Integrated Digital Architecture for JPEG Image Compression

Luciano Agostini* and Sergio Bampi*

***Abstract* – This paper presents the architecture and design of a JPEG compressor in hardware. The system is a functional unit of a compressor chip, divided in four major parts: color space converter and downsampler, 2-D DCT module, quantization and entropy coding. Architectures for these four parts were designed and described in VHDL. The results of the VHDL mapping into Altera Flex 10K FPGAs are also herein presented.**

1 Introduction

The Join Photographic Expert Group proposed the JPEG compression standard and the complete standard documentation can be found in [1]. This paper focuses only in the hardware implementation of a subset of the JPEG standard called baseline [2,3]. The baseline is the mode widely used in both software and hardware versions of the JPEG compression.

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.

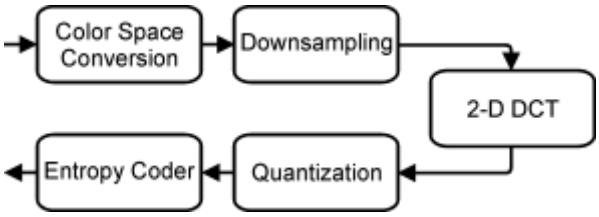

Figure 1 – Steps of a JPEG baseline compression

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 co-sine 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.

The paper sections present the architectures used in each one of the four parts mentioned above, their VHDL description and their synthesis results. The compressor architecture operates in pipeline, whose design is also addressed.

2 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).

$$\begin{aligned} Y &= 0{,}299\,R + 0{,}587\,G + 0{,}114\,B \\ Cb &= 0{,}564\,B - 0{,}564\,Y \\ Cr &= 0{,}713\,R - 0{,}713\,Y \end{aligned} \qquad (1)$$

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.

* Universidade Federal do Rio Grande do Sul, Microeletronics Group, P.O.Box 15 064, Porto Alegre, Brazil. E-mail: {agostini, bampi}@inf.ufrgs.br
Tel: +55 (51) 316-6812, Fax: +55 (51) 319-1576.

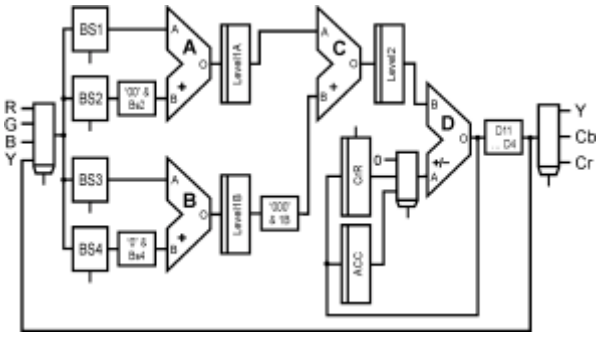

Figure 2 – Integrated architecture for color space conversion and downsampling

Table 1 presents the VHDL description results. The mapping in Altera [4] FPGAs was made using Flex10KE devices.

	Logic Cells	Period (ns)	Code Lines
Color Space Converter and Downsampler	441	49,7	869

Table 1 – VHDL mapping results for the integrated color space conversion and downsampling module.

3 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 m1 = $\cos(4\pi/16)$, m2 = $\cos(6\pi/16)$, m3 = $\cos(2\pi/16) - \cos(6\pi/16)$ and m4 = $\cos(2\pi/16) + \cos(6\pi/16)$).

The designed architecture for 1-D DCT calculation is presented in Fig. 3. This architecture is based on the architecture proposed by [6] and uses five ripple-carry adders and one multiplier. In the implementation, the multiplier is similar to that used in the integrated architecture of space color conversion and downsampling, using shift-and-adds.

Step 1		
$b0 = a0 + a7$	$B1 = a1 + a6$	$b2 = a2 - a4$
$b3 = a1 - a6$	$B4 = a2 + a5$	$b5 = a3 + a4$
$b6 = a2 - a5$	$B7 = a0 - a7$	
Step 2		
$c0 = b0 + b5$	$C1 = b1 - b4$	$c2 = b2 + b6$
$c3 = b1 + b4$	$C4 = b0 - b5$	$c5 = b3 + b7$
$c6 = b3 + b6$	$C7 = b7$	
Step 3		
$d0 = c0 + c3$	$D1 = c0 - c3$	$d2 = c2$
$d3 = c1 + c4$	$D4 = c2 - c5$	$d5 = c4$
$d6 = c5$	$D7 = c6$	$d8 = c7$
Step 4		
$e0 = d0$	$e1 = d1$	$e2 = m3 \times d2$
$e3 = m1 \times d7$	$e4 = m4 \times d6$	$e5 = d5$
$e6 = m1 \times d3$	$e7 = m2 \times d4$	$e8 = d8$
Step 5		
$f0 = e0$	$f1 = e1$	$f2 = e5 + e6$
$f3 = e5 - e6$	$f4 = e3 + e8$	$f5 = e8 - e3$
$f6 = e2 + e7$	$f7 = e4 + e7$	
Step 6		
$S0 = f0$	$S1 = f4 + f7$	$S2 = f2$
$S3 = f5 - f6$	$S4 = f1$	$S5 = f5 + f6$
$S6 = f3$	$S7 = f4 - f7$	

Table 2 – Scaled 1-D DCT algorithm

The input data in each step of the scaled algorithm is stored in ping-pong buffers to make possible the use of just one operator per step. This architecture operates in a 48-stages pipeline. One 8x8 input matrix is calculated at every 64 clock cycles with the pipeline full and with a pipeline latency of 48 cycles.

A transpose buffer connects the two 1-D DCT architectures. This buffer was designed with two small 64-word RAMs. When the first 1-D DCT architecture writes the results line by line in one memory, the second 1-D DCT architecture reads the input values column by column from the other memory.

The 2-D DCT VHDL results are presented in Table 3. Altera FPGA of Flex 10KE family was used.

	Logic Cells	Period (ns)	Memory Bits	Lines of Code
1-D DCT 1	2051	73,2	0	2446
1-D DCT 2	2473	80,8	0	2468
Trans.Buf.	274	36,5	1408	439
2-D DCT	4792	78,1	1408	5353

Table 3 – 2-D DCT VHDL synthesis results

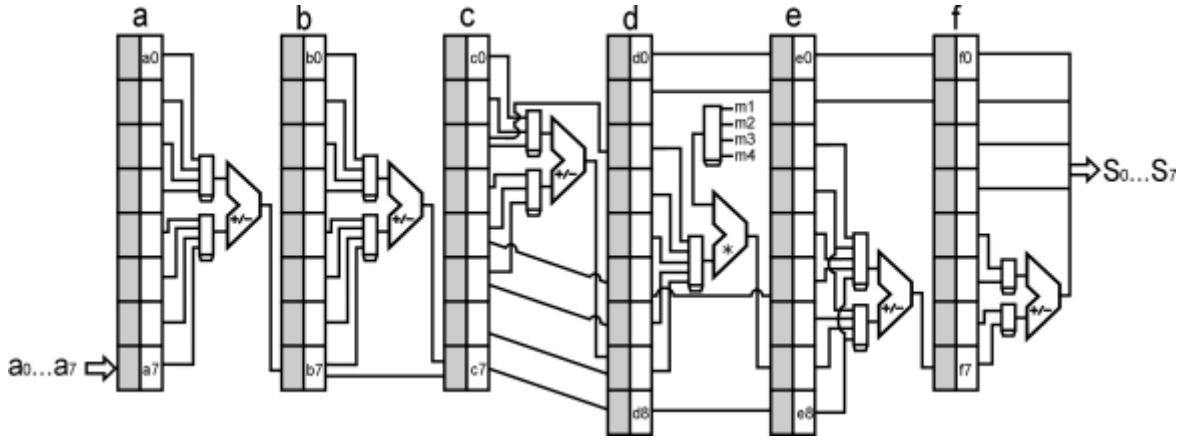

Figure 3 – 1-D DCT architecture

4 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.

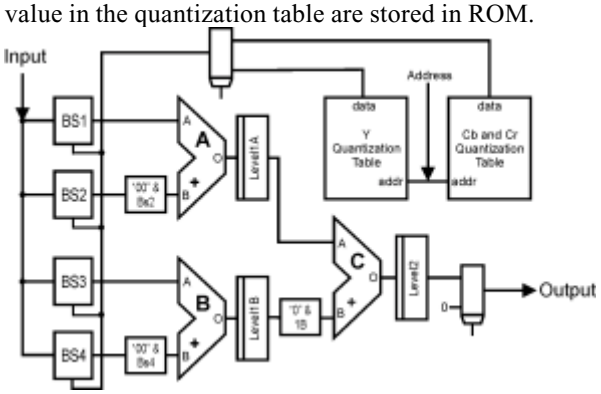

Figure 4 – Quantization architecture

The mapping of the quantization architecture was done in an Altera Flex 10KE FPGA device. The results of this mapping are presented in Table 4.

	Logic Cells	Period (ns)	Memory Bits	Lines of Code
Quantization	293	36,9	1536	676

Table 4 – VHDL synthesis results for the quantization.

5 Entropy Coder

The last stage of the JPEG compression is the entropy coding. After the quantization process, the resulting matrix will have a large amount of zero occurrences.

This matrix is read in zigzag order to increase the sequences of zeros that are compressed by RLE.

In entropy coding the DC and AC coefficients are handled separately, as shown in Fig. 5. The DC component is the first component in an 8x8 matrix (index 0,0) and the AC components are the remaining 63 elements.

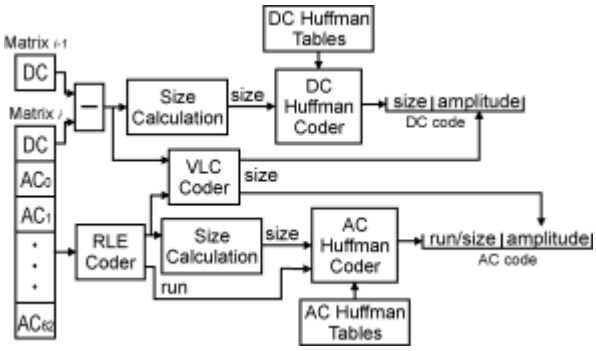

Figure 5 – Entropy coder

The DC components of successive 8x8 windows in an image have a high degree of correlation. Then the first step in a DC entropy coding is a differential coding

between the actual DC component and the DC component of the previous matrix. This differential code is coded by VLC were all non-significant bits are discarded (including the signal bit). The differential value is used also to calculate the number of significant bits that are generated by the VLC coder. This operation is made by a size calculation and generates the "size" field (Fig. 5). The "size" field is Huffman-coded. There are four Huffman tables in the JPEG baseline operation: one for DC luminance components, one for DC chrominance components, one for AC luminance components and the last one for the AC chrominance components. The values generated by Huffman coder and by VLC coder are concatenated to generate the JPEG DC code (Fig. 5) [7].

The first step in AC components coding is the RLE, which is simplified to be only a zeros counter. The RLE coder generates the "run" field that represents the number of zeros that precedes a non-zero value. The non-zeros values pass by a VLC coder and by a size calculation. The resulting concatenation of the fields "run" and "size" are Huffman-coded. Finally the Huffman coded fields "run/size" are concatenated with the field "amplitude", generated by the VLC coder, to generate the JPEG AC code [7].

The entropy coder architecture is being designed. Architecture implementation has been finished for the differential coder, the size calculation and the RLE encoder.

Differential coder is a single subtrator associated with two registers. The size calculation is made with a combinational logic and does not use memory to store the size table [1]. The RLE coder is shown in Fig. 6 and was designed using a zero comparator and one counter.

The Huffman coder will use the pre-determined Huffman tables suggested by the JPEG standard [1]. These tables will be stored in the FPGA internal memory.

The first VHDL description results for the entropy coder are presented in Table 5.

	Logic Cells	Period (ns)	Lines of Code
RLE	26	5,8	97
Size calculation	18	14,3	70
Diferential coding	46	4	108
Total (preliminary)	90	14,3	275

Table 5 – Preliminary results for entropy coding VHDL synthesis

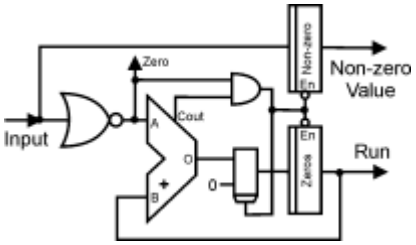

Figure 6 – RLE coder architecture

6 Conclusions

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.

Future work calls for the reutilization of the IP modules herein presented into a full compressor chip (with I/O and memory control functions also embedded in the system). The overall bus and interconnection architecture is to be proposed, while the JPEG modules addressed in this paper will compose a single, dedicated functional unit for this compressor chip. We expect to use a Flex10KE200 and up to 7,000 logic cells for this functional unit. The target clock period for the complete architecture is 80ns, which is a reasonable target, given the results already presented.

References

[1] The International Telegraph and Telephone Consultative Committee (CCITT). "Information Technology – Digital Compression and Coding of Continuous-Tone Still Images – Requirements and Guidelines". Rec. T.81, 1992.

[2] W. Pennebaker and J. Mitchell. JPEG Still Image Data Compression Standard, Van Nostrand Reinhold, USA, 1992.

[3] J. Miano. Compressed Image File Formats – JPEG, PNG, GIF, XBM, BMP, Addison Wesley Longman Inc, USA, 1999.

[4] Altera Data Book, Altera Corporation, 1995.

[5] Y. Arai, T. Agui and M. Nakajima. "A Fast DCT-SQ Scheme for Images". Transactions of IEICE, vol. E71, n°. 11, 1988, pp. 1095-1097.

[6] M. Kovac and N. Ranganathan. "JAGAR: A Fully Pipeline VLSI Architecture for JPEG Image Compression Standard". Proceedings of the IEEE, vol. 83, n°. 2, 1995, pp. 247-258.

[7] V. Bhaskaran and K. Konstantinides. Image and Video Compression Standards Algorithms and Architectures – Second Edition, Kluwer Academic Publishers, USA, 1999.