HARDWARE SUPPORT OF JPEG

Mohammed Elbadri, Raymond Peterkin, Voicu Groza, Dan Ionescu, and Abdulmotaleb El Saddik
School of Information Technology and Engineering, University of Ottawa
800 King Edward Avenue, Ottawa, ON, K1N 6N5, Canada
{melbadri, peterkin, groza, ionescu, elsaddik}@site.uottawa.ca

Abstract

Image formats specified by the Joint Photographic Expert Group (JPEG) are preferred for images with high colour content. Along with Graphic Interchange Format (GIF) images, JPEG is the most commonly used image format on the Internet and JPEG is the de facto still image format for digital cameras. All major digital camera manufacturers use JPEG as its exclusive or primary still image format. Most JPEG processing occurs in software programs like Microsoft Paint and Adobe Photoshop. Relatively few hardware solutions exist for processing JPEG images. This paper presents a comprehensive literature survey of hardware solutions for JPEG images. Wherever possible, different JPEG formats and accompanying hardware will be presented and compared to illustrate various advantages/disadvantages. The performance of hardware and software for JPEG is compared and an overview of commercial hardware for JPEG is also provided.

Keywords: Quantization; Discrete Cosine Transforms (DCT); entropy coding.

1. Introduction

The JPEG image format has seen an explosion in popularity since the proliferation of the Internet and the World Wide Web (WWW). Web pages typically display images in GIF or JPEG formats. JPEG images, however, allow for better representation of high color images. Digital photographers typically use JPEG images as their primary still image format because it maximizes the total number of images you can store, allows for fast writes to memory and compatibility with e-mail and the Internet in general. JPEG is commonly used in its lossy mode, which means that data is lost every time an image is opened, edited and saved. Conventional JPEG compression and decompression algorithms are widely implemented in software. Those implementations are integrated into image manipulation programs (Adobe Photoshop, Microsoft Paint, etc.) and utilized whenever the user employs the JPEG format.

When JPEG images must be compressed and/or decompressed in real-time, conventional software implementations are incompatible, too large or vastly inefficient when used in hardware devices, e.g., digital cameras or platforms such as Field Programmable Gate Arrays (FPGAs).

Hardware specific implementations of JPEG must be used. Those implementations use various performance tradeoffs and techniques to optimize the performance for a specific format.

This report presents a literature survey of hardware development for JPEG images. The next section describes the JPEG standard in great detail including three separate image modes and a description of various components for each type. Hardware solutions for each mode are presented. Architecture descriptions are given whenever possible along with experimental results. The following section provides a performance comparison between hardware and software implementations of JPEG. A brief description of commercial hardware for JPEG is given with an overview of digital camera technology. The final section of this paper summarizes the main points of the paper and mentions possible future work.

2. JPEG Architecture

The JPEG standard has various different images modes. Three of them will be described in this section along with relevant hardware developments that have taken place. Baseline JPEG is the most common JPEG modes and is widely used in digital technology and Internet based technologies. JPEG-LS is a lossless type of JPEG image type and JPEG-2000 which is based on wavelet technology. The following section describes JPEG Baseline including compression, decompression and solutions for those tasks.

2.1. JPEG Baseline

JPEG baseline can be divided into the five segments shown in Figure 1. The color space conversion modules transforms Red Green Blue (RGB) encoded data into YCbCr coding. Y represents luminance (based on inverse gamma-distorted data) and CbCr represents chrominance. Downsampling reduces the sampling rate of the converted data and the 2-D DCT transforms the data into the frequency domain. Quantization eliminates high frequency components and small amplitude elements and the entropy coder decreases the number of bits to represent the image.

2.1.1 Hardware Compression Solution. The hardware solution in [1] presents a hardware architecture for full compression of JPEG baseline images. The architecture is composed of four modules. The first module combines color space conversion and downsampling while separate modules

exist for the 2-D DCT, quantization and entropy encoding. The subsections below describe the details of those modules.

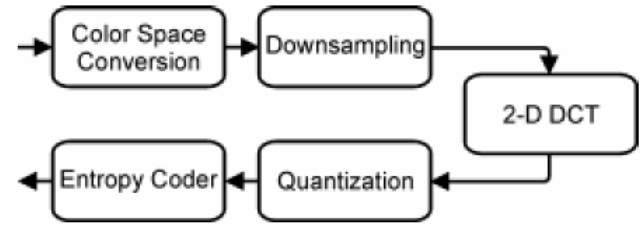

Figure 1 Steps in JPEG baseline compression

2.1.1.1 Color Space Converter and Downsampler. The color space converter and the downsampler are integrated to optimize these operations. The downsampling operation is only a control operation. The following equations describe the operations performed in color space conversion for this module.

$$Y = 0.299R + 0.587G + 0.114B \quad [1]$$

$$Cb = 0.564B - 0.564Y \quad [2]$$

$$Cr = 0.713R - 0.713Y \quad [3]$$

Figure 2 describes the implementation of the multiplication, addition and subtraction operations as well as the downsampling. The modules labelled BSx where x is 1, 2, 3 or 4 are parallel barrel shifters and the modules labelled A, B, C and D are ripple carry adders.

Operations occur in a three stage pipeline where the first two stages are used for multiplication and the last stage is used for addition. Values for Y, Cb and Cr are generated separately every four clock cycles. Implementation was done in VHDL with 441 logic cells and 869 lines of code.

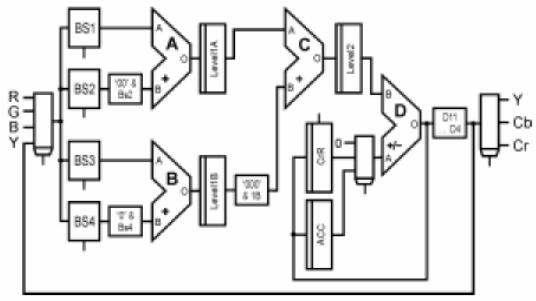

Figure 2 Architecture for color space conversion and downsampling

2.1.1.2 Two Dimensional (2-D) DCT Module. The DCT is the core of JPEG compression as it is the most critical module due to its complexity. A pipelined fast 2-D DCT architecture is fully described in [2]. A high level description of the architecture for the 2-D DCT module is described in Figure 3. This architecture was designed to reach a high operating frequency and to allow the use of pipeline techniques. Thus, the architecture was divided into two 1-D DCT architectures and one transpose buffer. The two 1-D DCT architectures are similar but the bit widths at each pipeline stage are different. The 1-D DCT architectures are organized in a six stage pipeline, one stage for each algorithm step. The transpose buffer operates like a temporal barrier between the first and the second 1-D DCT, allowing the use of a 2-D DCT global pipeline.

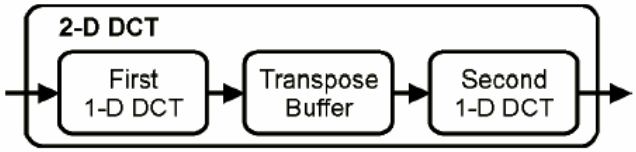

Figure 3 Two Dimensional (2-D) DCT architecture

The 2-D DCT inputs in this design are matrices of 8x8 elements eight-bit wide each. The first 1-D DCT receives and processes this matrix in a row-wise order. The transpose buffer receives the row-wise results and gives the column-wise inputs to the second 1-D DCT architecture. The second architecture processes the column-wise data and gives a column-wise data output.

Figure 4 illustrates the 1-D DCT architecture. The 1-D DCT algorithm is performed in a six stage pipeline to directly correspond with the steps of the theoretical algorithm. Five of the six stages perform addition/subtraction operations and the other stage (the fourth one) performs multiplication. Figure 5 illustrates the multiplier used in the 1-D DCT. It is composed entirely of shifts and adds to perform a multiplication operation. The modules labelled BSx where x is a number are barrel shifters. Barrel shifter outputs are fed into a series of adders and the outputs of the multiplication operation is ultimately produced. The multiplier design requires six cycles to perform a multiplication, saving 8 cycles through the strategic use and application of shifters and adders.

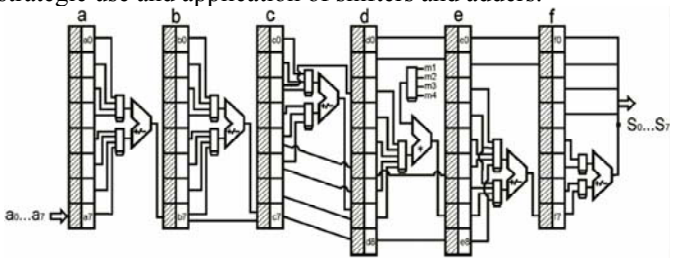

Figure 4 One Dimensional (1-D) DCT architecture

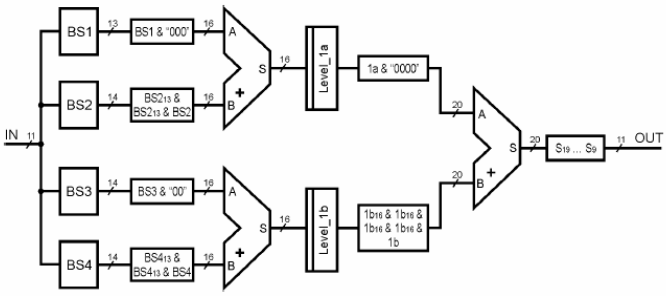

Figure 5 Multiplier used in 1-D DCT

Each stage uses a different set of register stored values to perform its operations during eight clock cycles. Single and separate arithmetic units are used at every stage to perform necessary operations. 1-D DCT outputs are generated in a serial manner. Figure 6 illustrates the architecture for the

transpose buffer. It is used to transpose (invert the rows and columns) of a 1-D DCT for another 1-D DCT. Two RAM modules are used that are 64 words long and 12 bits wide. FPGAs typically have internal RAM macroblocks for use at the designer's discretion so RAM was used in this design. It also saves logic cells and improves overall performance. When the first 1-D DCT stores values in a row wise manner (in either RAM module), the second 1-D DCT reads the other module in a column wise manner, thus performing a transpose operation in the process. Read, write, and control signals are used to manage the flow of information in and out of the transpose buffer.

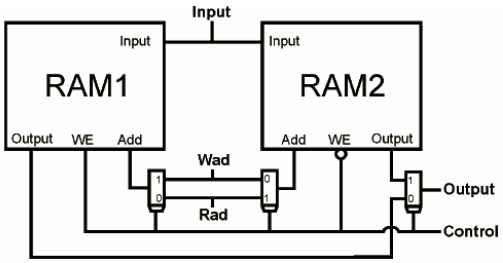

Figure 6 Transpose buffer architecture

2.1.1.3 Quantization Module. Figure 7 illustrates a high level description of the encoder for compressing JPEG images in hardware. Matrix elements at the far left are read in a zig-zag pattern. Separate modules exist for Huffman coding of DC and AC values, the appropriate Huffman tables a Variable Length Coding (VLC) coder (for non-zero values), a differential coder and a Run Length Encoder (RLE). Table values are taken directly from the JPEG standard and accessed whenever necessary. The differential coder performs a subtraction and is associated with two matrix values related to DC coefficients.

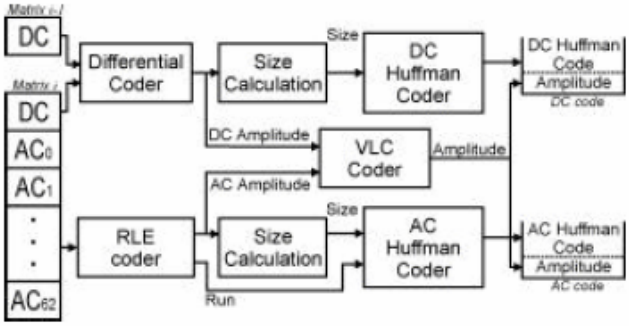

Figure 7 Entropy coder architecture

The architecture of the RLE is shown in Figure 8. It is an asynchronous circuit and has two restrictions based on the JPEG standard. The maximum values are limited by the number of bits used to represent the total number of elements. The second restriction is based on the theoretical possibility of getting pairs like 0/0 which cannot exist. Those restrictions aside, the RLE is implemented as a zeros counter that either indicates the presence of a non-zero value or generates the appropriate number of zeros.

Figure 9 demonstrates the architecture of the Huffman coder. Four separate tables exist to hold DC and AC values for luminance and chrominance. In performing Huffman coding logic, values are retrieved from a series of comparators and multiplexers that are used to determine the appropriate Huffman code and Huffman size.

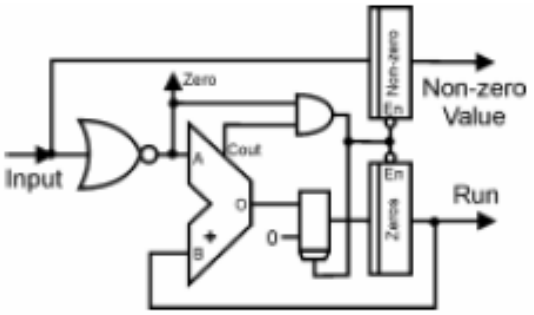

Figure 8 Run length encoder architecture

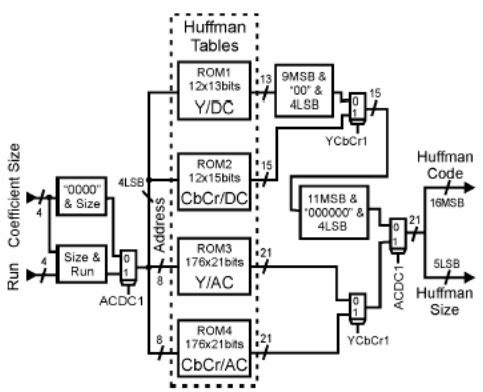

Figure 9 Huffman coder for color images

2.1.2 Hardware Decompression Solution. The decompression process is the inverse of the compression process. The decompression architecture is similar to the compression architecture, which is shown in Figure 1. All of the four modules are inverted in functionality to be utilized by the decompression architecture.

The hardware decompression solution presented in [3] shows a hardware architecture for full decompression of JPEG grey scale images. In comparison to the compression architecture, described in the previous subsections, this solution will not use a colour space converter and downsampler because this solution only covers grey scale images. The space converter and downsampler are enclosed in the architecture because the uncompressed images are colour scale images.

Figure 10 illustrates the block diagram of the JPEG decompression architecture. It comprises of three modules: a 2D IDCT, inverse quantization and decoding.

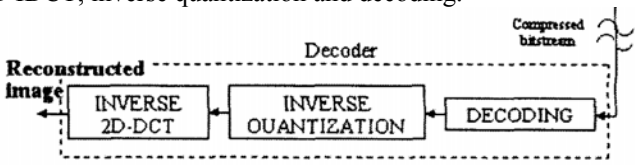

Figure 10 Block diagram of JPEG decoder

3. Hardware & Software Performance Comparison

Illustrating the differences between hardware and software implementations of JPEG requires a comprehensive comparison between various implementations. In this section, we summarize the experimental results of [5]. Hardware and software implementations of lossless and near-lossless JPEG are compared to show the difference in performance.

The difference in performance is best characterized by total execution time and the total number of cycles. Software implementations were done in C++ and run on a 400 MHz platform while hardware implementations were done in VHDL and executed on a 67 MHz platform. Five different image types were also used to illustrate their affect on overall performance. Those image types were satellite (overhead image of a city), pictorial, document, biometric (fingerprint) and medical (knee).

Table 1 illustrates the total decrease in cycles when comparing the hardware implementation to the software implementation for lossless compression. We see that the total number of cycles saved is consistently at or above 84% regardless of the image or whether it is being encoded or decoded. A slightly higher performance increase is observed when encoding images, which is consistent with the fact that encoding is a more complex process. Since the process is more complex there is more room for improvement and consequently we see a greater increase in performance. There was virtually no difference among the various image types although decoding the biometric image yielded the greatest improvement overall.

Table 1 Hardware performance increase for lossless compression

Image	Encoding Cycle Decrease	Decoding Cycle Decrease
Pictorial	88 %	85 %
Document	86 %	84 %
Medical	88 %	85 %
Satellite	87 %	84 %
Biometric	87 %	91 %

Table 2 shows the total decrease in cycles when comparing implementations for near lossless compression. We see virtually the same performance increases and trends for near lossless compression as was seen for lossless compression. The increases in encoding were extremely similar while the increases with decoding were slightly less similar, with the biometric image having the greatest improvement.

Table 2 Hardware performance increase for near lossless compression

Image	Encoding Cycle Decrease	Decoding Cycle Decrease
Pictorial	88 %	84 %
Document	87 %	84 %
Medical	89 %	85 %
Satellite	88 %	83 %
Biometric	88 %	87 %

Table 3 illustrates the total number of cycles saved between hardware and software. The performance increase is similar between encoding and decoding while there is a slightly greater increase with encoding images.

Table 3 Overall performance increase for hardware and software

Image	Encoding Cycle Decrease	Decoding Cycle Decrease
Lossless	87 %	85 %
Near lossless	88 %	84 %

It should be noted that these numbers represent the total number of execution cycles saved and not the total increase in time. The hardware implementation ran on a platform that was approximately six times slower. In spite of the disparity in processing speed, the hardware implementation improved execution time in each case.

4. Conclusion and Future Work

In conclusion, a survey of hardware JPEG implementations was performed to explore the research being done to minimize execution time. Since most JPEG implementations are done in software, they can be reduced in complexity and size through hardware.

The most commonly used JPEG image mode is JPEG Baseline. Compression and decompression of JPEG Baseline images was divided into 7 modules: DCT, IDCT, quantization, dequantization, encoding, decoding, and color transform module. The DCT and IDCT modules are the most processor intensive. Consequently they have the most potential for hardware optimization. Compressing images requires more processing than decompressing images due to Huffman and Run length tables as well as quantization.

When comparing the performance of hardware and software, between 84% and 88% of total execution cycles were saved with hardware. That performance increase was observed for both lossless and near lossless compression for five different image types. Encoding images yielded a slightly better performance increased than decoding because of the increase in complexity. Biometric images performed slightly better than the other image types when decoding.

References

[1] L. Agostini, S. Bampi, "Integrated Digital Architecture for JPEG Image Compression," *European Conference on Circuit Theory and Design*, Vol. III, pp. 181-184, 2001

[2] L. Agostini, I. Silva, S. Bampi. "Pipelined Fast 2-D DCT Architecture for JPEG Image Compression," *Symposium on Integrated Circuit Design and System Design*, September, 2002

[3] Z. Yusof, Z. Aspar, I. Suleiman, "Field Programmable Gate Array (FPGA) Based Baseline JPEG Decoder*," Conference on Convergent Technologies For The Asia-Pacific TENCON 2000,* Vol. 3, pp. 218-220, 2000

[4] S Lei, M Sun, "An Entropy Coding System for Digital HDTV Applications," *IEEE TRANSACTION ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, no. 1, pp. 147-154, March, 1991

[5] A Savakis, M. Piorun, "Benchmarking and Hardware Implementation of JPEG-LS," *International Conference on Image Processing*, Vol. 2, pp. 22-25, September, 2002