Design of Pipelined architecture for jpeg image compression with 2D-DCT and Huffman Encoding

Prashant Chaturvedi, Tarun Verma, Rita Jain
L.N.C.T Bhopal

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. In this paper we describe the design and implementation of a fully pipelined architecture for implementing the JPEG image compression standard. The architecture exploits the principles of pipelining and parallelism in order to obtain high speed and throughput. The design was synthesized using Xilinx9.2i and Spartan 3 FPGAs, and simulation was carried out using ModelSim environment. It has been estimated that the entire architecture can be implemented on a single FPGA to yield a clock rate of about 100 MHz which allow an input rate of 24 bit input RBG.

Index Terms— RLE , Compression, FPGA, JPEG, Huffman Encoding.

I. INTRODUCTION

The Joint Photographic Experts Group (JPEG) was formed in 1986 to define standards for image compression algorithms. The well-known JPG file extension has become the international standard for Internet image compression. By compression of the image, the overall file size decreases. To the casual viewer however, or even through a more detailed inspection of the image, there may not appear to be any loss of information from the original picture. JPEG compression has become one of the most popular techniques for image compression and is being used in a wide variety of applications. It is involved in digital cameras, the digital altering of images, loading pictures on the web.

Other applications. Nowadays, the focus has shifted to using reconfigurable hardware to implement the JPEG algorithm to increase its efficiency and hence reduce the cost of this technique. Lot of research is going on in this area and in this project the aim is to achieve JPEG compression using reconfigurable hardware.

The JPEG baseline can be divided into five main steps, as shown in Fig. 1: color space conversion, downsampling, 2-D DCT, quantization and entropy coding. This paper will present the architectures for these five modules. The first two operations are integrated in a single architecture.

The color space conversion transforms the RGB coding to the YCbCr color coding. The downsampling operation reduces the sampling rate of the color information (Cb and Cr). The 2-D DCT transform the pixel data from the spatial domain to the frequency domain. The quantization operation eliminates the high frequency components and the small amplitude coefficients of the cosine expansion. Finally, the entropy coding uses run-length encoding (RLE), Huffman, variable length coding (VLC) and differential coding to decrease the number of bits used to represent the image [1].
The JPEG compression is a lossy compression, since downsampling and quantization operations are irreversible [3]. But the losses can be controlled in order to keep the necessary image quality.

II. Color Space Converter & Downsampler

The first two steps of the JPEG compression are color space conversion and downsampling. The first one uses the input color components R, G and B to calculate each one of the Y, Cb and Cr components. Color spaces with luminance and chrominance components (like Y-Cb-Cr space) are more appropriate to be used with DCT [3].

The downsampling operation consists in reducing the number of samples of the chrominance components. These are less important to the human eye than the luminance components. The architecture uses a 4:1:1 sampling rate ratio for the Y, Cb and Cr data. Such downsampling results in a 50% reduction in the image data.

This paper integrates the architectures of the color space converter and the downsampler to optimize these operations. Such integration allows that are just calculated the values of Cb and Cr that will be used. The downsampling operation is only a control operation. The operation of the space color conversion is presented in (1).

\[
\begin{align*}
Y &= (0.299*R) + (0.587*G) + (0.114*B) \\
Cb &= (-0.1687*R) - (0.3313*G) + (0.5*B) + 128 \\
Cr &= (0.5*R) - (0.4187*G) - (0.0813*B) + 128
\end{align*}
\]

Multiplication, addition and subtraction operations are used in color space conversion. Our architecture uses shift-and-add in 04 parallel barrel shifters and 4 ripple-carry adders. The datapath shown in Fig. 2 operates in a three stage pipeline where the two first stages are used for multiplications and the last stage is used to add or subtract the multiplication results. The integrated architecture generates Y values at every 4 clock cycles with the pipeline full and has 7 clock cycles latency for a complete Y operation. The values for Cb and Cr are generated every 4 clock cycles.

III. DCT in Two Dimensions

The DCT in two dimensions (2-D DCT) is the core of the JPEG compression. This is the most critical module to be designed in hardware JPEG compressor because of its high algorithm complexity. There are many algorithms to solve the 2-D DCT with a small number of operations. The algorithm chosen in this implementation was proposed in [5] and modified by [6]. This algorithm calculates the DCT in one dimension (1-D DCT) and uses 29 additions and 5 multiplications. The 2-D DCT has the separability property.

Thus, using two 1-D DCT calculations it is possible to generate the 2-D DCT results. In an 8x8 input matrix, the first 1-D DCT is applied on the matrix lines then the second 1-D DCT is applied on the columns of the first 1-D DCT results matrix. This separation reduces the complexity of the calculation. The algorithm proposed by [5,6] is scaled and it makes possible an efficient pipeline exploration. This algorithm is presented in Table 2 (where \( m_1 = \cos\left(\frac{4\pi}{16}\right) \), \( m_2 = \cos\left(\frac{6\pi}{16}\right) \), \( m_3 = \cos\left(\frac{2\pi}{16}\right) - \cos\left(\frac{6\pi}{16}\right) \) and \( m_4 = \cos\left(\frac{2\pi}{16}\right) + \cos\left(\frac{6\pi}{16}\right) \)).

2D Discrete cosine transform core combined with level shift. Works on block of 64 samples. Take 8 bit input and produces 12 bit output. First, uncoded image is level shifted from unsigned integers with range \([0, 2^P - 1]\) to signed integers with range \([-2^{P-1}, 2^{P-1} - 1]\). \( x^p \) means here \( x \) to power of \( p \).

Then 2D DCT is performed using following equation:

\[
X(u,v) = \frac{2}{N} C(u) C(v) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} x(i,j) \cos \left( \frac{(2i+1)u\pi}{2N} \right) \cos \left( \frac{(2j+1)v\pi}{2N} \right)
\]

where \( C(u),C(v) = 2^{\frac{u}{2}} \) for \( u=0, v=0 \)

\[ = 1 \quad \text{otherwise} \]

\( x(i,j) \) – input sample at position \((i,j)\) in 8x8 block

\( X(u,v) \) – output sample at position \((u,v)\) in 8x8 block

\( N=64 \).

MDCT takes data row-wise but outputs column-wise. To get row-wise order it is necessary to transpose output DCT matrix.

III. ZIG ZAG

Zig-Zag block is responsible to perform so called zig-zag scan. It is simply reorder of samples positions in one 8x8 block according to following tables.
Table 1 shows the ZigZag

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></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>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>14</td>
<td>15</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>7</td>
<td>13</td>
<td>16</td>
<td>26</td>
<td>29</td>
<td>42</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>12</td>
<td>17</td>
<td>25</td>
<td>30</td>
<td>41</td>
<td>43</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>18</td>
<td>24</td>
<td>31</td>
<td>40</td>
<td>44</td>
<td>53</td>
</tr>
<tr>
<td>10</td>
<td>19</td>
<td>23</td>
<td>32</td>
<td>39</td>
<td>45</td>
<td>52</td>
<td>54</td>
</tr>
<tr>
<td>20</td>
<td>22</td>
<td>33</td>
<td>38</td>
<td>46</td>
<td>51</td>
<td>55</td>
<td>60</td>
</tr>
<tr>
<td>21</td>
<td>34</td>
<td>37</td>
<td>47</td>
<td>50</td>
<td>56</td>
<td>59</td>
<td>61</td>
</tr>
<tr>
<td>35</td>
<td>36</td>
<td>48</td>
<td>49</td>
<td>57</td>
<td>58</td>
<td>62</td>
<td>63</td>
</tr>
</tbody>
</table>

IV. Quantization

The quantization operation is an integer division of the 2-D DCT coefficients by pre-defined values. These pre-defined values are stored in tables called quantization tables. In JPEG baseline mode there are two quantization tables: one for luminance components (Y) and another for chrominance components (Cb and Cr). The optimum values of the components in quantization tables are dependent on the application, but the JPEG standard suggests typical tables that have a good efficiency for any application. This operation eliminates the 2-D DCT coefficients that are less perceptible to the human eye. The result of this operation in an 8x8 matrix of 2-D coefficients is a sparse matrix.

The quantization architecture designed in this paper is presented in Fig. 4 and uses two ROMs and one multiplier to calculate the quantized coefficients. The values in the standard quantization tables used for divisions were transformed into multiplier values. The multiplier in the quantization has similar architecture to that used in the color space conversion and in the 2-D DCT modules. The barrel shifter control words for each value in the quantization table are stored in ROM.

V. RLE

RLE core performs run-length encoding of data. For input block 8x8 samples, multiple symbol outputs are created consisting of RUNLENGTH, SIZE and AMPLITUDE.

RUNLENGTH is the number of consecutive zero-valued AC coefficients in the zig-zag sequence preceding the nonzero AC coefficient being represented. For DC coefficient RUNLENGTH is always zero. SIZE is the number of bits used to encode AMPLITUDE. SIZE value is between 1 and 10 for AC coefficient. SIZE value is between 1 and 11 for DC coefficient. AMPLITUDE is two complement’s signed integer. Value ranges from -2047 to 2047 for DC coefficient and from -1023 to 1023 for AC coefficient. When start_pb asserts: buf_sel used by ZIGZAG toggles rd_cnt is reset to 0 and starts counting 0..63. -> 64 values are read from QUANTIZER. wr_cnt is reset to 0. RLE encoding is done on 8x8 block RLE encoded data is written to DoubleFIFO.
RLE Core performs encoding of 64 input samples to a variable number of output symbols. Input data to RLE consists of 64 words. First word is DC coefficient, words 1..63 are AC coefficients.

VI. HUFFMAN Encoder

Encodes data sample into a pair of symbols: AMPLITUDE (encoded as 2's complement signed integer) and SIZE according to the following table:

<table>
<thead>
<tr>
<th>SIZE</th>
<th>AMPLITUDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1,1</td>
</tr>
<tr>
<td>2</td>
<td>-3,-2,2,3</td>
</tr>
<tr>
<td>3</td>
<td>-7..-4,4..7</td>
</tr>
<tr>
<td>4</td>
<td>-15..-8,8..15</td>
</tr>
<tr>
<td>5</td>
<td>-31..-16,16..31</td>
</tr>
<tr>
<td>6</td>
<td>-63..-32,32..63</td>
</tr>
<tr>
<td>7</td>
<td>-127..-64,64..127</td>
</tr>
<tr>
<td>8</td>
<td>-255..-128,128..255</td>
</tr>
<tr>
<td>9</td>
<td>-511..-256,256..511</td>
</tr>
<tr>
<td>10</td>
<td>-1023..-512,512..1023</td>
</tr>
<tr>
<td>11</td>
<td>-2,047..-1,024,1,024..2,047</td>
</tr>
</tbody>
</table>

Table 2 shows the Huffman calculation.

Huffman block performs Huffman encoding operation. It converts parallel data into serial bit stream. Serial bit stream output is packed into bytes which are stored in output FIFO. Huffman block operates 8x8 block-wise. Huffman encoded data are stored to output FIFO. Two FIFOs with size is 2x64x8 bits each. 2x 64x8 for assumption that JPEG encoded stream is no more size than 2x of input unencoded stream.

Double FIFO block is simply double buffer using two FIFOs. The idea is that while Huffman writes data for next block Byte Stuffer can read encoded data for previous block.

FIFO is used instead of RAM because Huffman encoder can write variable number of encoded words to FIFO thus FIFO implementation is easier than using RAM.
VI. RESULT

This paper presented the architecture of the five main modules of the JPEG compression: color space conversion, downsampling, 2-D DCT, quantization and entropy coding. The final results of the synthesis of the modules were also presented.

Measured from JPEG encoding start till encoding done: Input image 640x480 or 24 bit RGB color. New sample loaded every cycle until FIFO full. Quantization tables at 50% quality setting 7.3 ms processing time @ 100 MHz clock [2.3 clock cycles per input sample]1000/7.3=136 frames per second @ 100 MHzInput file size = 921 kB. Output file size = 44 Compression Ratio: 21.31:1 Bits per pixel: 1.13:1

VII. Conclusions

VIII. References


[7] “Home site of the JPEG and JBIG committees” <http://www.jpeg.org/> (21/04/01)

