INCREASING THE IMAGE QUALITY IN A JPEG COMPRESSOR THROUGH ARITHMETIC ERROR MINIMIZATION

Roger Endrigo Carvalho Porto[1], *Bruno Silveira Neves*[2],
Luciano Volcan Agostini[1][2], *José Luís Almada Güntzel*[1]

[1]Grupo de Arquiteturas e Circuitos Integrados (GACI)
Departamento de Informática – Universidade Federal de Pelotas (UFPEL)
Caixa Postal 354 – CEP 96010-900 – Pelotas – RS – Brasil

[2]Grupo de Microeletrônica (GME)
Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre – RS – Brasil

{rogerecp, guntzel, agostini}@ufpel.edu.br, {bsneves, agostini}@inf.ufrgs.br

ABSTRACT

This paper presents solutions to increase the quality of a JPEG compressor directed to gray scale images designed in hardware in previous works. The focus of this paper is to minimize the errors generated by three simplified multipliers that are present into the 2-D DCT and quantization calculations. The calculation error founded through simulation was of 0.8% when the architectural results are compared with ideal results. This is a very small error but it may cause a high image distortion in the decompression operation. The best solution designed in this work increases de quality of the compressed data in a rate higher than 93%. This solution generates an increase of 7.8% in the use of logic cells and memory bits in the complete JPEG compressor and it increases the compressor period in 19.2%. This solution is able to process 23 million of gray scale pixels per second.

1. INTRODUCTION

The principle of the JPEG compression is the use of controllable losses to reach high compression rates. In this context, the information is transformed to the frequency domain through 2-D DCT [1]. Since neighbor pixels in an image have high likelihood of showing small variations in color, the DCT output will group the higher amplitudes in the lower spatial frequencies [2]. Then, the higher spatial frequencies can be discarded by the quantization, generating a high compression rate and a small perceptible loss in the image quality.

This paper presents an analysis of the internal calculation errors generated by some simplifications that were realized into the original compressor to minimize the use of resources and to increase the compressor performance. The sources of errors and theirs impacts are presented in this paper and architectural solutions are proposed and designed to minimize these errors. The synthesis results of the designed architectures and the error minimization obtained with these architectures are also presented.

Section 2 of this paper presents a short introduction about JPEG compression. Section 3 presents the error evaluation in the original JPEG compressor. Section 4 presents the proposed solutions to minimize the errors presented in section 3 and the architectures that were designed to these proposed solutions. Section 5 presents the synthesis results of the designed architectures into the JPEG compressor. Finally, section 6 presents the conclusions of this paper.

2. ORIGINAL JPEG COMPRESSOR ARCHITECTURE

The JPEG compressor designed in hardware in previous works [3], focus of this paper, uses the JPEG [4] baseline compression mode [5]. The baseline compression mode can be divided in three main steps, as is showed in Fig. 1: 2-D DCT (Two Dimensional Discrete Cosine Transform), quantization and entropy coding.

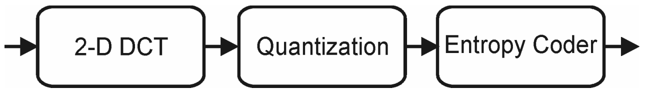

Figure 1 – JPEG baseline compression

The 2-D DCT and quantization architectures are shortly presented in this section because only these architectures perform arithmetical operations and, for consequence, may generate calculations errors in theirs outputs.

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 the 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 used in the original implementation was proposed in [6] and modified in [7]. 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 DCTs calculations it is possible to generate the 2-D DCT results, as showed in Fig. 2. 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 division reduces the calculation complexity.

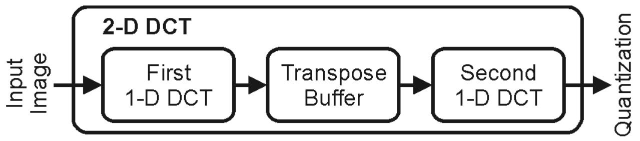

Figure 2 – Block diagram of the 2-D DCT architecture

The original architecture for 1-D DCT calculation is presented in Fig. 3. This architecture is based on the architecture proposed by [7] and it uses ripple-carry adders. The original multipliers was decomposed in shift-adds operations, then the 2-D DCT architecture is a multiplication free architecture. This multiplier is highlighted in Fig. 3 because it is the main error generator into the 1-D DCT architectures.

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 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 for gray scale images there is just one quantization table. The optimum values of the components in quantization table are dependent on the application, but the JPEG standard suggests a typical table that has a good efficiency for any application [5].

Quantization attenuates or eliminates the 2-D DCT coefficients that are less perceptible to the human eye. The result of this operation is a sparse matrix [2]. The quantization architecture used in this paper is presented in Fig. 4 and it uses one ROM and one multiplier to calculate the quantized coefficients. The multiplier used in the quantization is similar to that used in the two 1-D DCT modules and it is an error generator too. The barrel shifter control words for each value in the quantization table are stored in ROM.

The multipliers used into the two 1-D DCT and into the quantization architectures were decomposed in shifts and adds to minimize the use of hardware resources. Since one of the multiplier inputs is always a constant, it is possible to preview the number of shifts necessary for each calculation [3]. The shifts are performed by barrel shifters.

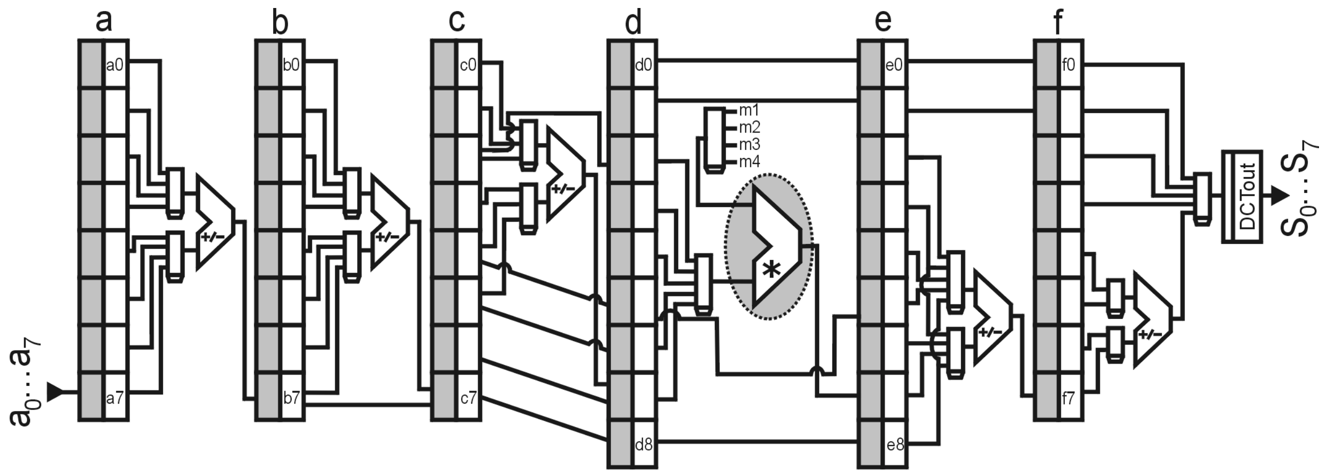

Figure 3 – 1-D DCT architecture

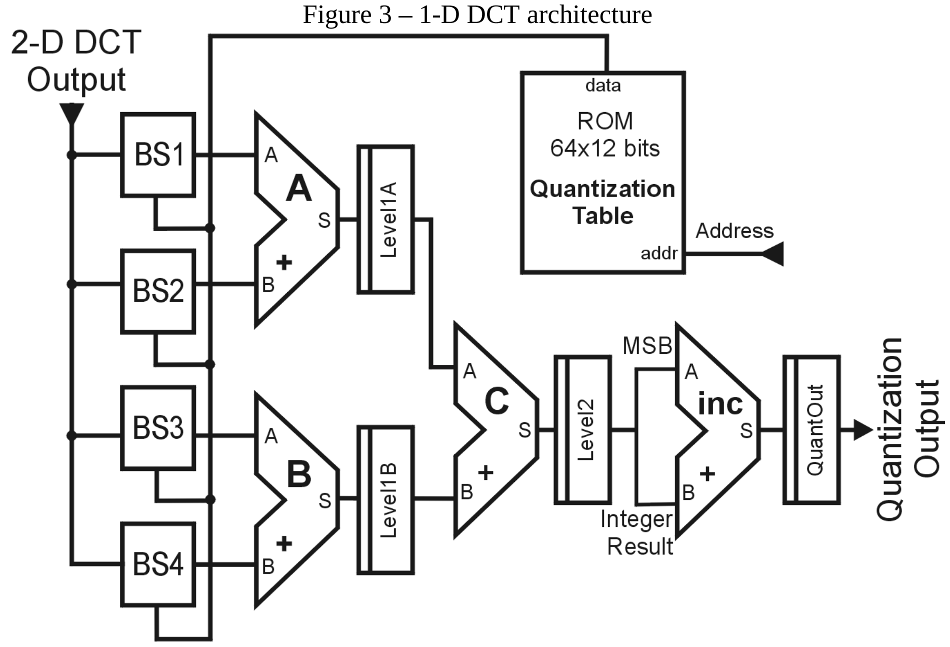

Figure 4 – Quantization architecture

Just four shift-adds are used in the original multipliers architecture to save arithmetic units. Internally to the multipliers all the significant bits were considered to maximize the precision of this calculation but the multiplier outputs are truncated to discard the fractionary part of this internal result. These simplifications generate errors into the calculated results because a finite number of additions of different shifts are used and the multiplication result is truncated. The original multiplier architecture is showed in Fig. 5.

All other arithmetic operations into the 2-D DCT and quantization architectures do not generate errors. These other operations are additions or subtractions of integer numbers and all architectural operators consider one additional bit when carry out is generated.

The last operation in a JPEG compression is the entropy coding that uses several losses less compression techniques to reduce the amount of bits necessary to represent the image [2]. This operation does not generate arithmetic calculation errors and then, it will not be detailed in this paper.

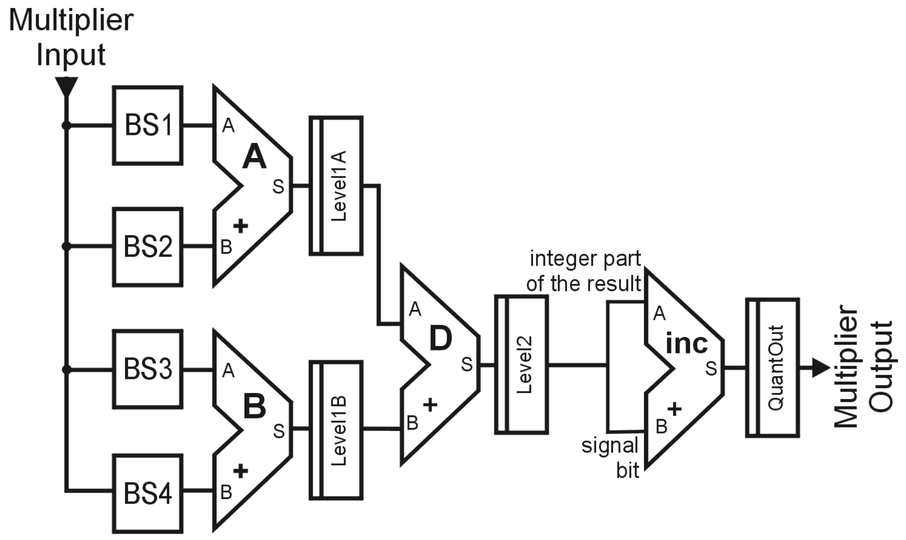

Figure 5 – Original Multiplier architecture used into the two 1-D DCT and into Quantization blocks

3. ERROR EVALUATION IN THE ORIGINAL JPEG COMPRESSOR

To evaluate the error generated into the JPEG compressor were designed two descriptions of the compressor in C language. The first one does not consider the architecture restrictions and, for consequence, it does not generate errors. This description was used to generate accurate results to be compared with the architectural results. The second C code is a literal and detailed description of the designed architecture of the JPEG compressor. The results generated by this second description were used to evaluate the JPEG compressor error.

Only the first and the second JPEG compressor blocks in Fig. 1 (2-D DCT and Quantization) operate data calculation and then, for this, only these blocks may generate arithmetical errors. These errors are caused by simplifications into the designed operators. Special attention was given to the multipliers used into the two 1-D DCT and into the quantization architectures. The original multipliers architectures were accurately designed in the C description of the architecture to make sure that the comparative results were considering the effects of the shift-adds in the image quality.

Another important aspect that was considered in the multipliers architecture description in C was the truncating realized in the multipliers outputs. This was realized to eliminate the fractionary part of the results, once the entropy coder, that uses the quantization results, is able to process just integer numbers. There is an incrementer in the original multipliers output, as showed in Fig 5, but this incrementer is used just to correct the negative results and it is not used to round the multiplier outputs. The absence of this rounding operation is another potential generator of arithmetical errors and it was preserved in the C description of the designed architecture.

It is important to emphasize that the errors that this paper aims to minimize are not the errors relative to the losses generated for the process of JPEG compression. The errors focused in this paper are generated by the truncations and rounds realized in the internal compression calculations. These truncations are performed because the designed JPEG compressor architecture uses fixed point in the internal results representation. The JPEG compression standard defines that these results must use a floating point representation. Basically, the operations in floating point that had been truncated or rounded to be represented in fixed point were multiplications by cosines.

The JPEG compression is a lossy compression and then, a part of the information included in the original image is eliminated in an irreversible way.

This loss of information is controlled to not harm the image perception by the human eye [3]. On the other hand, the truncations and rounds errors that are generated in the internal calculation of the proposed architecture can have a catastrophic influence in the image quality when the decompression process is realized. In this process, the truncations and rounds errors will modify significantly the value of some image pixels when the inverse quantization is performed. In this operation, an error of one unit in the input value can cause an error of up to 100 units in the output value and, therefore, these errors must be minimized or eliminated and this is the goal of this work.

Were designed a group of thirty matrixes of sixty four elements in each one to stimulate the two C descriptions designed to comparing the ideal results with the architectural results and for evaluating the architectural errors. These input matrixes were specially constructed to generate a high error in the outputs and to sensitize all the architectural critical modules.

The first simulation was made to discover the error that was generated by the hardware implementation of the original JPEG compressor. The quantization outputs of the implementation without errors it were compared with the quantization outputs of the architectural implementation. This comparison was made for all the thirty matrixes and it generates thirty error matrixes. The error matrixes were generated through a simple subtraction of the ideal value of each element of each matrix by the value of the same matrix element generated through the architectural description.

The modules of all elements of the thirty error matrixes were added to generate the absolute error. For these thirty matrixes, the absolute error was of 29 units. This number represents an error of 0.7932166 percent in all JPEG compressor calculations for this group of matrixes. This is a very small error, but may not be an acceptable error in applications where the image quality is the primordial factor. Then, some architectural alternatives were explored to minimize these errors. These alternatives are presented in the next section of this paper.

4. PROPOSED SOLUTIONS TO MINIMIZE THE ERROR

The comparisons between the ideal results and the architectural results confirm that the multipliers of the two 1-D DCT and quantization were the unique generators of arithmetic errors into the JPEG architecture. Then some improvements in these multipliers were proposed and designed.

The first alternative proposed to improve the multiplier results was the design of a rounding operator to be used in the multipliers output. With this solution, it was possible to eliminate the errors generated by the truncation of the results in the original architecture.

The designed solution uses the incrementer that is included in the original architecture to minimize the use of resources. The original incrementer was used to correct the negative results. The multiplier designed through shift-adds presents one unitary error in all negative results, then, the original incrementer was used to add one in all negative results and, for consequence, to correct these results. Then the original incrementer adds the signal bit that is the most significant bit (MSB) to the integer part of the multiplication results.

To generate a rounding operator, another incrementer is needed. This new incrementer is used to add one in the integer part of the multiplications results if the most significant bit of the fractionary part is one. Then, this new incrementer must add the most significant bit of the fractionary part of the result and the integer part of this result. A single architecture was designed to join these two incrementers in just one operator. This architecture is presented in Fig. 6.

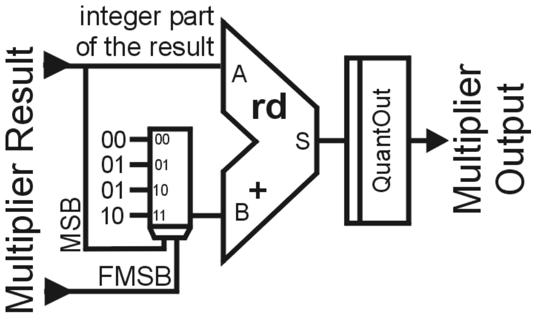

Figure 6 – Rounding architecture used in the multipliers outputs

The architecture presented in Fig. 6 uses one adder that receives as input **A** the integer part of the multiplier result and as input **B** a two bit value generated according Tab. 1. In Tab. 1 the column **MSB** indicates the value of the most significant bit of the multiplier result (that is the signal of the result) and the column **FMSB** indicates the value of the fractionary most significant bit. As showed in Fig. 6, to implement the function of the truth table presented in Tab. 1 was used one multiplexer with constants “00”, “01” and “10” as inputs and with the bits MSB and FMSB extracted from the multiplier result used as control signals.

The second alternative designed to improve the multiplier results was the use of more accuracy constants in the multiplications of the 2-D DCT and quantization. As explained in section 2, one of the multipliers inputs is always a constant and the other input is a variable value.

Table 1 – Truth table for the B input of the adder

MSB	**FMSB**	**B Input**
0	0	00
0	1	01
1	0	01
1	1	10

To increase the quality of the multiplications results it is necessary to improve the accuracy of the used constants. This means that the number of shifts that are added must increase. Then, three alternative solutions were designed in C language. These solutions use respectively five, six and seven shifts-adds into the three multipliers and they use the rounding architecture presented in Fig. 6. Comparing the results of these three alternative multipliers with the original multiplier and with the ideal multiplier it was possible to evaluate the results in quality improvements and the residual calculation error. These comparing results are presented in Tab. 2 and are related to the same group of thirty matrixes used previously.

Table 2 – Quality improvements and residual error generated through alternative multipliers architectures

Solution	**Quality Improvements**	**Residual Error**
Four shift-adds	-	0.793%
Five shift-adds	75.862%	0.192%
Six shift-adds	82.759%	0.137%
Seven shift-adds	93.104%	0.055%

From Tab. 2 is possible to notice that the use of rounding and the increase of the number of shit-adds, as expected, improve significantly the JPEG compressor results. The quality of the results with the multipliers using seven shifts was 93% better than with the original JPEG architecture multipliers. But there is a residual error in the final simulation results. This error is around 0.06% using seven shifts into the multipliers to the selected group of thirty matrixes. For typical matrixes, extracted from real images, this error will be near to zero.

Fig. 7 presents, as example, the architecture proposed for the multipliers using seven shifts to be added. As it can be notice, this architecture is significantly larger than the original architecture presented in Fig. 5. The figure of the multiplier architectures using five and six shifts will not be presented in this paper.

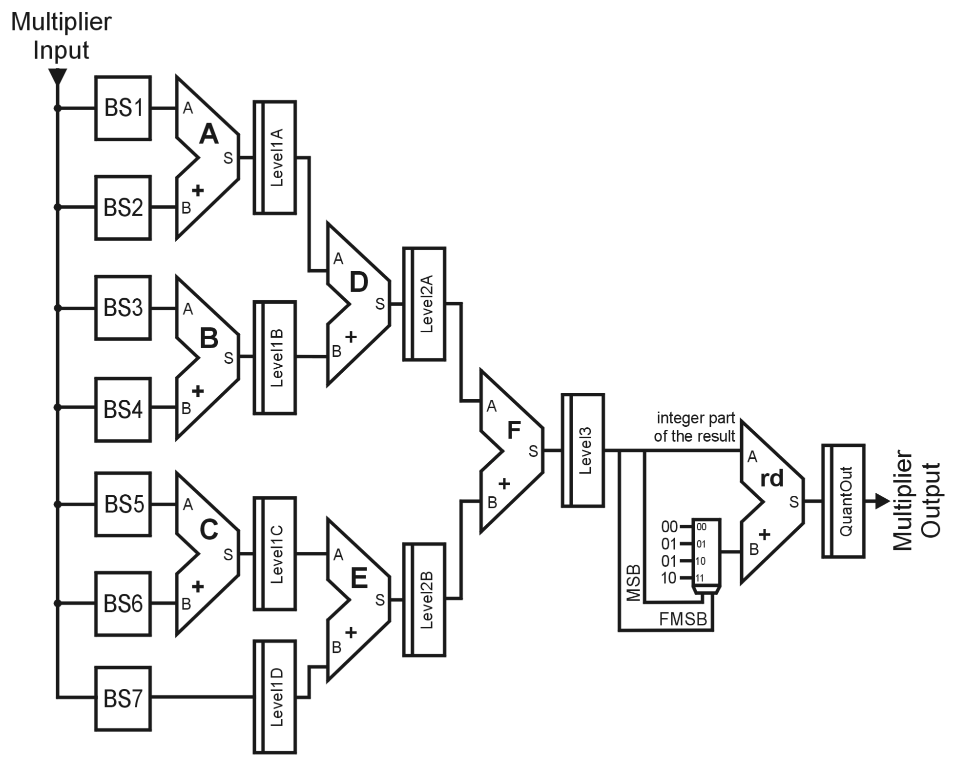

Figure 7 – Multiplier using seven shift-adds designed to minimize the arithmetic errors in the two 1-D DCT and in the quantization architectures

5. SYNTHESIS RESULTS

The proposed multipliers that were designed and simulated in C language were designed in VHDL and synthesized for Altera [8] FPGAs from Flex 10KE family [9]. Tab. 3 shows the JPEG compressor synthesis results to each alternative solution that was designed.

Table 3 – JPEG compressor synthesis results using the alternative multipliers

Solution	**Area** (LCs)	**Memory** (bits)	**Period** (ns)	**Frequency** (MHz)
Four shift-adds	4454	7436	36.5	27.40
Five shift-adds	4613	7628	39.7	25.19
Six shift-adds	4714	7820	41.5	24.10
Seven shift-adds	4802	8012	43.5	22.99

EPF10K130EQC240-1 FLEX10KE Altera device

As expected, the use of more shifts into the multipliers implies in an increase in the use of logic cells an in the use of memory bits. The use of logic cells occur because new barrel shifters, new registers and new adders are joined to the multiplier architecture at each increase in the number of used shifts. Tab. 4 presents this relation between the number of shifts and the hardware resources.

Tab. 4 presents also the number of bits used in the words of the quantization ROM to each designed solution. The barrel shifters controls are stored into this ROM and each one uses three bits in each memory word.

The increase in the number of shifts used into the multipliers caused also a reduction in the JPEG compressor operation frequency, as showed in Tab. 3.

Table 4 – Use of hardware resources into the multipliers solutions

Solution	**# of Barrel Shifters**	**# of Registers**	**# of Adders**	**Memory Word (bits)**
Four shift-adds	4	4	4	12
Five shift-adds	5	7	5	15
Six shift-adds	6	7	6	18
Seven shift-adds	7	8	7	21

Tab. 5 presents a comparison between the designed solutions. It presents the losses in terms of use of resources, the losses in terms of period and the improvements in terms of image quality.

The best multiplier solution in terms of image quality is the one with seven shifts. This solution is able to process 23 million of gray scale pixels per second, generating an arithmetical error of 0.05%. This architecture is able to process 74.8 frames of 640x480 pixels in one second.

The solution that uses six shift-adds is able to process 24 million of gray scale pixels per second, generating an error of 0.14%. This architecture is able to process 78.4 frames of 640x480 pixels in one second.

Table 5 – Comparative results between the alternative multipliers used into the JPEG compressor

Solution	**Four shift-adds**	**Five shift-adds**	**Six shift-adds**	**Seven shift-adds**
Logic Cells Losses	-	3.57%	5.84%	7.81%
Memory Bits Losses	-	2.58%	5.16%	7.75%
Period Losses	-	8.77%	13.70%	19.18%
Quality Improvements	-	75.86%	82.76%	93.10%

The compressor that uses multipliers with five shift-adds is able to process 25 million of gray scale pixels per second, generating an error of 0.19%. This compressor is able to process 82 frames of 640x480 pixels in one second.

The original architecture, using four shift-adds, is able to process 27 million of gray scale pixels per second, generating an error of 0.79%. The original architecture is able to compress 89 frames of 640x480 pixels in one second.

6. CONCLUSIONS

This paper presented solutions to increase the quality of the image generated by a JPEG compressor directed to gray scale images designed in previous works. To minimize the errors generated by this compressor the first step was the accurate evaluation of these errors through simulations in C language. These simulations indicated that the error was of 0.8% when the architectural results are compared with ideal results.

Some architectural modifications were designed in VHDL to increase the quality of the image generated by the original JPEG compressor architecture. These modifications were focused in three simplified multipliers that are used into the JPEG compressor. The designed solutions increase the image quality through the use of four, five, six or seven shift-adds into the multipliers and through the use of a rounding architecture in the multipliers output. The synthesis results of the designed solutions were presented and analyzed in this paper.

The best solution in terms of the quality of the compressed data uses seven shift-adds into its multipliers. This solution improves the image quality in a rate higher than 93% and it reduces the errors to 0.06%. This best solution generates an increase of 7.8% in the use of logic cells and memory bits and it increases the compressor period in 19.2%. This solution is able to process 23 million of gray scale pixels per second reaching a processing rate of 74.8 frames of 640x480 pixels per second.

The simulation and synthesis results of all designed solutions were presented in details in this paper. These results were considered satisfactory because the quality improvement was significant and the impacts in terms of use of resources and in terms of performance were not very high. An interesting investigation that was not designed in this paper is the use of faster adder operators joined with the solution that uses seven shift-adds into the multiplier. Then, the performance of the modified JPEG compressor should increase and it should be equal or higher than the original compressor performance with a calculation error near to zero.

7. REFERENCES

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

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

[3] L. V. Agostini, “Design of Architectures for JPEG Image Compression” (Portuguese). Master Dissertation – Federal University of Rio Grande do Sul. Informatics Institute. Pos-Graduation in Computer Science Program, Porto Alegre, RS, Brazil, 2002. 143p.

[4] JPEG and JBIG Committees, “Home Site of the JPEG and JBIG Committees” <http://www.jpeg.org>

[5] 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.

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

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

[8] ALTERA Corporation, *Altera: The Programmable Solutions Company*, San Jose: Altera Corporation, Available in: <http://www.altera.com> 2003.

[9] ALTERA Corporation. *FLEX 10KE – Embedded Programmable Logic Devices Data Sheet – version 2.3.* San Jose, Altera Corporation, 2001.