AVALIAÇÃO DOS IMPACTOS DO USO DE SOMADORES COMO MACRO FUNCTIONS EM UM PROJETO DE COMPRESSOR JPEG

Giovano Camaratta[1], Fábio Daitx[1], Luciano Agostini[1,2], Sergio Bampi[1]

[1] Grupo de Microeletrônica (GME)
Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15064 – CEP. 91501-970 – Porto Alegre /RS – Brasil
[2] GACI – UFPel – Pelotas/RS – Brasil

{grcamaratta, ffdaitx, agostini, bampi}@inf.ufrgs.br

RESUMO

Esse artigo apresenta uma variação no projeto de um compressor de imagens JPEG *baseline* em tons de cinza proposto, implementado e validado em trabalhos anteriores. As modificações do projeto visam explorar recursos adicionais do FPGA para aumentar o desempenho e diminuir a utilização de recursos, através da substituição dos somadores da arquitetura original por somadores *Macro Function* da biblioteca IEEE. A arquitetura do compressor, apresentada nesse artigo, foi descrita em VHDL e sintetizada para FPGAs Flex10KE da Altera. O compressor JPEG é pipelinezado e tem uma latência mínima de 238 ciclos de relógio, considerando o pipeline totalmente preenchido. Com as alterações propostas neste artigo, a freqüência máxima de operação atingiu 45,05MHz, sendo 24% maior que a freqüência original, permitindo o processamento de até 147 imagens de 640x480 pixels por segundo. Foram utilizadas 4612 células lógicas para a síntese completa do compressor JPEG, o que representou um acréscimo de apenas 2,01% em relação à arquitetura original.

1. INTRODUÇÃO

O padrão de compressão JPEG [1] foi proposto pelo *Joint Photographic Experts Group* em 1987 e, desde então, é o padrão mais conhecido e usado para compressão de imagens fotográficas. A compressão JPEG atinge taxas de compressão elevadas, com um impacto relativamente mínimo na qualidade das imagens comprimidas.

A disseminação do tratamento de imagens digitais (câmeras digitais, celulares, DVD players, etc...) estimulou o desenvolvimento de um projeto em hardware, mais rápido e com menor consumo de energia, para a compressão de imagens.

Tendo em vista as exigências críticas de redução de tamanho, aumento do desempenho e redução no consumo de energia de alguns destes sistemas embarcados, torna-se importante a otimização da arquitetura do compressor JPEG. É nesse contexto que se enquadra este trabalho, visando reduzir a área ocupada e melhorar o desempenho do compressor JPEG para imagens em tons de cinza. O compressor proposto foi projetado e implementado originalmente em [2] e utiliza muitos somadores, tendo em vista a elevada quantidade de cálculos necessários para a compressão JPEG. O compressor JPEG foi originalmente desenvolvido com somadores simples do tipo *Ripple Carry*, que são somadores lentos, principalmente para somas com números com elevada faixa dinâmica. Estes somadores foram modificados em trabalhos anteriores [3], na tentativa de obtenção de melhor desempenho através do uso de somadores rápidos. Este objetivo foi alcançado, mas estudos posteriores [4] apontaram para a hipótese de que a utilização dos somadores *Macro Function*, da própria biblioteca IEEE poderiam alcançar excelentes relações de desempenho e custo de hardware, por explorarem de forma otimizada alguns recursos disponíveis no FPGA. Além disso, o uso de *Macro Functions* nas descrições VHDL simplifica a tarefa do projetista.

Esse trabalho realizou a substituição de todos somadores da arquitetura do compressor JPEG por somadores *Macro Function* (MFA) e analisou os resultados obtidos. O somador *Macro Function* é uma função da biblioteca **ieee.unsigned.all** acessível através do código **"+"**, para os somadores ou **"-"**, no caso dos subtratores. Os resultados e a análise comparativa, bem como a explicação da arquitetura projetada para FPGAs da família FLEX 10KE da Altera [10] são apresentados nesse texto.

Esse artigo é dividido em cinco seções. A seção 2 introduz o padrão JPEG e a seção 3 apresenta a arquitetura

projetada para a compressão JPEG de imagens em tons de cinza. A seção 4 mostra os resultados da síntese do compressor JPEG com somadores *Macro Function* e compara esses resultados com os obtidos em trabalhos anteriores. Finalmente, as conclusões do trabalho são apresentadas na seção 5.

2. COMPRESSÃO DE IMAGENS JPEG

O núcleo da compressão JPEG é a Transformada Discreta do Cosseno de duas dimensões (DCT 2-D) a qual, com a quantização e com técnicas de compressão sem perdas, faz com que seja possível a redução significativa na quantidade de dados necessária para representar a imagem. O padrão JPEG 2000 [5] prevê a utilização da Transformada Discreta Wavelet (DWT) ao invés da DCT. A DWT conduz a taxas de compressão ainda maiores e propicia imagens de maior qualidade.. A compressão JPEG baseada na DCT foi utilizada no desenvolvimento desse artigo, uma vez que a maioria dos compressores JPEG implementados até o presente momento ainda usam a DCT.

A arquitetura apresentada neste trabalho foi projetada de forma modular e hierárquica; permitindo, a reutilização de todos os seus módulos em outros projetos.

A compressão JPEG para imagens em escala de cinza pode ser dividida em três operações principais, como é mostrado na Fig. 1: DCT 2-D, quantização e codificação de entropia [2,6].

Figura 1 – Operações da compressão JPEG para imagens em escala de cinza

Para cada uma das operações apresentadas, foram propostas arquiteturas, as arquiteturas foram descritas em VHDL em nível RTL, o VHDL foi sintetizado, simulado e validado utilizando uma implementação em software dos algoritmos envolvidos. Todos os blocos da arquitetura da Fig. 1 foram completamente propostos, codificados em VHDL, desenvolvidos e sintetizados em trabalhos anteriores [2].

3. PROJETO DA ARQUITETURA DO COMPRESSOR JPEG

A arquitetura do compressor JPEG foi desenvolvida considerando a compressão de imagens em tons de cinza. A imagem de entrada considerada neste trabalho é composta de *pixels* representados por números inteiros sem sinal com precisão de oito bits. A faixa de valores possíveis para os *pixels* da imagem de entrada é de 0 a 255. O *datapath* e o controle da arquitetura foram desenvolvidos de forma hierárquica. A divisão hierárquica do projeto agilizou o desenvolvimento e a implementação de toda a arquitetura, aumentando o seu desempenho. Além deste aspecto, a utilização de hierarquia oferece mais clareza e simplicidade ao projeto. Foi graças ao desenvolvimento hierárquico no projeto original que este trabalho foi possível, pois a substituição dos somadores por *Macro Function* foi tarefa relativamente simples.

O projeto do *pipeline* para o compressor de imagens em tons de cinza começa com a definição da taxa de dados de entrada, cujo valor ficou estabelecido em um consumo de um dado de 8 bits para cada ciclo de relógio. As palavras JPEG de saída possuem 32 bits e são entregues de forma assíncrona. Os detalhes de sincronização são fornecidos nas próximas seções desse artigo [2].

A latência global também é dependente do assincronismo permitido na saída do compressor. A latência mínima do compressor é de 238 ciclos de relógio. A latência elevada é um reflexo da estratégia de desenvolver um *pipeline* com muitos estágios, visando um alto desempenho e um esquema de entrada/saída simples.

3.1. A Arquitetura da DCT-2D

A arquitetura da transformada discreta do co-seno (DCT 2-D) desenvolvida nesse trabalho usa três simplificações de *hardware* principais. A mais significativa é o uso da propriedade da separabilidade da transformada dos co-senos, em que o cálculo da DCT 2-D é dividido em dois cálculos de DCT 1-D. A segunda simplificação é o uso da escalabilidade, em que a arquitetura da DCT 2-D entrega uma escala dos valores reais de saída de uma DCT 2-D. Esta escala deve ser corrigida no cálculo da quantização pela união fatorada de ambos: fatores de escala e constantes de quantização. A terceira simplificação, que também conduz a uma redução na utilização de recursos de hardware, é a decomposição da multiplicação em operações de soma e deslocamento. O projeto é altamente paralelo e é construído sem a utilização de multiplicadores.

O cálculo completo da DCT-2D 8x8 de um bloco com elementos de 64 pixels sem essas simplificações necessita de 4,096 adições e 4,096 multiplicações [6]. Com as simplificações, essa transformada 8x8 usará apenas 704 adições.

Os valores de entrada da primeira DCT 1-D são os mesmos pixels de entrada de 8 bits para o compressor JPEG. Anteriormente ao cálculo da DCT-2D, o pixel de entrada deve passar por uma operação de deslocamento de nível. Com o deslocamento de nível, a faixa de valores da entrada é modificada de [0,..,255] para [128,..,127] através de uma subtração de 128 de todos valores de entrada. Essa operação simplifica-se a uma inversão do bit mais

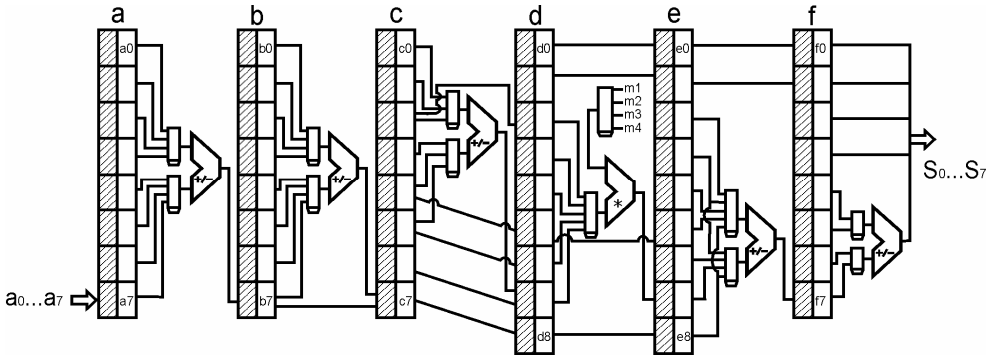

Figura 2 – Arquitetura do Cálculo da DCT 1-D

significativo (*MSB*) de todas as entradas e foi acoplada ao módulo da primeira DCT 1-D.

O algoritmo escolhido nesse trabalho para o cálculo da DCT em uma dimensão foi proposto em [7] e modificado por [8].

As simulações do algoritmo proposto em [8] apresentaram resultados que diferem dos valores esperados para cálculo da DCT 1-D. Esse comportamento inesperado foi pesquisado a fundo e, depois de uma análise detalhada dos resultados da simulação e de comparações com o algoritmo proposto em [7] concluiu-se que o algoritmo proposto em [8] tinha um pequeno erro em uma de suas operações. Esse erro foi corrigido e o algoritmo correto foi utilizado na implementação do compressor JPEG [2].

A arquitetura desenvolvida para o cálculo da DCT-1D é apresentada na Fig. 2, sendo baseada na arquitetura proposta em [8]. A arquitetura da DCT 1-D possui uma latência de 49 ciclos. Como o algoritmo tem seis passos bem definidos e independentes, o pipeline tem seis estágios. Apenas um operador aritmético é usado em cada estágio do pipeline e somente oito ciclos de relógio são usados para executar todas as operações de cada passo do algoritmo. O maior desafio para manter o uso de oito ciclos de relógio por passo do algoritmo está no estágio de multiplicação, mais especificamente, na arquitetura do multiplicador, que precisa efetuar cinco multiplicações por constantes em apenas oito ciclos.

Os multiplicadores utilizados no cálculo da DCT 1-D foram decompostos em somas e deslocamentos para maximizar o desempenho. Como uma das entradas é sempre uma constante, é possível determinar quais deslocamentos serão necessários para cada cálculo. O multiplicador usado neste trabalho usa seis ciclos de relógio para as 5 multiplicações previstas no algoritmo.

A DCT 2-D, mais especificamente as duas DCTs 1-D, representam a parte mais crítica do projeto com relação a quantidade de cálculos realizados.

Nessa etapa do projeto original foram realizados vários testes para avaliar as vantagens do uso de diferentes tipos de somadores nos diferentes estágios de pipeline. Os somadores usados originalmente nas duas arquiteturas da DCT 1-D foram: *Ripple Carry, Carry Look Ahead, Carry Select*. Foram realizados testes com todos os somadores de um mesmo tipo e com combinações refinadas visando maximizar o desempenho e diminuir a utilização de recursos de *hardware* tanto na arquitetura da DCT 2-D quanto na arquitetura completa do compressor JPEG [3,4].

Esse trabalho propôs e desenvolveu uma solução alternativa para a utilização de somadores, onde todos os somadores do projeto foram substituídos por somadores *Macro Functions* disponíveis nas bibliotecas da IEEE. Estes experimentos tiveram o objetivo de atingir resultados ainda melhores na relação entre desempenho e consumo de recursos, quando comparado aos resultados originais. Os resultados obtidos serão discutidos na seção 4 deste trabalho.

O *buffer* de transposição é usado para conectar as arquiteturas das duas DCTs 1-D, onde os resultados da primeira DCT 1-D devem ser armazenados linha a linha e lidos pela segunda DCT 1-D coluna a coluna.

A arquitetura projetada para o *buffer* de transposição utiliza duas pequenas memórias RAM, como está apresentado na Fig. 3. Essas duas memórias operam de maneira intercalada: enquanto uma é usada para escrita, a outra é usada para leitura.

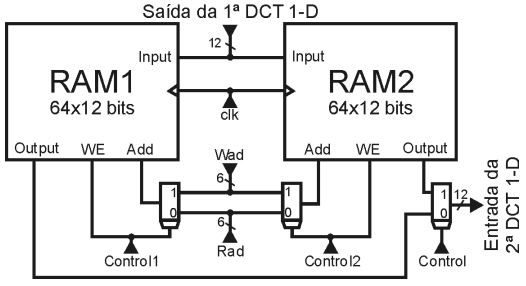

Figura 3 – Arquitetura do Buffer de Transposição

3.2. Arquitetura da Quantização

O funcionamento do módulo da quantização é baseado em uma divisão inteira de todos os coeficientes da DCT 2-D por constantes. A quantização tem como objetivo de gerar

uma matriz esparsa para ser utilizada pela operação do codificador de entropia. Adicionalmente, o cálculo relativo à correção de escala da DCT 2-D foi introduzido na arquitetura da quantização. A quantização e a correção do fator de escala são divisões por constantes e, por isso, estas duas operações podem ser integradas em apenas um módulo de hardware sem que seja aumentada a complexidade do módulo original da quantização. Cada um dos 64 elementos do resultado da DCT 2-D terá seu próprio fator de escala. Sendo assim, a quantização realizará a operação apresentada em (**1**).

$$Cq_{ij} = round\left(C_{ij} \times \frac{1}{Q_{ij} \times Fe_{ij}} \right) \qquad 0 \le i,j \le 7 \qquad (\mathbf{1})$$

em que:

- Cq_{ij} é o coeficiente quantizado;
- C_{ij} é o coeficiente da DCT 2-D;
- Q_{ij} é a constante de quantização e
- Fe_{ij} é o fator de escala da DCT 2-D.

A arquitetura proposta para a quantização está apresentada na Fig. 4. Esta arquitetura é composta, basicamente, por um multiplicador similar àquele que foi utilizado na arquitetura da DCT 1-D, usando operações de somas e deslocamentos.

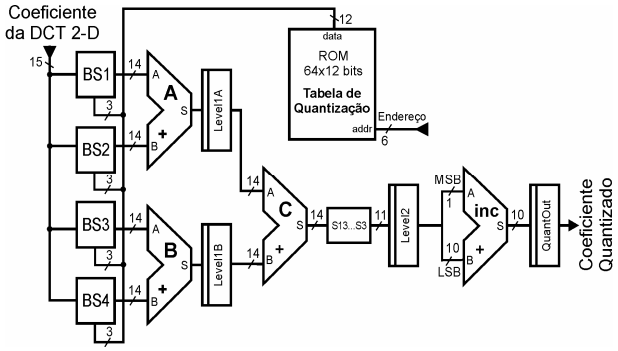

Figura 4 – Arquitetura de Quantização

A arquitetura da quantização utiliza uma memória ROM, que, ao contrário de armazenar a matriz de 64 elementos gerados pela execução do cálculo de $1/(Q_{ij} \times Fe_{ij})$, armazena o controle de quatro *barrel shifter*s usados nesta arquitetura, indicando qual dos deslocamentos devem ser executados em cada *barrel shifter* para cada elemento calculado.

O bloco da quantização foi projetado utilizando-se um pipeline com 4 estágios e uma latência de 4 ciclos de relógio.

3.3. Arquitetura do Buffer Ziguezague

A arquitetura desenvolvida para o Buffer Ziguezague é similar à arquitetura do buffer de transposição usada na arquitetura da DCT 2-D, consistindo de duas memórias RAM, as quais são intercaladas nas operações de leitura e escrita.

O Buffer Ziguezague recebe como entrada de dados os coeficientes da DCT 2-D quantizados (os quais estão dispostos em colunas por colunas) e fornece os dados para o codificador de entropia respeitando um critério de ordenamento em ziguezague. Esta operação é efetivada com o objetivo de maximizar a eficiência das operações do codificador de entropia. A latência do Buffer Ziguezague é de 66 ciclos de relógio.

3.4. Arquitetura do Codificador de Entropia

O último passo para efetuar a compressão JPEG está no codificador de entropia, cujo diagrama de blocos está apresentado na Fig. 5. Dessa maneira, o módulo arquitetural de codificação de entropia recebe os coeficientes da DCT 2-D que passaram pela quantização e fornece como saída as palavras JPEG comprimidas e montadas.

O Codificador de Entropia utiliza a codificação diferencial, a codificação por número de ocorrências (*Run Length Encoding* - RLE), a codificação de comprimento de palavra variável (*Variable Length Coding* - VLC) e a codificação de Huffman [2,5,6] para efetivar uma redução real no número de bits usados para representar a imagem após a compressão JPEG.

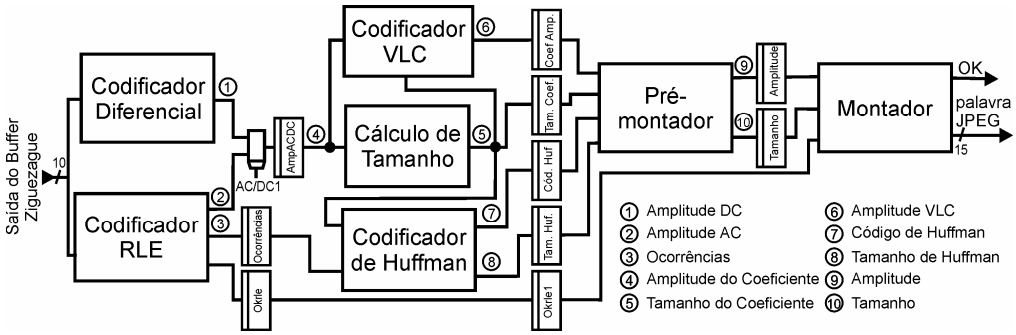

Figura 5 – Arquitetura genérica para o Codificador de Entropia

A matriz de entrada do Codificador de Entropia possui dois tipos de componentes. O componente na posição <0,0> (primeira linha e primeira coluna) é chamado de componente DC e todas os outros são chamados de componentes AC. O codificador de entropia processa os componentes DC e AC de maneira diferente. O projeto da arquitetura do codificador de entropia deste artigo é genericamente apresentado na Fig. 5.

A saída do Codificador de Entropia consiste de palavras JPEG de 32 bits, as quais são geradas de maneira assíncrona. O comportamento assíncrono inerente ao codificador é explicado nas próximas seções. Assim, a arquitetura apresentada na Fig. 5 consome um valor de entrada em cada ciclo de relógio e opera em um pipeline de quatro estágios.

3.4.1. Codificador Diferencial

O codificador diferencial é usado apenas para os componentes DC e realiza uma simples subtração entre o componente DC da matriz atual e o componente DC ma matriz anterior. O resultado gerado por esta operação é chamado de *Amplitude DC* nesse artigo.

3.4.2. Codificador RLE

A codificação por número de ocorrências (*Run Length Encoding – RLE*) é realizada somente nos componentes AC da entrada e é simplificada para na compressão JPEG, operando como um contador de zeros [6]. A arquitetura do codificador RLE é mostrada na Fig. 6.

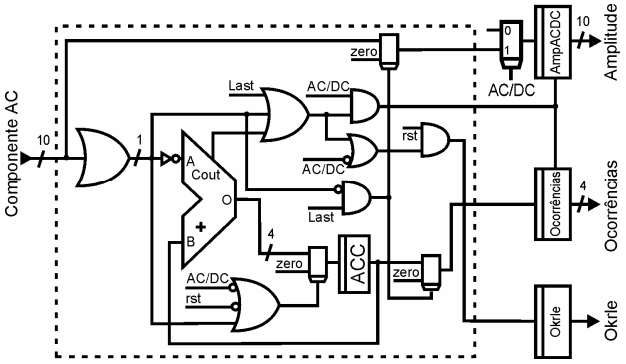

Figura 6 – Arquitetura do Codificador RLE

O codificador RLE é assíncrono. Enquanto o codificador conta a ocorrência de zeros, as saídas são congeladas. Quando uma entrada não nula é encontrada essas saídas são atualizadas com um novo par *"Ocorrências / Amplitude AC"*. A arquitetura do RLE tem um *flag* ("*Okrle*" na Fig. 6) para indicar que uma nova saída válida está disponível. Há duas restrições no codificador RLE que são impostas pelo padrão JPEG. A primeira é que o padrão JPEG define que o máximo valor para o campo "*ocorrências*" então, quando há mais que 16 zeros em seqüência, o par de saída *"Ocorrências / Amplitude AC"* deverá ser o par especial 15/0 [1]. A segunda restrição acontece quando uma matriz de entrada termina com zeros; então, o par especial 0/0 deve ser escrito nas saídas. Essas duas restrições foram consideradas no projeto da arquitetura.

Quando o codificador RLE está operando normalmente (sem as duas restrições acima) um zero de entrada inicializa a operação do contador, desabilita a escrita da saída e coloca o *flag "Okrle"* em nível lógico baixo, para indicar que a saída não está pronta ainda. Uma entrada não nula reinicializa o contador e armazena esse valor não nulo e o número de zeros que o precedem na saída. Então o *flag "Okrle"* é *setado* para 1, indicando que essas são saídas válidas.

3.4.3. Cálculo da Amplitude

As amplitudes DC e AC passam pelo cálculo de tamanho, que indica quantos bits são significativos (sem contar o bit de sinal) entre os 10 bits do valor de *"AmpACDC"*. A arquitetura projetada para efetuar este cálculo foi desenvolvida com uma lógica combinacional similar àquela proposta em [8]. Esta arquitetura gera um campo denominado *"Tamanho de Coeficiente"* nesse artigo. Esse campo *"Tamanho de Coeficiente"* é utilizado para controlar a arquitetura VLC e a arquitetura do pré-montador, além de servir como uma entrada para o codificador de Huffman.

3.4.4 Codificador VLC

O codificador de tamanho de palavra variável (*Variable Length Coder* - VLC) é utilizado para identificar apenas quais são os bits significativos na entrada "*AmpACDC"* e para descartar os bits não significativos, incluindo o bit de sinal. Os números negativos devem ser representados em complemento de um para serem codificados por VLC. Então, o bit de sinal é descartado também e a interpretação do sinal deve ser invertida: um número que começa com zero é negativo e um número que os começa com um é positivo.

Neste artigo, a arquitetura do codificador de VLC apenas identifica os bits significativos e prepara estes bits para serem montados pela arquitetura do montador.

A arquitetura do codificador VLC é um *barrel shifter* controlado pelo campo *"Tamanho de Coeficiente"*. Os valores das amplitudes AC e DC passam por este *barrel shifter,* o qual efetua o deslocamento para a esquerda do valor de *"AmpACDC"*, o que coloca o seu primeiro bit significativo (sem o bit de sinal) ocupando o bit mais significativo da palavra. A saída do codificador VLC é denominada *"Amplitude VLC"* neste artigo.

3.4.5. Codificador de Huffman

O campo "*Tamanho do Coeficiente"* (para os coeficientes DC) e a concatenação entre o campo "*Tamanho do*

Coeficiente" e o campo "*Ocorrências"* (para os coeficientes AC) são codificados por Huffman. Esse artigo usa as tabelas de Huffman estáticas propostas no padrão JPEG [1].

A utilização das tabelas propostas pelo padrão simplifica a arquitetura, utilizando menos hardware e obtendo um alto desempenho, mas diminuindo a taxa de compressão [6]. A Fig. 7 mostra a arquitetura do codificador de Huffman. Essa arquitetura usa apenas duas memórias ROM para armazenar as duas tabelas de Huffman utilizadas na codificação de entropia das imagens em tons de cinza: uma para os componentes DC e outra para os componentes AC.

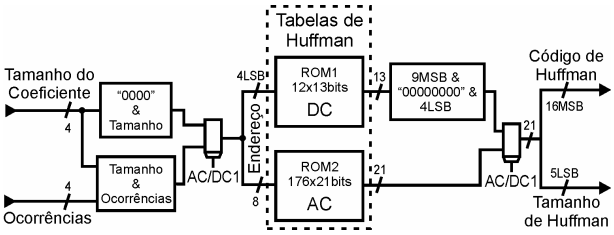

Figura 7 – Arquitetura do Codificador de Huffman

As tabelas de Huffman foram projetadas usando as memórias internas do FPGA. Os valores que serão codificados por Huffman são utilizados como endereços para essas memórias. Os códigos de Huffman e o tamanho desses códigos são armazenados nesses blocos de RAM. Então, dois campos formam a saída da memória: *"Código de Huffman"* e *"Tamanho de Huffman"*. O campo *"Tamanho de Huffman"* é usado para controlar a montagem de cada código de Huffman pela arquitetura do pré-montador.

3.4.6. Pré-Montador

A arquitetura do pré-montador, apresentada na Fig. 8, recebe quatro valores na entrada: "*Amplitude VLC"*, "*Tamanho do Coeficiente"*, "*Código de Huffman"* e "*Tamanho de Huffman"*. Duas saídas são geradas para serem utilizadas na arquitetura do montador: "*Amplitude"* e "*Tamanho"*.

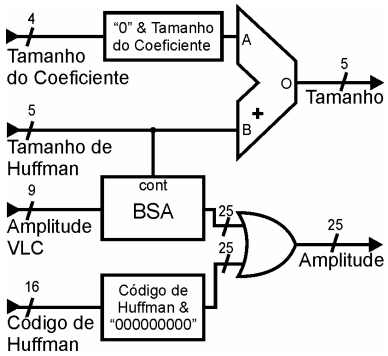

Figura 8 – Arquitetura do Pré-Montador

O pré-montador monta o "*Código de Huffman"* com a "*Amplitude VLC"* e gera a saída "*Amplitude"*. O número de bits mais significativos do campo "*Amplitude"* é dado pela soma do "*Tamanho de Huffman"* com o "*Tamanho do Coeficiente"*. Esta soma gera a saída "*Tamanho"*.

3.4.7. Montador

A arquitetura apresentada na Fig. 9 monta as palavras JPEG considerando apenas os bits mais significativos da entrada *"Amplitude"*. O processo de montagem usa um *barrel shifter* controlado pela entrada *"Tamanho"* e uma operação lógica OU para formar os bits mais significativos de diferentes entradas. Um somador que acumula os tamanhos das amplitudes da entrada controla a montagem da palavra. A arquitetura do montador descarta os bits menos significativos da codificação VLC, gerando um código real de comprimento variável. Essa arquitetura foi baseada na arquitetura proposta em [9].

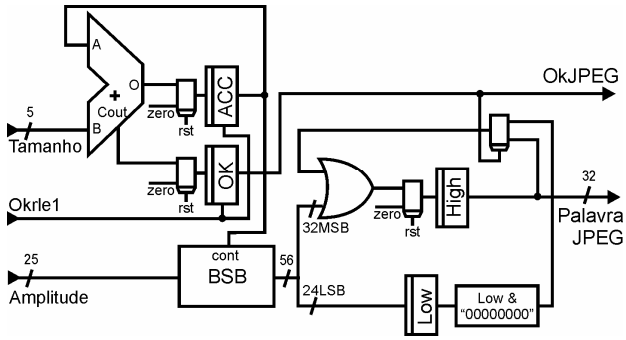

Figura 9 – Arquitetura do Montador

O montador usa dois registradores para montar as palavras JPEG. O registrador *"High"* armazena os primeiros 32 bits da saída e quando os 32 bits estão prontos, o novo valor válido é disponibilizado na saída. O registrador *"Low"* é usado para armazenar o *overflow* quando o valor gerado possuir mais que 32 bits. Este *overflow* é armazenado no registrador *"High"* quando uma nova palavra JPEG começa a ser montada. A operação de montagem é habilitada apenas quando o codificador RLE gera saídas válidas. O flag *"OK"* indica que uma palavra JPEG válida está pronta.

4. RESULTADOS DE SÍNTESE

O objetivo desse trabalho foi analisar o desempenho da arquitetura do compressor JPEG para imagens em tons de cinza, proposta e implementada em [2], com somadores MFA (*Macro Function Adder*) da biblioteca aritmética da IEEE [11]. A partir dos resultados obtidos para a implementação com somadores RCA (*Ripple Carry)* [2] e para diversos tipos de somadores misturados (heterogêneos

- HetA) [3], é possível verificar diferenças significativas e importantes com relação ao desempenho (em freqüência de operação), número de células lógicas (área física do FPGA) e custo de implementação do código.

A solução heterogênea (HetA) foi uma tentativa de alcançar o maior desempenho possível e minimizar a utilização de recursos. Onde existiam somadores com atrasos mais críticos, esses somadores foram substituídos por somadores mais rápidos e com mais consumo de recurso, mas, onde os atrasos dos somadores não eram tão relevantes, somadores mais lentos e com menor consumo de recurso foram utilizados. Deste modo, foi possível atingir uma freqüência de operação equivalente àquela atingida pela solução que utilizou os somadores mais rápidos, mas com um impacto reduzido no consumo de recursos.

A tabela 1 mostra um resumo dos resultados de síntese para a arquitetura completa do compressor JPEG, para 3 variações nos tipos de somadores utilizados, simulados no software Quartus II da Altera [12] para os FPGAs da família Flex10KE [10].

Tabela 1 – Comparação entre implementações do compressor JPEG com diferentes tipos de somadores

JPEG	Células Lógicas	Bits de Memória	Período (ns)	Latência (ciclos)
RCA	4521	7436	31,6	237
HetA	4700	7436	25,1	238
MFA	4612	7436	22,2	238

A síntese do compressor JPEG utilizando somadores MFA usou 4,612 células lógicas e 7,436 bits de memória do dispositivo EPF10K130EQC240-1 [10]. O período mínimo alcançado com essa arquitetura foi de 22,2 ns processando imagens com uma freqüência de até 45 MHz permitindo que uma imagem de 640 x 480 pixels seja completamente processada em 6,8 ms. O compressor JPEG para imagens em tons de cinza pode alcançar uma taxa de processamento de 147 imagens por segundo, a 640 x 480 pixels por frame.

Esses resultados são expressivos quando comparados às arquiteturas anteriores desenvolvidas com outros tipos de somadores. Em primeiro lugar é preciso considerar a facilidade de descrição do código com o MFA, que só precisa ser instanciada com um sinal de soma (+), o que já tornaria seu uso bem mais prático que os demais somadores. O desempenho alcançado acentuou ainda mais as vantagens do uso de tais somadores. Embora o número de células lógicas tenha tido um resultado um pouco superior que os somadores RCA, a relação global *desempenho/custo de hardware* foi bem melhor para a implementação homogênea com somadores MFA.

A parte mais crítica para o desempenho do compressor JPEG ocorre na etapa do cálculo da DCT-2D [2], devido a grande complexidade e quantidade de cálculos. Foi nessa parte do projeto em que foi realizada a maioria dos testes com somadores apresentados em [3] para otimizar o processamento dos cálculos. Assim, nesse trabalho, é apresentada uma análise comparativa específica para a DCT-2D. A etapa de síntese utilizou o software Quartus II e foi direcionada a um FPGA da família APEX20KE da Altera [12], mais especificamente, o dispositivo EP20K160ETC144-1. Este dispositivo foi utilizado para permitir a comparação com trabalhos anteriores [3] que utilizaram este dispositivo. Os resultados de síntese para as implementações da arquitetura da DCT-2D são apresentadas na tabela 2.

Tabela 2 – Comparação entre implementações da DCT-2D com diferentes tipos de somadores

Somador	Células Lógicas	Freqüência (MHz)	Acréscimo Freqüência	Acréscimo Recursos
RCA	3.664	45,57	-	-
CLA	3.777	49,24	8,05%	3,08%
CLAH	3.828	54,47	19,53%	4,48%
CSA	3.979	57,81	26,86%	8,6%
HetA	3.856	57,19	25,5%	5,24%
MFA	3.596	55,45	21,68%	-1,86%

Observando-se a tabela 2, constatam-se os seguintes resultados para a utilização de cada tipo de arquitetura nos somadores da DCT 2-D. Começando-se com a arquitetura RCA foram necessárias 3664 células lógicas, obtendo-se uma freqüência máxima de operação de 45,57 MHz.

Para a arquitetura CLA (*Carry Lookahead*) foram necessárias 3777 células lógicas, atingindo-se uma freqüência máxima de operação de 49,24 MHz. No caso da arquitetura CLAH (*Carry Lookahead* Hierárquico), foram necessárias 3828 células lógicas, atingindo-se uma freqüência máxima de operação de 54,47 MHz. Usando-se a arquitetura CSA (*Carry Select*) foram necessárias 3979 células lógicas, obtendo-se uma freqüência máxima de operação de 57,81 MHz. Para a arquitetura em que foi efetivado um ajuste fino com somadores mistos ou heterogêneos (HetA) foram necessárias 3856 células lógicas, obtendo-se uma freqüência máxima de operação de 57,19 Mhz.

Quanto ao aumento em desempenho, a arquitetura CLA apresentou uma freqüência máxima de operação 8,05% maior que a freqüência máxima da arquitetura RCA. No caso do CLA hierárquico o incremento foi de 19,53%. Para a arquitetura CSA, 26,86 %. E no caso da HetA, o aumento em desempenho foi de 25,5 %. Os incrementos em área ocupada foram de 3,08% para a arquitetura CLA,

4,48 % para a CLAH, 8,6% para a arquitetura CSA e 5,24% para a DCT 2-D com ajuste fino (HetA).

Para a arquitetura com somadores MFA foram necessárias 3596 células lógicas obtendo-se uma freqüência máxima de operação de 55,45 MHz , proposta nesse artigo. Houve um aumento de 21,68% no desempenho e uma diminuição de 1,86% no uso de recursos do FPGA com relação à implementação original RCA [2]. Esses resultados tiveram o melhor aproveitamento de espaço com relação aos dados do estudo referencial anterior [3], e um desempenho de processamento (freqüência de operação) similar aos mais elevados desempenhos das demais soluções. Dessa forma, o MFA representou a melhor relação *desempenho/custo de hardware*, validando a hipótese que motivou este trabalho.

As possíveis melhoras com a utilização desse tipo de somadores se devem, resumidamente, ao fato de que o somador MFA utiliza cadeias rápidas de propagação de *carry* do FPGA alvo, tratando as somas utilizando recursos em hardware que aceleram seus resultados, sem aumentar a quantidade de células lógicas necessárias para sua implementação. [4].

5. CONCLUSÕES

Esse artigo apresentou um resumo do projeto do compressor JPEG em hardware e os resultados de síntese para três implementações distintas, variando o tipo dos somadores, para a família de FPGAs Flex10KE da Altera. O impacto das otimizações da arquitetura do compressor resultou em uma latência mínima de 238 ciclos de relógio e o processamento de uma imagem de 640x480 pixels em 6,8ms, permitindo uma taxa de processamento de até 147 imagens por segundo, 4,612 células lógicas e 4,436 bits de memória do dispositivo EPF10K130EQC240-1 foram usados para mapear este projeto da arquitetura global do compressor. Além disso, também foi realizada uma análise do módulo da DCT-2D, que é a parte mais crítica do projeto. A análise mostrou que o somador MFA obteve sempre a melhor relação desempenho/custo de hardware, embora o MFA perca um pouco em desempenho para somadores CSA e HetA.

A principal contribuição desse trabalho é mostrar que realmente é válido e relevante optar pela utilização de somadores *Macro Function*, quando o projeto é direcionado para FPGAs. O somador MFA está pré-definido em uma biblioteca de funções da ferramenta de síntese de VHDL do fabricante de FPGA (neste caso, a Altera).

Os resultados obtidos foram satisfatórios, considerando o estímulo ao trabalho que foi encontrar vantagens na utilização dos somadores mais simples de serem utilizados na descrição do código. Esse trabalho demonstrou que é possível diminuir o tempo de descrição do código com a utilização do somador MFA que apresenta ainda as vantagens de desempenho e custo no projeto de hardware.

7. REFERÊNCIAS

[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] L. Agostini. Projeto de Arquiteturas Integradas para a Compressão de Imagens JPEG. Dissertação de Mestrado – Universidade Federal do Rio Grande do Sul. Instituto de Informática. Programa de Pós-Graduação em Ciência da Computação, Porto Alegre, Brazil-RS, 2002. 143p. http://www.ufpel.tche.br/%7Eagostini/pesquisa/AgostiniJPEG.pdf

[3] R. Porto, et all. "Experimentos com Somadores Rápidos para Uso na DCT 2-D". X Workshop IBERCHIP, 2004, Cartegena de Indias. Disponível em: <http://www.iberchip.org/iberchip2004/articles/117-1-AGOSTINI-ROGERIBERCHIP.PDF>

[4] M. Porto, et all. "Impactos do Uso de Diferentes Arquiteturas de Somadores em FPGAS Altera". XI Workshop IBERCHIP, 2005, Salvador. Disponível em: <http://www.iberchip.org/iberchip2005/articles/98/98--porto-Somadores_Completo.pdf>

[5] JPEG2000 Committee Drafts. JPEG2000 Final Committee Draft Version 1.0. http://www.jpeg.org/public/fcd15444-1.pdf

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

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

[8]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.

[9] S. Lei; M. Sun. “An Entropy Coding System for Digital HDTV Applications. *IEEE Transactions on Circuits and Systems for Video Technology*. 1991, pp. 147-155.

[10] Altera Corporation. “FLEX 10KE – Embedded Programmable Logic Devices Data Sheet – versão 2.3”. San Jose, Altera Corporation, 2001. 1 CD.

[11] Brown, S., *Fundamentals of Digital Logic With VHDL Design.* McGraw-Hill, EUA, 2002.

[12] Altera Corporation, “Altera: The Programmable Solutions Company”, <http://www.altera.com>, 2005.