qi + q \]

The architecture required for one step cordic iteration is shown in figure 6. Depending upon the number of iterations required for a given image above architecture is required in pipelined manner. In VHDL it can be done with the help of generate statement by simply passing generic parameter which is equal to the number of iterations as shown in following figure.

**Fig.6: one step cordic iteration**

This means that multiplying by \( K \) tends to initialized \( X \) with 0.607 Instead of 1. Thus all pixels are converted into angular values in the range 0 to 90 degrees [10].

\[ \mathbf{...} \]

By iteratively rotating \( \Phi \) towards 0, \( \sin^{-1}(\theta) \) can be calculated

**Step1.** Set \( \Phi = 45^0 \)

**Step2.** If \( 0^0 = \Phi \) then
\[ \Phi = \Phi + (45/2)^0 \]

**Step3.** If \( 0^0 = \Phi \) then
\[ \Phi = \Phi - (45/4)^0 \]

Continue by halving step size

For example, for 5 x 5 images as shown below:

\[ \begin{array}{cccccc}
180 & 200 & 197 & 250 & 198 \\
165 & 243 & 176 & 199 & 220 \\
120 & 189 & 221 & 245 & 195 \\
180 & 176 & 229 & 212 & 147 \\
167 & 134 & 172 & 178 & 189 \\
\end{array} \]

\[ P_{ij} = \]

After sufficient number of cordic iterations, all the gray level values are get converted into corresponding angular values.

Thus the resulting matrix is given as:

\[ \mathbf{...} \]

\[ Q_{ij}' = \]

Here \( P_{ij} \) is the gray level values of pixels in an image and \( Q_{ij}' \) is the respective angular values. Thus for representing the original image the value of pixels are 0 to 255 which requires 8 bit, but after applying the above algorithm, we need 7 bits for representation as it have the angular values from 0 to 90[11].
3.3 Reduction of bit rate of qi,j’ image

For further compression of an image the output is again processed by divider block so that it will convert the image into 6 bit form for further compression, now for further compression of an image, divide the image by 1.5 because the range of 0 to 90 after division will become 0 to 60 this will reduce the bit rate of the image[12].

Thus we have: \[ Q_{ij}'' = \frac{Q_{ij}'}{1.5} \]......(10)

<table>
<thead>
<tr>
<th>Q''i,j</th>
<th>31’36’34’60’34’</th>
<th>34’50’29’34’41’</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30’21’27’34’22’</td>
<td>30’29’43’38’24’</td>
</tr>
<tr>
<td></td>
<td>28’21’29’30’32’</td>
<td>28’21’29’30’32’</td>
</tr>
</tbody>
</table>

This will results the maximum value of angle is 60, for which \( 2^6 = 64 \) and hence we can represent it in 6 bits.

3.4 Bit plane slicing

In this approach, bit plane slicing is used for further compression of an image. Using one bit plane or two bit plane slicing the output is further compressed. A bit plane is a set of bits corresponding to a given bit position in each of the binary numbers in an image. It is used to determine the adequacy of numbers of bits used to quantize each pixel in the image [13]. From figure 8, it is seen that instead of highlighting gray level images, it is desired to highlight the contribution made to total image appearance by specific bits. Suppose that each pixel in an image is represented by 8 bits[14]. Now assume that the image is having eight, 1-bit planes ranging from bit plane1-0 (LSB) to bit plane 7 (MSB). In Qi,j it have 6 bit image in compressed form, now apply the bit plane slicing technique on Qi,j image.

In \( Q_i''(1,1) \) the gray level pixel with value 46(00101110) will be present in the 6th, 4th, 3rd and 2nd bit plane. After applying 2 bit plane slicing and 1 bit plane slicing on the binary number to get the image in 4 bit and 5 bit form as shown below.

46 → 101110 → 1011 (4 bit form)
46 → 101110 → 10111 (5 bit form)

Fig.8: Bit Plane slicing for 8 bit image

Suppose the original 8 bit input image is of size 100kb. So after applying the 0 bit plane slicing on Qi,j’ image the output image will be 2/8 of the original image i.e. this will able to compressed the size of image to 25%. For 1 bit plane slicing the output image will be 3/8 of the original image i.e. this will able to compress the image to 37.5%. For 2 bit plane slicing the output image will be 4/8 of the original image i.e. this will able to compress the image to 50%. Thus the proposed method is able to compressed an image upto 50%[15].

4. MODELING RESULTS

In this work, MATLAB codes are develop for simulation and verifying PSNR, MSE and compression ratio and compare the results with other existing methods such as DCT, Huffmans, RLE and Wavelet with different filters. Another set of experiment involves FPGA implementation for performing simulation and synthesis using Xilin ISE for the verification of area, power and delay of the proposed method. The set of experiments evaluate the effect of different methods on the quality of the reconstructed image. Experiments were conducted using the standard data base such as images 'lena', 'Football' and Cutebaby. The performance measures for analysis used is mainly Mean square Error (MSE), Peak Signal to Noise Ratio (PSNR), Compression Ratio and Image Quality[16]. Where MSE is the cumulative squared error between the compressed and the original image. Whereas PSNR is a measure of the peak error. For evaluating mean square error and peak signal to noise ratio, following formulae were used.

\[ \text{MSE} = \frac{1}{XY} \sum_{i=1}^{X} \sum_{j=1}^{Y} (Q_{ij} - Q_{ij}'')^2 \]......(11)

\[ \text{PSNR} = 20 \times \log_{10} \left( \frac{255}{\text{MSE}} \right) \]......(12)

Where \( Q(i,j) \) is the original image and \( Q(i,j)'' \) is the compressed version of the image and \( X,Y \) are the dimensions of given image, 255 is the peak signal value.
Figure 9: (a) Original lena image (b) Reconstructed image using DCT (c) Reconstructed image using RLE (d) Reconstructed image using Huffmans (e) Reconstructed image using wavelet dB2 filter (f) Reconstructed image using proposed method.

Fig. 9: (a) Original lena image (b) Reconstructed image using DCT (c) Reconstructed image using RLE (d) Reconstructed image using Huffmans (e) Reconstructed image using wavelet dB2 filter (f) Reconstructed image using proposed method.

Figure 10: (a) Original Football image (b-f) Reconstructed image using DCT, RLE, Huffmans, using wavelet dB2 filter and using proposed method.

Fig. 10: (a) Original Football image (b-f) Reconstructed image using DCT, RLE, Huffmans, using wavelet dB2 filter and using proposed method.

Figure 11: (a) Original Cutebaby image (b) Reconstructed image using DCT (c) Reconstructed image using RLE (d) Reconstructed image using Huffmans (e) Reconstructed image using wavelet dB6 filter (f) Reconstructed image using proposed method.

Fig. 11: (a) Original Cutebaby image (b) Reconstructed image using DCT (c) Reconstructed image using RLE (d) Reconstructed image using Huffmans (e) Reconstructed image using wavelet dB6 filter (f) Reconstructed image using proposed method.

Figure 12 shows the compressed images after VHDL simulation for the proposed architecture applied on different standard test images. After simulation the proposed method is synthesized using Xilinx ISE suite13.2 and tested it on Spartan-3 FPGA. Figure 13 shows the RTL (register transfer level) level diagram for proposed scheme and Table 2 shows the micro statistics of the proposed method. Figure 14 shows

Fig.12: Compressed images after VHDL simulation. Figure (a) represents compressed Football image (b) compressed cutebaby image (c) compressed Lena image.
Table I: Comparison of results for different images using MATLAB

<table>
<thead>
<tr>
<th>Image</th>
<th>Parameters</th>
<th>MMSE (dB)</th>
<th>PSNR (dB)</th>
<th>Compression Ratio (%)</th>
<th>Time Elapsed (s)</th>
<th>Proposed Method (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lena</td>
<td>DCT</td>
<td>3.7</td>
<td>42.39</td>
<td>74.90</td>
<td>0.1</td>
<td>1.29</td>
</tr>
<tr>
<td>Image</td>
<td>CRT</td>
<td>5</td>
<td>23.39</td>
<td>47.00</td>
<td>1.0</td>
<td>45.89</td>
</tr>
<tr>
<td></td>
<td>Huff</td>
<td>25.26</td>
<td>23.70</td>
<td>7.7</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wavelet</td>
<td>2.62</td>
<td>43.93</td>
<td>12.16</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Football</td>
<td>MMSE (dB)</td>
<td>1.0</td>
<td>47.00</td>
<td>64.12</td>
<td>0.5</td>
<td>0.84</td>
</tr>
<tr>
<td>Image</td>
<td>PSNR (dB)</td>
<td>3.2</td>
<td>27.92</td>
<td>48.37</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Compress</td>
<td>39.00</td>
<td>48.37</td>
<td>16.98</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Correction</td>
<td>90.00</td>
<td>48.37</td>
<td>48.72</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ratio (%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Time Elapsed (s)</td>
<td>1.0</td>
<td>0.32</td>
<td>1.89</td>
<td>0.085</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cutebaby</td>
<td>MMSE (dB)</td>
<td>1.3</td>
<td>40.00</td>
<td>64.12</td>
<td>0.5</td>
<td>0.245</td>
</tr>
<tr>
<td>Image</td>
<td>PSNR (dB)</td>
<td>5</td>
<td>38.32</td>
<td>48.37</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Compress</td>
<td>15.00</td>
<td>48.37</td>
<td>16.98</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Correction</td>
<td>5</td>
<td>48.37</td>
<td>48.72</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ratio (%)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Time Elapsed (s)</td>
<td>1.0</td>
<td>0.32</td>
<td>1.89</td>
<td>0.085</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Micro statistics of proposed method using Spartan 3 FPGA

<table>
<thead>
<tr>
<th>Method Parameters</th>
<th>Proposed</th>
<th>DCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>No.of ROM</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Multiplier</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Add/Sub</td>
<td>43</td>
<td>36</td>
</tr>
<tr>
<td>No.of Slices</td>
<td>66/768</td>
<td>75/768</td>
</tr>
<tr>
<td>Registers</td>
<td>16</td>
<td>24</td>
</tr>
<tr>
<td>Power(mW)</td>
<td>0.18</td>
<td>0.27</td>
</tr>
<tr>
<td>Delay(nSec)</td>
<td>6.216</td>
<td>13.714</td>
</tr>
</tbody>
</table>
5. CONCLUSION
This paper presented a new method based on angular domain using CORDIC algorithm. In this work, MATLAB codes as well as VHDL codes were developed. In first set of experiment the algorithm is simulated for MMSE, PSNR and compression ratio using MATLAB. The results are compared with standard existing methods such as DCT, RLE, Huffman and Wavelet with db62 and db6 filters. From the results it is observed that the proposed method works very efficiently and effectively and the results obtained are extremely good. The proposed method able to compress an image 50.00 % which is a very high compression rate. Also the visual quality of output image is intact and hence is exactly as same as input image. Moreover, the Peak Signal to Noise ratio (PSNR) value obtained is very high and is in between 50 dB to 60 dB and minimum mean square error (MMSE) value is very low and comes out to be 0.245 for image cutebaby. The other set of experiment shows the VHDL simulation and synthesis for analyzing area, power and delay for the proposed method. Table 2 summarizes the Micro statistics of proposed method using Spartan 3 FPGA the results of the proposed method were compared with standard DCT. It is seen that the propose method consumes very small power of 0.18 mWatts with only one ROM. Also the proposed method consumes only 9% of total area with small delay of about 6 nSec. Moreover, from the table 1 and table 2, it is clear that performance measures obtained are very good enough to show the high success rate of the research being done by means of this paper.

6. REFERENCES


