UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Projeto de Arquiteturas Integradas para a Compressão de Imagens JPEG

por

LUCIANO VOLCAN AGOSTINI

Dissertação submetida à avaliação,
como requisito parcial para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Sergio Bampi
Orientador

Porto Alegre, março de 2002.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Agostini, Luciano Volcan
Projeto de Arquiteturas Integradas para a Compressão de Imagens JPEG / Luciano Volcan Agostini – Porto Alegre: PPGC da UFRGS, 2002.
143p.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR – RS, 2002. Orientador: Bampi, Sergio.

1. Arquiteturas para compressão JPEG, 2. Compressão JPEG, 3. Compressão de imagens, 4. Arquiteturas para a compressão de imagens. I. Bampi, Sergio. II. Título.

Agradecimentos

Ao povo brasileiro, que financiou toda a minha formação, desde a primeira série até este mestrado. Este povo sofrido e oprimido, mas inundado de inesgotável esperança. A estes pretendo retribuir o investimento em mim realizado, não com este trabalho, mas com a vida dedicada a contribuir para que o Brasil seja um lugar melhor para se viver.

A minha amada, companheira e amiga Cristhianny Barreiro, com quem tenho tido o prazer de compartilhar a minha vida nos últimos nove anos. Devo a ela, além de meus lindos filhos, a certeza que tenho de que amar vale a pena e que lutar por aquilo que amamos e cremos é mais do que uma escolha, é uma necessidade, sem a qual nada mais faz sentido.

Aos meus filhos, Lucas e Isabela Agostini, que suportaram a minha ausência e o meu cansaço neste período de mestrado. Suportaram, compreenderam e, com seu amor e carinho, me deram a necessária energia para concluir mais esta etapa de nossas vidas. Graças a sua existência e ao seu pleno existir eu sou capaz de sonhar e de lutar por uma vida melhor.

A meus pais, Accelino e Neusa Agostini, aos quais devo minha vida, minha formação e grande parte de meus princípios. Com o olhar fixo nos seus exemplos tenho a certeza de que é possível lutar por toda uma vida pelo que se acredita, sem, no entanto, perder a alegria, o prazer ou a esperança.

A meu orientador, Sergio Bampi, que foi incumbido da árdua tarefa de me iniciar e me conduzir neste estimulante mundo acadêmico. Nele encontrei um grande intelectual, um grande orientador (no sentido mais amplo quanto possível para a palavra) e, acima de tudo, um grande amigo.

Ao bom amigo Ivan Saraiva Silva, com quem tive o prazer de compartilhar a vida neste último ano. Nas longas e estimulantes discussões, que iam do futebol à política, passando pela microeletrônica, teve importante papel no meu amadurecimento enquanto ser humano crítico e ator da construção de uma nova realidade.

Aos professores e pesquisadores do Grupo de Microeletrônica da UFRGS, que me acolheram e guiaram durante o mestrado, com destaque aos professores Altamiro Susin, Luigi Carro e Ricardo Reis, que com seu engajamento, sua inteligência, seu dinamismo e seu companheirismo, fizeram com que eu produzisse muito mais do que acreditava ser capaz.

Aos colegas do Grupo de Microeletrônica da UFRGS, que compartilharam bons e maus momentos ao meu lado neste período de mestrado e que muito contribuíram para a conclusão deste trabalho. Dentre todos, destaco Ana Pinto, Gaspar Stemmer e Flávio Zimmermman, que além de bons colegas, tornaram-se grandes amigos.

Ao companheiro Rogério Santanna, presidente da Companhia de Processamento de Dados do Município de Porto Alegre – PROCEMPA, que incentivou e apoiou o desenvolvimento deste trabalho.

A Hilda Xavier Volcan, minha avó que, com seu exemplo, ensinou-me o valor da humanidade, da empatia, do respeito, do carinho e da tolerância e que, quando se foi, deixou enorme e irreparável saudade.

A Oraídes Martinez, minha sogra e minha avó adotiva, uma voz amiga, carinhosa e experiente, cujas palavras, sempre de incentivo, tiveram grande relevância nas decisões que tomei neste período que tive o privilégio de, com ela, conviver.

Aos amigos Rodrigo Almeida, Romi Lamb, Vander Volcan, Jocelaine Volcan, Bruno Volcan, Marcos Grandi, Kelvin Reinhardt, Adriane Cardozo, entre tantos outros, que, pelo simples fato de existirem, contribuem com a minha existência.

Aos amigos Rogério Volcan, Adriane Niederauer, Frank Volcan e Pedro Alberto Lokschin que, além de existirem, me acolheram com carinho em suas casas em vários momentos desta caminhada.

Aos amigos cultivados no último ano junto à FAPERGS, em especial a Ivone Cassol, Geni Chaves, Nádia Gaio e Marilene Guidini.

A todas as pessoas que, de alguma forma, contribuíram para a minha formação e que, portanto, indiretamente, viabilizaram a execução deste trabalho.

Dedico este trabalho
aos meus três grandes amores:
Cristhianny, Lucas e Isabela.

O Fim

Quando penso no fim
penso no infinito
o fim é uma projeção
uma ficção
uma invenção

Não há fim
como não há começo
há apenas o contínuo
e o interminável

Não temos dimensão
não cabemos no tempo
mesmo no espaço

Somos muito mais
somos infinitos
fontes intermináveis
de sonhos, de sentimentos
de idéias, de desalentos

Nunca penso no fim
penso, sempre, no novo
e irrefutável começo
que existe depois de todo
suposto e equivocado fim.

Luciano Agostini
fevereiro de 2002

Sumário

Lista de Abreviaturas

AC	*Alternating Current*
APP_X	*Application specific marker*
ASCII	*American Standard Code for Information Interchange*
ASIC	*Application Specific Integrated Circuit*
BMP	*Bit Map*
COM	*Comment marker*
DAC	*Define Arithmetic Coding marker*
DC	*Direct Current*
DCT	*Discrete Cosine Transform*
DCT 1-D	*Discrete Cosine Transform in One Dimension*
DCT 2-D	*Discrete Cosine Transform in Two Dimensions*
DHP	*Define Hierarchical Progression marker*
DHT	*Define Huffman Table marker*
DNL	*Define Number of Lines marker*
DQT	*Define Quantization Table marker*
DRI	*Define Restart Interval marker*
DSP	*Digital Signal Processor*
DVD	*Digital Versatile Disk*
EOB	*End Of Block marker*
EOI	*End Of Image marker*
EXP	*Expand component marker*
FPGA	*Field Programmable Gate Array*
GIF	*Graphics Interchange Format*
IP	*Intellectual Property*
JFIF	*JPEG File Interchange Format*
JPEG	*Joint Photographic Experts Group*
JPG_X	*JPEG reserved marker*
LSB	*Less Significant Bit*
MCU	*Minimum Coded Unit*
MSB	*More Significant Bit*
PAL	*Phase Alternating Line*
PC	*Personal Computer*
PCI	*Peripheral Component Interconnect*
PNG	*Portable Network Graphics*
RAM	*Random Access Memory*
RES	*Reserved marker*
RLE	*Run Length Encoder*
ROM	*Read Only Memory*
RST_X	*Restart marker*
SOF_X	*Start Of Frame marker*
SOI	*Start Of Image marker*
SOS	*Start Of Scan marker*
TEM	*Temporary marker*
VHDL	*VHSIC Hardware Description Language*
VLC	*Variable Length Coder*
ZRL	*Zero Run Length marker*

Lista de Figuras

Lista de Tabelas

Resumo

Esta dissertação apresenta o desenvolvimento de arquiteturas para a compressão JPEG, onde são apresentadas arquiteturas de um compressor JPEG para imagens em tons de cinza, de um compressor JPEG para imagens coloridas e de um conversor de espaço de cores de RGB para YCbCr. As arquiteturas desenvolvidas são detalhadamente apresentadas, tendo sido completamente descritas em VHDL, com sua síntese direcionada para FPGAs da família Flex10KE da Altera.

A arquitetura integrada do compressor JPEG para imagens em tons de cinza possui uma latência mínima de 237 ciclos de *clock* e processa uma imagem de 640x480 *pixels* em 18,5ms, permitindo uma taxa de processamento de 54 imagens por segundo. As estimativas realizadas em torno da taxa de compressão obtida indicam que ela seria de aproximadamente 6,2 vezes ou de 84 %.

A arquitetura integrada do compressor JPEG para imagens coloridas foi gerada a partir de adaptações na arquitetura do compressor para imagens em tons de cinza. Esta arquitetura também possui a latência mínima de 237 ciclos de *clock*, sendo capaz de processar uma imagem coloria de 640 x 480 *pixels* em 54,4ms, permitindo uma taxa de processamento de 18,4 imagens por segundo. A taxa de compressão obtida, segundo estimativas, seria de aproximadamente 14,4 vezes ou de 93 %.

A arquitetura para o conversor de espaço de cores de RBG para YCbCr possui uma latência de 6 ciclos de *clock* e é capaz de processar uma imagem colorida de 640x480 *pixels* em 84,6ms, o que permite uma taxa de processamento de 11,8 imagens por segundo. Esta arquitetura não chegou a ser integrada com a arquitetura do compressor de imagens coloridas, mas algumas sugestões e estimativas foram realizadas nesta direção.

Palavras-Chave: compressão de imagens, compressão JPEG, arquiteturas para compressão de imagens, arquiteturas para compressão JPEG.

TITLE: “DESIGN OF ARCHITECTURES FOR JPEG IMAGE COMPRESSION”

Abstract

This dissertation presents the design of architectures for JPEG image compression. Architectures for a gray scale images JPEG compressor that were developed are herein presented. This work also addresses a color images JPEG compressor and a color space converter. The designed architectures are described in detail and they were completely described in VHDL, with synthesis directed for Altera Flex10KE family of FPGAs.

The integrated architecture for gray scale images JPEG compressor has a minimum latency of 237 *clock* cycles and it processes an image of 640x480 *pixels* in 18,5ms, allowing a processing rate of 54 images per second. The compression rate, according to estimates, would be of 6,2 times or 84%, in percentage of bits compression.

The integrated architecture for color images JPEG compression was generated starting from incremental changes in the architecture of gray scale images compressor. This architecture also has the minimum latency of 237 *clock* cycles and it can process a color image of 640 x 480 *pixels* in 54,4ms, allowing a processing rate of 18,4 images per second. The compression rate, according to estimates, would be of 14,4 times or 93%, in percentage of bits compression.

The architecture for space color conversor from RBG to YCbCr has a latency of 6 *clock* cycles and it is able to process a color image of 640 x 480 *pixels* in 84,6ms, allowing a processing rate of 11,8 images per second. This architecture was finally not integrated with the color images compressor architecture, but some suggestions, alternatives and estimates were made in this direction.

Keywords: image compression, JPEG compression, architectures for image compression, architectures for JPEG compression.

1 Introdução

Nos dias de hoje existe, no mundo, um movimento radical na direção da digitalização de informações. Este movimento está transformando o comportamento das pessoas e, deste modo, está gerando um significativo impacto na sociedade. Estas transformações se fazem sentir já há alguns anos e não é possível prever quando ou como irão estabilizar, nem quais serão as reais conseqüências deste processo para a civilização humana.

Os mais pessimistas acreditam que este movimento é excludente e discriminatório, pois a maior parte do planeta não pode, nem poderá, usufruir dos benefícios gerados pelo uso de tecnologia digital e, então, estaria sendo criada uma nova e irreversível forma de exclusão, a chamada "exclusão digital". Por outro lado, os mais otimistas acreditam que, em um curto espaço de tempo, toda a população do planeta estará conectada à internet e terá acesso aos benefícios gerados pela tecnologia digital.

Otimismos e pessimismos a parte, a tendência à digitalização global é notória. Não bastasse a explosão no número de usuários da internet, a cada dia que passa fervilham novos produtos com suporte à informação digital: computadores, celulares, televisões, máquinas fotográficas, filmadoras, DVDs, e tantos outros.

A viabilidade de grande parte destes produtos e das aplicações para as quais eles foram desenvolvidos passa, essencialmente, pelo tempo de processamento ou de transmissão dos dados que estão sendo manipulados. Por isso, muitas aplicações não são viáveis com o uso de processadores comerciais ou DSPs com software embutido, sendo necessária a migração dos algoritmos para hardware. O advento de lógica programável e de ferramentas de síntese contribuiu significativamente para esta migração, diminuindo o tempo de projeto e aumentando a sua flexibilidade.

As aplicações que manipulam imagens estáticas ou em movimento (vídeo) são das mais críticas em termos de desempenho, exigindo muito processamento, pois são as que contêm mais dados a serem manipulados. Então, cada vez mais, estas aplicações têm sido desenvolvidas em hardware, viabilizando produtos como filmadoras digitais, DVDs, máquinas fotográficas digitais, televisão digital, entre outras. Há um grande interesse da indústria no desenvolvimento de arquiteturas para as mais diversas operações sobre imagens e vídeos, tanto que, nesta área, são poucas as publicações científicas que apresentam soluções completas e com o aprofundamento necessário para o desenvolvimento ou aperfeiçoamento do que é publicado. Na maioria dos casos, as publicações são superficiais ou omitem informações essenciais.

Uma área muito importante para a viabilização de aplicações que processam vídeo ou imagens é a compressão de dados. A compressão dos dados de imagens ou vídeo permite que sejam otimizados recursos de transmissão e armazenamento, uma vez que os dados do vídeo ou da imagem irão utilizar um número menor de bits após a compressão.

O interesse desta dissertação é justamente a compressão de imagens fotográficas digitais, mais especificamente, a compressão conhecida como JPEG. A compressão JPEG foi padronizada pelo *Joint Photographic Experts Group* [JPE 2001] e tem base na utilização da Transformada Discreta do Coseno (DCT), que transforma a informação do domínio espacial para o domínio das freqüências. Neste domínio, a compressão JPEG descarta as freqüências para as quais o olho humano é menos sensível.

O padrão JPEG foi desenvolvido para comprimir imagens fotográficas, por isso, tem um desempenho muito melhor para este tipo de imagem do que para imagens de

desenhos. Uma imagem fotográfica de 1MB no formato BMP não comprimido pode chegar a menos que 50KB com o uso da compressão JPEG [MIA 99].

O padrão JPEG é o foco deste trabalho porque a principal aplicação que o motivou foi a monitoração de trânsito, onde a imagem a ser comprimida é uma fotografia do carro infrator ou do estado atual da via (engarrafada, trânsito livre, trânsito lento, etc.). O sistema de monitoração de trânsito tomado como referência é o que está em operação em Porto Alegre, capital do Rio Grande do Sul. Este sistema não utiliza a compressão de imagens, possuindo um custo de armazenamento e transporte das imagens adquiridas muito elevado, uma vez que são utilizados discos rígidos para este fim. Com o uso da compressão, este sistema teria a sua capacidade de armazenamento incrementada significativamente, o que reduziria, também, o custo da operação de transporte das imagens, que é manual, pois um mesmo disco rígido seria capaz de armazenar um número muito maior de imagens. O custo gerado pelos defeitos nos discos rígidos também seria reduzido, uma vez que a sua manipulação seria menor.

O uso de compressão de imagens também viabilizaria uma futura conexão em rede de todos os monitores com a central, na qual as imagens seriam transmitidas via rede e não seriam mais necessárias as operações de transporte manual dos discos rígidos.

As imagens geradas pelos atuais monitores de trânsito são tomadas como referência para o trabalho desenvolvido nesta dissertação. Os monitores atuais geram imagens de 640 x 480 *pixels* em tons de cinza, como a imagem apresentada na fig. 1.1, que é uma imagem real gerada por um monitor de trânsito. Na fig. 1.1 os dígitos da placa foram, propositadamente, apagados para evitar, neste texto, a identificação do veículo infrator.

FIGURA 1.1 – Exemplo de imagem gerada por monitores de trânsito de Porto Alegre

Todos os exemplos citados na dissertação que tratam de imagens completas, irão utilizar imagens de 640 x 480 *pixels*, tal qual as imagens geradas por estes monitores.

O foco principal desta dissertação é o desenvolvimento de uma arquitetura de compressor JPEG para imagens em tons de cinza, uma vez que a aplicação motriz deste trabalho gera imagens não comprimidas em tons de cinza. Ainda assim, foi desenvolvido um compressor JPEG para imagens coloridas que também é apresentado nesta dissertação. O compressor para imagens em tons de cinza, por ser o foco principal, é explicado com elevado grau de detalhamento, ocupando a maior parte desta dissertação.

O texto desta dissertação está dividido em três capítulos principais. O capítulo 2 irá introduzir o padrão JPEG, com especial destaque para o modo de operação conhecido como *baseline*, que será implementado nas arquiteturas desenvolvidas.

O capítulo 3 apresenta a arquitetura desenvolvida para o compressor JPEG para imagens em tons de cinza, com seus quatro blocos principais: DCT 2-D, quantizador, *buffer* ziguezague e codificador de entropia. Como o compressor para imagens em tons de cinza é o foco principal desta dissertação, a arquitetura é detalhadamente descrita e são apresentados dados de síntese, estimativas de desempenho e resultados de simulações tanto para o compressor quanto para seus quatro blocos principais.

O capítulo 4 apresenta as arquiteturas desenvolvidas para o compressor JPEG para imagens coloridas, tendo sido consideradas duas situações principais, abordadas separadamente: uma que é formada pelos quatro blocos do compressor para imagens em tons de cinza, adaptados para o processamento de imagens coloridas e outra onde, além destes quatro blocos, é considerado um bloco adicional, o da conversão de espaço de cores. A arquitetura do compressor JPEG para imagens coloridas, bem como a arquitetura do conversor de espaço de cores, foram completamente desenvolvidas, sintetizadas e simuladas e os dados obtidos neste processo estão apresentados. A arquitetura do compressor para imagens coloridas não chegou a ser integrada com a arquitetura do conversor de espaço de cores, mas são apontados caminhos nesta direção e também são realizadas estimativas de uso de recursos e de desempenho para cada um dos caminhos propostos.

Por fim, o capítulo 5 apresenta as conclusões desta dissertação e sugere trabalhos futuros a serem realizados tomando como base o trabalho ora desenvolvido.

2 A Compressão de Imagem JPEG

O padrão JPEG foi definido pelo *Joint Photographic Experts Group* [JPE 2001] no ano de 1992 e rapidamente tornou-se a referência mais conhecida e usada para compressão de imagens fotográficas. O padrão JPEG é muito extenso e define vários modos de operação, utilizando várias técnicas de compressão [THE 92], que cobrem a compressão sem perdas e a compressão com perdas. A compressão JPEG com perdas é a mais utilizada sendo quase um sinônimo de compressão JPEG, pois é onde o padrão destaca-se por obter as maiores taxas de compressão dentre os padrões existentes.

Este capítulo pretende introduzir a compressão JPEG e seus conceitos, para utilizá-los como base para os dois próximos capítulos, que tratam do desenvolvimento arquitetural propriamente dito. O padrão JPEG é superficialmente apresentado neste capítulo, com ênfase nos conceitos e definições que são utilizados no decorrer do texto da dissertação. A base para a escrita deste capítulo foi uma revisão da literatura e do estado da arte, para a elaboração de uma síntese suficientemente completa para introduzir o problema que pretende ser resolvido e suficientemente sintética para não dispersar o objetivo principal da dissertação, que é o desenvolvimento de arquiteturas para a compressão JPEG.

O enfoque principal do capítulo será dado ao modo de operação conhecido como *baseline* e para os conceitos e as técnicas de compressão utilizados neste modo de operação. O modo *baseline* é definido como o modo que contém o menor grupo de requisitos para considerar uma compressão como compressão JPEG [PEN 92], sendo o modo mais simples e o que é mais utilizado na prática, tanto para aplicações em software como para aplicações em hardware [BHA 99].

O núcleo da compressão JPEG é o uso da Transformada Discreta do Coseno (DCT) que, em conjunto com técnicas de quantização e de compressão sem perdas, tornam possível uma significativa redução na quantidade de dados necessários para representar a imagem.

Existem perdas de informação no modo de compressão JPEG *baseline* e as perdas ocorrem, essencialmente, nas operações de *downsampling* e de quantização (que serão detalhadas nos itens 2.3.2 e 2.3.4 deste capítulo). Estas perdas podem ser controladas de modo a ter uma influência quase imperceptível ao olho humano.

O capítulo foi dividido em três partes principais. A primeira aborda alguns conceitos e definições básicas da compressão JPEG, que são necessários para a compreensão do padrão. A seguir são descritos, superficialmente, os modos de operação JPEG, com o objetivo de apresentar o padrão JPEG em toda a sua abrangência e de contextualizar o modo de operação *baseline*, foco principal desta dissertação. Por fim são apresentadas as operações realizadas pela compressão JPEG, quando operando no modo *baseline*, onde pretende-se introduzir, de maneira sólida e rápida, a completa operação de um compressor JPEG.

2.1 Conceitos e Definições Básicas do Padrão JPEG

Este item do capítulo irá tecer algumas definições básicas, a maioria extraída do próprio padrão JPEG, que serão utilizadas no decorrer dos demais itens do texto da dissertação e que auxiliam na compreensão da compressão JPEG.

Segundo o padrão JPEG, *scan* é uma passada pelos dados de um ou mais componentes de cor da imagem original [THE 92]. A compressão pode ocorrer com o uso de um ou mais *scans*, dependendo do modo de operação e das definições do compressor. Os compressores apresentados nos capítulos 3 e 4 realizam o processamento da imagem com uma única passada pelos dados da imagem de entrada, por isso, utilizam apenas um *scan*.

A imagem pode ser dividida em vários *frames*, onde um *frame* é uma coleção de um ou mais *scans* [MIA 99]. A divisão da imagem em mais de um *frame* não é utilizada no modo *baseline*, onde existe apenas um *frame*, que engloba o único *scan* da imagem.

Uma unidade de dados é a menor unidade lógica dos dados da imagem original, que pode ser processada em determinado modo de operação [PEN 92]. No caso dos modos de compressão com base na DCT, como o *baseline*, uma unidade de dados é um bloco de 8 x 8 amostras de um componente de cor da imagem. O tamanho de bloco é definido pelo cálculo da DCT, como será explicado no item 2.3.3 desta dissertação. Dentro da unidade de dados os *pixels* são ordenados da esquerda para a direita e do topo para a base. Se a imagem é colorida, cada janela de 8 x 8 *pixels* possui três unidades de dados, uma para cada componente de cor. Se a imagem está representada em tons de cinza, cada janela de 8 x 8 *pixels* é formada por uma única unidade de dados. A fig. 2.1 apresenta as três unidades de dados relativas a uma imagem colorida e a unidade de dados relativa a uma imagem em tons de cinza.

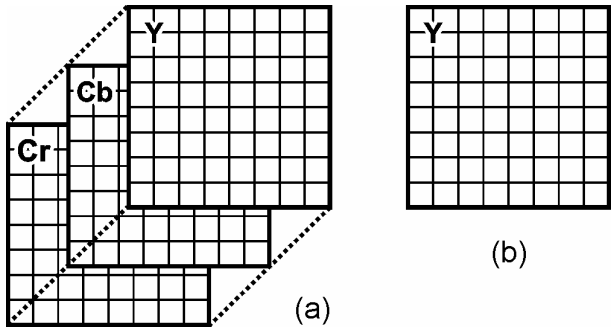


FIGURA 2.1 – Unidades de dados de imagens coloridas (a) e em tons de cinza (b)

Se a imagem de entrada for colorida, as unidades de dados, dentro do *scan*, podem estar dispostas de uma forma intercalada ou não. O intercalamento, neste caso, é referente a organização das unidades de dados dos componentes de cor. Se, em um único *scan*, são processadas unidades de dados relativas a vários componentes de cor, então diz-se que os componentes de cor estão intercalados [BHA 99]. Se, em um *scan*, são processadas unidades de dados relativas a um único componente de cor de uma imagem colorida, então, os componentes não estão intercalados e são necessários múltiplos *scans* para o completo processamento da imagem [BHA 99]. Para imagens em tons de cinza, como existe um único componente de cor, o conceito de intercalamento não se aplica. Nesta dissertação optou-se pelo uso do intercalamento para o processamento de imagens coloridas, uma vez que, desta forma, há uma maior eficiência no processamento e no armazenamento da imagem [BHA 99]. A fig. 2.2 apresenta dois exemplos de *scans*: um é codificado de maneira não intercalada e outro de maneira intercalada (considerando uma imagem de 32 x 16 *pixels* no espaço de cores YCbCr). Na fig. 2.2 (a) estão apresentadas oito unidades de dados de cada um dos componentes Y, Cb e Cr do *scan* intercalado, enquanto que na fig. 2.2 (b), estão apresentados os três *scans* referentes aos elementos Y, Cb e Cr, também com suas oito unidades de dados.

Para imagens intercaladas, as unidades de dados são agrupadas em unidades mínimas codificadas ou *minimum coded units* (MCU). Um MCU possui o menor conjunto possível de dados intercalados [BHA 99]. No exemplo da fig. 2.2 (a) estão dispostos oito MCUs. Para imagens em tons de cinza, um MCU contém uma única unidade de dados, por possuir apenas um componente de cor.

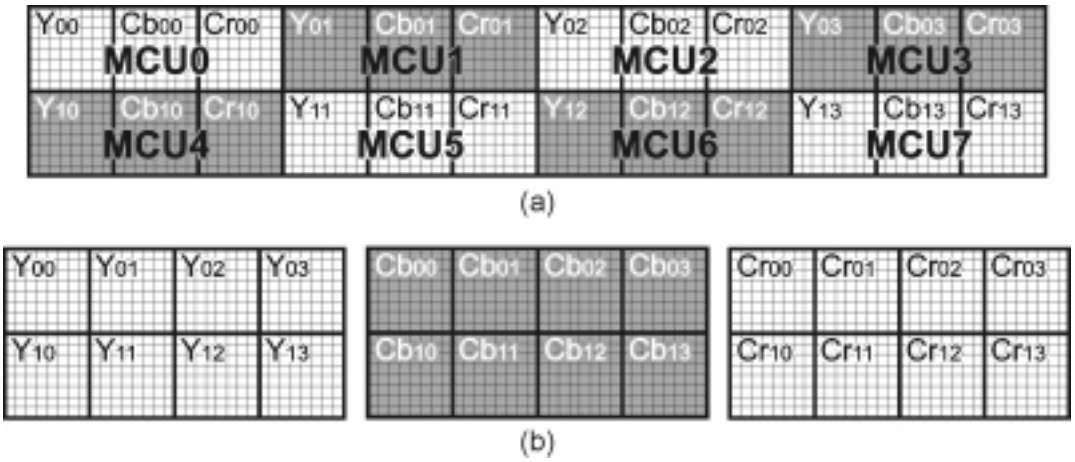


FIGURA 2.2 – Comparação entre um *scan* intercalado (a) e *scans* não intercalados (b)

No exemplo da fig. 2.2 é possível perceber que cada MCU possui três unidades de dados. No padrão JPEG, por definição, o número máximo de unidades de dados por MCU é igual a dez [BHA 99].

2.2 Os modos de Operação da Compressão JPEG

Existem quatro modos de operação descritos no padrão JPEG: seqüencial, progressivo, hierárquico e sem perdas [THE 92]. A fig. 2.3 apresenta estes modos de operação divididos em relação às diferentes técnicas de compressão utilizadas e em relação ao número de bits utilizados para representar cada informação de cor. A parte da fig. 2.3 que está em cinza é onde o modo *baseline* se enquadra.

Seqüencial				**Progressivo**				**Sem Perdas**		**Hierárquico**
Huffman		Aritmético		Huffman		Aritmético		Original	JPEG LS	
8bits	12bits	8bits	12bits	8bits	12bits	8bits	12bits			

FIGURA 2.3 – Modos de compressão do padrão JPEG

De todos os modos de operação apresentados na fig. 2.3, os mais utilizados são o modo seqüencial e o modo progressivo. São raras as aplicações que utilizam os modos sem perdas e o modo hierárquico [MIA 99].

O modo de compressão sem perdas é pouco utilizado porque atinge taxas de compressão menores que o de outros formatos sem perdas já desenvolvidos, como o GIF [COM 87] e o PNG [ROE 2002].

No modo progressivo, os componentes são codificados em múltiplos *scans* (mínimo de 2 e máximo de 896 [MIA 99]). O *scan* inicial cria uma versão da imagem com reduzido número de detalhes e os *scans* subsequentes vão refinando a imagem até que ela fique completa. Neste modo, a imagem vai sendo mostrada à medida que é

decodificada e isto é útil para aplicações em rede, em que o usuário pode ter uma boa idéia da imagem antes da chegada de todos os seus dados, como no exemplo apresentado na fig. 2.4. O modo progressivo suporta amostragem de 8 ou 12 bits para cada componente de cor e pode usar a codificação de entropia (que será discutida no item 2.3.5 desta dissertação) baseada na codificação de Huffman ou na codificação aritmética. A desvantagem deste modo está na maior complexidade em relação ao modo seqüencial, o que exige mais processamento.

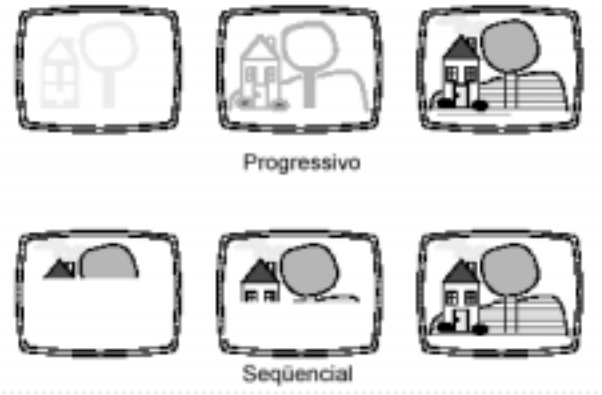


FIGURA 2.4 – Exemplo de modo de operação seqüencial e progressivo

O modo hierárquico é um modo super progressivo e divide a imagem em um grupo de *frames*. O modo hierárquico tem o mesmo objetivo do modo progressivo no sentido de criar versões da imagem com qualidade incremental. A diferença é que o modo hierárquico divide a imagem em mais um nível de hierarquia, em comparação ao modo progressivo, possibilitando uma progressividade com um maior número de passos. Por efetuar este grande número de divisões no processamento das imagens, o modo hierárquico acaba por ter uma complexidade muito elevada, que o faz ser pouco utilizado.

No modo seqüencial, as imagens são codificadas do topo para a base de forma contígua, como também pode ser observado no exemplo da fig. 2.4. Este modo suporta dados de cor de 8 ou 12 bits. No modo seqüencial, cada componente de cor é completamente codificado em não mais que um *scan*. Se o processamento for intercalado, a imagem irá possuir um único *scan*, relativo aos três componentes de cor. Existem duas alternativas para a codificação de entropia no modo seqüencial: Huffman e Aritmética. O modo *baseline* é um subgrupo do modo seqüencial, no qual são suportados 8 bits nos dados de cor e a codificação de entropia tem como base a codificação de Huffman [MIA 99].

2.3 As Operações da Compressão JPEG

Este item da dissertação irá abordar as operações envolvidas no processo de compressão JPEG. O enfoque será restrito às operações realizadas no modo *baseline*, que é o modo de interesse desta dissertação, uma vez que é utilizado como referência para as arquiteturas desenvolvidas nos capítulos 3 e 4.

A compressão JPEG, quando operando no modo *baseline*, é formada, basicamente, por três operações principais, como está apresentado na fig. 2.5: Transformada Discreta do Coseno em duas dimensões (DCT 2-D), quantização e codificação de entropia [THE 92] [BHA 99]. Cada uma destas operações é abordada nos próximos itens deste capítulo.

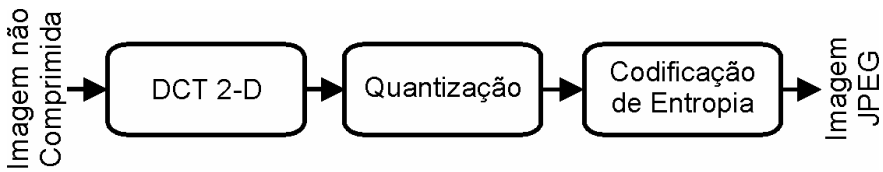


FIGURA 2.5 – Operações da compressão JPEG

As três operações apresentadas na fig. 2.5 são realizadas na compressão de imagens coloridas ou em tons de cinza e formam o núcleo da compressão JPEG [THE 92].

Caso a imagem a ser processada seja uma imagem colorida, mais duas operações podem ser inseridas no processo de compressão: conversão do espaço de cores e *downsampling*. Neste caso, a compressão JPEG é formada por cinco operações, como está apresentado na fig. 2.6.

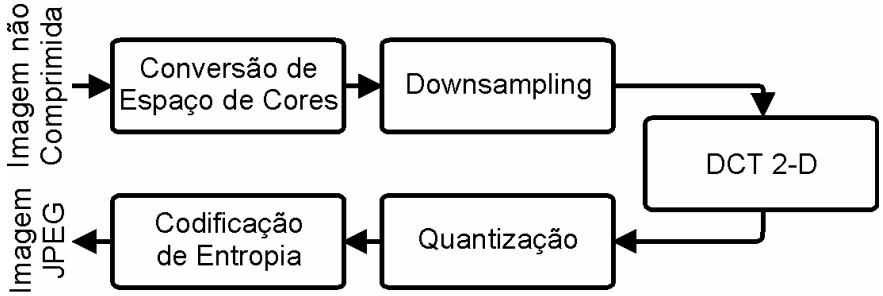


FIGURA 2.6 – Operações da compressão JPEG incluindo a conversão do espaço de cores e o *downsampling*

A conversão de espaço de cores é necessária se a imagem a ser comprimida estiver no espaço de cores RGB (*Red, Green and Blue*) [MUR 96], como uma imagem no formato Windows BMP [MUR 96] por exemplo. Então, o espaço de cores deve ser transformado de RGB para um espaço do tipo luminância e crominância [BHA 99], como será explicado no próximo item.

A operação de *downsampling* não é obrigatória, mas contribui significativamente para a redução dos dados da imagem que está sendo comprimida. Esta operação será detalhada no item 2.3.2.

Tanto a operação de conversão do espaço de cores quanto a operação de *downsampling* só são aplicadas a imagens coloridas, não tendo sentido para imagens em tons de cinza. Esta questão ficará mais clara quando as operações forem explicadas em maiores detalhes nos itens 2.3.1 e 2.3.2, respectivamente.

A seguir serão apresentadas cada uma das cinco operações envolvidas na compressão JPEG. Primeiro são abordadas as duas operações não obrigatórias, que são a conversão do espaço de cores e o *downsampling*. A seguir são abordadas as operações que formam o núcleo da compressão JPEG, ou seja: o cálculo da DCT 2-D, a quantização e a codificação de entropia.

2.3.1 Conversão do Espaço de Cores

A conversão do espaço de cores é a primeira operação a ser realizada por um compressor JPEG, quando este recebe como entrada imagens no espaço de cores RGB. Os componentes R, G e B possuem um elevado grau de correlação, tornando difícil o processamento de cada uma das informações de cor de forma independente, por isso, a

compressão JPEG tem uma eficiência muito maior para espaços de cores do tipo luminância e crominância do que para o espaço RGB [MIA 99]. Então, os *pixels* da imagem que estão no espaço de cores RGB, são convertidos para um espaço de cores do tipo luminância e crominância para a posterior operação de compressão [PEN 92].

A conversão de espaço de cores só tem sentido quando a imagem que está sendo processada for uma imagem colorida, não se aplicando a imagens em tons de cinza. Uma imagem em tons de cinza possui apenas um componente de cor, que não necessita de nenhum tipo de conversão para ser processada por um compressor JPEG.

Existem vários espaços de cores do tipo luminância e crominância. O espaço que será adotado nesta dissertação é chamado de YCbCr [MUR 96], onde o componente Y contém a informação de luminância da imagem, ou seja, contém as informações sobre os tons de cinza. Os componentes Cb e Cr contêm as informações de cores da imagem, sendo que o componente Cb contém a informação relativa à cor azul e o componente Cr contém a informação relativa à cor vermelha. Este espaço de cor é embasado no espaço YUV, que é utilizado no padrão europeu de televisão PAL, onde os componentes Cb e Cr são escalas deslocadas dos componentes U e V [LI 2002].

Os cálculos realizados na conversão do espaço de cores de RGB para YCbCr estão apresentados abaixo. São consideradas parcelas de cada um dos três componentes de cor dos *pixels* de entrada no cálculo dos componentes de cor dos *pixels* no espaço YCbCr [BHA 99].

$$Y_{i,j} = 0{,}299R_{i,j} + 0{,}587G_{i,j} + 0{,}114B_{i,j}$$
$$Cb_{i,j} = -0{,}169R_{i,j} - 0{,}331G_{i,j} + 0{,}5B_{i,j}$$
$$Cr_{i,j} = 0{,}5R_{i,j} - 0{,}419G_{i,j} - 0{,}081B_{i,j}$$

2.3.2 *Downsampling*

Quando aplica-se a operação de *downsampling*, começa a acontecer a redução nos dados necessários para armazenar a informação da imagem ou, em outras palavras, começa a acontecer a compressão. A operação de *downsampling* não é obrigatória e só pode ser aplicada a imagens coloridas.

O olho humano é menos sensível às informações de crominância da imagem (Cb e Cr) do que à informação de luminância (Y). Com base nesta constatação, é possível eliminar parte da informação de crominância no processo chamado de *downsampling* [MIA 99]. Alguns componentes de luminância (tipicamente 2, 3 ou 4) são, então, associados a componentes de crominância (tipicamente 1 ou 2), em uma relação diferente da relação de entrada, que é de 1:1:1. Existem várias formas para relacionar os componentes de luminância com os de crominância na implementação do *downsampling* dependendo da aplicação alvo. Esta dissertação irá considerar uma relação de 4:1:1, ou seja, quatro componentes Y associados a apenas um componente Cb e um componente Cr.

Com esta relação se obtém uma taxa de compressão de 50% em relação à imagem da entrada, com perdas não muito significativas na sua qualidade. Esta redução ocorre porque são descartados três em cada quatro componentes Cb e Cr. Como exemplo, se for considerada uma fração de 4 *pixels* de uma imagem colorida qualquer, onde cada componente de cor utilize 8 bits, são necessários 32 bits para representar cada uma das três informações de cor destes quatro *pixels* e, no total, são utilizados 96 bits para representar esta fração da imagem. Com a operação de dowsampling na taxa de 4:1:1, a informação de luminância continua a utilizar 32 bits (4 x 8bits), mas as informações de crominância passam a utilizar 8 bits cada (1 x 8bits), em um total de 48

bits. A taxa de compressão obtida é de duas vezes ou de 50%, pois a fração de imagem que utilizava 96 bits antes da operação de *downsampling*, passa a utilizar 48 bits após esta operação.

Por definição, nesta dissertação as unidades de dados são processadas de maneira intercalada, como já explicado, formando vários MCUs. Cada MCU será formado por seis unidades de dados, quatro relativas a componentes Y, uma referente a Cb e uma referente a Cr. A fig. 2.7 apresenta o fator de forma das matrizes Y, Cb e Cr de entrada em relação às matrizes Y, Cb e Cr entregues após a operação de *downsampling*, com seus respectivos MCUs. Cada quadrado na fig. 2.7 eqüivale a uma unidade de dados, ou seja, a uma matriz de 8 x 8 elementos.

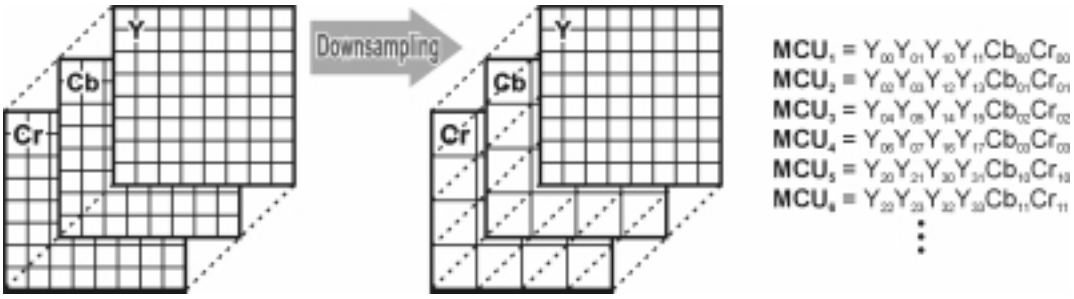


FIGURA 2.7 – Fator de forma das matrizes Y, Cb e Cr antes e depois do *downsampling*

2.3.3 Transformada Discreta do Coseno - DCT 2-D

A transformada discreta do coseno em duas dimensões - DCT 2-D, é utilizada para transformar a representação da informação do domínio espacial para o domínio das freqüências. Após a transformação de domínio, as freqüências mais elevadas, que tendem a contribuir menos com a informação da imagem [MIA 99], são atenuadas, ou mesmo eliminadas, pelo processo chamado de quantização, que é a operação que sucede a DCT 2-D na compressão JPEG e que será abordada no item 2.3.4 deste trabalho.

Para atenuar ou eliminar as freqüências menos perceptíveis ao olho humano, a decomposição da imagem para o domínio das freqüências deve gerar componentes de freqüência independentes entre si, ou seja, estes componentes devem ser decorrelacionados .

Este trabalho não aprofundará os detalhes e justificativas matemáticas da DCT 2-D, sendo que em [PEN 92], [JAI 89], [GON 93], [BHA 99] são encontradas abordagens da DCT, direcionada para o processamento de imagens.

A computação básica em uma DCT em duas dimensões é a transformação de uma unidade de dados (ou seja, uma matriz de 8 x 8 *pixels*) para o domínio das freqüências, portanto, a imagem de entrada deve ser dividida em blocos antes da aplicação da DCT 2-D. O tamanho do bloco foi padronizado em 8 x 8 por vários motivos [BHA 99]:

- este tamanho não impõe requisitos significativos de memória nas implementações em hardware ou software;
- a complexidade computacional deste tamanho de bloco não é demasiadamente grande;
- um tamanho de bloco maior não causa uma melhora significativa na taxa de compressão.

Uma propriedade importante da DCT 2-D, que será utilizada neste trabalho, é a separabilidade, ou seja, o cálculo da DCT 2-D pode ser realizado calculando-se primeiro

a DCT unidimensional (DCT 1-D) das linhas da matriz 8 x 8 de entrada e, a seguir, calculando-se novamente a DCT 1-D sobre as colunas da matriz resultante do primeiro cálculo [BHA 99]. Desta forma o cálculo da DCT 2-D fica com a complexidade significativamente reduzida. Esta propriedade é muito empregada em implementações da DCT 2-D em hardware, uma vez que o número de operações e a quantidade de hardware utilizados é menor do que em implementações em que a DCT 2-D é calculada diretamente.

A aplicação da DCT 2-D nos componentes da matriz 8 x 8 de entrada gera uma matriz 8 x 8 de coeficientes. O cálculo do coeficiente C_{ij}, onde i e j representam a linha e a coluna do coeficiente C, é dado por [MIA 99]:

$$T_{ij} = c_{ij} \sum_{x=0}^{7} \sum_{y=0}^{7} V_{yx} \cos\left(\frac{(2y+1)i\pi}{16}\right) \cos\left(\frac{(2x+1)j\pi}{16}\right)$$

onde V_{yx} é o componente na linha y e coluna x da matriz de entrada, $0 \leq i, j \leq 7$ e

$$c_{ij} = \begin{cases} \frac{1}{8} & \text{se i ou j} = 0 \\ \frac{1}{4} & \text{se i e j} \neq 0 \end{cases}$$

Dada a importância da propriedade da separabilidade da DCT 2-D, faz-se necessária a definição da DCT 1-D. A matriz de entrada para a DCT 2-D possui oito colunas por oito linhas. Para aplicar dois cálculos da DCT 1-D sobre esta matriz, primeiramente aplica-se a DCT 1-D sobre as oito linhas da matriz de entrada e, a seguir, aplica-se novamente a DCT 1-D sobre as oito colunas da matriz resultante do primeiro cálculo da DCT 1-D. Então pode-se perceber que, em ambos os cálculos, a DCT 1-D recebe como entrada um vetor de oito elementos. A DCT 1-D para um vetor de oito elementos é dada por [MIA 99]:

$$T_i = c_i \sum_{x=0}^{7} V_x \cos\left(\frac{(2x+1)i\pi}{16}\right)$$

onde Vx é o componente x do vetor de entrada, $0 \leq i \leq 7$ e

$$c_i = \begin{cases} \frac{1}{2\sqrt{2}} & \text{se i} = 0 \\ \frac{1}{2} & \text{se i} \neq 0 \end{cases}$$

A separação do cálculo da DCT 2-D em dois cálculos da DCT 1-D diminui a complexidade total da operação, diminuindo a área ocupada para implementações em hardware e, consequentemente, aumentando o seu desempenho. Esta decomposição simples reduz a complexidade do cálculo por um fator de quatro [BHA 99]. O algoritmo da DCT 2-D sem a separabilidade utiliza 64 multiplicações e 64 adições para o cálculo de cada coeficiente da DCT. Assim, para cada matriz de 8x8 *pixels*, são necessárias 4.096 multiplicações e 4.096 adições. Usando a decomposição, são necessários 16 cálculos da DCT 1-D (oito para as linhas e oito para as colunas) resultando em um total de 1.024 multiplicações e 1.024 adições para a mesma matriz de 8x8 *pixels* [BHA 99].

O algoritmo escolhido para o cálculo da DCT 2-D, neste trabalho, além de utilizar o princípio da separabilidade, é também um algoritmo otimizado para o cálculo da DCT, inserido na compressão JPEG. O algoritmo proposto por [ARA 88] e

modificado por [KOV 95] para cálculo da DCT 2-D prevê que parte do cálculo da DCT seja incorporado ao passo da quantização, que será abordada no próximo item. Sendo a quantização uma divisão inteira dos coeficientes da DCT 2-D por constantes, é possível inserir neste cálculo algumas das divisões por constantes que são necessárias ao cálculo da DCT. Como ambas as operações são de divisão por constantes, é possível gerar uma terceira constante cujo valor é o produto das duas primeiras. Desta forma, o cálculo da quantização não tem sua complexidade alterada, enquanto o cálculo da DCT fica significativamente simplificado. As saídas são, então, uma escala dos resultados esperados para a DCT 2-D e, portanto, os algoritmos que adotam esta simplificação não realizam o cálculo completo da DCT 2-D. Outras propostas que podem transferir parte dos cálculos da DCT 2-D para um pós processamento, como a quantização, são apresentadas em [CHE 77] e [LEE 84].

Com o uso do algoritmo proposto por [ARA 88] e [KOV 95] são utilizadas 29 adições e 5 multiplicações para o cálculo de uma DCT 1-D sobre um vetor de oito elementos. Então são necessárias 80 multiplicações e 464 adições para o cálculo da DCT 2-D de uma matriz de 8x8 *pixels*. Esta solução utiliza 1,95% das multiplicações e 11,33% das adições do algoritmo sem separabilidade, e 7,81% das multiplicações e 45,31% das adições do algoritmo com separabilidade que não gera resultados em escala. Esta é a mais eficiente DCT 1-D conhecida [PEN 92] para aplicações como a compressão JPEG.

A tab. 2.1, extraída de [BHA 99], traça um comparativo entre diversas soluções simplificadas de DCT 1-D, tomando como referência o número de multiplicações e adições necessárias a cada solução. Dentre as soluções apresentadas na tab. 2.1, a solução utilizada nesta dissertação é a que utiliza o menor número de operações.

TABELA 2.1 – Comparativo entre alguns algoritmos rápidos para cálculo da DCT 1-D

Algoritmo	Multiplicações	Adições
Chen [CHE 77]*	16	26
Lee [LEE 84]*	12	29
Lee [LEE 84]**	11	29
Chen [CHE 77]**	8	26
Arai [ARA 88]**	5	29

* usa separabilidade
** usa separabilidade e escala

Uma outra alternativa para simplificar o cálculo da DCT 2-D na compressão JPEG é considerar as perdas que existem no processo de quantização e, ao invés de efetuar o cálculo completo, efetuar um cálculo aproximado da DCT 2-D, como em [TRA 2000] e em [HOF 2001]. Desta forma é possível simplificar o cálculo da DCT 2-D, sem prejudicar a qualidade da compressão, uma vez que a saída do processo de quantização será igual àquela gerada a partir do cálculo completo da DCT 2-D. Este método foi parcialmente utilizado nesta dissertação, pois as multiplicações por constantes, necessárias ao cálculo da DCT 1-D, são obtidas a partir da soma de deslocamentos, o que é uma aproximação do cálculo completo, como será detalhado no item 3.1.3.4.

A matriz resultante do cálculo da DCT 2-D é composta de 64 elementos e, por definição do padrão JPEG, o coeficiente da freqüência zero é chamado de componente DC, sendo todos os demais coeficientes chamados de componentes AC. Esta divisão é

necessária porque, na compressão JPEG, os componentes DC e AC são codificados de maneira diferenciada, como será visto no item 2.3.5 deste capítulo.

2.3.4 Quantização

Os coeficientes da DCT 2-D passam pelo processo chamado de quantização. A quantização, juntamente com o *downsampling* (caso seja aplicado), são os reais responsáveis pela perda de informação na compressão JPEG.

O processo de quantização é uma divisão inteira dos coeficientes da DCT 2-D por uma determinada constante, chamada de constante de quantização, e o resultado é arredondado para o menor inteiro mais próximo [MIA 99]. O objetivo da quantização é que o resultado arredondado seja zero para o maior número possível de freqüências que não interfiram, ou interfiram pouco, na qualidade da imagem, potencializando assim o uso de técnicas de codificação sem perdas, que serão abordados no item 2.3.5 deste trabalho. As constantes de quantização são armazenadas em matrizes 8x8 chamadas de tabelas de quantização. No modo *baseline*, por definição, são usadas, no máximo, duas tabelas de quantização. Para a quantização de imagens coloridas são utilizadas as duas tabelas de quantização, uma para o componente de luminância (Y) e outra para os componentes de crominância (Cb e Cr). Por outro lado, para imagens em tons de cinza é utilizada apenas a matriz de quantização referente ao componente de luminância.

O cálculo realizado pela operação de quantização está apresentado abaixo, onde Cq_{ij} é o coeficiente quantizado, C_{ij} é o coeficiente da DCT 2-D e Q_{ij} é o *quantum* extraído da tabela de quantização [BHA 99].

$$Cq_{ij} = round\left(\frac{C_{ij}}{Q_{ij}}\right) \qquad 0 \leq i,j \leq 7$$

Imagens típicas de fotografias apresentam amplitudes pequenas nas grandes freqüências, indicando que a maior parte da energia da imagem está localizada nas freqüências mais baixas [BHA 99]. Então, as tabelas de quantização são desenvolvidas com valores menores para as regiões de baixa freqüência e valores maiores para as regiões de alta freqüência. Desta forma as freqüências mais importantes para a percepção do olho humano são menos atenuadas que as demais freqüências. Como resultado desta operação obtém-se uma matriz esparsa, de acordo com o objetivo anteriormente exposto.

Para cada imagem existe um conjunto de tabelas de quantização ótimo, mas é possível generalizar estas tabelas para que sejam usadas em várias imagens de uma determinada aplicação. O próprio padrão JPEG [THE 92] sugere tabelas de quantização típicas para luminância (Y) e para crominância (Cb e Cr). Estas tabelas típicas têm um bom desempenho para a maior parte das imagens [BHA 99]. Este trabalho utilizará as tabelas propostas pelo padrão, que estão apresentadas no anexo 2 desta dissertação, nas quais pode-se perceber que a menor divisão efetuada na quantização é por 10 e a maior é por 199.

As tabelas de quantização usadas na compressão JPEG são inseridas no arquivo da imagem comprimida (como será visto em maiores detalhes no anexo 1 deste trabalho) para que a operação possa ser desfeita. Como a quantização é uma divisão inteira, todos os valores à direita da vírgula são eliminados no resultado, não podendo ser recuperados na operação inversa, quando a imagem é descomprimida. Estas perdas podem ser controladas pelos valores contidos nas tabelas de quantização. Quanto mais próximos da unidade forem os valores da tabela, menor a perda de informação da

imagem, mas também, menor será a taxa de compressão obtida pela codificação de entropia, que será apresentada no próximo item deste capítulo.

2.3.5 Codificação de Entropia

Após o processo de quantização, a matriz resultante terá muitas ocorrências de zeros. Os valores diferentes de zero têm maior probabilidade de estarem concentrados no canto superior esquerdo da matriz de coeficientes por dois motivos principais. O primeiro, porque a DCT 2-D concentra a maior parte da energia da imagem no canto superior esquerdo da matriz de coeficientes [BHA 99], então, as freqüências localizadas nesta região da matriz são as que possuem as maiores amplitudes e, por isso, são as que têm maior probabilidade de terem valores diferentes de zero após a quantização. O segundo motivo diz respeito às matrizes de quantização, que possuem *quanta* menores para os elementos desta região.

Dada a maior concentração de valores diferentes de zero no canto superior esquerdo da matriz resultante da quantização, esta matriz é lida em ziguezague, para que as ocorrências de zeros apareçam em grandes seqüências, potencializando as técnicas de compressão utilizadas pelo codificador de entropia. A fig. 2.8 apresenta a ordenação em ziguezague sobre a matriz resultante da quantização.

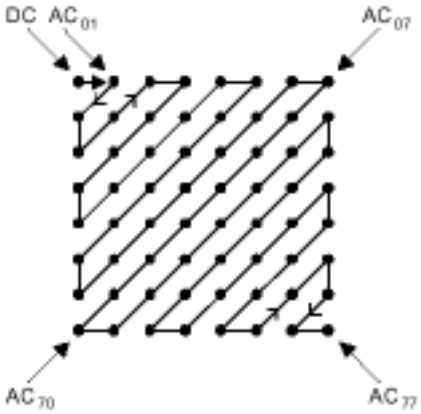


FIGURA 2.8 – Ordenamento em ziguezague

A redução real na quantidade de bits da imagem é realizada nas operações de *downsampling* (se aplicada) e de codificação de entropia. As demais operações são as responsáveis pela preparação dos dados para que a codificação de entropia atinja as elevadas taxas de compressão obtidas pela compressão JPEG.

A codificação de entropia utiliza, em conjunto, várias técnicas de compressão sem perdas, com o intuito de obter o menor número possível de bits para representar cada matriz de coeficientes quantizados. Estas técnicas são a codificação diferencial, a codificação de comprimento de palavra variável (*Variable Length Coding* – VLC), a codificação por número de ocorrências (*Run-Length Encoding* – RLE) e a codificação de Huffman.

Na codificação de entropia os componentes DC e AC dos blocos de entrada são tratados de forma diferenciada, como pode ser observado na fig. 2.9, por isso, este item foi dividido em dois sub-itens, um sobre a codificação de entropia para componentes DC e outro sobre a codificação de entropia para componentes AC.

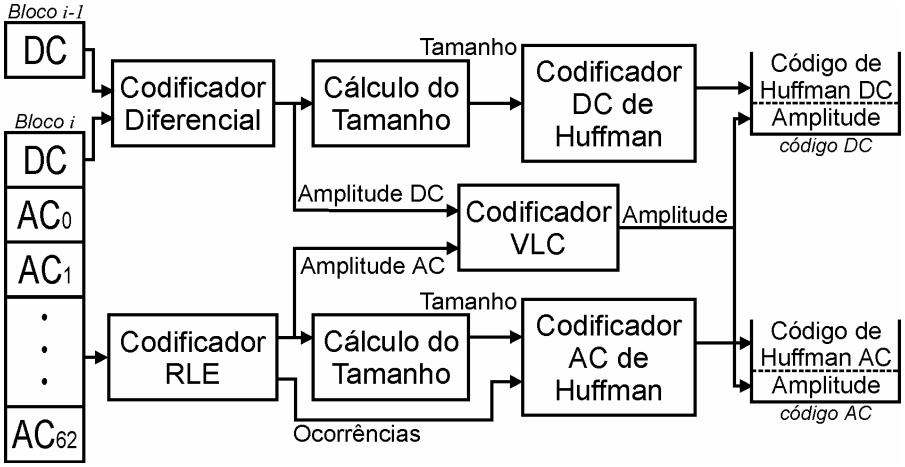


FIGURA 2.9 – Etapas da codificação de entropia

2.3.5.1 Codificação de Entropia de Componentes DC

A primeira etapa na codificação de entropia dos componentes DC é a codificação diferencial. Como os componentes DC de unidades de dados consecutivas de um mesmo componente de cor possuem um alto grau de correlação [BHA 99], o valor da diferença entre dois componentes DC consecutivos possui uma amplitude menor do que a amplitude original de um componente DC. A codificação diferencial é efetuada com uma subtração do valor DC do bloco anterior pelo valor DC do bloco atual ($DC_i - DC_{i-1}$), gerando o campo *amplitude DC*. Para a compressão de imagens coloridas, é preciso considerar o intercalamento entre as matrizes dos componentes de cor Y, Cb e Cr, uma vez que a codificação diferencial deve ocorrer entre os elementos DC consecutivos de um mesmo componente de cor.

Após a codificação diferencial, a codificação de entropia determina qual é o menor número de bits necessários para representar o campo *amplitude DC* através da tabela de tamanhos [THE 92], apresentada na tab. 2.2, gerando o campo *tamanho*.

TABELA 2.2 – Tabela de tamanhos

Tamanho	Faixa de Valores
0	0
1	–1,1
2	–3,–2,2,3
3	–7....–4, 4...7
4	–15...–8, 8...15
5	–31...–16, 16...31
6	–63...–32, 32...63
7	–127...–64, 64...127
8	–255...–128, 128...255
9	–511...–256, 256...511
10	–1023...–512, 512...1023
11	–2047...–1024, 1024...2047

O campo *tamanho* é, então, codificado por Huffman [MUR 96]. A codificação de Huffman é uma codificação de comprimento de palavra variável que utiliza a estatística para determinar qual palavra deve ser associada a cada símbolo. Os símbolos com maior número de ocorrências recebem os menores códigos e os símbolos com menor número de ocorrências recebem os maiores códigos.

A codificação de Huffman pode ser estática ou dinâmica. Na codificação dinâmica as estatísticas são montadas para cada diferente imagem, gerando um resultado ótimo em termos de compressão. Por outro lado, na codificação estática tabelas de estatísticas predefinidas são utilizadas, diminuindo significativamente a complexidade do processo de compressão, mas a eficiência da codificação também diminui.

O padrão JPEG [THE 92] sugere tabelas de Huffman estáticas para uso geral, que serão utilizadas nas arquiteturas implementadas nesta dissertação. Estas tabelas estão apresentadas no anexo 3. A compressão JPEG *baseline* prevê a existência de até quatro tabelas de Huffman para a codificação de imagens coloridas: uma para os componentes DC dos dados de luminância (Y), uma para os coeficientes AC dos dados de luminância, uma para os coeficientes DC dos dados de crominância (Cb e Cr) e uma para os coeficientes AC dos dados de crominância. Quando são codificadas imagens em tons de cinza, são utilizadas as duas tabelas de Huffman para luminância, uma para elementos DC e outra para elementos AC. As tabelas de Huffman utilizadas no processo de compressão estão apresentadas no anexo 3 e são inseridas no arquivo JPEG, para possibilitar a operação inversa quando da descompressão.

A codificação de Huffman do campo *tamanho* irá gerar o campo *código de Huffman DC*, através da tabela de Huffman DC, que pode ser do componente de luminância ou de crominância. Se a compressão for de imagens em tons de cinza, a tabela de Huffman DC utilizada é sempre referente ao componente de luminância.

A codificação VLC, da mesma forma que a codificação de Huffman, utiliza palavras de comprimento variável para representar os símbolos codificados. A diferença é que a codificação VLC não utiliza dados estatísticos, ela apenas descarta todos os bits que não são significativos no símbolo de entrada, onde mesmo o bit de sinal é descartado. A codificação VLC de números negativos exige que estes números estejam em complemento de um [BHA 99] e não em complemento de dois, por isso, todos os números negativos devem ser subtraídos da unidade antes de serem codificados por VLC. Como, em todos os casos, os números positivos iniciarão por 1 e os números negativos começarão por 0, a interpretação do sinal deve ser inversa à convencional, se o número começa por zero, então, este número é negativo e está em complemento de um. Por outro lado, se o número começa por um, então, é positivo. A codificação VLC recebe como entrada o campo *amplitude DC* e gera o campo *amplitude*.

Por fim, o *código de Huffman DC* é concatenado com o campo *amplitude*, gerando o par *código de Huffman DC / amplitude*, que é o componente DC já codificado pelo codificador de entropia.

Como exemplo, será considerado um componente DC de uma matriz de luminância com a *amplitude DC* igual a 195. Neste caso, o campo *tamanho* será igual a 8, através da tab. 2.2. O *código de Huffman DC* para o valor 8, conforme a tab. A3.1 do anexo 3, é igual a 111110. A codificação VLC para o valor 195 gera um campo *amplitude* igual a 11000011. Então a codificação de entropia para o exemplo é gerada com a concatenação dos campos *código de Huffman DC* e *amplitude*, sendo igual a 11111011000011.

Em outro exemplo, considerando um componente DC de uma matriz de luminância com a *amplitude DC* igual a–195, novamente, o campo *tamanho* será igual a

8 e *código de Huffman DC* será igual a 111110. A codificação VLC gera o campo *amplitude* através do complemento de um do valor –195, que é igual a 00111100. Então a codificação de entropia deste exemplo será igual a 11111000111100.

2.3.5.2 Codificação de Entropia de Componentes AC

Para os componentes AC, a primeira etapa é a codificação RLE [MUR 96]. O princípio básico da codificação RLE é a substituição de um conjunto de símbolos repetidos em uma seqüência de dados pelo número de símbolos repetidos e o próprio símbolo [MUR 96]. Por isso, a codificação RLE é indicada para casos onde exista um grande número de repetições de símbolos em seqüência. Na compressão JPEG existem grandes seqüências de zeros após a ordenação em ziguezague da matriz resultante da quantização, por isso, a codificação RLE foi simplificada para apenas contar a ocorrência deste símbolo. Além disso, como são contados apenas os zeros, não é necessário que o símbolo apareça após o número de ocorrências.

A codificação RLE, na compressão JPEG, conta quantos zeros existem antes de um componente não zero e gera um par *ocorrências / amplitude AC*, onde o campo *ocorrências* informa o número de zeros que antecedem o valor não zero e o campo *amplitude AC* é o próprio valor do componente não zero. Na fig. 2.10 é apresentado um exemplo de possíveis componentes AC, de uma matriz de entrada para a codificação de entropia, e os seus respectivos pares *ocorrências / amplitude AC*, gerados pela codificação RLE. O primeiro elemento da matriz do exemplo é *X* porque seu valor não importa para a codificação RLE, uma vez que este é o elemento DC da matriz.

FIGURA 2.10 – Exemplo de codificação RLE no padrão JPEG

Existem dois símbolos especiais previstos pela compressão JPEG, identificados como EOB (*End of Block*) e ZRL (*Zero Run Length*), que são inseridos nos dados durante a codificação RLE [MIA 99]. O primeiro indica que a matriz de entrada termina com zeros. O segundo indica a ocorrência de uma seqüência de quinze componentes zeros, seguida de zero. Como podem ser usados apenas quatro bits para representar a quantidade de ocorrências de zeros [PEN 92], seqüências maiores que dezesseis precisam ser divididas. O marcador EOB tem o valor 0/0 e o marcador ZRL tem o valor 15/0. Ambos os marcadores estão no exemplo da fig. 2.10.

Após a codificação RLE, a codificação de entropia do componente AC, de forma similar ao que ocorre na codificação dos componentes DC, determina qual é o menor número de bits necessários para representar o campo *valor* através da tabela de tamanhos, também gerando o campo chamado de *tamanho*. A tabela de tamanhos para os componentes AC é similar a dos componentes DC, apresentada na tab. 2.2, a única diferença é que os tamanhos 0 e 11 não existem na tabela de tamanhos AC.

O próximo passo para codificação de entropia dos componentes AC é a concatenação do campo *ocorrências*, gerado pela codificação RLE, com o campo *tamanho*, gerado a partir da tabela de tamanhos. Esta concatenação gera o par *ocorrências / tamanho*, que é codificado por Huffman, gerando o campo *código de*

Huffman AC, de modo similar ao que ocorre com o campo *tamanho*, na codificação dos componentes DC. A diferença está na tabela de Huffman que será utilizada.

Simultaneamente à codificação de Huffman do par *ocorrências / tamanho*, o campo *amplitude AC* passa pela codificação VLC, que é efetuada da mesma forma que para os componentes DC, gerando o campo *amplitude*.

Por fim o *código de Huffman AC* é concatenado com o campo *amplitude*, gerando o par *código de Huffman AC / amplitude*, que é o componente AC já codificado pelo codificador de entropia.

Para exemplificar a codificação de componentes AC será considerado o seguinte trecho AC de uma matriz de luminância: 0 0 7 ... A codificação RLE irá transformar esta seqüência no par *ocorrências / amplitude AC*, que será igual a 2 / 7. Segundo a tabela de tamanhos (tab. 2.2), a *amplitude AC* de valor 7 irá gerar um campo *tamanho* com valor igual a 3. A concatenação do campo *ocorrências* com o campo *tamanho*, 2 / 3, possui um *código de Huffman AC* igual a 1111110111 segundo a tab. A3.3 do anexo 3. O valor 7 gera um campo *amplitude* igual a 111, de acordo com a codificação VLC. A codificação de entropia para o exemplo é, então, finalizada através da concatenação do *código de Huffman AC* com o campo *amplitude*, sendo igual a 1111110111111.

Outro exemplo é o trecho AC: 0 0 –7 ..., também de uma matriz de luminância. A codificação RLE deste exemplo gera um par *ocorrências / amplitude AC* igual a 2 / -7. O valor -7 tem um *tamanho* igual a 3. O *código de Huffman AC* da concatenação do campo *ocorrências* com o campo *tamanho*, 2 / 3, é o mesmo do exemplo anterior, ou seja, 1111110111. O complemento de um do valor -7 gera um campo *amplitude* igual a 000, através da codificação VLC. Deste modo, a codificação de entropia para este exemplo é igual a 1111110111000.

3 A Arquitetura do Compressor JPEG para Imagens em Tons de Cinza

Este capítulo enfocará a arquitetura desenvolvida para o compressor JPEG de imagens em tons de cinza. A imagem de entrada, considerada neste capítulo, 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. Este é o padrão mais usado para imagens em tons de cinza [MUR 96]. O valor de cada *pixel* representa a intensidade de preto que existe no *pixel* da imagem.

Para imagens em tons de cinza, são três as operações principais realizadas pela compressão JPEG: Transformada Discreta do Coseno em Duas Dimensões (DCT 2-D), quantização e codificação de entropia, conforme está apresentado na fig. 3.1.

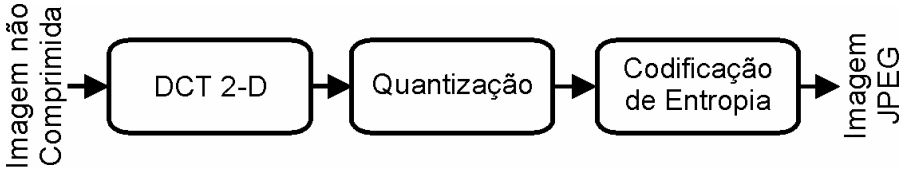


FIGURA 3.1 – Operações da compressão JPEG para imagens em tons de cinza

Antes de ser processado o cálculo da DCT 2-D, o *pixel* de entrada deve passar por um deslocamento de nível. Este deslocamento faz com que o valor médio dos elementos da entrada passe de cento e vinte e oito para zero, possibilitando uma maior uniformidade no processamento da imagem [BHA 99]. Esta operação resume-se a uma simples subtração de 128 em todos os componentes de entrada, que passam, então, a estar contidos na faixa de valores de -128 a 127.

O *datapath* e o controle da arquitetura foram desenvolvidos de forma hierárquica. Cada nível da hierarquia gera, da maneira mais independente possível, os seus sinais de controle. Essa divisão hierárquica agilizou o desenvolvimento da arquitetura, aumentando o seu desempenho e a sua clareza, além de possibilitar o reuso das partes integrantes da arquitetura como *cores* IP em outros projetos. A arquitetura do compressor será abordada de acordo com a hierarquia desenvolvida, com detalhamentos incrementais dentro de cada nível, até o nível de menor abstração.

O nível mais abstrato da hierarquia está apresentado na fig. 3.2 e possui apenas quatro macro estágios de *pipeline*: DCT 2-D, quantizador, *buffer* ziguezague e codificador de entropia. Cada macro estágio possui vários estágios de *pipeline*, que são necessários para o seu nível da hierarquia.

Os dados de entrada, para o compressor de imagens em tons de cinza, são consumidos na taxa de um dado de 8 bits para cada ciclo de *clock*. As palavras JPEG de saída possuem 32 bits e são entregues de forma assíncrona, como será detalhado no item 3.3. Por conta deste assincronismo na saída, a arquitetura do compressor JPEG utiliza um *flag*, chamado de *OK* na fig. 3.2, para indicar que a saída é válida.

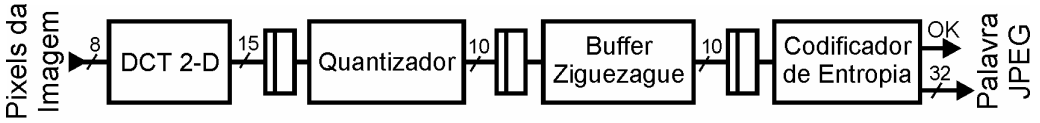


FIGURA 3.2 – Arquitetura genérica do compressor JPEG para imagens em tons de cinza

A latência global é dependente do codificador de entropia, cujas saídas são entregues de forma assíncrona. De qualquer modo, a latência da arquitetura da DCT 2-D é de 163 ciclos de *clock*, a latência do quantizador é de 3 ciclos e a latência do *buffer* ziguezague é de 66 ciclos de *clock*. A latência mínima do codificador de entropia é de 5 ciclos, portanto a latência mínima do compressor é de 237 ciclos de *clock*. A latência elevada é um reflexo da complexa operação que é realizada e da estratégia de desenvolver um *pipeline* com muitos estágios, visando um alto desempenho. Uma imagem de 640 x 480 *pixels*, por exemplo, é completamente processada em, no máximo, 307.437 ciclos, considerando o *pipeline* vazio, e em, no máximo, 307.200, ciclos, considerando o *pipeline* preenchido. Do exemplo é possível perceber que o impacto do preenchimento do *pipeline* e, por conseqüência, da latência, é pequeno quando considerada, na entrada, uma imagem de tamanho real. Este impacto fica ainda menos relevante, se forem consideradas seqüências de imagens para a compressão, tal qual em vídeo digital ou outras aplicações.

O compressor JPEG para imagens em tons de cinza foi completamente descrito em VHDL estrutural. A descrição VHDL foi direcionada para a síntese em FPGAs da família FLEX10E [ALT 2001a], fabricados pela Altera [ALT 2001]. O ambiente Maxplus2, também da Altera, foi utilizado, tanto para a codificação dos blocos em VHDL, quanto para as sínteses e simulações que foram realizadas para os diversos níveis hierárquicos. No total, foram desenvolvidos 80 arquivos de descrições VHDL que, em conjunto, formam o compressor JPEG. Um total aproximado de 6.400 linhas de VHDL foram escritas para descrever o compressor JPEG para imagens em tons de cinza.

Em alguns dos módulos da arquitetura do compressor foram utilizados blocos de memória que estão disponíveis internamente à família de dispositivos para a qual a síntese foi direcionada. O uso da memória interna torna os códigos VHDL, que descrevem estes módulos, dependentes do fabricante dos dispositivos. Portanto, estes módulos não são sintetizáveis em outra ferramenta, direcionada para outro fabricante de FPGAs ou para outra metodologia de síntese, como *standard cell*, por exemplo. O uso de módulos pré-sintetizados de memória interna se justifica porque a sua substituição por registradores ocuparia um espaço imenso em termos de células lógicas, inviabilizando a síntese da arquitetura completa em um único dispositivo. Estas memórias, caso a síntese seja direcionada a outro fabricante de FPGAs, devem ser descritas novamente, de acordo com as exigências da nova família de dispositivos alvo. Caso os códigos VHDL do compressor JPEG sejam utilizados para outra metodologia de projeto, como *standard cell*, as memórias devem ser sintetizadas a partir de um gerador automático de memória.

Os módulos que são dependentes do fabricante do dispositivo, em função do uso de memória interna, são:

- *Buffer* de transposição (utilizado pela arquitetura da DCT 2-D);
- Quantizador;
- *Buffer* ziguezague e
- Codificador de Huffman (utilizado pelo codificador de entropia).

A síntese de todo o compressor utilizou 6.199 células lógicas e 7.436 bits de memória do dispositivo EPF10K130EQC240-1 [ALT 2001a] da Altera, sendo utilizado 93% das células lógicas e 11% dos bits de memória disponíveis neste dispositivo. Foram utilizados 10 pinos de entrada e 33 pinos de saída, isto sem contar com os pinos de alimentação. A freqüência máxima atingida por esta arquitetura foi de 16,6MHz, permitindo que uma imagem de 640x480 *pixels* seja completamente processada em

18,5ms. O compressor desenvolvido pode atingir uma taxa de processamento de 54 imagens de 640 x 480 *pixels* por segundo.

O desempacotamento da imagem de entrada e a montagem do arquivo JPEG são tarefas do processador de entrada e saída, que não foi desenvolvido nesta dissertação. O desempacotamento consiste em disponibilizar apenas os *pixels* presentes no arquivo de entrada para a arquitetura do compressor, eliminando os cabeçalhos e rodapés. A montagem do arquivo JPEG é realizada através das palavras JPEG geradas pelo codificador de entropia e também através das tabelas de quantização e de Huffman utilizadas, sempre seguindo o modelo de arquivo proposto no anexo 1 desta dissertação. Cabe ao processador de entrada e saída dividir a imagem de entrada em janelas de 8x8 *pixels*, que serão utilizadas como entrada para a DCT 2-D e, por conseqüência, para o compressor JPEG de imagens em tons de cinza.

O controle global do compressor é muito simples, uma vez que o controle das operações realizadas pelo compressor é realizado de maneira distribuída, estando inserido no nível hierárquico mais baixo quanto possível. Desta maneira, o controle global deve preocupar-se apenas com a inicialização da arquitetura da DCT 2-D.

Este capítulo será dividido em cinco partes principais, onde são enfocadas as arquiteturas dos quatro módulos do compressor JPEG para imagens em tons de cinza: DCT 2-D, quantizador, *buffer* ziguezague e codificador de entropia, além das considerações finais sobre o compressor.

3.1 A Arquitetura da DCT 2-D

O cálculo da DCT 2-D tem um alto grau de complexidade computacional, o que restringe o seu uso para muitas aplicações. Os primeiros algoritmos desenvolvidos para calcular a DCT em duas dimensões possuíam desempenho computacional muito reduzido, dado o elevado número de operações que eram necessárias. Este número de operações acabava, também, por inviabilizar comercialmente as aplicações direcionadas para hardware, pois exigiam um número demasiado grande de operadores. Mas muitas soluções algorítmicas alternativas, como em [CHE 77], [LEE 84], [ARA 88], [FEI 92] e outras, foram propostas para minimizar a complexidade do cálculo, aumentando o seu desempenho e viabilizando o seu uso em diversas aplicações, incluindo aquelas voltadas para sua implementação diretamente em hardware dedicado.

Existem muitas propostas de arquiteturas, disponíveis na literatura, para o cálculo da DCT 2-D, como em [MAD 95], [WAN 95], [KOV 95], [LEE 97], [MEL 2000] e outras. O princípio básico da arquitetura para cálculo da DCT 2-D desenvolvida nesta dissertação é o uso da propriedade da separabilidade, conforme foi abordado no capítulo 2. Então, a arquitetura da DCT 2-D é transformada em duas arquiteturas similares de DCT 1-D e um *buffer* de transposição, que recebe os resultados do primeiro cálculo da DCT 1-D, os armazena linha a linha e os entrega coluna a coluna para o segundo cálculo da DCT 1-D. A saída da DCT 2-D, desta forma, fica organizada coluna a coluna, ou seja, as primeiras oito saídas formam a coluna zero da matriz resultado. A arquitetura genérica da DCT 2-D está apresentada na fig. 3.3.

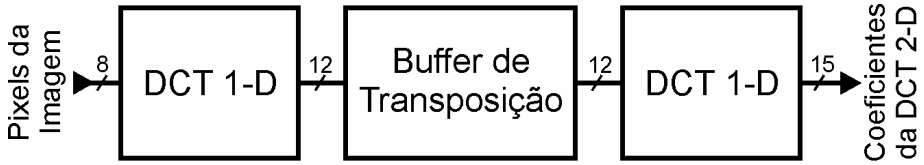


FIGURA 3.3 – Arquitetura genérica da DCT 2-D

As duas arquiteturas para cálculo da DCT 1-D, necessárias ao cálculo da DCT 2-D, são similares e, através de uma simples realimentação, a segunda arquitetura poderia ser utilizada para efetuar os dois cálculos. Nesta dissertação optou-se por usar duas arquiteturas separadas como forma de aumentar o desempenho do cálculo da DCT 2-D, que é o cálculo mais crítico da compressão JPEG. O uso de duas arquiteturas independentes duplica o desempenho do cálculo, considerando o *pipeline* da DCT 2-D preenchido. Por outro lado, a área utilizada é significativamente superior.

Além do uso de duas arquiteturas de DCT 1-D independentes, optou-se por desenvolver duas arquiteturas distintas para permitir a minimização no uso de recursos de hardware. As duas arquiteturas diferem apenas no número de bits utilizados em cada estágio do *pipeline*. A arquitetura da segunda DCT 1-D poderia ser replicada e usada para efetuar o cálculo da primeira DCT 1-D mas, para obter a pretendida minimização, a primeira DCT 1-D foi simplificada de modo a utilizar o menor número de bits possível em cada estágio do *pipeline*. Os cálculos realizados para determinar o número de bits necessários em cada estágio do *pipeline* estão resumidamente apresentados no anexo 4 desta dissertação.

Os valores de entrada na primeira DCT 1-D são os próprios valores de entrada para o compressor JPEG de imagens em tons de cinza, então, estas entradas possuem 8 bits e estão na faixa entre 0 e 255. Com a aplicação do deslocador de nível, esta faixa é alterada para –128 a 127 através de uma subtração de 128 de todos os valores da entrada, como anteriormente mencionado. Esta operação foi simplificada à simples inversão do oitavo bit da entrada, estando acoplada à arquitetura da primeira DCT 1-D.

As arquiteturas para cálculo da DCT 1-D são organizadas em um *pipeline* de seis estágios e estão fortemente baseadas na arquitetura proposta por [KOV 95]. A conexão entre as duas arquiteturas é realizada pelo *buffer* de transposição, que está organizado em um *pipeline* de 65 estágios.

A latência das arquiteturas desenvolvidas para o cálculo da DCT 1-D é de 49 ciclos de *clock*, enquanto que o *buffer* de transposição possui uma latência de 65 ciclos. Desta forma, a latência global da arquitetura da DCT 2-D é de 163 ciclos de *clock*, enquanto que a latência da arquitetura proposta por [KOV 95] é de 172 ciclos. Esta melhora é apresentada em detalhes no item 3.1.3 da dissertação.

A síntese da DCT 2-D para um dispositivo EPF10K100EQC208-1 [ALT 2001a] da Altera, utilizou 4.181 células lógicas e 1.536 bits de memória, atingindo uma freqüência máxima de operação de 18,9MHz. Esta arquitetura seria capaz de processar uma imagem em tons de cinza de 640 x 480 *pixels* em 16,2ms, permitindo uma taxa de processamento de 58 imagens por segundo.

A arquitetura da DCT 2-D e os resultados obtidos através do processo de síntese foram parcialmente apresentados em [AGO 2001a].

A apresentação dos detalhes da arquitetura para o cálculo da DCT 2-D é realizada nos próximos itens da dissertação, onde inicialmente está apresentado o bloco de controle da DCT 2-D, seguido do algoritmo rápido usado para o cálculo da DCT 1-D e das arquiteturas desenvolvidas para efetuar o cálculo da DCT 1-D e para o *buffer* de transposição. Por fim são apresentadas as considerações finais sobre a arquitetura da DCT 2-D.

3.1.1 O Bloco de Controle da DCT 2-D

O bloco de controle global da DCT 2-D é bastante simples, uma vez que as operações de controle foram descentralizadas na arquitetura do compressor. O controle da DCT 2-D gera sinais de inicialização para o *buffer* de transposição e para a segunda DCT 1-D, a partir dos *flags* gerados pelos blocos de controle das duas DCT 1-D e pelo

buffer de transposição. O *reset* que inicia a operação da arquitetura do bloco de controle é o mesmo que inicia a operação na primeira DCT 1-D. Este sinal é o próprio sinal de inicialização do compressor. Todos os demais sinais de controle são gerados pelos blocos de controle da DCT 1-D e do *buffer* de transposição.

3.1.2 Algoritmo Rápido Usado para o Cálculo da DCT 1-D

O algoritmo escolhido nesta dissertação para o cálculo da DCT em uma dimensão foi proposto por [ARA 88] e modificado por [KOV 95] e está apresentado abaixo. Este algoritmo possui seis passos completamente independentes entre si, o que possibilita o uso de *pipeline* entre estes passos.

As simulações do algoritmo proposto em [KOV 95] apresentaram resultados diferentes dos esperados para o cálculo da DCT. Desta forma, após uma análise detalhada dos resultados de simulação e após comparações com o algoritmo proposto em [ARA 88] (que serviu de base para o algoritmo desenvolviodo em [KOV 95]), concluiu-se o algoritmo proposto em [KOV 95] possuia um erro no cálculo da variável b_2, definida como a_2 - a_4. Este erro distorce completamente os dados da imagem que está sendo processada. O erro foi corrigido no algoritmo utilizado neste trabalho, onde o cálculo de b_2, como está apresentado no algoritmo abaixo exposto, é obtido através do cálculo a_3 - a_4.

Para o algoritmo apresentado abaixo, tem-se que:

- $m1 = \cos(4\pi/16)$;
- $m2 = \cos(6\pi/16)$;
- $m3 = \cos(2\pi/16) - \cos(6\pi/16)$ e
- $m4 = \cos(2\pi/16) + \cos(6\pi/16)$.

Passo 1

b0 = a0 + a7	*b1 = a1 + a6*	*b2 = a3 – a4*
b3 = a1 – a6	*b4 = a2 + a5*	*b5 = a3 + a4*
b6 = a2 – a5	*b7 = a0 – a7*	

Passo 2

c0 = b0 + b5	*c1 = b1 – b4*	*c2 = b2 + b6*
c3 = b1 + b4	*c4 = b0 – b5*	*c5 = b3 + b7*
c6 = b3 + b6	*c7 = b7*	

Passo 3

d0 = c0 + c3	*d1 = c0 – c3*	*d2 = c2*
d3 = c1 + c4	*d4 = c2 – c5*	*d5 = c4*
d6 = c5	*d7 = c6*	*d8 = c7*

Passo 4

e0 = d0	*e1 = d1*	*e2 = m3 x d2*
e3 = m1 x d7	*e4 = m4 x d6*	*e5 = d5*
e6 = m1 x d3	*e7 = m2 x d4*	*e8 = d8*

Passo 5

f0 = e0	*f1 = e1*	*f2 = e5 + e6*
f3 = e5 – e6	*f4 = e3 + e8*	*f5 = e8 – e3*
f6 = e2 + e7	*f7 = e4 + e7*	

Passo 6

S0 = f0	*S1 = f4 + f7*	*S2 = f2*
S3 = f5 – f6	*S4 = f1*	*S5 = f5 + f6*
S6 = f3	*S7 = f4 – f7*	

3.1.3 A arquitetura da DCT 1-D

A arquitetura desenvolvida para o algoritmo exposto no item anterior está apresentada na fig. 3.4, sendo fortemente baseada na arquitetura proposta por [KOV 95]. Os resultados obtidos a partir do desenvolvimento da arquitetura da DCT 1-D foram publicados em [AGO 2001b]. Como o algoritmo possui seis passos bem definidos e independentes entre si, o *pipeline* desenvolvido teve, também, seis estágios. Cinco dos seis estágios são de soma ou subtração e um de multiplicação sendo que apenas um operador aritmético é utilizado em cada estágio, seguindo a proposta de [KOV 95]. Com a restrição do número de operadores, estabeleceu-se, como objetivo para esta implementação, o uso de, no máximo, oito ciclos de *clock* em cada estágio do *pipeline*.

No que diz respeito aos estágios de soma existem, no máximo, oito operações em um único passo do algoritmo, desta forma, para respeitar a restrição anteriormente posta, cada soma deve, necessariamente, ser processada em um único ciclo, o que é possível mesmo com arquiteturas convencionais de somadores (como *ripple carry*).

O maior desafio para manter o uso de oito ciclos de *clock* por estágio está no estágio de multiplicação, mais especificamente, na arquitetura do multiplicador, que precisa efetuar cinco multiplicações por constantes em apenas oito ciclos. Para respeitar esta restrição foi desenvolvida uma arquitetura em *pipeline* de dois estágios que efetua as cinco multiplicações em seis ciclos e será explicada em detalhes no item 3.1.3.4.

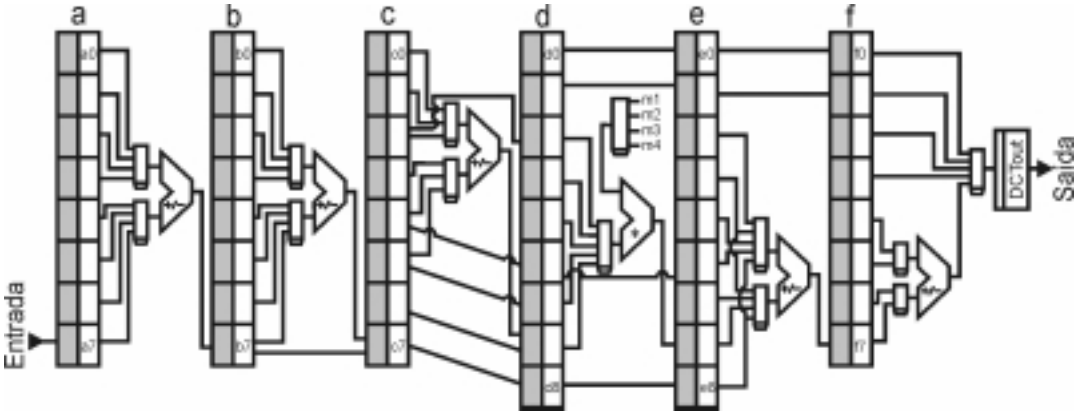


FIGURA 3.4 – Arquitetura para o cálculo da DCT 1-D

Na fig. 3.4, os dados chegam na arquitetura a uma taxa de um novo valor de oito bits a cada ciclo de *clock*, de acordo com a definição imposta pela arquitetura global do compressor. Estes valores são armazenados em um *buffer* do tipo ping-pong (a arquitetura genérica utilizada para este *buffer* e o seu princípio de operação são apresentados no item 3.1.3.2), que é capaz de manter os dados estáveis na entrada dos multiplexadores, que estão conectados às entradas do operador, durante os oito ciclos de *clock* necessários para que todos os cálculos previstos no estágio sejam efetuados. Todas as entradas dos estágios de *pipeline* estão conectadas à saída de um *buffer* ping-pong, cujas entradas estão conectadas às saídas do estágio anterior, como pode ser observado na fig. 3.4. Através do bloco de controle, utilizando os multiplexadores apresentados na fig. 3.4, é possível selecionar quais valores devem estar disponíveis nas entradas do operador a cada ciclo de *clock*. Desta maneira, são realizadas, seqüencialmente, todas as operações previstas nos passos do algoritmo anteriormente apresentado.

A saída das arquiteturas da DCT 1-D é sincronizada através de um registrador, cuja a entrada está conectada a um multiplexador, que seleciona qual das saídas deve ser

armazenada no registrador a cada ciclo de *clock*. Esta sincronização da saída é necessária porque parte dos resultados do último nível da DCT 1-D são gerados de forma paralela e parte são gerados de forma seqüencial, como pode ser observado no passo 6 do algoritmo já apresentado. Como resultado desta sincronização, após o preenchimento do *pipeline*, um novo valor válido é gerado na saída das arquiteturas de cálculo da DCT 1-D a cada ciclo de *clock*.

A diferença entre as arquiteturas da primeira e da segunda DCT 1-D reside no número de bits utilizados em cada estágio do *pipeline*, refletindo-se nas larguras de palavra dos *buffers* ping-pong, dos multiplexadores e dos operadores dos respectivos estágios. Além, é claro, de influenciar na área ocupada e no desempenho das duas arquiteturas. A tab. 3.1 apresenta as diferenças no número de bits utilizados na entrada de cada estágio, cujos cálculos estão apresentados no anexo 4.

TABELA 3.1 – Diferença no número de bits usados na entrada de cada estágio do *pipeline* da primeira e da segunda arquitetura da DCT 1-D.

Estágio		Nº de bits (entrada)	
		1ª DCT 1-D	2ª DCT 1-D
1	(soma)	8	12
2	(soma)	9	13
3	(soma)	10	14
4	(multiplicação)	11	15
5	(soma)	11	15
6	(soma)	12	15

A latência da arquitetura de cálculo da DCT 1-D é de 49 ciclos de *clock*, já considerando a escrita no registrador de saída, usado para o *pipeline*. Foram usados 10 ciclos a menos que na arquitetura proposta por [KOV 95], que possui uma latência de 59 ciclos. Esta melhora é obtida a partir da redução do número de ciclos de *clock* utilizados por cada estágio do *pipeline*. Na arquitetura desenvolvida, são usados oito ciclos por estágio, enquanto que a proposta por [KOV 95], são usados nove ciclos para os estágios de soma e quatorze para o estágio de multiplicação. Estes ganhos são relativos a simplificações realizadas nos *buffers* ping-pong e no multiplicador.

Usando o mesmo exemplo utilizado para introduzir a arquitetura da DCT 2-D, uma imagem de 640 x 480 *pixels*, em tons de cinza, é completamente processada, pela arquitetura desenvolvida para a DCT 1-D, em 307.249 ciclos com o *pipeline* vazio e em 307.200 ciclos com o *pipeline* preenchido.

A síntese da arquitetura proposta para a primeira DCT 1-D utilizou 1.660 células lógicas de um dispositivo EPF10K30ETC144-1 [ALT 2001a] da Altera, atingindo uma freqüência máxima de operação de 22,1MHz. Desta maneira, seria possível processar a imagem do exemplo acima em 13,9ms, permitindo uma taxa de processamento de 72 imagens por segundo. A arquitetura da segunda DCT 1-D ocupou 2.241 células lógicas do dispositivo EPF10K50ETC144-1 [ALT 2001a] da Altera e atingiu uma freqüência de operação máxima de 19MHz, processando a mesma imagem em 16,2ms, permitindo uma taxa de processamento de 62 imagens por segundo. Como era esperado, a segunda DCT 1-D obteve um desempenho inferior que a primeira DCT 1-D.

Os próximos itens irão abordar o bloco de controle e cada um dos três principais elementos presentes no *datapath* da arquitetura da DCT 1-D: *buffers* ping-pong, somadores *ripple carry* e o multiplicador.

3.1.3.1 O Bloco de Controle da DCT 1-D

O bloco de controle da DCT 1-D gera os sinais de habilitação da escrita nos *buffers* ping-pong, os sinais de controle dos multiplexadores, os sinais de controle de operação dos somadores (adição ou subtração) e o sinal de *reset* do multiplicador, sendo que o controle das duas arquiteturas de cálculo da DCT 1-D é idêntico.

A geração dos sinais de controle é determinada pela seqüência da operações exigidas pelo algoritmo escolhido e pelo *pipeline* desenvolvido para implementar este algoritmo. Com o *pipeline* preenchido, a geração dos sinais de controle se repete a cada oito ciclos de *clock*

A arquitetura do bloco de controle consiste, basicamente, de uma máquina de estados com oito estados que se repetem e que possui um *reset* assíncrono. O preenchimento do *pipeline* é controlado através de sinais de seqüenciamento internos ao bloco de controle que, incrementalmente, habilitam a escrita nos registradores ping-pong, a cada nova repetição da máquina de estados.

A tab. 3.2 apresenta o sincronismo definido pelo controle, onde estão apresentados, para cada estado da máquina de estados, os valores que estarão sendo entregues nas entradas do operador de cada estágio do *pipeline*, bem como a operação que este operador irá realizar, considerando que o *pipeline* está preenchido.

Na tab. 3.2, o *X* indica a presença de bolhas nos respectivos estágios do *pipeline*, ou seja, os operadores que possuem *X* como entradas não realizam nenhuma operação significativa durante estes estados. O controle destas bolhas é realizado através da desabilitação da escrita nos registradores ping-pong.

TABELA 3.2 – Operações na arquitetura da DCT 1-D a partir do bloco de controle

	Estágio						**Estágio**					
	a	**b**	**c**	**d**	**e**	**f**	**a**	**b**	**c**	**d**	**e**	**f**
	Estado 1						*Estado 5*					
Entrada A	A0	b0	c0	m3	e5	X	a2	b0	X	m2	e2	X
Entrada B	A7	b5	c3	d2	e6	X	a5	b5	X	d4	e7	X
Operação	Som.	Som.	Som.	Mult.	Som.	X	Som.	Sub.	X	Mult.	Som.	X
	Estado 2						*Estado 6*					
Entrada A	A1	b1	c0	m1	e5	f4	a3	b3	X	X	e4	f5
Entrada B	A6	b4	c3	d7	e6	f7	a4	b7	X	X	e7	f6
Operação	Som.	Sub.	Sub.	Mult.	Sub.	Som.	Som.	Som.	X	X	Som.	Som.
	Estado 3						*Estado 7*					
Entrada A	a3	b2	c1	m4	e3	X	a2	b3	X	X	X	X
Entrada B	a4	b6	c4	d6	e8	X	a5	b6	X	X	X	X
Operação	Sub.	Som.	Som.	Mult.	Som.	X	Sub.	Som.	X	X	X	X
	Estado 4						*Estado 8*					
Entrada A	a1	b1	c2	m1	e8	f5	a0	X	X	X	X	f4
Entrada B	a6	b4	c5	d3	e3	f6	a7	X	X	X	X	f7
Operação	Sub.	Som.	Sub.	Mult.	Sub.	Sub.	Sub.	X	X	X	X	Sub.

O diagrama temporal simplificado do *pipeline* está apresentado na fig. 3.5, onde estão representados os cálculos parciais de duas matrizes 8 x 8, identificadas pelas letras *x* e *y*. Em cada estágio, está representada a faixa de elementos da matriz de entrada que está sendo usada no cálculo naquele estágio.

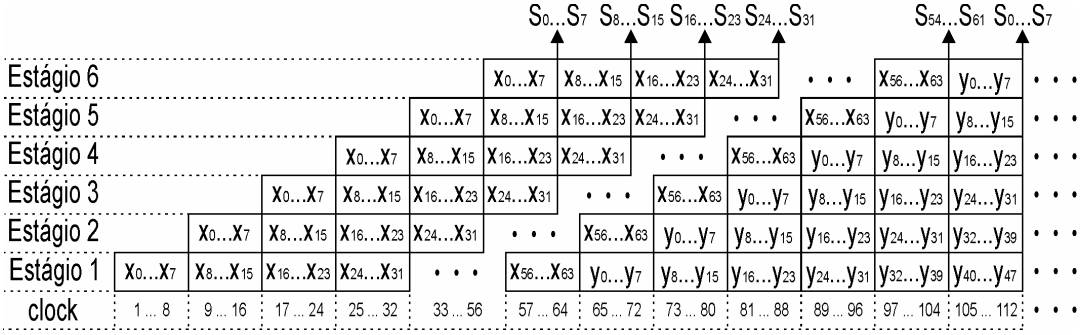


FIGURA 3.5 – Diagrama temporal simplificado do *pipeline* da arquitetura da DCT 1-D

A fig. 3.6 apresenta um *zoom* no diagrama da fig. 3.5, entre os ciclos de *clock* 57 a 64, em que pode-se perceber quais cálculos estão sendo efetuados em cada estágio do *pipeline*. No fragmento do *pipeline* apresentado na fig. 3.6 estão sendo calculadas as últimas seis linhas da matriz *x*, sendo que a terceira linha está pronta ao final do ciclo 64. O estágio 4 apresenta o *pipeline* de dois estágios do multiplicador, que será detalhado no item 3.1.3.4.

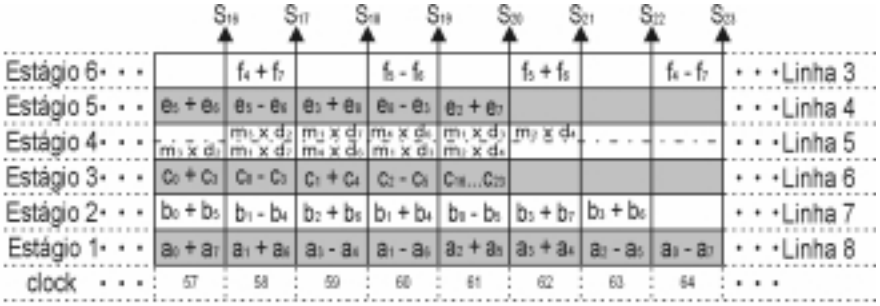


FIGURA 3.6 – *Zoom* no diagrama temporal entre os ciclos de *clock* 57 e 64

3.1.3.2 *Buffers* Ping-Pong

Na arquitetura da DCT 1-D, os *buffers* ping-pong [KOV 95] são responsáveis por toda a temporização e sincronização dos dados, permitindo o uso de *pipeline*. A arquitetura típica deste tipo de *buffer* está apresentada na fig. 3.7 e é formada por duas linhas de registradores identificadas na fig. 3.7 por *ping* e *pong*. Os dados chegam serialmente nos registradores *ping* através do pino *In* e estes dados são deslocados à direita a cada novo ciclo de *clock*. Esta operação de deslocamento é contínua, do início do processamento da imagem até o seu final. Quando todos os dados corretos estão disponíveis nos registradores *ping*, estes são copiados em paralelo para os registradores *pong*. Esta operação se repete a cada oito ciclos de *clock* e, para tanto, durante o sétimo ciclo de *clock* de cada repetição, o sinal de controle *Enable* recebe nível alto habilitando a escrita nos registradores *pong*. Então, na borda do oitavo ciclo da repetição ocorre a cópia em paralelo do conteúdo dos registradores *ping* para os registradores *pong* e o sinal *Enable* recebe nível baixo.

Todas as saídas dos registradores *pong* estão conectadas aos multiplexadores, cujas saídas serão utilizadas como entradas no operador aritmético do respectivo

estágio. Durante oito ciclos de *clock* os dados ficam estáveis nos registradores *pong*, podendo ser usados para os cálculos exigidos por cada passo do algoritmo.

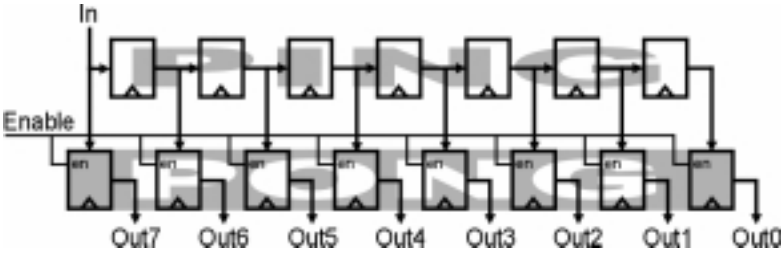


FIGURA 3.7 – Exemplo de registrador ping-pong

Para otimizar o número de ciclos de *clock* utilizados nesta operação, o valor de entrada *In* está conectado, tanto ao primeiro registrador *ping*, quanto ao primeiro registrador *pong*. Desta maneira, a cópia dos conteúdos dos registradores *ping* para os registradores *pong* é feita simultaneamente com o armazenamento do oitavo valor da entrada, o que permite a redução de um ciclo de *clock* em cada estágio do *pipeline* da DCT 1-D em relação à arquitetura proposta em [KOV 95]. Quando o oitavo dado está disponível na entrada, a escrita nos registradores *pong* é habilitada, o dado de entrada é escrito nos registradores *ping* e *pong* mais significativos e os dados armazenados nos registradores *ping* são copiados para os demais registradores *pong*.

3.1.3.3 Somadores *Ripple Carry*

Os somadores utilizados nos cinco estágios de somas (ou subtrações) e no multiplicador do estágio de multiplicações, são do tipo *ripple carry* [WES 95]. Estes somadores possuem arquiteturas bastante simples e amplamente difundidas. A reduzida área ocupada é a principal vantagem dos somadores *ripple carry* e a grande desvantagem reside em seu baixo desempenho, causado pela cadeia de *carry out* que acaba por definir o seu atraso. Não se entrará, nesta dissertação, em detalhes sobre a arquitetura dos somadores *ripple carry*, uma vez que este tipo de arquitetura é amplamente difundida.

Os operadores aritméticos utilizados nos estágios de soma ou subtração, no cálculo da DCT 1-D, devem estar preparados para operar no modo de subtração. Para utilizar o mesmo operador aritmético para soma ou subtração foi desenvolvida uma pequena lógica envolvendo a entrada B, o *carry* de entrada do somador e o sinal de controle de operação *AddSub*, de acordo com a tab. 3.3.

TABELA 3.3 – Lógica de controle do modo de operação dos somadores

AddSub	Carry In	Entrada B	Operação Realizada
0	0	B	A + B
1	1	~B	A + (~B + 1) ou A – B

A geração de *carry out* está prevista apenas para parte dos somadores utilizados, uma vez que, a partir da determinação dos máximos valores gerados por cada operador (apresentada no anexo 4), foi possível identificar quais somadores não iriam gerar jamais um *carry out* significativo. Desta forma, foi possível utilizar o mínimo tamanho de palavra em cada estágio e, em conseqüência, o mínimo de recursos.

A tab. 3.4 apresenta, para cada estágio das duas arquiteturas da DCT 1-D, o número de bits na entrada de cada somador, quais dos somadores possuem a geração de *carry out* e quais somadores podem operar no modo de subtração. Na tab. 3.4, o estágio

do multiplicador (estágio 4) apresenta os três somadores utilizados por sua arquitetura, que será apresentada no próximo item deste capítulo.

TABELA 3.4 – Somadores usados em cada estágio da primeira e da segunda DCT 1-D

Estágio	1ª DCT 1-D			2ª DCT 1-D		
	nº bits	Cout	Add/Sub	nº bits	Cout	Add/Sub
1	8	sim	sim	12	sim	sim
2	9	sim	sim	13	sim	sim
3	10	sim	sim	14	sim	sim
4	16	não	não	20	não	não
	16	não	não	20	não	não
	20	não	não	24	não	não
5	11	sim	sim	15	não	sim
6	12	não	sim	15	não	sim

Os somadores utilizados na arquitetura da primeira DCT 1-D possuem operadores com tamanho mínimo de 8 bits e máximo de 20 bits, enquanto que os operadores da segunda DCT 1-D possuem operadores com tamanho mínimo de 12 bits e máximo de 24 bits, como pode ser observado na tab. 3.4. É a propagação de *carry out* destes somadores que define o caminho crítico da arquitetura de cálculo da DCT 1-D e, por conseqüência, da própria DCT 2-D.

O uso de soluções alternativas, como somadores *carry look ahead* [WES 95], por exemplo, melhoraria significativamente o desempenho da arquitetura, mas também causaria um impacto na área ocupada. As alternativas arquiteturais para os somadores não foram exploradas nesta dissertação, mas este é um bom caminho a ser percorrido, caso a questão de desempenho passe a ser crítica. Como a arquitetura do compressor foi desenvolvida de modo hierárquico, a substituição dos somadores é bastante simples e não envolve os níveis hierárquicos superiores.

3.1.3.4 Multiplicador da DCT 1-D

Os multiplicadores utilizados nas arquiteturas de cálculo da DCT 1-D são similares, onde a diferença reside apenas no número de bits de entrada. As multiplicações foram decompostas em somas de deslocamentos como forma de maximizar o desempenho dos multiplicadores. Como uma das entradas é sempre uma constante, é possível prever quais serão os deslocamentos necessários para cada cálculo.

Esta solução é diferente da desenvolvida em [KOV 95], que utiliza uma árvore de Wallace [HWA 79] de seis estágios para fazer as multiplicações. A solução usada em [KOV 95] efetua as cinco multiplicações necessárias ao cálculo da DCT 1-D em 14 ciclos de *clock*. Os multiplicadores projetados nesta dissertação, usando deslocamentos e somas, usam 6 ciclos de *clock* para as mesmas cinco multiplicações, economizando 8 ciclos de *clock*.

A arquitetura desenvolvida para os multiplicadores está apresentada na fig. 3.8, na qual podem ser percebidos os quatro *barrel shifters* (responsáveis pelos deslocamentos das entradas) e os três somadores (responsáveis pelas somas dos quatro deslocamentos). Esta arquitetura foi desenvolvida para operar em um *pipeline* de dois estágios. No primeiro estágio, são efetuados os quatro deslocamentos e a soma parcial destes deslocamentos. No segundo estágio, é efetuada a soma dos resultados parciais obtidos no primeiro estágio.

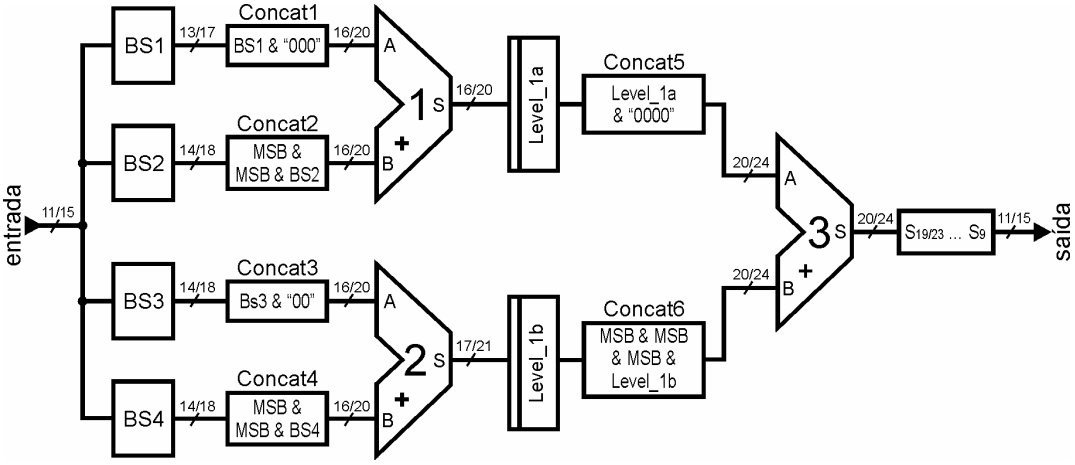


FIGURA 3.8 – Arquitetura do multiplicador

Na fig. 3.8 estão apresentados o número de bits utilizados em cada uma das partes da arquitetura do multiplicador, onde o primeiro número é referente à arquitetura do multiplicador usado na primeira DCT 1-D e o segundo número é referente à arquitetura do multiplicador usado na segunda DCT 1-D.

Os somadores presentes nos multiplicadores, como já foi dito no item anterior, são *ripple carry* e possuem os maiores operadores dentre todos os somadores utilizados nas arquiteturas de cálculo da DCT 1-D, definindo, portanto, a sua freqüência de operação. Estes somadores não geram *carry out* e operam apenas no modo de adição, não efetuando subtrações.

Como, na arquitetura proposta, são efetuadas apenas quatro somas de deslocamentos para cada multiplicação, as constantes da multiplicação têm que ser arredondadas para respeitar esta restrição. Por isso os resultados das multiplicações são aproximações dos resultados ideais. As quatro constantes utilizadas pelo multiplicador estão apresentadas na tab. 3.5, onde é possível perceber, em termos decimais, o impacto do uso de apenas quatro deslocamentos para representar estas constantes.

TABELA 3.5 – As quatro constantes usadas pelo multiplicador

Constante	Valor Decimal	Aproximação Utilizada	
		Binário	Decimal
$m1 = \cos(4\pi/16)$	0,707107	0,101101000	0,703125
$m2 = \cos(6\pi/16)$	0,382683	0,011000101	0,384766
$m3 = \cos(2\pi/16) - \cos(6\pi/16)$	0,541196	0,100010101	0,541016
$m4 = \cos(2\pi/16) + \cos(6\pi/16)$	1,306563	1,010011000	1,296875

A tab. 3.6 apresenta os quatro *barrel shifters* (BS1 a BS4) e os deslocamentos associados a cada uma das quatro constantes (*m1* a *m4*) utilizadas nas multiplicações. Na tab. 3.6, *i* é o valor de entrada e [*x*] significa que *i* é deslocado *x* bits à direita.

Os resultados finais da multiplicação são truncados e apenas os bits da parte inteira são entregues na saída dos multiplicadores. Este truncamento é necessário porque a arquitetura de cálculo da DCT 2-D processa números inteiros. Para evitar erros gerados por truncamento nas operações internas ao multiplicador, todos os bits significativos dos resultados intermediários foram considerados.

TABELA 3.6 – Deslocamentos associados a cada constante

	BS1	**BS2**	**BS3**	**BS4**
m1	*i* [1]	*i* [3]	*i [4]*	*i* [6]
m2	*i* [2]	*i* [3]	*i* [7]	*i* [9]
m3	*i* [1]	*i* [5]	*i* [7]	*i* [9]
m4	*i* [0]	*i* [2]	*i* [5]	*i* [6]

A arquitetura dos multiplicadores foi simplificada para minimizar o uso de recursos. Como as multiplicações são por constantes pode-se prever exatamente quais deslocamentos devem ser realizados em cada *barrel shifter* em cada ciclo de *clock*. Então é possível definir quantos e quais são os deslocamentos que cada *barrel shifter* deve estar apto a realizar. Desta forma os *barrel shifters* foram simplificados para executar somente os deslocamentos necessários, eliminando os deslocamentos que não são utilizados. A tab. 3.6 apresenta, em suas colunas, os deslocamentos associados a cada *barrel shifter*, de onde pode-se perceber, como exemplo, que o deslocador *BS1* efetua somente 0, 1 ou 2 deslocamentos à direita, enquanto que o deslocador *BS4* efetua somente 6 ou 9 deslocamentos, também à direita.

Considerando os deslocamentos mínimo e máximo efetuados por cada *barrel shifter*, é possível desprezar os bits que são constantes nas saídas de cada deslocador. Desta forma, é realizada mais uma simplificação na arquitetura do multiplicador, com impacto nos próprios *barrel shifters*, nos somadores do primeiro estágio e nos registradores de *pipeline*. Com esta simplificação, as saídas dos *barrel shifters* são geradas com escalas diferentes, que devem ser igualadas para que as saídas possam ser somadas. A correção na escala é realizada através de concatenações, que consistem, basicamente, da inserção de zeros à direita ou da extensão do bit de sinal, como está apresentado na fig. 3.8. A extensão do sinal, na fig. 3.8, está representada pela repetição do bit mais significativo da entrada (MSB).

Sem as simplificações, os quatro *barrel shifters* seriam idênticos e precisariam estar aptos a efetuar qualquer deslocamento, de 0 a 9 bits à direita. Com as simplificações foi possível minimizar significativamente o número de bits utilizados no controle dos deslocamentos e nas saídas dos deslocadores. Na tab. 3.7 estão apresentadas as simplificações realizadas em cada *barrel shifter* de cada multiplicador.

TABELA 3.7 – Impacto das simplificações nos *barrel shifters*

		Sem simplificação	**Com simplificação**
BS1	Bits de Controle	4	2
	Bits de Saída 1º Mult.	20	13
	Bits de Saída 2º Mult.	24	17
BS2	Bits de Controle	4	2
	Bits de Saída 1º Mult.	20	14
	Bits de Saída 2º Mult.	24	18
BS3	Bits de Controle	4	2
	Bits de Saída 1º Mult.	20	14
	Bits de Saída 2º Mult.	24	18
BS4	Bits de Controle	4	2
	Bits de Saída 1º Mult.	20	14
	Bits de Saída 2º Mult.	24	18

Com a utilização de deslocadores não simplificados os três somadores utilizados no multiplicador da primeira DCT 1-D teriam que ter palavras de 20 bits. Com as simplificações foi possível minimizar de 20 bits para 16 bits a largura de palavra dos dois somadores do primeiro estágio do *pipeline*. A mesma minimização foi possível nos registradores de *pipeline*, que sem simplificação teriam que ser de 20 bits e com a simplificação utilizam 16 e 17 bits.

No multiplicador da segunda DCT 1-D o impacto é semelhante, sendo que os somadores do primeiro nível do *pipeline*, com simplificação, utilizam 20 bits ao invés de 24 bits. Os registradores utilizam 20 e 21 bits ao invés de 24 bits.

Os blocos de controle de ambos os multiplicadores são idênticos e preocupam-se com a geração dos sinais de controle dos *barrel shifters* e com a inserção de três bolhas no *pipeline*, para sincronizar a operação do multiplicador com a operação dos demais blocos da arquitetura da DCT 1-D. O controle da DCT 1-D preocupa-se em inicializar os multiplicadores e em gerenciar as suas entradas. Desta forma, o controle do multiplicador é simples, consistindo em uma máquina de estados de oito estágios, com *reset* assíncrono. Os resultados são ordenados pelo *buffer* ping-pong ao qual o multiplicador está conectado.

A fig. 3.9 apresenta o diagrama temporal do *pipeline* do multiplicador. Nesta figura é possível perceber as três saídas com valores não utilizados pelo cálculo da DCT 1-D (saídas em X), referentes às bolhas inseridas no *pipeline*.

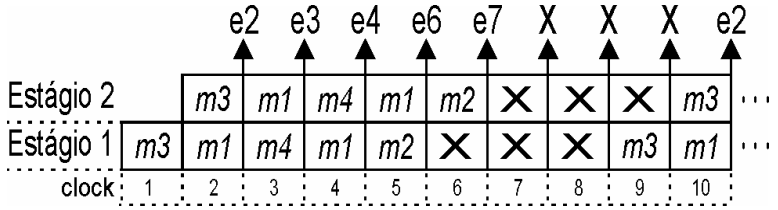


FIGURA 3.9 – Diagrama temporal do *pipeline* do multiplicador

A síntese isolada dos multiplicadores da primeira e segunda DCT 1-D utilizou, respectivamente, 231 e 307 células lógicas, atingindo uma freqüência de operação de 38 e 31,7MHz, em um dispositivo EPF10K30ETC144-1 [ALT 2001a] da Altera.

Um exemplo de operação do multiplicador, considerando uma multiplicação de um número qualquer *i* pela constante *m4* (com valor binário igual a 1,010011000), é apresentada a seguir. Temos que:

$$i \times m4 = i \times 1{,}010011000$$

que pode ser decomposto em somas de deslocamentos como segue:

$$i \times m4 = i \times 2^{0} + i \times 2^{-2} + i \times 2^{-5} + i \times 2^{-6}$$

Cada uma das parcelas a ser somada é um simples deslocamento do dado *i* de entrada. Como o sinal do expoente é negativo, temos que os deslocamentos são à direita. O número de deslocamentos à direita para cada parcela da soma é definido pelo valor do coeficiente. A tab. 3.8 apresenta os resultados das operações realizadas pelo primeiro multiplicador, incluindo as concatenações para correção da escala, onde $i = 00000011111_{b} = 31_{10}$.

No exemplo apresentado na tab. 3.8, a parte inteira do resultado da multiplicação (que efetivamente será utilizada) é exatamente igual ao resultado obtido através de um multiplicador convencional, tal qual um multiplicador matricial ou de Booth [HWA 79]. Este cálculo específico não apresentou nenhum erro na parte inteira, mas é importante salientar que pequenos erros podem ser gerados, dependendo do cálculo a ser realizado.

TABELA 3.8 – Exemplo de operação do multiplicador

Entrada $i = 00000011111_b = 31_{10}$			
Constante $m4 = 1{,}010011_b = 1{,}296875_{10}$			
BS1out 000000111110	**BS2out** 00000011111000	**BS3out** 00000001111100	**BS4out** 00000011111000
Concat1out (2^0) 0000001111110000	**Concat2out (2^{-2})** 0000000011111000	**Concat3out (2^{-5})** 0000000111110000	**Concat4out (2^{-6})** 0000000011111000
Somador1out 0000010011011000		**Somador2out** 00000001011101000	
Concat5out 000001001101100000000		**Concat6out** 0000000001011101000	
Somador3out 00000101000001101000			
Saída $e4 = 00000101000_b = 40_{10}$			

3.1.4 *Buffer* de Transposição

O *buffer* de transposição é usado para conectar as duas arquiteturas de cálculo da DCT 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 desenvolvida para o *buffer* de transposição utiliza duas pequenas memórias RAM de porta simples, como está apresentado na fig. 3.10. Optou-se pelo uso de memórias ao invés de registradores por questões específicas da implementação desejada. Como o objetivo desta dissertação é mapear para um FPGA a arquitetura completa de um compressor JPEG, o uso de memórias RAM se justifica duplamente. Primeiro, porque os dispositivos alvo deste trabalho possuem memória interna que não é completamente utilizada pelas demais partes da arquitetura, sendo, então, um recurso ocioso. Segundo, porque registradores mapeados em FPGAs utilizam muitas células lógicas, o que dificultaria o mapeamento de toda a arquitetura do compressor em um único dispositivo. Por outro lado, o uso de memória interna do FPGA, torna esta parte da descrição VHDL dependente de dispositivo e fabricante, como já foi discutido.

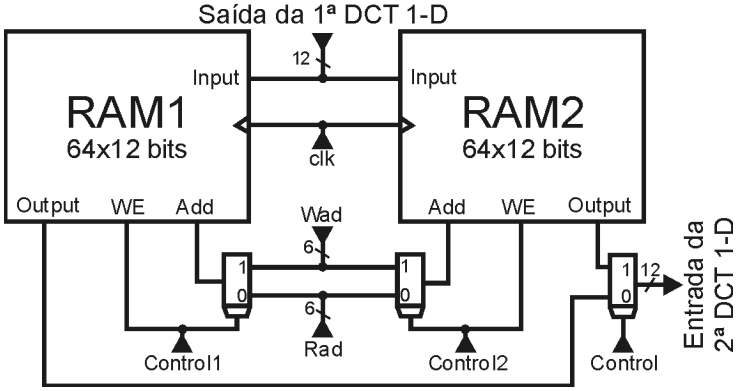


FIGURA 3.10 – Arquitetura do *buffer* de transposição

As saídas da primeira DCT 1-D, como foi justificado no anexo 4, possuem palavras de 12 bits, portanto as duas memórias terão 12 bits de comprimento de palavra.

As memórias utilizadas possuem a escrita e o endereçamento sincronizados com o *clock* do compressor e cada memória deve ser capaz de armazenar uma matriz de resultados da primeira DCT 1-D, ou seja, cada memória deve armazenar 64 elementos e, por isso, são necessários seis bits de endereçamento.

A arquitetura do *buffer* de transposição, apresentada na fig. 3.10, é bastante simples, sendo formada por duas memórias RAM, por três multiplexadores e pelo controle. Na figura, com exceção da entrada e da saída, todos os sinais apresentados são gerados pelo controle do *buffer* de transposição. Essencialmente, o controle gera os endereços de escrita (*Rad*) e de leitura (*Wad*), bem como os sinais *Control1* e *Control2*, que são usados para o controle dos multiplexadores de endereçamento e para a habilitação de escrita nas memórias e o sinal *Control*, que é usado para o controle do multiplexador de saída. Como já foi explicado anteriormente, o endereço de escrita é gerado linha a linha e o endereço de leitura é gerado coluna a coluna.

As duas memórias operam de maneira intercalada, enquanto uma é usada para a escrita, a outra é usada para leitura. Considerando as duas memórias preenchidas, a memória usada para escrita fica completamente preenchida pelos dados gerados pela primeira DCT 1-D exatamente no mesmo momento em que foi consumido, pela segunda DCT 1-D, todo o conteúdo da memória usada para leitura. Então, no próximo ciclo de *clock* as situações se invertem e a memória que estava sendo usada para leitura passa a ser usada para escrita e vice e versa. Os sinais *Control1* e *Control2*, apresentados na fig. 3.10, são os responsáveis pelo controle desta atuação intercalada, através da habilitação de escrita na memória e da conexão dos endereços de escrita e de leitura nas memórias. O sinal *Control* controla a saída do *buffer* de transposição que, ora está conectada à saída de uma memória, ora de outra.

O endereço de escrita está sempre conectado à memória que está habilitada para a escrita, enquanto que a memória que não está habilitada para a escrita recebe sempre o endereço de leitura. Quando há a inversão do modo de operação das memórias, os endereços conectados a elas também são invertidos e a memória que recebia o endereço de leitura passa a receber o endereço de escrita e vice e versa. A entrada *Input* está conectada às duas memórias simultaneamente, mas seu conteúdo só é efetivamente armazenado na memória que está habilitada para escrita.

O controle do *buffer* de transposição foi desenvolvido a partir de uma máquina de estados com 64 estados com *reset* assíncrono. Estes estados se repetem, formando um ciclo, que eqüivale à escrita e leitura de uma matriz completa de 64 elementos em cada uma das memórias. Em cada um dos 64 estados são gerados os corretos endereços de escrita e leitura, que se repetem a cada novo ciclo. Por outro lado, os sinais *Control1* e *Control2* são invertidos sempre no primeiro estado de cada ciclo de repetição da máquina de estados, realizando o intercalamento das memórias. O sinal *Control* é o sinal *Control1* atrasado em um ciclo de *clock*. O sinal *Control* é responsável pela seleção de qual saída de memória será conectada à saída do *buffer* de transposição e é gerado em atraso com relação ao sinal *Control1*, porque a saída correta da memória está disponível um ciclo após a geração do endereço de leitura, devido ao registrador de sincronismo de entrada utilizado para a memória. O controle do *buffer* de transposição também é responsável pela geração de um *flag* indicando que existe um valor válido na saída. Este *flag* é usado pelo controle global da DCT 2-D para inicializar a segunda DCT 1-D.

A latência do *buffer* de transposição é de 65 ciclos de *clock*. A síntese do *buffer* de transposição indicou que esta arquitetura, isoladamente, utilizaria 276 células lógicas, 1.536 bits de memória e seria capaz de operar a uma freqüência de 20,8MHz em um dispositivo EPF10K30ETC144-1 [ALT 2001a] da Altera.

3.1.5 Considerações Finais sobre a Arquitetura da DCT 2-D

A arquitetura da DCT 2-D foi dividida em diversos níveis hierárquicos para ser mais facilmente descrita e validada em VHDL. Os três principais níveis desta hierarquia, como já foi exposto, são: primeira DCT 1-D, *buffer* de transposição e segunda DCT 1-D. A tab. 3.9 apresenta um resumo comparativo dos resultados obtidos para a arquitetura da DCT 2-D, com relação aos três níveis hierárquicos citados. Da tab. 3.9, é possível perceber claramente que o caminho crítico da DCT 2-D é a segunda DCT 1-D, como já era esperado, uma vez que esta arquitetura é a que possui os operadores de maior largura de palavra. A síntese foi realizada para dispositivos da família Flex10KE [ALT 2001a] da Altera, utilizando, para tanto, o software MaxPlus2, que também foi utilizado para o processo de simulação da arquitetura da DCT 2-D.

TABELA 3.9 – Resultados arquiteturais da DCT 2-D

	Células Lógicas	**Bits de Memória**	**Período** (ns)	**Freqüência** (MHz)	**Latência** (nº ciclos)
1ª DCT 1-D	1.660	0	45,2	22,1	49
***Buffer* de Transposição**	276	1.536	48	20,8	65
2º DCT 1-D	2.241	0	52,6	19	49
DCT 2-D	4.181	1.536	52,7	19	163

A arquitetura de cálculo da DCT 2-D seria capaz de processar uma imagem de 640 x 480 *pixels* em 16,2ms, permitindo uma taxa de processamento de 62 imagens por segundo. Este desempenho pode ser melhorado ainda mais com a substituição dos operadores aritméticos, que são todos embasados em somadores *ripple carry*. Com a substituição destes somadores por arquiteturas mais rápidas, como de somadores *carry look ahead* [WES 95], o desempenho arquitetural iria crescer significativamente. Como a arquitetura da DCT 2-D foi construída de forma hierárquica, a substituição dos operadores é bastantes simples, não sendo necessárias alterações nos níveis superiores da hierarquia.

Foram realizadas diversas simulações com o intuito de estimular os pontos críticos da arquitetura, durante as quais, alguns erros foram descobertos e corrigidos. No anexo 5 da dissertação está a simulação completa do compressor JPEG para imagens em tons de cinza, onde a saída da DCT 2-D também é apresentada. Na simulação apresentada no anexo, a matriz de entrada faz parte de uma imagem real e foi colhida da literatura [BHA 99]. Paralelamente à simulação, o algoritmo utilizado na arquitetura da DCT 2-D, foi descrito em software e foi estimulado com a mesma matriz 8x8. Os resultados foram colhidos e comparados. A matriz de entrada foi a seguinte:

168	161	161	150	154	168	164	154
171	154	161	150	157	171	150	164
171	168	147	164	164	161	143	154
164	171	154	161	157	157	147	132
161	161	157	154	143	161	154	132
164	161	161	154	150	157	154	140
161	168	157	154	161	140	140	132
154	161	157	150	140	132	136	128

A tab. 3.10 apresenta os resultados obtidos na simulação da arquitetura comparados como os resultados obtidos através da implementação, do mesmo algoritmo, em software.

Como é possível perceber, existem diferenças entre os valores simulados e os valores calculados via software, decorrentes do fato que, no software, não foram considerados os erros gerados pelo uso de quatro somas de deslocamentos para efetuar cada multiplicação.

TABELA 3.10 – Comparação dos resultados da DCT 2-D em software e da simulação da arquitetura desenvolvida

Linha 0	**Simulado**	1716	533	-31	193	-82	-3	5	-11
	Software	1716	547	-29	184	-82	-6	3	-13
Linha 1	**Simulado**	377	-374	149	152	50	-13	90	-30
	Software	382	-383	153	164	54	-25	89	-16
Linha 2	**Simulado**	-63	-54	108	-110	27	-24	27	31
	Software	-63	-53	108	-112	27	-22	28	29
Linha 3	**Simulado**	82	-128	53	36	-141	78	30	15
	Software	78	-137	56	41	-143	73	28	18
Linha 4	**Simulado**	-100	57	-15	-15	-118	53	-23	1
	Software	-100	55	-15	-16	-118	62	-23	-5
Linha 5	**Simulado**	28	80	-63	20	27	-32	-46	9
	Software	30	83	-64	19	27	-33	-48	4
Linha 6	**Simulado**	7	-8	14	-8	5	10	-5	-5
	Software	7	-9	16	-4	5	10	-8	-7
Linha 7	**Simulado**	1	-2	1	4	8	-5	-6	-2
	Software	-2	1	-1	4	6	-3	-5	-2

É importante ressaltar que a arquitetura da DCT 2-D gera, em suas saídas, uma escala dos resultados reais de um cálculo da DCT 2-D. A correção desta escala é integrada ao cálculo da quantização, como será abordado no próximo item deste capítulo. Cada um dos 64 elementos da matriz resultado da arquitetura desenvolvida para o cálculo da DCT 2-D, terá um fator de escala próprio. Os fatores de escala das saídas das DCT 1-D são definidos como:

$$Fe(i) = \frac{C(i)}{2 \times \cos(i \times \pi/16)} \qquad \begin{cases} C(0) = 1/\sqrt{2} \\ C(i) = 1/2 \end{cases} \qquad 1 \le i \le 7$$

A matriz de fatores de escala da saída da DCT 2-D é formada a partir de combinações dos fatores de escala da saída da DCT 1-D. Esta matriz de fatores de escala é definida como:

$$Fe(ij) = Fe(i) \times Fe(j) \qquad 0 \le i, j \le 7$$

Calculando cada um dos 64 elementos gerados através da fórmula acima, é gerada a matriz de fatores de escala apresentada abaixo.

8	11,10	10,45	8	6,29	4,33	2,21
11,10	15,39	14,50	11,10	8,72	6,01	3,06
10,45	14,50	13,66	10,45	8,21	5,66	2,88
9,41	13,05	12,29	9,41	7,39	5,09	2,60
8	11,10	10,45	8	6,30	4,33	2,21
6,29	8,72	8,21	6,30	4,94	3,40	1,73
4,33	6,01	5,66	4,33	3,40	2,34	1,20
2,21	3,06	2,88	2,21	1,73	1,20	0,61

A correção na escala das saídas obtidas através da simulação da arquitetura da DCT 2-D foi simulada via software, para comparar os resultados completos da DCT 2-D com resultados colhidos na literatura [BHA 99], para a mesma matriz de entrada. Os resultados desta comparação estão apresentados na tab. 3.11, de onde pode-se perceber que as simplificações realizadas nos multiplicadores possuem pequeno impacto nos resultados da DCT 2-D. Este impacto, de acordo com o que era esperado, é praticamente todo absorvido pela arquitetura da quantização, que será apresentada no próximo item deste capítulo.

TABELA 3.11 – Comparação dos resultados da arquitetura da DCT 2-D em relação aos resultados colhidos na literatura

Linha 0	**Simulado**	214	48	-2	20	-10	0	1	-4
	Literatura	214	49	-3	20	-10	-1	1	-6
Linha 1	**Simulado**	33	-24	10	11	4	-1	14	-9
	Literatura	34	-25	11	13	5	-3	15	-6
Linha 2	**Simulado**	-6	-3	7	-8	2	-2	4	10
	Literatura	-6	-4	8	-9	3	-3	5	10
Linha 3	**Simulado**	8	-9	4	3	-14	10	5	5
	Literatura	8	-10	4	4	-15	10	6	6
Linha 4	**Simulado**	-12	5	-1	-1	-14	8	-5	0
	Literatura	-12	5	-1	-2	-15	9	-5	-1
Linha 5	**Simulado**	4	9	-7	2	4	-6	-13	5
	Literatura	5	9	-8	3	4	-7	-14	2
Linha 6	**Simulado**	1	-1	2	-1	1	2	-2	-4
	Literatura	2	-2	3	-1	1	3	-3	-4
Linha 7	**Simulado**	0	0	0	1	3	-2	-5	-3
	Literatura	-1	1	0	2	3	-2	-4	-2

O próximo item irá apresentar a arquitetura desenvolvida para o quantizador, já com a inserção do cálculo para a correção da escala da DCT 2-D.

3.2 A Arquitetura do Quantizador

A quantização, como já foi exposto anteriormente, é uma divisão inteira de todos os coeficientes da DCT 2-D por constantes. Para cada elemento da matriz 8x8 resultante do cálculo da DCT 2-D, existe uma constante específica a ser utilizada e estas constantes, em conjunto, formam a tabela de quantização. Como a arquitetura discutida neste capítulo é para imagens em tons de cinza, apenas a tabela de quantização referente ao componente de luminância é utilizada no cálculo.

A quantização é a operação onde efetivamente ocorrem perdas no processo de compressão JPEG para imagens em tons de cinza. O objetivo da quantização é gerar, como resultado, uma matriz esparsa, para potencializar a operação do codificador de entropia. A tabela de quantização utilizada para o compressor JPEG de imagens em tons de cinza está apresentada no anexo 2, tendo sido extraída do próprio padrão JPEG [THE 92]. A tabela de quantização utilizada é montada junto aos dados da imagem comprimida no arquivo JPEG e é essencial para o processo de descompressão, portanto, o processador de entrada e saída terá que utilizar exatamente a mesma tabela de quantização usada pela arquitetura.

Inserido junto ao cálculo da quantização, está o cálculo relativo aos fatores de escala, complementar ao cálculo da DCT 2-D. As operações de quantização e de correção da escala da saída da DCT 2-D são divisões por constantes, que podem ser integradas sem aumentar a complexidade do quantizador. Os elementos da matriz de fatores de escala apresentada no item anterior são multiplicados pelos elementos da matriz de quantização, gerando as 64 constantes que serão utilizadas nas divisões realizadas pelo quantizador.

Então, o quantizador irá realizar a seguinte operação:

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

onde:

- 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 multiplicação dos componentes Q_{ij} da matriz de quantização pelos componentes Fe_{ij} da matriz de fatores de escala gera a matriz apresentada abaixo.

128	122,06	104,53	150,51	192	251,42	220,81	134,64
133,16	184,69	202,97	247,91	288,50	505,66	360,32	168,38
146,34	188,47	218,51	294,98	418,10	468,11	390,32	161,50
131,70	221,81	270,40	320,78	479,76	643,02	407,28	160,91
144	244,12	386,74	526,79	544	685,13	445,95	169,95
150,85	305,14	451,69	473,03	509,13	513,61	384,39	159,55
212,15	384,34	441,24	442,92	445,95	411,61	281,18	120,65
158,92	281,66	273,97	254,35	247,21	173,42	123,04	60,29

A operação básica do quantizador proposto é a multiplicação dos componentes C_{ij} pelo componente $1/(Q_{ij} \times Fe_{ij})$. A arquitetura proposta para o quantizador,

apresentada na fig. 3.11, é formada, basicamente, por um multiplicador similar ao utilizado na arquitetura da DCT 1-D, ou seja, formado por somas de deslocamentos, onde também são somados quatro deslocamentos distintos do dado da entrada.

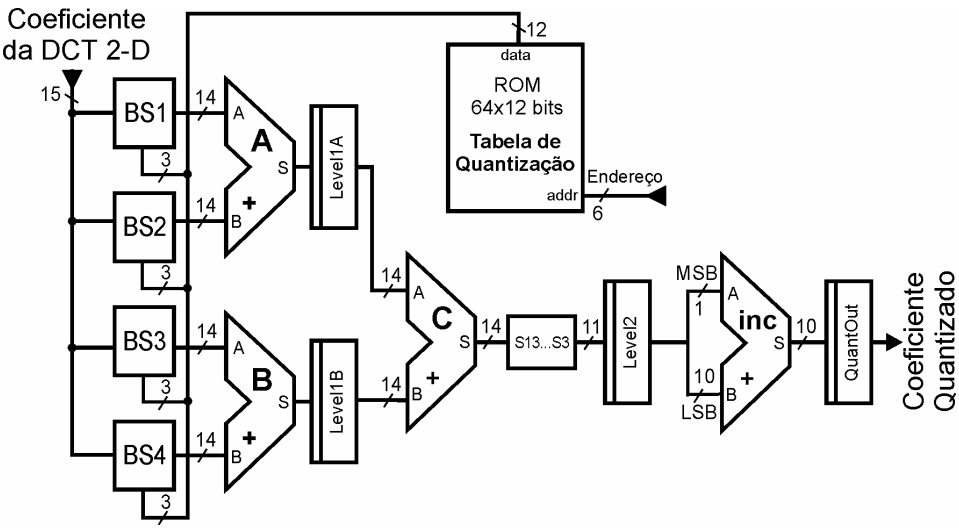


FIGURA 3.11 – Arquitetura proposta para o quantizador

A arquitetura do quantizador utiliza uma memória ROM, que ao invés de armazenar a matriz de 64 elementos gerada a partir do cálculo de $1/(Q_{ij} \times Fe_{ij})$, armazena o controle dos quatro deslocadores, indicando quais deslocamentos devem ser realizados em cada deslocador para cada elemento calculado. O conteúdo desta memória, bem como o seu relacionamento com a tabela de quantização e com a matriz de fatores de escala, está disponível no anexo 2.

Cada palavra de memória utiliza 12 bits, sendo que, cada deslocador utiliza 3 bits para controlar o número de deslocamentos, por isso, cada deslocador pode realizar até oito diferentes deslocamentos sobre o dado da entrada. O deslocador *BS1* recebe os 3 bits mais significativos dos 12 bits da saída da memória, enquanto que o deslocador *BS4* recebe os três bits menos significativos.

A entrada do quantizador possui 15 bits e a sua saída possui 10 bits. Esta redução no número de bits é conseqüência da divisão inteira realizada, sendo que, nenhum bit da parte inteira do resultado é perdido pelo quantizador.

O quantizador foi projetado em um *pipeline* de três estágios. No primeiro estágio, estão a leitura na memória ROM (que não possui sincronismo com o *clock* do compressor), os deslocamentos e as somas realizadas pelos somadores A e B. No segundo estágio é gerada, a partir do somador C, a soma dos resultados gerados pelos somadores A e B. Por fim, o resultado inteiro da quantização é gerado a partir do incrementador D, que realiza o arredondamento nos valores gerados pelo somador C, preservando apenas os bits inteiros deste resultado.

A latência da arquitetura do quantizador é de 3 ciclos de *clock* e a sua síntese utilizou 351 células lógicas e 768 bits de memória do dispositivo EPF10K30ETC144-1 da Altera. A freqüência máxima de operação do quantizador, caso fosse utilizado separadamente, seria de 21,6MHz, permitindo o processamento de uma imagem de 640 x 480 *pixels* em 14,2ms e uma taxa de processamento de 70,4 imagens por segundo.

O uso de soma de deslocamentos, da mesma forma que nos multiplicadores da DCT 2-D, faz com que as constantes utilizadas sejam uma aproximação das constantes ideais.

É importante ressaltar que a saída da DCT 2-D e, portanto, a entrada do quantizador, está organizada coluna a coluna. Desta maneira, foi preciso adequar o cálculo da quantização a este ordenamento diferenciado dos dados de entrada. Na prática, a memória ROM foi organizada com as constantes dispostas, também, coluna a coluna. Então, é possível acessar a memória seqüencialmente na mesma taxa em que os dados chegam para ser processados. A saída do quantizador é entregue coluna a coluna para a arquitetura do *buffer* ziguezague, que, ao reorganizar os dados considera que sua entrada está organizada coluna a coluna.

As mesmas simplificações realizadas no multiplicador da DCT 1-D foram realizadas no quantizador. Os deslocadores foram minimizados para gerar apenas os deslocamentos que são efetivamente necessários, sendo apresentados na tab. 3.12. Além disso, para manter a palavra da memória ROM com 12 bits, cada deslocador teve que realizar, no máximo, 8 diferentes deslocamentos, como pode ser observado na tab. 3.12. Esta restrição no número de deslocamentos realizados em cada deslocador foi possível através de uma reordenação das operações, com o objetivo de agrupar o maior número possível de deslocamentos iguais, para diferentes constantes, no mesmo deslocador. Esta reorganização está apresentada no anexo 2. Devido a esta restrição, deslocamentos maiores que 13 não são realizados.

TABELA 3.12 – Deslocamentos realizados pelos deslocadores da quantização

Controle	**Número de Deslocamentos**			
	BS1	**BS2**	**BS3**	**BS4**
0	4	zeros	zeros	zeros
1	5	6	5	7
2	6	7	8	8
3	7	8	9	9
4	8	9	10	10
5	X	10	11	11
6	X	11	12	12
7	X	12	13	13

Nos cálculos internos à quantização, foram consideradas quatro casas fracionárias. Se todos os bits fracionários tivessem sido considerados, o número de bits dos somadores e registradores teriam um significativo acréscimo. Este truncamento na quarta casa fracionária, para os cálculos internos ao quantizador, pode inserir um pequeno erro em alguns resultados. O erro gerado é praticamente irrelevante, uma vez que a divisão é inteira e que, portanto, todos os bits da parte fracionária do resultado são descartados na saída da arquitetura.

O controle do quantizador é bastante simples, consistindo na geração dos endereços para a leitura da memória ROM e na geração de um *flag* indicando que a saída do quantizador é válida. A geração dos endereços é incremental, iniciando no endereço 0 e indo até o endereço 64 e se repete até o final da imagem que está sendo processada. Por isso, os endereços foram gerados a partir de um incrementador de 6 bits, que é inicializado com zero através do sinal de *reset* da quantização.

A arquitetura do quantizador foi simulada visando a sua validação. Vários foram os estímulos gerados no intuito de sensibilizar as partes críticas da arquitetura. O exemplo de simulação utilizado para a DCT 2-D e apresentado no item anterior, foi também simulado no quantizador. Esta simulação está apresentada no anexo 5, em

conjunto com a simulação do compressor JPEG para imagens em tons de cinza, e utilizou como entrada a matriz resultante da simulação da DCT 2-D, ou seja:

1716	533	-31	193	-82	-3	5	-11
377	-118	149	152	50	-13	90	-30
-63	-54	108	-110	27	-24	27	31
82	-128	53	36	-141	78	30	15
-100	57	-15	-15	-118	53	-23	1
28	80	-63	20	27	-32	-46	9
7	-8	14	-8	5	10	-5	-5
1	-2	1	4	8	-5	-6	-2

A operação de quantização também foi realizada via software, utilizando como entrada a matriz resultante do cálculo da DCT 2-D realizado por software. Desta forma, foi possível comparar os resultados esperados com os resultados simulados e perceber o impacto que as simplificações realizadas impuseram na qualidade da matriz de saída. Os resultados colhidos da literatura [BHA 99], para o exemplo, foram idênticos aos obtidos via software. A comparação dos resultados das diferentes fontes está apresentada na tab. 3.13, onde os campos identificados como *"Soft./Lit."* apresentam os dados colhidos da simulação via software e da literatura, enquanto que os campos identificados como "Simulado" foram obtidos através da simulação da arquitetura.

TABELA 3.13 – Comparação dos resultados da quantização em software e da simulação da arquitetura desenvolvida

Linha 0	**Simulado**	13	4	0	1	0	0	0	0
	Soft./Lit.	13	4	0	1	0	0	0	0
Linha 1	**Simulado**	3	-2	1	1	0	0	0	0
	Soft./Lit.	3	-2	1	1	0	0	0	0
Linha 2	**Simulado**	0	0	0	0	0	0	0	0
	Soft./Lit.	0	0	0	0	0	0	0	0
Linha 3	**Simulado**	1	0	0	0	0	0	0	0
	Soft./Lit.	1	-1	0	0	0	0	0	0
Linha 4	**Simulado**	-1	0	0	0	0	0	0	0
	Soft./Lit.	-1	0	0	0	0	0	0	0
Linha 5	**Simulado**	0	0	0	0	0	0	0	0
	Soft./Lit.	0	0	0	0	0	0	0	0
Linha 6	**Simulado**	0	0	0	0	0	0	0	0
	Soft./Lit.	0	0	0	0	0	0	0	0
Linha 7	**Simulado**	0	0	0	0	0	0	0	0
	Soft./Lit.	0	0	0	0	0	0	0	0

A partir da tab. 3.13 é possível perceber que, para a matriz 8x8 utilizada como exemplo, apenas um resultado foi diferente, quando comparadas as saídas da arquitetura do quantizador com os resultados da simulação realizada via software ou dos dados colhidos da literatura. Da tab. 3.13 é possível concluir que, para o exemplo, os erros gerados pela decomposição das multiplicações da DCT 2-D e do quantizador, em quatro

somas de deslocamentos, foram pouco significativos e prejudicaram minimamente a qualidade da matriz resultante. Ainda assim, é importante ressaltar que esses erros podem ter maior relevância para outras matrizes de entrada. De qualquer modo, é possível afirmar que estes erros serão muito pequenos e pouco relevantes na qualidade final do processo de compressão.

O próximo item irá abordar a arquitetura do *buffer* ziguezague, responsável por ordenar os dados gerados pela quantização e entregá-los para o codificador de entropia.

3.3 A Arquitetura do *Buffer* Ziguezague

A arquitetura desenvolvida para o *buffer* ziguezague, apresentada na fig. 3.12, é muito similar à arquitetura do *buffer* de transposição utilizado na arquitetura da DCT 2-D, consistindo de duas memórias RAM que se intercalam nas operações de leitura e escrita. Dada esta grande similaridade, neste item apenas serão abordados os detalhes específicos do *buffer* ziguezague.

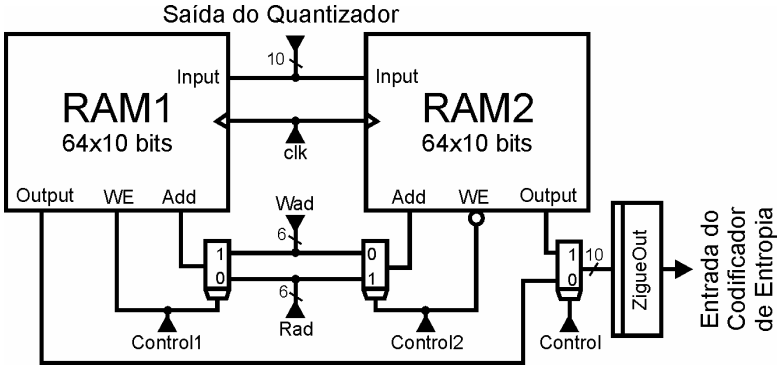


FIGURA 3.12 – Arquitetura do *buffer* ziguezague

O *buffer* ziguezague recebe como entrada os coeficientes da DCT 2-D quantizados, que estão organizados coluna a coluna, e entrega os dados para o codificador de entropia dispostos em ziguezague. Esta operação visa potencializar a codificação de entropia dos dados gerados pelo quantizador, tal qual já discutido no item 2.3.5 deste trabalho.

Na arquitetura proposta, a complexidade da operação do *buffer* ziguezague é dividida entre as operações de escrita e de leitura. A entrada, que está organizada coluna a coluna, é escrita no *buffer* ziguezague, linha a linha, e a ordenação em ziguezague é realizada através da operação de leitura. O responsável por este gerenciamento das escritas e leituras é o bloco de controle do *buffer* ziguezague, que gera os endereços para leitura e escrita na ordem correta. O bloco de controle também gera os sinais *Control1*, *Control2* e *Control* que são responsáveis pelo intercalamento nas operações de escrita e leitura das memórias.

Tanto as entradas quanto as saídas do *buffer* ziguezague possuem 10 bits e a latência do *buffer* é de 66 ciclos de *clock*.

A síntese do *buffer* ziguezague utilizou 372 células lógicas e 1.280 bits de memória de um dispositivo EPF10K30ETC144-1 [ALT 2001a] da Altera. Esta arquitetura, se utilizada em separado, seria capaz de operar a uma freqüência da 25,8MHz, processando uma imagem de 640 x 480 *pixels* em 11,9ms e permitindo uma taxa de processamento de 84 imagens por segundo.

As memórias RAM utilizadas possuem as entradas de dados e de endereços sincronizadas com o *clock* geral do compressor. A saída do *buffer* ziguezague passa por

um registrador de sincronismo, que também é utilizado como barreira temporal para o *pipeline* global do compressor.

O *buffer* ziguezague foi simulado no intuito de validar a arquitetura e identificar possíveis erros. O anexo 5 apresenta a simulação do *buffer* ziguezague operando em conjunto com o compressor JPEG para imagens em tons de cinza.

3.4 A Arquitetura do Codificador de Entropia

A codificação de entropia é a última operação na compressão JPEG, tendo como entrada os resultados da quantização organizados pelo *buffer* ziguezague. A codificação de entropia foi explicada em detalhes no item 2.3.5, no qual pode-se perceber que o processo de codificação é bastante complexo, por envolver muitos tipos de operações, por envolver manipulação de palavras com comprimento variável, por gerar assincronismo em algumas operações e por possuir um fluxo de dados bastante intrincado. Por outro lado, cada uma das operações, quando consideradas em separado, possui complexidade reduzida, por isso o codificador pode ter um elevado desempenho em termos de freqüência de operação.

A arquitetura proposta e desenvolvida para o codificador de entropia está apresentada, genericamente, na fig. 3.13. O codificador de entropia recebe como entrada valores de 10 bits gerados pelo *buffer* ziguezague. A saída do codificador de entropia consiste de palavras JPEG de 32 bits, que são entregues de forma assíncrona, como será detalhado a seguir.

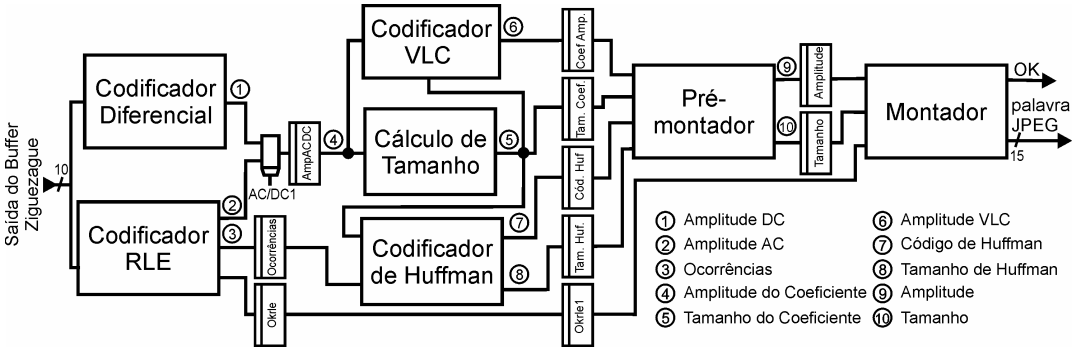


FIGURA 3.13 – Arquitetura genérica para o codificador de entropia

A arquitetura apresentada na fig. 3.13 consome um valor de entrada a cada ciclo de *clock* e opera em um *pipeline* de quatro estágios. Na fig. 3.13 estão destacados os registradores usados como barreira temporal para viabilizar o *pipeline.*

A entrada da arquitetura do codificador de entropia é síncrona, mas a sua saída é assíncrona. O assincronismo é gerado por dois motivos. O primeiro, diz respeito à arquitetura do codificador RLE, que possui uma saída assíncrona (como será visto em detalhes no item 3.4.3) propagando este assincronismo para os demais blocos da arquitetura. O impacto do assincronismo gerado pelo codificador RLE é um problema que deve ser tratado pela arquitetura do montador, que será vista no item 3.4.8. O segundo gerador de assincronismo são as codificações VLC e de Huffman, que possuem comprimento de palavra variável, sendo que também este assincronismo deve ser tratado pelo montador. Então, os 32 bits da saída do codificador de entropia são entregues a uma taxa completamente dependente do conjunto de entradas consideradas em cada caso.

A latência do codificador de entropia é dependente do assincronismo gerado pelo codificador RLE e pelo montador, sendo que a latência mínima é de 5 ciclos de *clock*. A síntese do codificador de entropia no dispositivo EPF10K30ETC144-1 da Altera utilizou 1.307 células lógicas e 3.852 bits de memória. Se o codificador de entropia fosse utilizado separadamente, poderia operar a uma freqüência de 25,6MHz e seria capaz de processar uma imagem de 640 x 480 *pixels* em 12ms, permitindo uma taxa de processamento de 83,3 imagens por segundo.

A entrada do codificador de entropia está conectada ao registrador de saída do *buffer* ziguezague. Este registrador funciona como barreira temporal entre os macro estágios do *pipeline* relativos ao *buffer* ziguezague e ao codificador de entropia. A saída do codificador de entropia é entregue de forma assíncrona e é disponibilizada diretamente na saída do compressor JPEG. A arquitetura do processador de entrada e saída, que não foi desenvolvida nesta dissertação, utilizará a saída do compressor para montar o arquivo JPEG de acordo com o formato JFIF [HAM 92] que está apresentado no anexo 1.

A seguir, serão apresentadas, em detalhes, as partes formadoras do codificador de entropia. Inicialmente é apresentado o bloco de controle, seguido das sete partes apresentadas na fig. 3.13 ou seja: codificador diferencial, codificador RLE, cálculo de tamanho, codificador de Huffman, codificador VLC, pré-montador e montador.

3.4.1 Controle do Codificador de Entropia

O controle global da arquitetura do codificador de entropia é muito simples e consiste na geração de três sinais principais: *AC/DC*, *Last* e *rst*.

O sinal *AC/DC* indica, para o primeiro estágio do *pipeline* do codificador de entropia, se o valor na entrada é AC ou DC. Este sinal é atrasado em um ciclo para ser usado no segundo estágio do *pipeline*, sendo chamado, então, de *AC/DC1*. O sinal *AC/DC* é usado para controlar os codificadores diferencial e RLE. O sinal *AC/DC1* é usado para controlar o codificador de Huffman e para selecionar a entrada do codificador VLC e do cálculo de tamanho, como pode ser observado na fig. 3.13.

O sinal *Last* indica que o valor de entrada é o último valor de uma matriz 8 x 8, ou seja, indica que o valor de entrada é o 64º elemento da matriz. Este sinal é utilizado pelo codificador RLE e está apresentado na fig. 3.13.

O sinal *rst* é responsável pela inicialização das arquiteturas dos codificadores RLE e diferencial. Este sinal é atrasado em três ciclos de *clock* para inicializar o montador, sendo, então, chamado de *rst3*. O sinal *rst* zera o conteúdo do registrador *Ant* do codificador diferencial (fig. 3.15) e dos registradores *ACC* e *Okrle* do codificador RLE (fig. 3.16), para que estes codificadores possam iniciar corretamente a sua operação. O sinal *rst3*, por sua vez, zera o conteúdo dos registradores *ACC*, *OK* e *High* quando da inicialização da arquitetura do montador (fig. 3.20). Enquanto os sinais *rst* e *rst3* forem mantidos em nível lógico baixo, as arquiteturas dos codificadores diferencial e RLE e do montador não entram em operação, mantendo todo o codificador de entropia fora de operação.

Um outro sinal de controle muito importante para a arquitetura é o sinal *Okrle*, que é um *flag* gerado pelo codificador RLE indicando que a sua saída é válida. Portanto, não é a arquitetura do controle que gera o sinal *Okrle*. Este *flag* é importante pois a operação do codificador RLE é assíncrona em relação às suas saídas, então o *flag* é usado para sincronizar a operação do codificador RLE com a operação do montador, como será detalhado nos itens 3.4.3 e 3.4.8 a seguir. O sinal *Okrle* é atrasado em um ciclo de *clock* para ser utilizado pelo montador, gerando o sinal *Okrle1*. Estes dois sinais estão apresentados na fig. 3.13, que descreve a arquitetura global.

A fig. 3.14 apresenta um trecho que exemplifica o diagrama temporal do *pipeline* do codificador de entropia, respeitando a seguinte legenda:

- DIF – Codificador diferencial em operação;
- RLE – Codificador RLE em operação;
- Huf. DC – Tabela de Huffman DC em uso;
- Huf. AC – Tabela de Huffman AC em uso;
- X – Resultado ou valor não importante;
- B – Bolha no *pipeline* (parada forçada do montador).

No trecho do diagrama temporal do *pipeline* apresentado na fig. 3.14 estão também apresentados os valores dos principais sinais de controle, bem como as entradas da arquitetura.

Montador			DC	AC	AC	B	B	AC	B	...		B	AC	DC	...
Pré-montador		DC	AC	AC	X	X	AC	X		...		X	AC	DC	AC	...
Tam./VLC/Huf.	Huf.DC	Huf.AC	Huf.AC	X	X	Huf.AC	X		...		X	Huf.AC	Huf.DC	Huf.AC	Huf.AC	...
Dif/RLE	DIF	RLE	RLE	RLE	RLE	RLE	RLE		...		RLE	RLE	DIF	RLE	RLE	RLE	...
clock	1	2	3	4	5	6	7	8	9	10...62	63	64	65	66	67	68	...
Entrada	X	≠0	≠0	0	0	≠0	0	X	X	X	0	0	X	≠0	≠0	≠0	...
sinais de controle: AC/DC	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	...
sinais de controle: Last	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	...
sinais de controle: Okrle1	X	X	1	1	1	0	0	1	0	X	0	X	0	1	1	1	...

FIGURA 3.14 – Diagrama temporal do *pipeline* do codificador de entropia

A arquitetura do codificador RLE e a arquitetura do montador, bem como as demais arquiteturas do codificador de entropia, serão explicadas nos próximos itens, apresentando mais detalhadamente a função dos sinais de controle em cada uma destas arquiteturas.

3.4.2 Codificador Diferencial

O codificador diferencial é usado apenas para os componentes DC, sendo a primeira operação realizada na codificação de entropia destes componentes. Como já foi detalhado no capítulo 2, a operação básica realizada pelo codificador diferencial é uma subtração. O resultado gerado por esta operação será chamado de *Amplitude DC*.

A arquitetura desenvolvida para o codificador diferencial, apresentada na fig. 3.15, utiliza 10 bits para a entrada e 10 bits para a saída. A arquitetura consiste de um subtrator de 10 bits e de um registrador para armazenar o código DC do bloco anterior. A escrita no registrador *Ant.* é habilitada pelo sinal *AC/DC*, que indica se o valor de entrada é um valor AC ou DC, sendo que a escrita só é permitida caso o valor seja DC.

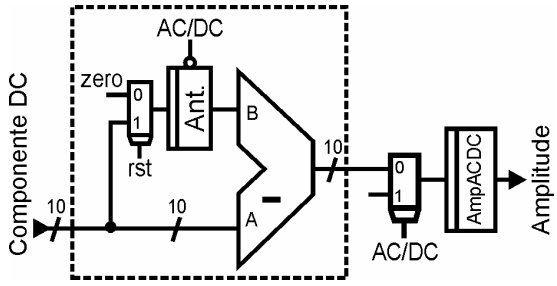


FIGURA 3.15 – Arquitetura do codificador diferencial

A área tracejada na fig. 3.15 eqüivale ao bloco chamado de *Codificador Diferencial* na fig. 3.13. O registrador *AmpACDC* é o mesmo registrador que armazena a saída do codificador RLE, estando conectado nas duas arquiteturas através do multiplexador que seleciona qual das duas saídas deve ser armazenada no registrador *AmpACDC*. Caso o sinal *AC/DC* indique que o valor atual é DC, o registrador *AmpACDC* armazena o conteúdo da saída do codificador diferencial.

A síntese isolada do codificador diferencial utilizou 28 células lógicas de um dispositivo EPF10K30ETC144-1 da Altera e poderia operar a uma freqüência de 57,5MHz.

3.4.3 Codificador RLE

A codificação RLE é realizada somente nos componentes AC da entrada, sendo simplificada para atuar como um contador de zeros, como já foi exposto no capítulo 2.

A arquitetura desenvolvida para o codificador RLE está apresentada na fig. 3.16, onde os registradores *AmpACDC***,** *Ocorrências* e *Okrle* são os mesmos apresentados na arquitetura global (fig. 3.13). A linha tracejada na fig. 3.16 delimita o bloco chamado de *Codificador RLE* na fig. 3.13.

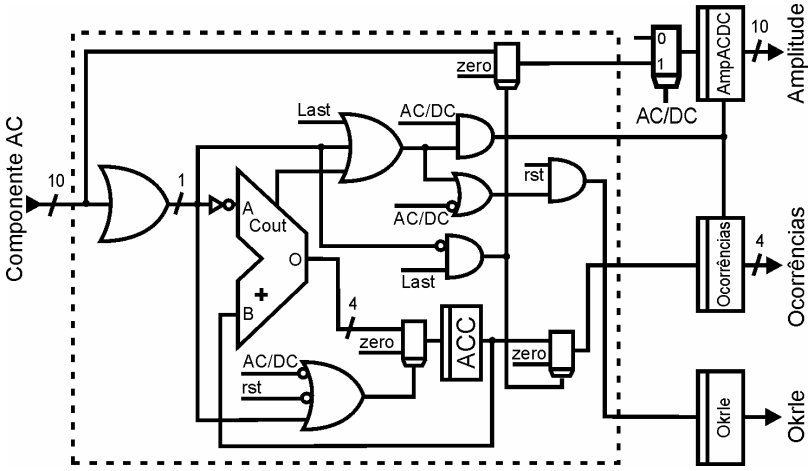


FIGURA 3.16 – Arquitetura do codificador RLE

A saída da arquitetura do codificador RLE é gerada de forma assíncrona, enquanto que os valores da entrada são consumidos de forma síncrona, a uma taxa de um valor por ciclo de *clock*.

O codificador RLE em conjunto com o controle do codificador de entropia, controlam a escrita no registrador *AmpACDC* apresentado na fig. 3.13. A escrita da saída do codificador RLE nos registradores *AmpACDC* e *Ocorrências* só é permitida quando o valor de entrada for AC e quando esta entrada for diferente de zero. Existem duas outras situações especiais onde a entrada é AC e igual a zero e a escrita nos registradores de saída é habilitada. Estas situações especiais são definidas pelo padrão JPEG e serão detalhadas nos próximos parágrafos.

A primeira situação especial é causada pela restrição existente no tamanho do campo *Ocorrências*, que tem tamanho fixo de 4 bits [THE 92]. Por isso, seqüências de mais de 16 zeros não são permitidas, então, caso existam 15 zeros seguidos de mais um zero na entrada, uma nova saída válida deve ser escrita. Assim, o registrador *Ocorrências* deve receber o valor 15 e o registrador *AmpACDC* deve receber o valor zero. Esta situação é controlada pela arquitetura proposta, através do *carry out* do contador de zeros que, quando tem valor igual a um (16º zero), habilita a escrita na

saída. Desta forma, o par "*Ocorrências / Amplitude AC*" é gerado corretamente porque a entrada é zero, sendo armazenada no registrador *AmpACDCI*. O conteúdo do registrador *ACC* é 15 e este valor é armazenado no registrador *Ocorrências*.

A segunda situação especial ocorre quando a matriz de entrada termina com uma seqüência de zeros. Neste caso, o valor escrito deve ser zero para os registradores *Ocorrências* e *AmpACDC*. Esta situação é controlada pelo sinal *Last*, apresentado na fig. 3.16. Este sinal indica que o valor de entrada é o último componente da matriz 8x8. Se este for o caso e a entrada for zero, então, os registradores de saída são zerados.

Uma entrada igual a zero, que não se enquadre nas situações especiais descritas acima, congela a escrita nos registradores *AmpACDC* e *Ocorrências*, habilitando a operação do contador de zeros e mantendo o *flag Okrle* em nível lógico baixo, o que indicará que a saída não é válida. Se um valor diferente de zero for posto na entrada, a escrita nos registradores *AmpACDC* e *Ocorrências* será habilitada, reiniciando o contador de zeros e colocando o *flag Okrle* em nível lógico alto, o que indica que uma nova saída válida está disponível. O sinal *Okrle* também é posto em nível lógico alto quando a entrada é um componente DC, possibilitando que este sinal seja usado diretamente pelo montador sem nenhuma composição adicional.

Não é possível prever quando um valor não zero irá aparecer na matriz de entrada ou quando uma das situações especiais irá ocorrer, por isso, é necessária a existência de um sinal de sincronismo que indica que o valor na saída é um novo valor válido. Este sinal está apresentado na fig. 3.13 como *Okrle*. O sinal *Okrle* é utilizado apenas pela arquitetura do montador, enquanto que as demais arquiteturas do codificador de entropia não precisam preocupar-se com o assincronismo da saída do codificador RLE.

O codificador RLE está no mesmo estágio de *pipeline* que o codificador diferencial, por isso as operações realizadas pelos dois codificadores devem consumir exatamente o mesmo número de ciclos de *clock*, para que seja possível uma ligação transparente com o estágio seguinte do *pipeline*.

A saída do registrador *AmpACDC* é chamada, neste trabalho, de *Amplitude do Coeficiente*.

A síntese isolada do codificador RLE utilizou 27 células lógicas do dispositivo EPF10K30ETC144-1 da Altera e a sua freqüência de operação poderia chegar a 61,7MHz.

3.4.4 Cálculo de Tamanho

O campo *Amplitude do Coeficiente* passa por um cálculo de tamanho, que indica quantos bits são significativos (sem contar com o bit de sinal) neste campo.

A arquitetura desenvolvida para este cálculo está apresentada na fig. 3.17. Esta arquitetura gera a saída chamada *Tamanho do Coeficiente*, que possui 4 bits. A saída *Tamanho do Coeficiente* será usada para controlar o codificador VLC e o pré-montador e como entrada para o codificador de Huffman. O *Tamanho do Coeficiente* poderia ser determinado a partir de uma tabela gravada em memória ROM, tal qual a tabela de tamanhos proposta pelo padrão JPEG [THE 92], mas optou-se pelo desenvolvimento de um circuito combinacional para realizar o cálculo, devido ao menor custo em termos de hardware.

O operador na entrada do circuito apresentado na fig. 3.17 é um decrementador, que recebe como entrada o campo *Amplitude do Coeficiente*, que será decrementado, e o bit de sinal deste campo, que irá controlar a operação do decrementador. Desta forma, se o número é negativo, o decremento é realizado, e se o número for positivo, nenhuma operação é realizada. Este decremento dos números negativos da entrada é realizado

para equalizar números positivos e negativos em termos da posição do bit mais significativo, sem considerar o bit de sinal. Esta equalização irá possibilitar a correta operação da segunda parte da arquitetura, que tem como base a comparação dos bits da saída do decrementador com o seu bit de sinal, para identificar qual é o primeiro bit significativo.

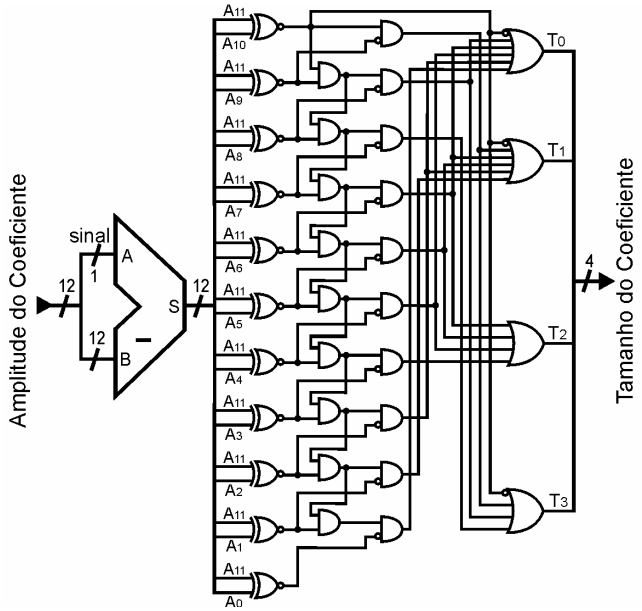


FIGURA 3.17 – Lógica para cálculo do tamanho do coeficiente

A arquitetura desenvolvida calcula o tamanho de acordo com a tabela de tamanhos proposta pelo padrão JPEG. Esta tabela prevê um tamanho máximo igual a onze, considerando doze bits de entrada. Como a saída do quantizador proposto neste trabalho possui, no máximo, dez bits, são estes dez bits que são codificados pelo codificador diferencial ou pelo codificador RLE, gerando o campo *Amplitude do Coeficiente*, que possui dez bits e é a entrada para a arquitetura de cálculo do tamanho. Então, os dois bits mais significativos da arquitetura recebem a extensão do sinal do campo *Amplitude do Coeficiente* para que existam, na entrada, os doze bits exigidos pela arquitetura. A arquitetura do cálculo de tamanho poderia ter sido simplificada para que sua entrada considerasse apenas dez bits, mas esta simplificação não foi realizada porque traria uma economia de recursos muito pequena, não tendo impacto no desempenho do codificador de entropia.

A síntese do circuito para cálculo do tamanho utilizou 33 células lógicas do dispositivo EPF10K30ETC144-1 da Altera e a sua freqüência de operação, caso fosse utilizado separadamente, poderia chegar a 47,4MHz.

3.4.5 Codificador VLC

O codificador de tamanho de palavra variável ou *Variable Length Coder* (VLC) é usado para identificar quais bits, dentre os 10 bits da *Amplitude do Coeficiente*, são significativos, com o objetivo de descartar os bits não significativos, incluindo o bit de sinal.

Os números negativos devem estar em complemento de um para serem codificados por VLC [BHA 99], por isso, na entrada do codificador há um decrementador controlado pelo bit de sinal. Se o número na entrada é negativo (em

complemento de dois) ocorre o decremento e o número passa a estar representado em complemento de um. Se o número é positivo não ocorre operação alguma.

A arquitetura proposta identifica os bits significativos e prepara o campo *Amplitude do Coeficiente* para ser montado pelo pré-montador, gerando a saída *Amplitude VLC*. Esta saída não possui um tamanho de palavra variável, pois os bits não significativos são descartados a posteriori, pela arquitetura do montador. A arquitetura do codificador VLC organiza os bits significativos para que os bits não significativos possam ser descartados pelo montador.

O codificador VLC é um *barrel shifter* controlado pelo campo *Tamanho do Coeficiente*. O valor do campo *Amplitude do Coeficiente* é deslocado para a esquerda para que o seu primeiro bit significativo ocupe a posição mais significativa da palavra.

A tab. 3.14 apresenta o número de deslocamentos à esquerda para cada valor do campo *Tamanho do Coeficiente* e, como pode ser observado, o deslocamento mínimo para a esquerda realizado pelo codificador VLC é igual a um. Isto significa que, pelo menos o bit de sinal está sendo descartado, independentemente de qual é o elemento de entrada. Então, a saída terá um bit a menos em relação à entrada, ou seja, a saída possui 9 bits.

TABELA 3.14 – Deslocamentos para a esquerda gerados pelo VLC para cada valor do campo Tamanho do Coeficiente

Tamanho	Nº de deslocamentos
0	10
1	9
2	8
3	7
4	6
5	5
6	4
7	3
8	2
9	1

A síntese do codificador VLC utilizou 75 células lógicas do dispositivo EPF10K30ETC144-1 da Altera, podendo operar a uma freqüência de 41,3MHz, caso fosse utilizado em separado.

3.4.6 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. A arquitetura proposta utiliza as tabelas de Huffman estáticas propostas pelo padrão JPEG [THE 92]. A arquitetura do codificador de Huffman está apresentada na fig. 3.18.

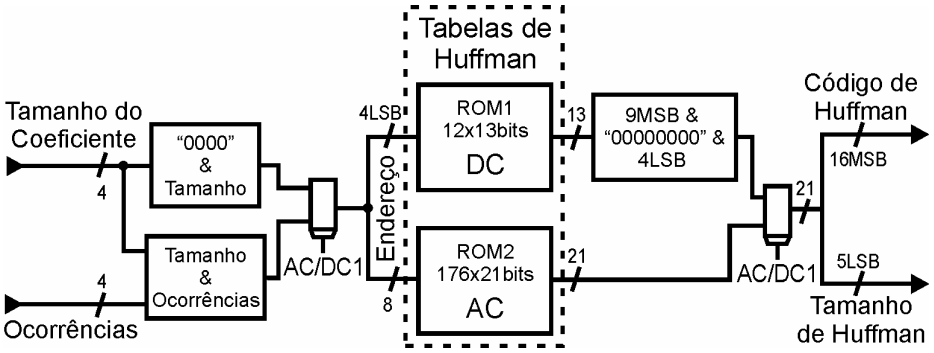


FIGURA 3.18 – Codificador de Huffman

Como o compressor JPEG, que está sendo apresentado neste capítulo, visa a compressão de imagens em tons de cinza, o codificador de Huffman utiliza apenas duas tabelas de Huffman: uma para os elementos DC e outra para os elementos AC. As tabelas de Huffman foram projetadas utilizando memórias ROM internas ao FPGA. Os próprios valores a serem codificados por Huffman são usados como endereços para estas memórias.

Cada posição de memória contém, além do código de Huffman, o tamanho deste código, em números de bits. O tamanho do código de Huffman é necessário para a montagem das palavras JPEG e poderia ter sido calculado através de uma arquitetura similar à proposta para o *Cálculo de Tamanho* (fig. 3.17). Optou-se por armazenar os tamanhos na memória porque os tamanhos de código são conhecidos para todos os códigos de Huffman, uma vez que estão sendo usadas tabelas de Huffman estáticas e que isto evita o atraso que seria causado por um novo cálculo de tamanho. Então, a saída das memórias e do próprio codificador de Huffman será formada por dois campos: o *Código de Huffman* e o *Tamanho de Huffman*.

As memórias ROM utilizadas na arquitetura do codificador de Huffman operam de maneira assíncrona, isto é, não foram utilizados os registradores de sincronismo para os endereços, para a entrada e para a saída. No mesmo ciclo de *clock* em que os valores *Tamanho do Coeficiente* e *Ocorrências* são disponibilizados na entrada, o *Código de Huffman* e o *Tamanho de Huffman*, referentes a estas entradas, são entregues na saída.

O desenvolvimento desta arquitetura buscou a minimização dos recursos utilizados para o armazenamento das tabelas de Huffman e para o seu uso. Neste sentido, a geração de endereços foi simplificada para usar diretamente os valores a serem codificados como endereços para as memórias. Ainda em termos de geração de endereços, ao invés de usar o par *Ocorrências / Tamanho* (proposto pelo padrão JPEG) como endereço na tabela de Huffman AC, utilizou-se o par invertido *Tamanho / Ocorrências*, refletindo esta inversão no conteúdo da tabela de Huffman. Esta inversão minimizou o número de palavras utilizadas pela tabela de Huffman AC, considerando o uso do par como endereço para esta memória. Caso o par proposto pelo padrão fosse utilizado, várias posições de memória reservadas não seriam usadas. Este enfoque diferenciado em relação ao padrão não implica em nenhuma incompatibilidade, uma vez que a diferença reside no processo e não no produto, no caso as palavras JPEG, que são idênticas às propostas pelo padrão. Ainda assim, 14 posições de memória não são utilizadas. Sem a inversão, seriam 90 as posições não utilizadas.

Os códigos de Huffman referentes aos elementos AC são em número de 162, sendo usadas 176 posições de memória para esta tabela, com 8 bits de endereço. Por outro lado, são 12 os códigos relativos aos elementos DC, com 12 posições de memória utilizadas e 4 bits de endereço.

A tabela de Huffman DC utiliza 9 bits para os códigos de Huffman e 4 bits para o tamanho do código. A tabela AC usa 16 bits para o código de Huffman e 5 para o tamanho do código.

Como o tamanho dos códigos de Huffman é variável, optou-se por colocar o primeiro bit significativo de cada código no bit mais significativo de cada palavra de memória, de forma similar ao que ocorre na codificação VLC, simplificando o processo de montagem das palavras JPEG.

Os conteúdos das memórias ROM e seus respectivos endereços estão apresentados no anexo 3 desta dissertação, de onde é possível visualizar os códigos de Huffman e os seus respectivos tamanhos, organizados de acordo com os endereços gerados. Neste anexo, estão apresentados os conteúdos das memórias relativas às tabelas de Huffman AC e DC dos componentes de luminância, que são usados para a codificação de imagens em tons de cinza, e dos componentes de crominância que, junto aos componentes de luminância, são usados para a codificação de imagens coloridas, conforme será apresentado no próximo capítulo.

Como as saídas das duas memórias possuem tamanhos de palavra diferentes são inseridos 8 zeros, entre o código e o tamanho, na saída da memória que contém a tabela de Huffman DC, padronizando o número de bits na saída do codificador de Huffman.

A seleção de qual saída de memória deve ser entregue na saída do codificador de Huffman é feita por um multiplexador controlado pelo sinal *AC/DC1*, que indica se o valor que está sendo processado é AC ou DC.

Os dois campos gerados pelo codificador de Huffman são entregues para o pré-montador e são obtidos através de uma simples separação entre os 21 bits da saída do multiplexador. Os 16 bits mais significativos formam o *Código de Huffman*, enquanto que os 5 bits menos significativos formam o *Tamanho de Huffman*.

É importante salientar que, do mesmo modo que no codificador VLC, os códigos de Huffman são entregues na saída com um tamanho fixo de palavra, no caso 16 bits. Os bits não significativos, dos 16 bits entregues, são eliminados pela arquitetura do pré-montador.

A síntese do codificador de Huffman utilizou 21 células lógicas e 3.852 bits de memória de um dispositivo EPF10K30ETC144-1 da Altera. O codificador de Huffman, se fosse usado isoladamente, poderia operar a uma freqüência de 58,8MHz.

3.4.7 Pré-Montador

A arquitetura do pré-montador, apresentada na fig. 3.19, recebe quatro valores na entrada, gerados a partir dos blocos já explicados: *Amplitude VLC*, *Tamanho do Coeficiente*, *Código de Huffman* e *Tamanho de Huffman*. Na saída, são entregues dois valores para o montador: *Amplitude* e *Tamanho*.

Os bits da *Amplitude VLC* são deslocados para a direita pelo *barrel shifter BSA*, que é controlado pelo *Tamanho de Huffman*. Nenhum bit é perdido nesta operação.

Este valor deslocado é montado com o *Código de Huffman* por uma operação lógica *ou*, sendo que o *Código de Huffman* é concatenado com zeros à direita, que servem de máscara para os valores deslocados da *Amplitude do Coeficiente*. Esta operação de montagem preserva apenas os bits significativos do código de Huffman, realizando, de fato, a codificação de comprimento variável referente à codificação de Huffman. O resultado da operação lógica *ou* é a saída *Amplitude* que será entregue ao montador.

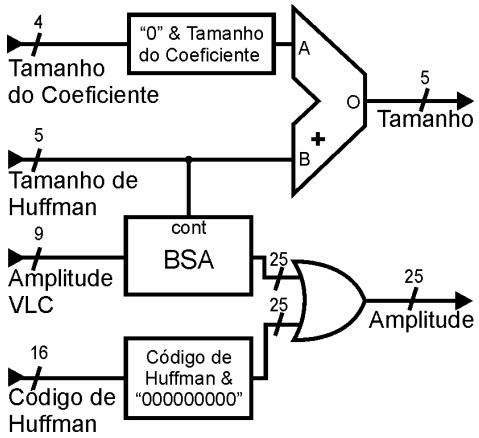


FIGURA 3.19 – Arquitetura do pré-montador

O número de bits significativos do campo *Amplitude* é dado pela soma do *Tamanho de Huffman* com o *Tamanho do Coeficiente*. Esta soma gera a saída *Tamanho*.

A saída *Amplitude* utilizará 25 bits, uma vez que o tamanho máximo de um código de Huffman é de 16 bits e o tamanho máximo de um coeficiente é de 9 bits. Então, para representar os 25 distintos tamanhos de amplitude, a saída *Tamanho* utilizará 5 bits.

A síntese do pré-montador, quando considerada isoladamente, utilizou 223 células lógicas do dispositivo EPF10K30ETC144-1 da Altera, podendo operar a uma freqüência de 33MHz.

3.4.8 Montador

A montagem final das palavras JPEG é realizada pela arquitetura do montador. Esta arquitetura, apresentada na fig. 3.20, monta as palavras JPEG considerando apenas os bits significativos do valor *Amplitude*, efetivando a codificação de comprimento variável realizada pelo codificador VLC.

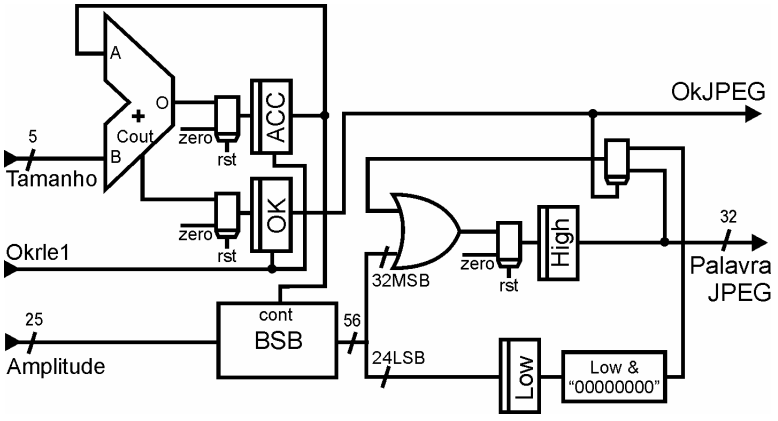


FIGURA 3.20 – Arquitetura do montador

O campo *Tamanho* gerado pelo pré-montador indica quantos bits, dentre os 25 bits da *Amplitude*, são bits significativos. A arquitetura do montador foi inspirada na arquitetura proposta por [LEI 91].

A operação realizada pelo montador é similar à operação realizada pelo pré-montador, com a diferença que, no montador, as operações são realizadas com realimentação para tornar possível a montagem de palavras completas. O montador

consiste de um *barrel shifter* (*BSB*), controlado pela acumulação dos valores do campo *Tamanho*, e de uma operação lógica *ou*, para montar e agrupar os bits significativos das diferentes entradas. A montagem das palavras é controlada por um somador que acumula os diferentes tamanhos das amplitudes de entrada através do registrador *ACC*.

O montador usa um par de registradores para realizar a montagem final das palavras JPEG. O registrador *High* armazena os 32 bits mais significativos da saída do *barrel shifter BSB* e quando 32 bits significativos estão montados, um novo valor válido é disponibilizado na saída. O registrador *Low* é usado para armazenar o *overflow*, quando o valor gerado pelo *barrel shifter BSB* tiver mais que 32 bits significativos. Este *overflow* é armazenado no registrador *High* quando uma nova palavra JPEG começa a ser montada.

O tamanho máximo da entrada *Amplitude* é de 25 bits e o maior deslocamento possível gerado pelo *barrel shifter* é de 31 bits. Portanto, para que nenhum bit seja perdido, a saída do *barrel shifter* deve possuir 56 bits. Destes 56 bits, 32 bits são usados para a operação lógica *ou*, cujo resultado será armazenado no registrador *High* e os 26 bits restantes são relativos ao *overflow*, que é armazenado no registrador *Low*.

O *flag OkJPEG* indica que uma nova palavra JPEG válida está pronta na saída do montador e, por conseqüência, na saída da arquitetura do codificador de entropia e da própria arquitetura do compressor JPEG. O sinal *OkJPEG* também é utilizado para controlar o multiplexador que define qual será a entrada da operação *ou*. Se a palavra JPEG está pronta, então, uma nova palavra deve começar a ser construída e o valor armazenado no registrador *Low* deve ser utilizado na operação *ou*. Caso contrário, será utilizado o conteúdo do registrador *High*.

O montador possui entradas assíncronas devido ao codificador RLE, como já foi explicado, mas diferentemente das arquiteturas do cálculo de tamanho, do codificador VLC, do codificador de Huffman e do pré-montador, onde o assincronismo da entrada não influencia no resultado das operações, o assincronismo na entrada influencia no resultado da montagem por causa da realimentação que existe no somador que controla os deslocamentos. Então, é necessária a existência de um sinal de sincronismo com o codificador RLE, como já foi discutido anteriormente. Este sinal está apresentado na fig. 3.20 como *Okrle1* e indica para o montador se a próxima entrada é válida ou não e, caso não seja válida, a escrita nos registradores *ACC* e *OkJPEG* fica desabilitada, paralisando o processo de realimentação até que uma nova entrada válida esteja disponível.

Como o codificador RLE está no primeiro estágio do *pipeline* e o montador está no último estágio, o sinal *Okrle* gerado pelo codificador RLE é atrasado em um ciclo, gerando o *Okrle1*, tal qual pode ser observado na arquitetura global do codificador, apresentada na fig. 3.13.

A síntese isolada do montador, para um dispositivo EPF10K30ETC144-1 da Altera, utilizou 858 células lógicas e poderia operar em uma freqüência de 22,2MHz.

3.4.9 Considerações Finais sobre a Arquitetura do Codificador de Entropia

O codificador de entropia foi completamente descrito em VHDL, sintetizado para dispositivos da família Flex10KE [ALT 2001a] da Altera e validado através de simulações. A síntese do codificador de entropia, quando isolado dos demais blocos do codificador JPEG, permitiria o processamento de uma imagem de 640 x 480 *pixels* em 12ms e uma taxa de processamento de 83,3 imagens por segundo para este tamanho de imagem. A tab. 3.15 apresenta um resumo da síntese dos diversos blocos integrantes da arquitetura do codificador.

TABELA 3.15 – Resultados arquiteturais do codificador de entropia

	Células Lógicas	**Bits de Memória**	**Período** (ns)	**Freqüência** (MHz)
Codificador Diferencial	28	0	17,4	57,5
Codificador RLE	27	0	16,2	61,7
Cálculo do Tamanho	33	0	21,1	47,4
Codificador de Huffman	21	3852	17	58,8
Codificador VLC	75	0	24,2	41,3
Pré-montador	223	0	30,3	33
Montador	858	0	45	22,2
Codificador de Entropia	1307	3852	39,1	25,6

Da tab. 3.15 pode-se perceber que a arquitetura mais lenta, dentre as arquiteturas do codificador de entropia, é a do montador. Esta arquitetura está alocada no quarto estágio do *pipeline*, sendo responsável pela definição do *clock* geral do codificador.

O anexo 5 apresenta a simulação do codificador de entropia, realizada no escopo da simulação do compressor JPEG para imagens em tons de cinza, considerando, como entrada, a matriz apresentada abaixo, que foi gerada pelo quantizador (apresentado no item 3.2) e ordenada em ziguezague.

13	4	3	0	-2	0	1	1
0	1	-1	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

A tab. 3.16 apresenta os resultados da codificação de entropia obtidos através da simulação da arquitetura desenvolvida. A primeira coluna indica se o elemento que está sendo processado é AC ou DC. A segunda coluna apresenta, para a palavra que será montada, qual a amplitude que será considerada, ou seja, é o próprio coeficiente quantizado, que irá ser codificado pelo codificador de entropia. A terceira coluna apresenta o par *Ocorrências/Tamanho*, para cada uma das amplitudes da entrada, onde o primeiro número indica quantos zeros precedem a amplitude e o segundo número indica qual o tamanho, em número de bits, desta amplitude. O elemento DC não possui o par *Ocorrências/Tamanho*, possuindo apenas o campo *Tamanho*.

A quarta e a quinta colunas da tab. 3.16 apresentam a codificação de entropia para cada entrada considerada, sendo que a quarta coluna apresenta o código de Huffman referente ao par *Ocorrências/Tamanho*, para elementos AC, ou ao componente *Tamanho*, para elementos DC, e a quinta coluna apresenta o código VLC para cada uma das amplitudes.

TABELA 3.16 – Códigos gerados pela codificação de entropia para a simulação realizada

	Amplitude	Ocorrências/ Tamanho	Codificação de Entropia	
			Huffman	VLC
DC	13	4	101	1101
AC	4	0/3	100	100
AC	3	0/2	01	11
AC	-2	1/2	11011	01
AC	1	1/1	1100	1
AC	1	0/1	00	1
AC	1	1/1	1100	1
AC	-1	0/1	00	0
AC	1	2/1	11100	1
AC	0	15/0 (ZRL)	11111111001	
AC	0	15/0 (ZRL)	11111111001	
AC	0	15/0 (ZRL)	11111111001	
AC	0	0/0 (EOB)	1010	

A tab. 3.17 apresenta a montagem das palavras JPEG a partir da concatenação dos códigos de Huffman e VLC apresentados na tab. 3.16. Os dados que estão em negrito representam a parte da palavra JPEG que já está montada. Quando a palavra JPEG está completa, o sinal *OkJPEG* recebe nível lógico alto, indicando a existência de uma nova palavra válida na saída. As palavras completas estão destacadas com fundo cinza na tab. 3.17.

TABELA 3.17 – Montagem das palavras JPEG

Palavra JPEG	OkJPEG
101110100000000000000000000000000	0
1011101100100000000000000000000	0
10111011001000111000000000000000	0
10111011001000111110110100000000	0
10111011001000111110110111001000	0
10111011001000111110110111001001	**1**
1100100000000000000000000000000	0
1100100000000000000000000000000	0
11001000111001000000000000000000	0
110010001110011111111110010000000	0
11001000111001111111110011111111	**1**
100111111111001000000000000000000	0
100111111111001101010000000000000*	0

* palavra incompleta

A codificação de entropia é a última etapa na compressão JPEG, tendo sido proposta, descrita em VHDL e validada. O próximo item deste capítulo irá tecer as considerações finais sobre o compressor JPEG para imagens em tons de cinza.

3.5 Considerações Finais sobre o Compressor JPEG para Imagens em Tons de Cinza

O compressor JPEG para imagens em tons de cinza foi particionado em quatro blocos principais, no intuito de facilitar o seu desenvolvimento. Estes quatro blocos, DCT 2-D, quantizador, *buffer* ziguezague e codificador de entropia, foram detalhadamente descritos neste capítulo. A tab. 3.18 apresenta um resumo dos resultados de síntese, na qual estão presentes os dados relativos a cada um dos quatro blocos, bem como os dados da síntese de todo o compressor.

A partir da tab. 3.18 é possível perceber, como já era esperado, que o bloco da DCT 2-D é o mais lento dentre os quatro blocos, definindo a freqüência máxima de operação do compressor. Além disso, a DCT 2-D utiliza mais que duas vezes mais células lógicas do que todas as outras arquiteturas em conjunto. A DCT 2-D é responsável por dois terços da latência global do compressor JPEG.

TABELA 3.18 – Resumo da síntese do compressor JPEG para imagens em tons de cinza

	Células Lógicas	**Bits de Memória**	**Período** (ns)	**Freqüência** (MHz)	**Latência** (nº ciclos)
DCT 2-D	4181	1536	52,7	19	163
Quantizador	351	768	46,3	21,6	3
***Buffer* Ziguezague**	372	1280	38,7	25,8	66
Codificador de Entropia	1307	3852	39,1	25,6	5*
Compressor JPEG	6199	7436	60,1	16,6	237*

* latência mínima

A síntese do compressor JPEG utilizou 6.199 células lógicas e 7.436 bits de memória do dispositivo EPF10K130EQC240-1 [ALT 2001a] da Altera, sendo utilizado 93% das células lógicas e 11% dos bits de memória disponíveis neste dispositivo. Foram utilizados 10 pinos de entrada e 33 pinos de saída. A freqüência máxima atingida por esta arquitetura foi de 16,6MHz, permitindo que uma imagem de 640 x 480 *pixels* seja completamente processada em 18,5ms. O compressor desenvolvido pode atingir uma taxa de processamento de 54 imagens de 640 x 480 *pixels* por segundo.

O compressor JPEG foi simulado, considerando a síntese do compressor completo, no intuito de validar a sua operação, incluindo cada um dos quatro blocos principais. Foram várias as simulações realizadas, dentre as quais, uma está apresentada no anexo 5 desta dissertação.

A seguir, será apresentada uma síntese dos resultados extraídos da simulação do compressor, que se encontra, na íntegra, no anexo 5. A matriz de entrada para o compressor e, por conseqüência, para o cálculo da DCT 2-D, está apresentada abaixo.

168	161	161	150	154	168	164	154
171	154	161	150	157	171	150	164
171	168	147	164	164	161	143	154
164	171	154	161	157	157	147	132
161	161	157	154	143	161	154	132
164	161	161	154	150	157	154	140
161	168	157	154	161	140	140	132
154	161	157	150	140	132	136	128

As saídas da arquitetura da DCT 2-D, considerando esta matriz de entrada, está apresentada a seguir. É importante lembrar que este resultado está ordenado coluna a coluna e que o resultado é uma escala do resultado real da DCT 2-D.

1716	533	-31	193	-82	-3	5	-11
377	-118	149	152	50	-13	90	-30
-63	-54	108	-110	27	-24	27	31
82	-128	53	36	-141	78	30	15
-100	57	-15	-15	-118	53	-23	1
28	80	-63	20	27	-32	-46	9
7	-8	14	-8	5	10	-5	-5
1	-2	1	4	8	-5	-6	-2

A arquitetura do quantizador recebe como entrada a saída do cálculo da DCT 2-D, gerando a matriz de resultados apresentada abaixo. Salienta-se que além do cálculo da quantização propriamente dita, está inserida no cálculo do quantizador, a correção da escala dos resultados da arquitetura da DCT 2-D. Novamente os dados estão organizados coluna a coluna.

13	4	0	1	0	0	0	0
3	-2	1	1	0	0	0	0
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

A matriz gerada pelo quantizador é entregue ao *buffer* ziguezague, que recebe os dados ordenados coluna a coluna e os entrega organizados em ziguezague. A matriz abaixo apresenta as saídas do *buffer* ziguezague.

13	4	3	0	-2	0	1	1
0	1	-1	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Por fim, a matriz gerada pelo *buffer* ziguezague é entregue ao codificador de entropia, que gera as palavras JPEG apresentada na tab. 3.19.

TABELA 3.19 – Codificação de entropia para o exemplo simulado

Ordem	Palavra JPEG
1	**1011101100100011111010110111001001**
2	**1100100011100111111111001111111**
3*	**100111111111001101**0XXXXXXXXXXXX

*palavra incompleta (apenas 19 bits montados)

A matriz 8x8 de entrada utiliza 8 bits para cada um dos seus 64 elementos, portanto, são utilizados 512 bits para qualquer matriz de entrada. A taxa de compressão obtida é dependente da matriz de entrada, sendo que, para o exemplo acima, foram utilizados 83 bits para representar a matriz de entrada, após o processo de compressão. Então, para o exemplo, foi obtida uma taxa de compressão de 6,2 vezes ou de 83,8 %. Esta taxa de compressão elevada é relativa à matriz utilizada como exemplo, sendo que, para outras matrizes, esta taxa pode ser menor. É importante perceber que o arquivo JPEG possui, além dos dados da imagem comprimida, as tabelas de quantização e de Huffman utilizadas na compressão, bem como outros dados necessários ao processo de descompressão, como está apresentado no anexo 1, por isso, a taxa de compressão calculada não é a taxa efetiva obtida no processo completo de compressão, que compara o número de bits utilizados pelo arquivo da imagem de entrada com o número de bits utilizados pela arquivo da imagem comprimida.

A arquitetura do compressor JPEG para imagens em tons de cinza atingiu resultados animadores, tanto em termos de recursos consumidos, quanto em termos de desempenho, sendo perfeitamente adequado ao uso nas aplicações motivadoras deste trabalho.

Para obter um desempenho ainda mais elevado, uma alternativa simples e que surtiria efeitos significativamente positivos seria a substituição dos somadores *ripple carry* por somadores mais rápidos, como *carry look ahead* [WES 95]. Infelizmente, esta exploração arquitetural não foi realizada no escopo deste trabalho, sendo uma interessante investigação para futuros trabalhos.

4 A Arquitetura do Compressor JPEG para Imagens Coloridas

A compressão JPEG de imagens coloridas, de maneira genérica, pode ser dividida em cinco blocos principais de acordo com a fig. 4.1: conversão do espaço de cores, *downsampling*, DCT 2-D, quantização e codificação de entropia, onde a imagem de entrada está no espaço de cores RGB.

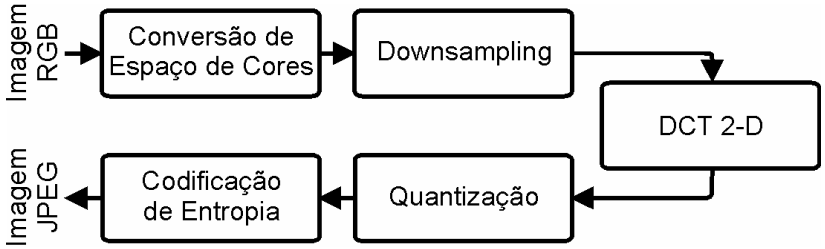


FIGURA 4.1 – Compressão JPEG de imagens coloridas

As duas primeiras operações apresentadas na fig. 4.1, conversão de espaço de cores e *downsampling*, não são obrigatórias na compressão JPEG para imagens coloridas. Por definição, a compressão JPEG possui, obrigatoriamente, as operações de DCT 2-D, quantização e codificação de entropia [THE 92] [BHA 99]. Se a imagem de entrada já está no espaço de cores YCbCr, não é necessária a operação de conversão de espaço de cores e, se a taxa de *downsampling* for de 1:1:1 entre os componentes Y, Cb e Cr, então a operação de *downsampling* não é usada. A operação de *downsampling* pode, ainda, ser transferida para o processador de entrada e saída, que é o responsável por disponibilizar os dados na entrada da arquitetura. Esta alternativa gera um pequeno impacto na complexidade da operação do processador de entrada e saída, sendo essa a solução adotada pelas arquiteturas desenvolvidas nesta dissertação.

Então, o compressor JPEG para imagens coloridas pode ficar restrito às mesmas três operações presentes no compressor para imagens em tons de cinza, ou seja: DCT 2-D, quantização e codificação de entropia. Embora as operações sejam as mesmas, os operadores devem sofrer algumas adaptações, como ficará claro no decorrer deste capítulo, uma vez que a entrada é uma imagem colorida no espaço de cores YCbCr, como está apresentado na fig. 4.2.

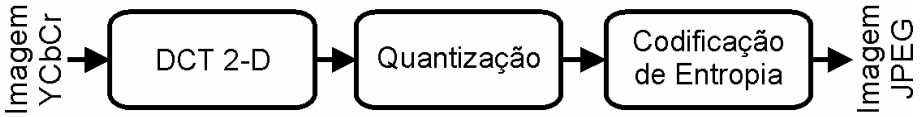


FIGURA 4.2 – Compressão JPEG simplificada para imagens coloridas

Este capítulo será dividido em dois itens principais. O primeiro item irá abordar o desenvolvimento da arquitetura do compressor JPEG para imagens coloridas, sem considerar a operação de conversão de espaço de cores. Esta arquitetura foi completamente desenvolvida, descrita em VHDL, sintetizada e simulada e os dados obtidos neste processo são apresentados. Como as operações realizadas pela compressão JPEG, abordada neste item, são as mesmas da compressão JPEG para imagens em tons

de cinza, a arquitetura desenvolvida no capítulo 3 foi reaproveitada, sofrendo alguns ajustes, para ser utilizada na compressão de imagens coloridas.

O segundo item do capítulo irá abordar a arquitetura do compressor JPEG considerando a operação de conversão do espaço de cores. O bloco de três operações desenvolvido no primeiro item é utilizado sem alterações no segundo item, sendo que a operação de conversão de espaço de cores deve ser anexada antes deste bloco. A arquitetura proposta para a conversão de espaço de cores foi completamente desenvolvida, descrita em VHDL, sintetizada e simulada. Então, neste item, teremos dois blocos principais: conversor do espaço de cores e compressor JPEG para imagens coloridas. Os dois blocos foram completamente desenvolvidos, mas não foram interligados. Algumas propostas para a integração destes dois blocos são também apresentadas.

Para as duas arquiteturas principais apresentadas neste capítulo, cada componente de cor, de cada *pixel* da imagem de entrada, possui 8 bits de precisão, em um total de 24 bits para cada *pixel*. Desta forma, para um mesmo tamanho de imagem de entrada, são processados três vezes mais dados pelo compressor JPEG para imagens coloridas do que pelo compressor JPEG para imagens em tons de cinza.

O processador de entrada e saída seleciona os *pixels* da imagem de entrada que devem ser disponibilizados para a arquitetura do compressor. A operação de *downsampling*, anteriormente definida, é realizada pelo processador de entrada e saída que, de maneira intercalada, deve entregar para a arquitetura do compressor quatro matrizes 8x8 relativas ao componente de cor Y, seguidas de uma matriz 8x8 relativa ao componente Cb e de uma relativa ao componente Cr.

A síntese das arquiteturas desenvolvidas neste capítulo foi direcionada a dispositivos da família Flex10KE [ALT 2001a] da Altera, tendo sido realizada através da ferramenta Maxplus2. As simulações também foram realizadas através desta ferramenta.

4.1 Compressor JPEG para Imagens Coloridas sem Conversão de Espaço de Cores

A arquitetura do compressor JPEG, desenvolvida neste item, recebe como entrada componentes de cor de imagens coloridas no espaço de cores YCbCr. O processador de entrada e saída, não desenvolvido nesta dissertação, é responsável por extrair uma janela de 8x8 elementos da matriz do componente de cor que será processado. Esta janela de 64 elementos é a entrada para a arquitetura do compressor JPEG para imagens coloridas.

As quatro arquiteturas que foram desenvolvidas para o compressor JPEG de imagens em tons de cinza, DCT 2-D, quantizador, *buffer* ziguezague e codificador de entropia, foram adaptadas para o processamento de imagens coloridas. As arquiteturas da DCT 2-D e do *buffer* ziguezague não precisam sofrer nenhum tipo de alteração, uma vez que o processamento, do ponto de vista destas arquiteturas, é exatamente o mesmo para os dois tipos de compressão. A operação de deslocamento de nível, que é anexada à arquitetura da DCT 2-D no compressor para imagens em tons de cinza, deve sofrer uma pequena alteração, pois apenas os componentes de luminância devem passar por este deslocamento de nível. Os componentes de luminância possuem um valor médio de 128, que é reduzido para zero através do deslocador de nível, enquanto que os componentes de crominância já possuem o valor médio em zero [BHA 99]. Desta forma, o inversor utilizado para o bit mais significativo da entrada, realizando o

deslocamento de nível, deve ser convertido a um inversor controlado, que só inverte a entrada caso o componente processado seja de luminância.

As arquiteturas do quantizador e do codificador de entropia precisam ser modificadas para o processamento de imagens coloridas. Estas adaptações serão descritas nos próximos itens deste capítulo, onde serão enfocadas apenas as diferenças da implementação adaptada ao processamento de imagens coloridas, em relação à implementação para imagens em tons de cinza.

A arquitetura do compressor JPEG para imagens coloridas continuará a ser capaz de consumir um dado de entrada a cada ciclo de *clock* e continuará com a mesma latência.

A síntese do compressor JPEG para imagens coloridas utilizou 6.315 células lógicas e 12.080 bits de memória de um dispositivo EPF10K130EQC240-1 [ALT 2001a] da Altera, utilizando 94% e 18%, respectivamente, dos recursos deste dispositivo. A arquitetura utiliza 10 pinos de entrada e 33 pinos de saída. O compressor JPEG para imagens coloridas é capaz de operar a uma freqüência máxima de 16,9MHz, sendo capaz de processar uma imagem colorida de 640 x 480 *pixels* em 54,4ms e permitindo uma taxa de processamento de 18,4 imagens por segundo.

4.1.1 As Adaptações na Arquitetura do Quantizador

A arquitetura do quantizador do compressor de imagens coloridas, apresentada na fig. 4.3, é muito parecida com a arquitetura do quantizador utilizada no compressor para imagens em tons de cinza. A diferença principal é a existência de uma memória ROM adicional, que irá conter a tabela de quantização para os componentes Cb e Cr [THE 92]. Todo o princípio de funcionamento é o mesmo que o apresentado no item 3.2. A multiplicação realizada é decomposta em quatro somas de deslocamentos e tanto os somadores, quanto os deslocadores, são exatamente os mesmos utilizados no quantizador anteriormente apresentado. As memórias contêm os deslocamentos necessários para cada constante de quantização, já devidamente multiplicada pelo fator de escala da DCT 2-D, como foi explicado no item 3.1.5.

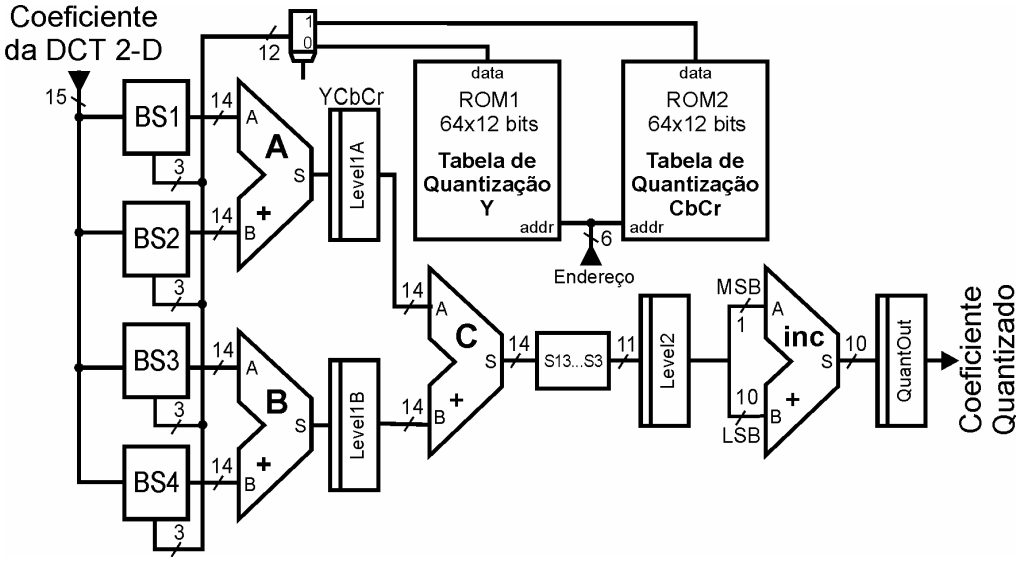


FIGURA 4.3 – Quantizador para o compressor de imagens coloridas

A memória ROM adicional é igual à primeira memória ROM, possuindo 6 bits de endereço e utilizando 64 posições de memória de 12 bits. O funcionamento das memórias não é sincronizado pelo *clock* geral do compressor.

A geração de endereços também é exatamente igual a apresentada no item 3.2. O endereço gerado é disponibilizado, simultaneamente, para as duas memórias ROM. A seleção de qual conteúdo deve ser utilizado em cada cálculo é realizada através do multiplexador apresentado na fig. 4.3, que é controlado pelo sinal *YCbCr*. O sinal *YCbCr* é gerado pelo controle da arquitetura e indica se o cálculo atual é de um componente de luminância (Y) ou de um componente de crominância (Cb ou Cr).

As adaptações na arquitetura do quantizador para que suporte o processamento de imagens coloridas não afeta a sua latência e o seu desempenho é pouco afetado, porque os estágios do *pipeline* continuam os mesmos e porque apenas um multiplexador foi adicionado ao caminho dos dados que estão sendo processados. Estas comparações de desempenho serão apresentadas no item 4.1.3 deste capítulo.

A síntese do quantizador do compressor de imagens coloridas utilizou 378 células lógicas e 1.536 bits de memória do dispositivo EPF10K30ETC144-1 [ALT 2001a] da Altera. Esta arquitetura seria capaz de operar em uma freqüência de 19,3MHz, processando uma imagem colorida de 640x480 *pixels* em 47,7ms e permitindo uma taxa de processamento de 21 imagens por segundo.

4.1.2 As Adaptações na Arquitetura do Codificador de Entropia

O codificador de entropia desenvolvido para imagens em tons de cinza foi adaptado para a codificação de imagens coloridas, onde apenas o controle geral do codificador e as arquiteturas dos codificadores diferencial e de Huffman foram alteradas. Todas as demais arquiteturas são exatamente as mesmas do codificador de entropia desenvolvido para o compressor JPEG para imagens em tons de cinza.

O controle do codificador de entropia foi modificado para geração do sinal de controle *YCbCr*, que indica se o dado que está sendo processado é um componente Y, Cb ou Cr. O sinal possui dois bits, cujas combinações indicam qual elemento de cor está sendo processado, de acordo com a tab. 4.1.

TABELA 4.1 – Sinal de controle *YCbCr*

YCbCr	**Componente Processado**
00	Y
01	Cb
10	Cr
11	X

A síntese isolada do codificador de entropia do compressor JPEG para imagens coloridas utilizou 1.405 células lógicas e 7.728 bits de memória de um dispositivo EPF10K30ETC144-1 [ALT 2001a] da Altera. Esta arquitetura poderia operar a uma freqüência máxima de 23,1MHz e seria capaz de processar uma imagem colorida de 640x480 *pixels* em 40ms, permitindo uma taxa de processamento de 25 imagens por segundo.

O desempenho do codificador de entropia sofreu pequeno impacto com as adaptações realizadas, uma vez que apenas um multiplexador foi adicionado ao caminho dos dados da arquitetura do codificador diferencial e outro ao caminho dos dados da arquitetura do codificador de Huffman. Este impacto é discutido no item 4.1.3 deste

capítulo. A latência do codificador de entropia não foi alterada e continua dependente do assincronismo gerado pelo codificador RLE e pelo montador.

4.1.2.1 Codificador Diferencial para Imagens Coloridas

O codificador diferencial para imagens coloridas foi modificado em relação ao codificador diferencial para imagens em tons de cinza, uma vez que em imagens coloridas, existem três componentes de cor e a correlação existente entre os componentes DC, anteriormente explicada, refere-se aos consecutivos componentes DC de um mesmo componente de cor. Como os componentes de cor são processados de maneira intercalada, conforme já foi explicado, é necessário ao codificador diferencial para imagens coloridas armazenar o componente DC da última matriz processada de cada um dos três componentes de cor. Então, foram inseridos mais dois registradores na arquitetura do codificador diferencial, que está apresentada na fig. 4.4.

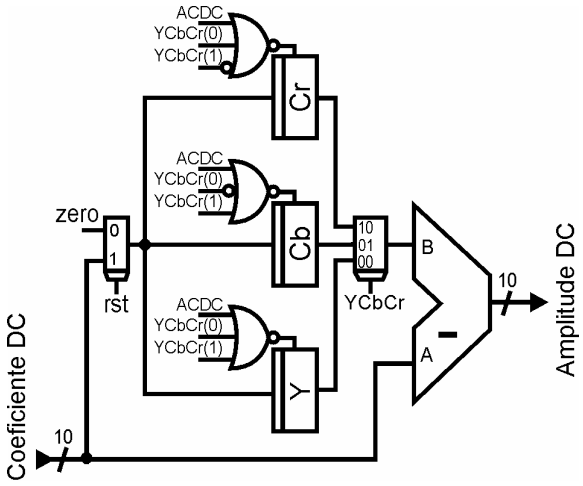


FIGURA 4.4 – Codificador diferencial para imagens coloridas

A habilitação para a escrita nos registradores passou a utilizar, além do sinal *ACDC*, o sinal *YCbCr*, para que o componente DC seja armazenado no correto registrador a ele destinado, obedecendo a seqüência apresentada na tab. 4.2.

TABELA 4.2 – Habilitação da escrita nos registradores do codificador diferencial

YCbCr	ACDC	Habilitação para escrita		
		Y	Cb	Cr
00	0	SIM	não	não
01	0	não	SIM	não
10	0	não	não	SIM
11	0	não	não	não
X	1	não	não	não

O sinal *YCbCr* também é utilizado para controlar o multiplexador que define a entrada *B* do subtrator, de acordo com a fig. 4.4.

4.1.2.2 Codificador de Huffman para Imagens Coloridas

O codificador de Huffman desenvolvido para o compressor de imagens em tons de cinza foi adaptado ao processamento de imagens coloridas através do uso de duas novas tabelas de Huffman, uma para os elementos DC relativos aos componentes

crominância e outra para os elementos AC, também relativos aos componentes de crominância. Então, mais duas memórias ROM foram inseridas na arquitetura desenvolvida, com o objetivo de armazenar as novas tabelas de Huffman. A organização das novas memórias é igual à adotada nas memórias originais, ou seja, em cada posição de memória estão armazenados o código de Huffman e o tamanho deste código em número de bits. As tabelas de Huffman armazenadas nestas novas memórias estão apresentadas no anexo 3 desta dissertação.

Na arquitetura desenvolvida, que está apresentada na fig. 4.5, além das duas memórias extras adicionadas em relação ao codificador para imagens em tons de cinza, foram inseridos mais dois multiplexadores, para permitir a seleção de qual das quatro saídas de memória será conectada à saída do codificador. Os multiplexadores inseridos são controlados pelo sinal *YCbCr* e selecionam a saída das memórias de acordo com o tipo de componente de cor que está sendo processado, se de luminância (Y) ou de crominância (Cb ou Cr). Após a seleção do tipo de componente de cor, é necessário selecionar a saída da memória de acordo com o correto elemento da matriz de entrada que está sendo processado, DC ou AC. Este controle é o mesmo que já estava presente na arquitetura original e é realizado pelo multiplexador controlado pelo sinal *ACDC*.

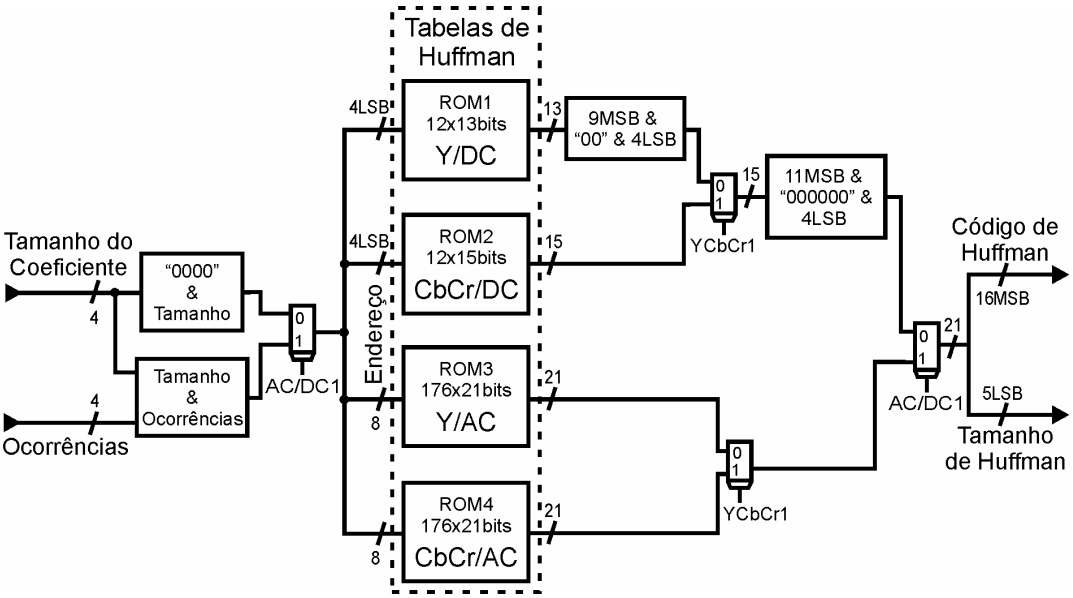


FIGURA 4.5 – Codificador de Huffman para imagens coloridas

4.1.3 Considerações Finais sobre o Compressor JPEG para Imagens Coloridas

O compressor JPEG para imagens coloridas foi gerado a partir da arquitetura do compressor JPEG para imagens em tons de cinza. A arquitetura do compressor para imagens coloridas foi desenvolvida, descrita em VHDL, sintetizada e simulada utilizando o ambiente Maxplus2 da Altera. A síntese foi direcionada para FPGAs da família Flex10KE [ALT 2001a].

A tab. 4.3 apresenta o resumo do processo de síntese do compressor, de onde pode-se perceber que, como no compressor para imagens em tons de cinza, a DCT 2-D é o bloco mais lento, que utiliza o maior número de células lógicas e que possui a maior latência.

A síntese do compressor para imagens coloridas utilizou 6.315 células lógicas e 12.080 bits de memória, ocupando, respectivamente, 94 % e 18 % dos recursos do dispositivo EPF10K130EQC240-1 [ALT 2001a]. Foram utilizados 10 pinos de entrada e 33 pinos de saída. Com a freqüência máxima de 16,9MHz, o compressor é capaz de processar uma imagem colorida de 640x480 *pixels* em 54,4ms, permitindo uma taxa de processamento de 18,4 imagens por segundo.

TABELA 4.3 – Resumo da síntese do compressor para imagens coloridas

	Células Lógicas	**Bits de Memória**	**Período** (ns)	**Freqüência** (MHz)	**Latência** (nº ciclos)
DCT 2-D	4181	1536	52,7	19	163
Quantizador	378	1536	51,8	19,3	3
***Buffer* Ziguezague**	372	1280	38,7	25,8	66
Codificador de Entropia	1405	7728	43,4	23,1	5*
Compressor JPEG	6315	12080	59	16,9	237*

* latência mínima

A simulação do compressor para imagens coloridas não está anexada a esta dissertação devido ao fato de ser muito extensa e bastante similar com a simulação do compressor JPEG para imagens em tons de cinza. Apenas um resumo de uma das simulações realizadas está apresentado nos próximos parágrafos. Nesta simulação, para a compressão dos dados de luminância (Y), foram utilizadas como entrada, quatro matrizes iguais a do exemplo dado no capítulo 3, que está apresentada abaixo.

168	161	161	150	154	168	164	154
171	154	161	150	157	171	150	164
171	168	147	164	164	161	143	154
164	171	154	161	157	157	147	132
161	161	157	154	143	161	154	132
164	161	161	154	150	157	154	140
161	168	157	154	161	140	140	132
154	161	157	150	140	132	136	128

A simulação dos componentes Y ocorre do mesmo modo apresentado no capítulo anterior, sendo que, até a saída do *buffer* ziguezague, os resultados são idênticos para as quatro matrizes processadas e para o exemplo apresentado no capítulo 3. A matriz entregue pelo *buffer* ziguezague para as quatro matrizes Y está apresentada abaixo.

13	4	3	0	-2	0	1	1
0	1	-1	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Na codificação de entropia, os resultados não serão idênticos para as quatro matrizes Y, uma vez que a codificação diferencial considera, para realizar a subtração já explicada no item 3.3.2, o elemento DC da última matriz do componente que está sendo processado. A codificação diferencial da primeira matriz possui um valor DC anterior igual a zero, portanto a sua saída será igual à entrada, ou seja, será igual a 13. Para a codificação diferencial das outras três matrizes Y, o valor DC anterior será igual a 13, uma vez que as quatro matrizes Y de entrada são iguais. Deste modo, o resultado da codificação diferencial será igual a zero para as três últimas matrizes Y. Os códigos gerados pelo codificador de entropia, para esta simulação, estão apresentados na tab. 4.4, onde estão destacados os diferentes códigos gerados para os elementos DC.

A primeira coluna da tab. 4.4 indica se o elemento que está sendo processado é AC ou DC. A segunda coluna apresenta o coeficiente quantizado, que irá ser codificado pelo codificador de entropia. A terceira coluna apresenta o par *Ocorrências/Tamanho*, para cada uma das amplitudes da entrada, onde o primeiro número indica quantos zeros precedem a amplitude e o segundo número indica qual o tamanho, em número de bits, desta amplitude. O elemento DC não possui o par *Ocorrências/Tamanho*, possuindo apenas o campo *Tamanho*.

A quarta e a quinta coluna apresentam a codificação de entropia para cada entrada considerada, sendo que a quarta coluna apresenta o código de Huffman referente ao par *Ocorrências/Tamanho* (para elementos AC) ou ao componente *Tamanho* (para elementos DC) e a quinta coluna apresenta o código VLC para cada uma das amplitudes.

TABELA 4.4 – Códigos gerados pela codificação de entropia para os componentes de luminância

	Amplitude	**Ocorrências/ Tamanho**	**Codificação de Entropia**	
			Huffman (Y)	VLC
DC	13	4	101	1101
DC	13	0	00	
AC	4	0/3	100	100
AC	3	0/2	01	11
AC	-2	1/2	11011	01
AC	1	1/1	1100	1
AC	1	0/1	00	1
AC	1	1/1	1100	1
AC	-1	0/1	00	0
AC	1	2/1	11100	1
AC	0	15/0 (ZRL)	11111111001	
AC	0	15/0 (ZRL)	11111111001	
AC	0	15/0 (ZRL)	11111111001	
AC	0	0/0 (EOB)	1010	

Foram utilizados um total de 317 bits para representar as quatro matrizes Y, que serão montados em conjunto com as saídas referentes aos componentes Cb e Cr, como será apresentado na tab. 4.6. A primeira matriz Y utilizou 83 bits, enquanto que as três últimas matrizes Y utilizaram 78 bits

A matriz de entrada utilizada para a compressão dos dados de crominância (Cb e Cr) está apresentada abaixo.

40	33	33	22	26	40	36	26
43	26	33	22	29	43	22	36
43	40	19	36	36	33	15	26
36	43	26	33	29	29	19	4
33	33	29	26	15	33	26	4
36	33	33	26	22	29	26	12
33	40	29	26	33	12	12	4
26	33	29	22	12	4	8	0

A saída da arquitetura da DCT 2-D, considerando a matriz de crominância, está apresentada a seguir. É importante notar que o resultado da DCT 2-D para as matrizes de crominância, utilizadas como exemplo, é exatamente igual ao resultado obtido para matrizes de luminância, que foi apresentado no item 3.4, mesmo com matrizes de luminância e crominância distintas. Isto ocorreu propositadamente, para salientar que as matrizes de luminância passam por um pré-processamento que não é realizado para as matrizes de crominância. Este pré-processamento é o deslocamento de nível e já foi explicado no capítulo 3, sendo uma subtração de 128 de todos os elementos da matriz de luminância. A matriz de crominância usada como entrada foi gerada a partir de uma subtração de 128 de todos os elementos da matriz de luminância. Desta forma, após deslocamento de nível, as matrizes de luminância e crominância serão exatamente iguais para o processamento da DCT 2-D.

1716	533	-31	193	-82	-3	5	-11
377	-118	149	152	50	-13	90	-30
-63	-54	108	-110	27	-24	27	31
82	-128	53	36	-141	78	30	15
-100	57	-15	-15	-118	53	-23	1
28	80	-63	20	27	-32	-46	9
7	-8	14	-8	5	10	-5	-5
1	-2	1	4	8	-5	-6	-2

A arquitetura do quantizador recebe como entrada a saída do cálculo da DCT 2-D, gerando a matriz de resultados apresentada abaixo, usando, para tanto, a tabela de quantização para os componentes de crominância.

12	2	0	0	0	0	0	0
3	-1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

A matriz gerada pelo quantizador é entregue ao *buffer* ziguezague, que gera a matriz apresentada a seguir.

12	3	2	0	-1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Por fim, a matriz gerada pelo *buffer* ziguezague é entregue ao codificador de entropia, que gera o resultado apresentado na tab. 4.5. Este resultado é idêntico para as matrizes de Cb e Cr.

TABELA 4.5 – Códigos gerados pela codificação de entropia para os componentes de crominância para a simulação realizada

	Amplitude	**Ocorrências/ Tamanho**	**Codificação de Entropia**	
			Huffman (CbCr)	VLC
DC	12	4	1110	1100
AC	3	0/2	100	11
AC	2	0/2	100	10
AC	-1	1/1	1011	0
AC	0	15/0 (ZRL)	1111111010	
AC	0	15/0 (ZRL)	1111111010	
AC	0	15/0 (ZRL)	1111111010	
AC	0	0/0 (EOB)	00	

O processamento completo das seis matrizes 8x8, quatro do componente Y, uma do componente Cb e uma do componente Cr, gera as palavras JPEG apresentadas na tab. 4.6, onde estão apresentadas, além das palavras JPEG, a origem dos dados contidos em cada palavra, com relação às matrizes de componentes de cor da entrada. As palavras JPEG geradas para os diferentes componentes são montadas de forma intercalada, como já foi explicado no capítulo 2.

Para o exemplo, a compressão das quatro matrizes Y utilizou 317 bits, enquanto que a compressão das matrizes Cb e Cr utilizam 55 bits para cada matriz. Então, são utilizados, no total, 427 bits para representar as seis matrizes de entrada comprimidas.

As seis matrizes da entrada são oriundas da operação de *downsampling* realizada pelo processador de entrada e saída e representam 12 matrizes de entrada, quatro para cada componente de cor. Cada matriz de entrada utiliza 8 bits para cada um dos seus 64 elementos, em um total de 512 bits. Então, para as 12 matrizes da imagem de entrada são utilizados 6.144 bits. Para o exemplo, foi obtida uma taxa de compressão, sobre as matrizes de entrada, de 14,4 vezes ou de 93,1 %. É importante perceber que o arquivo JPEG possui, além dos dados da imagem comprimida, as tabelas de quantização e de Huffman utilizadas na compressão, bem como outros dados necessários ao processo de descompressão, por isso, a taxa de compressão calculada não é a taxa efetiva obtida no processo completo de compressão, que compara o número de bits utilizados pelo arquivo da imagem de entrada com o número de bits utilizados pela arquivo da imagem comprimida.

TABELA 4.6 – Saídas do compressor para o exemplo

Matriz processada	Palavra JPEG
Y_0	10111011001000111110110111001001
Y_0	11001000111001111111110011111111
Y_0 e Y_1	10011111111100110100010010001111
Y_1	10110111001001110010001110011111
Y_1	11110011111111100111111111001101
Y_2 e Y_2	00010010001111101101110010011100
Y_2	10001110011111111100111111111001
Y_2 e Y_3	11111111001101000100100011111011
Y_3	01110010011100100011100111111111
Y_3 e Cb_0	00111111111001111111110011010111
Cb_0	01100100111001010110111111101011
Cr_0 e Cr_0	11111010111111101000111011001001
Cr_0	11001010110111111101011111110101
Cr_0*	11111101000xxxxxxxxxxxxxxxxxxxxxx

* palavra incompleta

O fato da taxa de compressão ser mais elevada do que a obtida pela compressão JPEG para imagens em tons de cinza é função da operação de *downsampling*, que é responsável pela redução de 50% dos dados da imagem, antes mesmo do processo de compressão propriamente dito, de acordo com o que foi explicado no item 2.3.2.

O desempenho do compressor para imagens coloridas, em termos de freqüência máxima de operação e de taxa de processamento de imagens, é, como esperado, menor do que o desempenho do compressor para imagens em tons de cinza.

No que diz respeito a freqüência de operação, foram inseridos novos componentes em alguns dos blocos, que fizeram a freqüência máxima destes blocos ser menor do que a freqüência dos blocos eqüivalentes do compressor para imagens em tons de cinza. A tab. 4.7 apresenta o resumo comparativo entre os resultados da síntese do compressor colorido e do compressor para imagens em tons de cinza.

TABELA 4.7 – Comparação dos resultados de síntese do compressor para imagens coloridas e do compressor para imagens em tons de cinza

	JPEG colorido			JPEG tons de cinza		
	Células Lógicas	Bits de Memória	Freqüência (MHz)	Células Lógicas	Bits de Memória	Freqüência (MHz)
Quantizador	378	1536	19,3	351	768	21,6
Codificador de Entropia	1405	7728	23,1	1307	3852	25,6
Codificador diferencial	81	-	40	28	-	57,5
Codificador de Huffman	50	7728	52,1	21	3852	58,8
Compressor	6315	12080	16,9	6199	7436	16,6

Da tab. 4.7 é possível perceber o impacto das adaptações realizadas em termos de células lógicas, de bits de memória e de freqüência máxima de operação.

O principal impacto está no uso de bits de memória, que é 62% superior no compressor para imagens coloridas em relação ao compressor para imagens em tons de cinza. Este significativo incremento no número de bits utilizados é função das memórias inseridas no quantizador e no codificador de Huffman. O compressor para imagens coloridas utilizou 1,8% mais células lógicas que o compressor para imagens em tons de cinza, com um impacto praticamente insignificante.

A freqüência máxima de operação teve um impacto negativo pequeno, mas perceptível, nos blocos alterados, ou seja, no quantizador, no codificador diferencial e no codificador de Huffman. No entanto, em termos da arquitetura completa do compressor, o impacto pode ser desprezado, pois os estágios de *pipeline* mais críticos, em termos de freqüência, estão na arquitetura da DCT 2-D, que não sofreu alteração alguma para o uso no compressor de imagens coloridas. Por isso, a diferença nas freqüências de operação dos dois compressores não é significativa.

A comparação entre o desempenho do compressor para imagens coloridas e do compressor para imagens em tons de cinza é função, além das respectivas freqüências máximas de operação, do próprio tipo de imagem a ser processado. Para imagens de mesmo número de *pixels*, existem três vezes mais dados a serem processados pela arquitetura do compressor para imagens coloridas do que pela arquitetura do compressor para imagens em tons de cinza. Por isso, o compressor para imagens coloridas é capaz de processar uma imagem de 640x480 (colorida) em 54,4ms, enquanto que o compressor para imagens em tons de cinza processa uma imagem de mesmas dimensões em 18,5ms. A taxa de processamento para imagens destas dimensões é de 18,4 imagens por segundo para o compressor para imagens coloridas e 54 imagens por segundo para o compressor de imagens em tons de cinza.

Dos dados citados é possível perceber que o impacto na freqüência de operação tem pouca influência no desempenho global, uma vez que o compressor para imagens em tons de cinza é, quase que exatamente, três vezes mais lento que o compressor para imagens em tons de cinza.

Ainda assim, a taxa de processamento de 18,4 imagens por segundo é suficiente para muitas aplicações. Por isso o compressor JPEG para imagens coloridas pode ser utilizado como um ASIC ou como um IP para o desenvolvimento de aplicações mais complexas.

Para aumentar ainda mais o desempenho do compressor para imagens coloridas, um caminho possível é substituir os somadores utilizados, que são *ripple carry*, por somadores mais rápidos, como *carry look ahead* [WES 95], por exemplo. Outra alternativa seria a utilização de três arquiteturas em paralelo, uma para cada componente de cor. Deste modo, a taxa de processamento seria praticamente igual a do compressor para imagens em tons de cinza. O uso de três arquiteturas em paralelo insere um custo adicional de recursos muito elevado.

4.2 Compressor JPEG para Imagens Coloridas com Conversão de Espaço de Cores

Este item do trabalho irá enfocar o desenvolvimento da arquitetura do compressor JPEG para imagens coloridas, considerando a conversão de espaço de cores. Resultados parciais obtidos no desenvolvimento desta arquitetura foram publicados em [AGO 2001].

Para muitas aplicações, a imagem de entrada está no espaço de cores RGB e deve ser convertida para algum espaço de cores do tipo luminância e crominância para ser processada pelo compressor JPEG. Nesta dissertação, optou-se pelo espaço YCbCr, como já foi mencionado.

Neste item será proposta uma arquitetura para a conversão do espaço de cores. A integração da arquitetura do conversor de espaço de cores com a arquitetura do compressor JPEG para imagens coloridas não foi desenvolvida nesta dissertação, no entanto, esta questão é discutida e algumas soluções são propostas.

4.2.1 A Arquitetura para o Conversor de Espaço de Cores

A arquitetura desenvolvida para o conversor do espaço de cores de RGB para YCbCr toma por base a arquitetura proposta em [AGO 2001c] e será apresentada, em detalhes, neste item da dissertação.

A síntese da arquitetura do conversor de espaço de cores utilizou 281 células lógicas, 28 pinos de entrada e 9 pinos de saída de um dispositivo EPF10K30ETC144-1 [ALT 2001a] da Altera. Esta arquitetura é capaz de operar a uma freqüência máxima de 32,7MHz, processando uma imagem colorida de 640 x 480 *pixels* em 84,6ms e permitindo uma taxa de processamento de 11,8 imagens por segundo.

A arquitetura foi validada através de várias simulações e uma destas simulações está apresentada no anexo 6.

4.2.1.1 O Datapath do Conversor de Espaço de Cores

Os cálculos realizados pelo datapath do conversor de espaço de cores são, basicamente, multiplicações por constantes e somas, como já foi apresentado no capítulo 2. Estes cálculos são:

$$Y = 0{,}299R + 0{,}587G + 0{,}114B$$

$$Cb = -0{,}169R - 0{,}331G + 0{,}5B$$

$$Cr = 0{,}5R - 0{,}419G - 0{,}081B$$

A arquitetura desenvolvida, apresentada na fig. 4.6, foi projetada para operar em um *pipeline* de quatro estágios e possui uma latência de seis ciclos de *clock*. Um novo resultado válido é gerado a cada três ciclos de *clock* quando o *pipeline* está preenchido.

As somas foram desenvolvidas através do uso de somadores *ripple carry*, enquanto que as multiplicações foram desenvolvidas através do uso de somas de deslocamentos, como nos multiplicadores utilizados na DCT 1-D e na quantização.

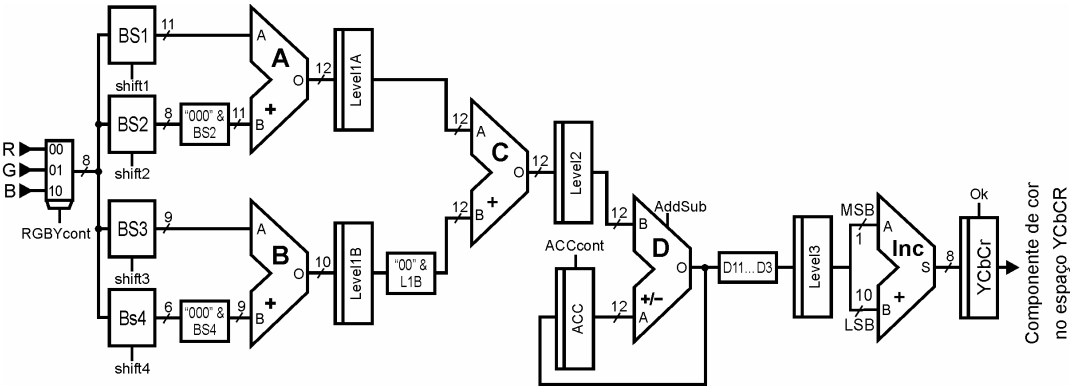


FIGURA 4.6 – *Datapath* da arquitetura integrada

Os deslocamentos foram obtidos através de *barrel shifters* otimizados e as somas, novamente com o uso de *ripple carry*. Os deslocamentos associados a cada constante estão apresentados na tab. 4.8, onde *i* é o dado de entrada e *[x]* indica o número de deslocamentos à direita, realizado sobre o dado de entrada.

TABELA 4.8 – Deslocamentos realizados por cada *barrel shifter*

Constante	BS1	BS2	BS3	BS4
0,299	*i* [2]	*i* [5]	*i* [6]	*i* [8]
0,587	*i* [1]	*i* [4]	*i* [7]	*i* [6]
0,114	*i* [4]	*i* [5]	*i* [6]	*i* [8]
0,169	*i* [3]	*i* [5]	*i* [7]	*i* [8]
0,331	*i* [2]	*i* [4]	*i* [6]	*i* [9]
0,5	*i* [1]	-	-	-
0,419	*i* [2]	*i* [5]	*i* [3]	*i* [6]
0,081	*i* [4]	*i* [6]	-	*i* [9]

Cada multiplicação é obtida através da soma de quatro deslocamentos, que são realizadas pelos somadores *A*, *B* e *C*. O somador *D* faz a acumulação das multiplicações e pode realizar somas ou subtrações, de acordo com o componente de cor que está sendo calculado. O incrementador *Inc* arredonda o resultado da multiplicação, somando com a parte inteira do resultado o bit mais significativo dentre os bits da parte fracionária.

A tab. 4.9 apresenta a entrada de controle, o número de deslocamentos à direita e o número de bits na saída, para cada um dos quatro *barrel shifters* apresentados na fig. 4.6. Os *barrel shifters* foram simplificados, gerando o mínimo número possível de deslocamentos e eliminando os bits que são constantes para todos os deslocamentos gerados. Estas simplificações têm impacto também nos somadores e registradores utilizados na arquitetura, muito embora gerem números em diferentes escalas nas saídas dos *barrel shifters*.

TABELA 4.9 – Operação dos *barrel shifters*

	BS1		BS2		BS3		BS4	
Controle	des.	bits	des.	bits	des.	bits	des.	Bits
00	1	11	X	8	X	9	X	6
01	2		4		3		6	
10	3		5		6		8	
11	4		6		7		9	

As saídas dos *barrel shifters* foram truncadas na quarta casa decimal como forma de minimizar os recursos utilizados pela arquitetura. O impacto deste truncamento é mínimo, uma vez que a arquitetura entrega números inteiros em sua saída e, então, mesmo as quatro casas fracionárias, consideradas nos cálculos internos, são descartadas na saída, que possuirá apenas oito bits.

Para somar os valores entregues por diferentes *barrel shifters* é necessário colocar os números na mesma escala e isto é feito através da concatenação com zeros, como está apresentado na fig. 4.6. Os números são concatenados com zeros porque a entrada possui apenas números positivos e, até o segundo nível do *pipeline*, são

efetuadas apenas adições, então, não é necessário controlar a extensão do sinal, que é conhecido e tem o valor zero.

Os componentes de cor Y, Cb e Cr calculados pela arquitetura são armazenados no registrador de saída *YCbCr*. Este registrador permite o sincronismo da saída da arquitetura do conversor de espaço de cores, formando a barreira temporal que viabiliza a operação do *pipeline*.

Cada *pixel* de entrada, com seus componentes R, G e B, deve ficar disponível durante três ciclos de *clock* para o cálculo de cada componente no espaço de cores YCbCr e quem realiza este gerenciamento é o processador de entrada e saída.

4.2.1.2 Controle do Conversor de Espaço de Cores

O controle da arquitetura do conversor é responsável pelo seqüenciamento das operações realizadas no *pipeline* e pelo gerenciamento dos cálculos realizados em cada ciclo de *clock*.

O controle de qual componente está sendo calculado a cada instante é realizado pelo sinal externo *YCbCrcontrol*, que indica se o componente calculado é Y, Cb ou Cr. Desta forma, a arquitetura do conversor de espaço de cores pode se adaptar a qualquer taxa de *downsampling* e pode ser utilizada em arquiteturas com o processamento intercalado ou não. A arquitetura do compressor para imagens coloridas, desenvolvida nesta dissertação, considera o processamento intercalado dos componentes de cor e uma taxa de *downsampling* de 4:1:1, isto é, o conversor de espaço de cores deve gerar quatro matrizes 8x8 de componentes Y, uma matriz 8x8 de componentes Cb e uma matriz 8x8 de componentes Cr. Este controle deve ser feito pelo controle global, caso as duas arquiteturas sejam integradas em uma única arquitetura.

O bloco de controle gera os sinais de controle dos *barrel shifters*, indicando qual deslocamento deve ser realizado por cada *barrel shifter* para cada cálculo. A entrada dos *barrel shifters* é selecionada através do multiplexador que é controlado pelo sinal *RGBYcont*, selecionando o correto componente de cor necessário para cada cálculo.

O controle também gera o sinal que seleciona o tipo de operação realizada pelo somador *D*, que será adição ou subtração, de acordo com cada novo cálculo.

A entrada *A* do somador *D* recebe o conteúdo do registrador *ACC*, que deve ser zerado no início de um novo cálculo. A inicialização do registrador *ACC* é realizada pelo sinal *ACCcont*, que é gerado pelo bloco de controle.

A escrita no registrador de saída *YCbCr* é controlada pelo sinal *Ok*, que indica que uma nova saída válida foi gerada pela arquitetura. O sinal *Ok* é atrasado em um ciclo de *clock* para a geração do sinal *OkYCbCr*, que é o sinal de sincronismo da arquitetura do conversor de espaço de cores, indicando que o valor armazenado no registrador *YCbCr* é um novo valor válido.

4.2.1.3 A Simulação do Conversor de Espaço de Cores

Foram realizadas diversas simulações com o intuito de estimular os pontos críticos da arquitetura. Uma destas simulações está apresentada no anexo 6. Paralelamente à simulação, os cálculos da conversão do espaço de cores foram realizados em software para validar a arquitetura desenvolvida. Foram consideradas as mesmas matrizes R, G e B para as duas situações, para viabilizar a comparação dos resultados. As matrizes R, G e B estão apresentas abaixo, sendo que as matrizes R e B são iguais.

Matriz G							
168	161	161	150	154	168	164	154
171	154	161	150	157	171	150	164
171	168	147	164	164	161	143	154
164	171	154	161	157	157	147	132
161	161	157	154	143	161	154	132
164	161	161	154	150	157	154	140
161	168	157	154	161	140	140	132
154	161	157	150	140	132	136	128

Matrizes R e B							
160	160	160	160	160	160	160	160
160	160	160	160	160	160	160	160
160	160	160	160	160	160	160	160
160	160	160	160	160	160	160	160
160	160	160	160	160	160	160	160
160	160	160	160	160	160	160	160
160	160	160	160	160	160	160	160
160	160	160	160	160	160	160	160

A versão em software do conversor de espaço de cores não levou em consideração os truncamentos nos resultados intermediários e a restrição causada pelo uso de quatro somas de deslocamentos, por isso, alguns elementos calculados pela arquitetura apresentam um pequeno erro em relação aos mesmos elementos calculados via software. Os resultados comparativos entre a conversão de cores realizada via software e através da simulação da arquitetura desenvolvida estão apresentados nas tabs. 4.10, 4.11 e 4.12, referentes, respectivamente, aos elementos Y, Cb e Cr.

TABELA 4.10 – Comparação dos resultados obtidos via software e via simulação da arquitetura desenvolvida para o componente Y

Linha 0	**Simulado**	165	161	161	154	156	165	162	156
	Software	164	160	160	154	156	164	162	156
Linha 1	**Simulado**	166	156	161	154	158	166	154	162
	Software	166	156	160	154	158	166	154	162
Linha 2	**Simulado**	166	165	152	162	162	161	150	156
	Software	166	164	152	162	162	160	150	156
Linha 3	**Simulado**	162	166	156	161	158	158	152	144
	Software	162	166	156	160	158	158	152	143
Linha 4	**Simulado**	161	161	158	156	150	161	156	144
	Software	160	160	158	156	150	160	156	143
Linha 5	**Simulado**	162	161	161	156	154	158	156	148
	Software	162	160	160	156	154	158	156	148
Linha 6	**Simulado**	161	165	158	156	161	148	148	144
	Software	160	164	158	156	160	148	148	143
Linha 7	**Simulado**	156	161	158	154	148	144	146	141
	Software	156	160	158	154	148	143	145	141

TABELA 4.11 – Comparação dos resultados obtidos via software e via simulação da arquitetura desenvolvida para o componente Cb

Linha 0	**Simulado**	-2	0	0	4	2	-2	-1	2
	Software	-2	0	0	3	1	-2	-1	1
Linha 1	**Simulado**	-3	2	0	4	1	-3	4	-1
	Software	-3	1	0	3	0	-3	3	-1
Linha 2	**Simulado**	-3	-2	5	-1	-1	0	6	2
	Software	-3	-2	4	-1	-1	0	5	1
Linha 3	**Simulado**	-1	-3	2	0	1	1	5	10
	Software	-1	-3	1	0	0	0	4	9
Linha 4	**Simulado**	0	0	1	2	6	0	2	10
	Software	0	0	0	1	5	0	1	9
Linha 5	**Simulado**	-1	0	0	2	4	1	2	7
	Software	-1	0	0	1	3	0	1	6
Linha 6	**Simulado**	0	-2	1	2	0	7	7	10
	Software	0	-2	0	1	0	6	6	9
Linha 7	**Simulado**	2	0	1	4	7	10	8	11
	Software	1	0	0	3	6	9	7	10

TABELA 4.12 – Comparação dos resultados obtidos via software e via simulação da arquitetura desenvolvida para o componente Cr

Linha 0	**Simulado**	-3	-1	-1	4	2	-3	-2	2
	Software	-3	0	0	4	2	-3	-1	2
Linha 1	**Simulado**	-4	2	-1	4	1	-4	4	-2
	Software	-4	2	0	4	1	-4	4	-1
Linha 2	**Simulado**	-4	-3	5	-2	-2	-1	7	2
	Software	-4	-3	5	-1	-1	0	7	2
Linha 3	**Simulado**	-2	-4	2	-1	1	1	5	12
	Software	-1	-4	2	0	1	1	5	11
Linha 4	**Simulado**	-1	-1	1	2	7	-1	2	12
	Software	0	0	1	2	7	0	2	11
Linha 5	**Simulado**	-2	-1	-1	2	4	1	2	8
	Software	-1	0	0	2	4	1	2	8
Linha 6	**Simulado**	-1	-3	1	2	-1	8	8	12
	Software	0	-3	1	2	0	8	8	11
Linha 7	**Simulado**	2	-1	1	4	8	12	10	13
	Software	2	0	1	4	8	11	10	13

4.2.2 Discussão sobre a Integração das Arquiteturas do Conversor de Espaço de Cores e do Compressor JPEG para Imagens Coloridas

A arquitetura do conversor de espaço de cores, considerando o *pipeline* preenchido, gera um novo resultado válido a cada três ciclos de *clock*. A arquitetura da DCT 2-D, primeira arquitetura do compressor JPEG, consome um valor de entrada por ciclo de *clock*. Então, existe um óbvio problema na integração destas duas arquiteturas, uma vez que a taxa de consumo da DCT 2-D é superior à taxa de produção da arquitetura do conversor de espaço de cores. Este item da dissertação irá discutir esta questão e apontar algumas possíveis soluções. Nenhuma das soluções propostas chegou a ser implementada, mas alguns dos possíveis caminhos nesta direção estão indicados neste item.

4.2.2.1 Integração dos Datapaths Originais

As duas arquiteturas principais podem ser unidas de maneira direta, onde a saída do conversor de espaço de cores é conectada diretamente na entrada da arquitetura da DCT 2-D, primeira arquitetura do compressor para imagens coloridas.

Neste caso, a arquitetura do compressor para imagens coloridas irá ficar dois terços do tempo em inatividade, o que é possível através da inserção de bolhas no seu *pipeline*. Então, se faz necessária uma alteração no controle da arquitetura da DCT 2-D e das demais arquiteturas que se seguem no compressor, no sentido de manter o sincronismo com a arquitetura do conversor de espaço de cores, considerando apenas os valores válidos gerados por esta arquitetura.

Nesta solução, a arquitetura do conversor de espaço de cores irá produzir um valor válido a cada 3 ciclos de *clock* e, então, a arquitetura do compressor adaptado passaria 66,7 % do tempo inativa. Todo este tempo perdido é um problema, pois a complexidade da compressão e o uso de recursos estão concentrados na arquitetura do compressor JPEG para imagens coloridas e mantê-la parada por tanto tempo é um preço alto a ser pago, principalmente quando pretende-se um desempenho elevado, que justifique a implementação do compressor em hardware.

4.2.2.2 Arquitetura do Conversor de Espaço de Cores Paralela

Esta solução passa por um incremento significativo na arquitetura do conversor de espaço de cores, visando a obtenção uma taxa de produção mais elevada do que a obtida na primeira solução.

O datapath da arquitetura, apresentado na fig. 4.7, é capaz de gerar um componente de cor válido na sua saída a cada ciclo de *clock*, considerando o *pipeline* preenchido.

O principal inconveniente desta solução deve-se ao fato do significativo incremento em recursos utilizados pelo conversor de espaço de cores, pois o número de somadores cresceu 175 % em relação à arquitetura original. Por outro lado, a arquitetura do compressor JPEG para imagens coloridas seria utilizada 100% do tempo, já que uma nova saída válida é disponibilizada a cada ciclo de *clock*. Com essa solução, além de obter a melhor taxa possível de processamento, a arquitetura original do compressor para imagens coloridas não precisaria ser alterada.

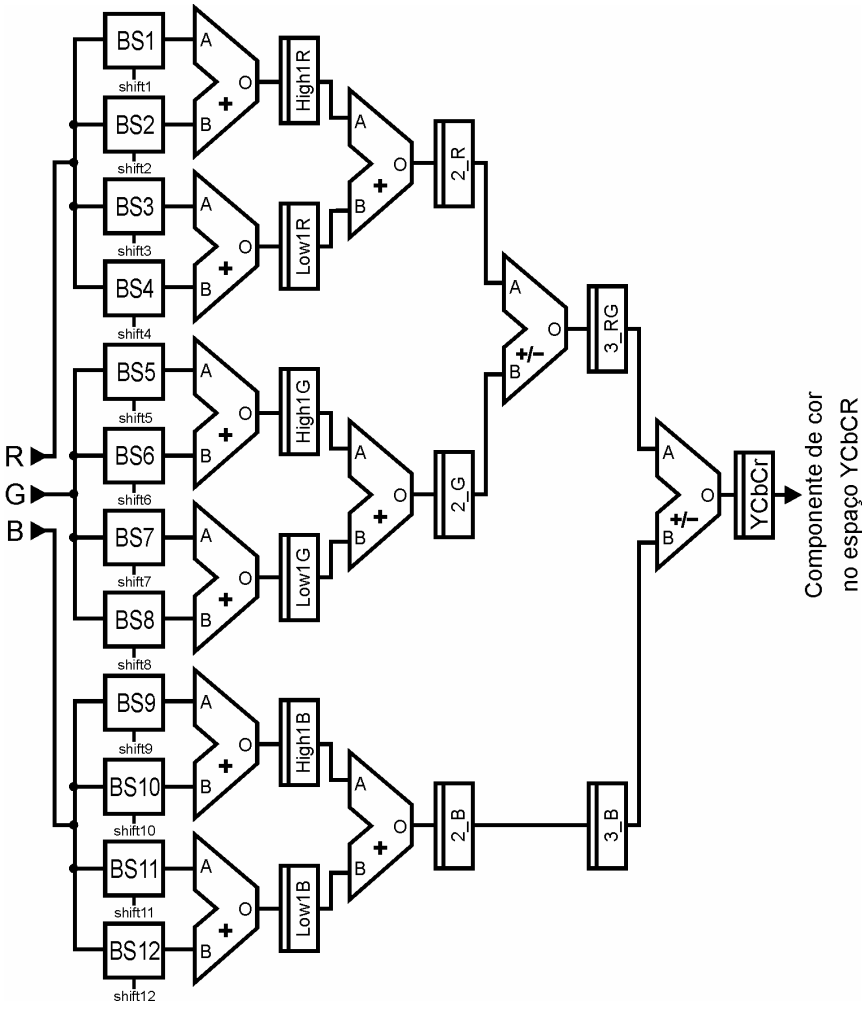


FIGURA 4.7 – *Datapath* paralelo do conversor de espaço de cores

4.2.2.3 Utilização de *Clocks* Múltiplos do *Clock* Principal

Uma possibilidade para igualar as taxas de produção e consumo das duas arquiteturas é utilizar um *clock* mais elevado na arquitetura do conversor de espaço de cores e um *clock* mais lento para a arquitetura do compressor JPEG para imagens coloridas, utilizando as duas arquiteturas originais.

Neste caso, o *clock* do conversor de espaço de cores teria que ser três vezes mais rápido que o do compressor para imagens coloridas. Segundo os resultados obtidos através da síntese das arquiteturas, a freqüência máxima de operação da arquitetura do conversor de espaço de cores é de 32,7MHz e a freqüência máxima da arquitetura do compressor JPEG para imagens coloridas é de 16,9MHz. Então, a solução com *clocks* múltiplos seria viável, por exemplo, com o uso de um *clock* de 10,9MHz na arquitetura do conversor de espaço de cores e um *clock* de 32,7MHz na arquitetura do compressor integrado.

Para a conexão das duas arquiteturas é necessária a existência de um *buffer* para armazenar as saídas válidas do conversor, de acordo com o seu *clock*, e permitir a leitura para o compressor JPEG para imagens coloridas, como o apresentado na fig. 4.8, onde utiliza-se o registrador *Sinc* com *clock* igual ao do conversor de espaço de cores. A escrita só é permitida quando um valor válido for produzido por esta arquitetura.

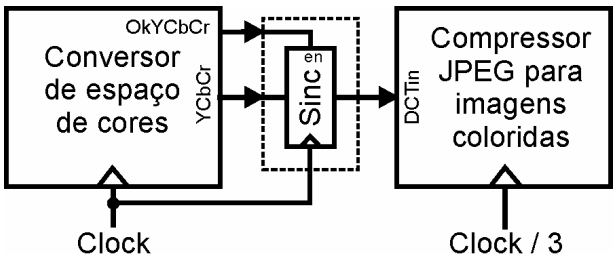


FIGURA 4.8 – Proposta de *buffer* de sincronismo

Com esse tipo de solução a arquitetura do compressor adaptado pode ser utilizada 100% do tempo em relação ao seu *clock*. A questão é que o compressor não pode operar na sua freqüência máxima, o que degrada o seu desempenho. Para que o compressor seja utilizado em sua freqüência máxima, o conversor de espaço de cores teria que operar a, pelo menos, 50MHz.

O principal problema desta solução está na garantia do perfeito casamento das fases dos dois *clocks*. Outro problema é a dificuldade de se trabalhar com *clocks* múltiplos, principalmente para aplicações que visem a síntese em FPGAs, como é o caso deste trabalho.

4.2.3 Considerações Finais sobre o Compressor JPEG para Imagens Coloridas com Conversão de Espaço de Cores

O compressor JPEG para imagens coloridas, considerando a conversão de espaço de cores, não foi completamente desenvolvido nesta dissertação, como já foi explicado. Mas as arquiteturas para o conversor de espaço de cores e para o compressor JPEG para imagens coloridas foram desenvolvidas, descritas em VHDL, sintetizadas e simuladas.

A síntese da arquitetura desenvolvida para o conversor de espaço de cores utilizou 281 células lógicas, sendo capaz de operar a uma freqüência máxima de 32,7MHz, processando uma imagem colorida de 640 x 480 *pixels* em 84,6ms e permitindo uma taxa de processamento de 11,8 imagens por segundo.

A síntese do compressor para imagens coloridas utilizou 6.315 células lógicas e 12.080 bits de memória, sendo capaz de operar a uma freqüência máxima de 16,9MHz, processando uma imagem colorida de 640x480 *pixels* em 54,4ms e permitindo uma taxa de processamento de 18,4 imagens por segundo.

A integração das arquiteturas do conversor de espaço de cores e do compressor JPEG para imagens coloridas não chegou a ser implementada, como já foi explicado, mas, ainda assim, é possível realizar algumas estimativas em termos de área ocupada e de desempenho.

A união das duas arquiteturas de maneira direta, como foi proposto no item 4.2.2.1, utilizaria cerca de 6.600 células lógicas e 12.080 bits de memória, sendo possível o uso do mesmo dispositivo necessário ao compressor JPEG para imagens coloridas, ou seja, o FPGA EPF10K130EQC240-1 [ALT 2001a] da Altera. A freqüência máxima de operação ficaria em torno dos 16MHz, um pouco menor do que a freqüência do compressor JPEG para imagens coloridas. Considerando as bolhas que seriam inseridas no *pipeline* do compressor, a arquitetura integrada seria capaz de processar uma imagem colorida de 640x480 *pixels* em 173ms, permitindo uma taxa de processamento de 5,8 imagens por segundo. Esta taxa de processamento estimada é bem inferior à taxa obtida pelo compressor JPEG de imagens coloridas, que é de 18,1 imagens por segundo, por causa do número excessivo de bolhas inseridas no *pipeline*.

A integração das duas arquiteturas com o conversor de espaço de cores paralelo, que foi proposta no item 4.2.2.2, causaria um sensível impacto em termos de células lógicas ocupadas. A arquitetura paralela não chegou a ser desenvolvida, mas é possível estimar que esta arquitetura utilizaria três vezes mais células lógicas do que a arquitetura não paralela. Então, a arquitetura integrada utilizaria algo em torno de 7.200 células lógicas e 12.080 bits de memória. Deste modo, o FPGA EPF10K130EQC240-1, utilizado para o compressor, não poderia ser utilizado para a síntese da arquitetura integrada, já que este dispositivo possui 6.656 células lógicas [ALT 2001a]. A freqüência máxima de operação seria a mesma da primeira solução, tendo como base a freqüência do compressor JPEG para imagens coloridas, ficando próxima dos 16MHz. A arquitetura integrada seria capaz de processar uma imagem colorida de 640x480 *pixels* em 58ms, permitindo uma taxa de processamento de 17,2 imagens por segundo, ficando muito próxima da taxa de processamento da arquitetura do compressor JPEG para imagens coloridas.

A solução que prevê o uso de *clocks* múltiplos, proposta no item 4.2.2.3 utilizaria praticamente os mesmos recursos da primeira solução, pois exige a inserção do *buffer* de sincronização entre as arquiteturas, mas descarta as adaptações para permitir a inserção de bolhas no *pipeline* do compressor para imagens coloridas. Então, seriam utilizadas cerca de 6.600 células lógicas e 12.080 bits de memória e também seria possível o uso do FPGA EPF10K130EQC240-1 da Altera. Se forem utilizados os *clocks* propostos de 32,7MHz para o conversor de espaço de cores e de 10,9MHz para o compressor para imagens coloridas, a arquitetura integrada seria capaz de processar uma imagem colorida de 640x480 em 84,6ms, com uma taxa de processamento de 11,8 imagens por segundo. Esta solução apresenta uma melhora significativa, em termos de desempenho, em relação à primeira solução, mas perde para a segunda solução, pois o compressor não pode operar em sua freqüência máxima.

A tab. 4.13 apresenta um comparativo entre as estimativas realizadas para as três diferentes soluções propostas para a integração do conversor de espaço de cores com o compressor JPEG para imagens coloridas.

TABELA 4.13 – Comparações entre estimativas para soluções de integração das arquiteturas do conversor de espaço de cores e do compressor JPEG para imagens coloridas

	Células Lógicas	**Bits de Memória**	**Taxa de Processamento***
Solução 1 Arquiteturas Originais	6.600	12.080	5,8
Solução 2 Arquitetura do conversor paralela	7.200	12.080	17,2
Solução 3 Uso de *clocks* múltiplos	6.600	12.080	11,8

* em imagens/seg, considerando imagens coloridas de 640 x 480 *pixels*

As três soluções propostas para a integração das arquiteturas do conversor de espaço de cores e do compressor JPEG para imagens coloridas possuem vantagens e desvantagens. A solução paralela (nº 2 na tab. 4.13) é a mais vantajosa em termos de desempenho e possui um pequeno custo adicional em termos de células lógicas, por isso, esta é a solução de maior viabilidade dentre as três soluções propostas.

5 Conclusões e Trabalhos Futuros

Esta dissertação apresentou o desenvolvimento de arquiteturas direcionadas à compressão de imagens JPEG. Inicialmente, o próprio padrão JPEG foi superficialmente apresentado, como forma de dar suporte teórico aos capítulos que trataram do desenvolvimento das arquiteturas.

Uma arquitetura integrada de compressor JPEG para imagens em tons de cinza foi completamente desenvolvida, onde a arquitetura foi detalhadamente explicada tendo sido apresentados os resultados da síntese, direcionada a FPGAs da Altera, e das simulações realizadas. Esta arquitetura possui uma latência mínima de 237 ciclos de *clock* e processa uma imagem de 640 x 480 *pixels* em 18,5ms, permitindo uma taxa de processamento de 54 imagens por segundo. São utilizadas 6.199 células lógicas, 7.436 bits de memória, 10 pinos de entrada e 33 pinos de saída do dispositivo EPF10K130EQC240-1. As estimativas realizadas em torno da taxa de compressão obtida por esta arquitetura indicam que ela seria de, aproximadamente, 6,2 vezes ou de 84 %.

Também foi completamente desenvolvida uma arquitetura de compressor JPEG para imagens coloridas, a partir de adaptações na arquitetura para a compressão de imagens em tons de cinza. As adaptações realizadas foram apresentadas em detalhes, juntamente com os resultados de síntese. Na arquitetura do compressor JPEG para imagens coloridas foram utilizadas 6.315 células lógicas, 12.080 bits de memória, 10 pinos de entrada e 33 pinos de saída do dispositivo EPF10K130EQC240-1 da Altera. Esta arquitetura também possui a latência mínima de 237 ciclos de *clock*, sendo capaz de processar uma imagem colorida de 640 x 480 *pixels* em 54,4ms, permitindo uma taxa de processamento de 18,4 imagens por segundo. A taxa de compressão obtida, segundo estimativas, seria de, aproximadamente, 14,4 vezes ou de 93 %, considerando a taxa de *downsampling* de 4:1:1, que é obtida através do processador de entrada e saída, o qual não foi desenvolvido no escopo desta dissertação**.** Esta taxa é um pouco inferior à esperada quando da proposta do trabalho, onde estimava-se, empiricamente, uma taxa de compressão de 20 vezes [AGO 2000].

Por fim, foi desenvolvida e apresentada uma arquitetura para o conversor de espaço de cores, a ser usado em aplicações cujas imagens estejam representadas no espaço de cores RGB. O conversor foi completamente desenvolvido, sintetizado e simulado e os dados obtidos neste processo foram apresentados. Esta arquitetura não chegou a ser integrada com a arquitetura do compressor de imagens coloridas, mas algumas sugestões e estimativas foram realizadas nesta direção. A arquitetura desenvolvida para o conversor utilizou 281 células lógicas, 28 pinos de entrada e 9 pinos de saída de um dispositivo EPF10K30ETC144-1. O conversor possui uma latência de 6 ciclos de *clock* e é capaz de processar uma imagem colorida de 640 x 480 *pixels* em 84,6ms, o que permite uma taxa de processamento de 11,8 imagens por segundo.

Os resultados obtidos foram considerados satisfatórios e encorajam o uso e reuso das arquiteturas desenvolvidas em outros projetos. Considerando a principal aplicação alvo, isto é, a monitoração de trânsito, a arquitetura do compressor JPEG para imagens em tons de cinza atingiu taxas de compressão e de desempenho que justificariam plenamente a sua utilização em conjunto com os monitores atuais.

São indicados alguns trabalhos futuros principais, no sentido de dar continuidade a esta dissertação. O primeiro deles, diz respeito a avaliação dos impactos dos erros

gerados pelas arquiteturas desenvolvidas, em termos da distorção das imagens comprimidas. Então, sugere-se o desenvolvimento, em software, do compressor apresentado e desenvolvido em hardware, bem como o desenvolvimento, também em software, de um compressor JPEG sem as aproximações utilizadas no primeiro compressor, com o intuito de comparar as diferenças dos resultados obtidos, considerando, para tanto, grupos de imagens variadas. Este desenvolvimento em software também é importante para uma avaliação mais precisa das taxas médias de compressão obtidas.

Outro trabalho futuro, diz respeito a realização de explorações arquiteturais sobre as arquiteturas que foram desenvolvidas. A primeira tarefa neste sentido, seria a substituição dos somadores *ripple carry* por somadores *carry look ahead* e, então, avaliar os ganhos em termos de desempenho e as perdas em termos de utilização de recursos.

O desenvolvimento, em hardware e software, do processador de entrada e saída também é indicado como trabalho futuro, pois não foi desenvolvido no escopo desta dissertação e tem papel crucial para o uso efetivo das arquiteturas propostas.

Por fim, é sugerida a integração da arquitetura do compressor JPEG (compressor e processador de entrada e saída) com a arquitetura de uma interface padrão PCI com o objetivo de viabilizar o uso do compressor JPEG como coprocessador de um processador convencional inserido em um microcomputador tipo PC. A descrição arquitetural da PCI pode ser adquirida como um *core* IP, ser reutilizada a partir de outros trabalhos acadêmicos ou ser desenvolvida por completo.

Anexo 1 O Formato de Arquivo JFIF

Uma das características mais surpreendentes no padrão JPEG é que ele não define um formato de arquivo, de forma a explicitar o que uma aplicação precisa fazer para criar imagens que possam ser abertas por outras aplicações [MIA 99]. Este vácuo foi inicialmente preenchido pelo formato JFIF (*JPEG File Interchange Format*), criado por Eric Hamilton [HAM 92]. O formato JFIF acabou tornando-se um sinônimo de arquivo JPEG e será abordado com maiores detalhes neste anexo.

As regras definidas pelo formato de arquivo devem ser seguidas pelo processador de entrada e saída, cuja arquitetura não foi desenvolvida no escopo desta dissertação. O processador de entrada e saída é quem monta o arquivo JPEG e, para tanto, utiliza as tabelas de Huffman e de quantização usadas na compressão, bem como as palavras JPEG geradas pelo compressor. Embora o processador de entrada e saída não tenha sido desenvolvido, muitas definições realizadas para o compressor refletem na operação do processador de entrada e saída e, por isso, este anexo irá apresentar, além do formato de arquivo JFIF, algumas destas definições. Desta forma, este anexo, além de apresentar o formato JFIF, também servirá como um guia para o futuro desenvolvimento do processador de entrada e saída.

A estrutura de um arquivo JFIF para a compressão JPEG no modo *baseline*, que é utilizada neste dissertação, está apresentada na fig. A1.1, de onde pode-se perceber que o padrão JPEG define marcadores para separar as estruturas dentro do arquivo.

FIGURA A1.1 – Organização do arquivo JFIF

Os marcadores definidos pelo padrão JPEG são formados por 2 bytes e começam sempre por FF_{16}. Pode-se dividir os marcadores em dois grupos principais: os que não contém dados associados e os que contém. Os marcadores com dados associados são seguidos imediatamente por um valor de 2 bytes que indica o número de bytes contidos no marcador. As tabs. A1.1 e A1.2 apresentam os marcadores dos dois tipos.

TABELA A1.1 – Marcadores JPEG com dados

Valor	Símbolo Usado no Padrão JPEG	Descrição
FFC0	SOF0	Começo do *frame* (*baseline*)
FFC1*	SOF1	Começo do *frame* (seqüencial estendido)
FFC2*	SOF2	Começo do *frame* (progressivo)
FFC3*	SOF3	Começo do *frame* (sem perdas)
FFC4	DHT	Define a tabela de Huffman
FFC5*	SOF5	Começo do *frame* (seqüencial diferencial)
FFC6*	SOF6	Começo do *frame* (progressivo diferencial)
FFC7*	SOF7	Começo do *frame* (sem perdas diferencial)
FFC8*	JPG	Reservado
FFC9*	SOF9	Começo do *frame* (seqüencial estendido com codificação aritmética)
FFCA*	SOF10	Começo do *frame* (progressivo com codificação aritmética)
FFCB*	SOF11	Começo do *frame* (sem perdas com codificação aritmética)
FFCC*	DAC	Define as condições da codificação aritmética
FFCD*	SOF13	Começo do *frame* (seqüencial diferencial com codificação aritmética)
FFCE*	SOF14	Começo do *frame* (progressivo diferencial com codificação aritmética)
FFCF*	SOF15	Começo do *frame* (sem perdas diferencial com codificação aritmética)
FFDA	SOS	Início do *scan*
FFDB	DQT	Define as tabelas de quantização
FFDC*	DNL	Define o número de linhas
FFDD*	DRI	Define o intervalo de recomeço
FFDE*	DHP	Define a progressão hierárquica
FFDF*	EXP	Expande componentes de referência
FFE0 – FFEF**	APP0 – APP15	Dados específicos da aplicação
FFFE*	COM	Comentário
FFF0 – FFFD*	JPG0 – JPG13	Reservado
FF02 – FFBF*	RES	Reservado

* Marcadores não utilizados pelo processador de entrada e saída definido para esta dissertação

** Apenas os marcadores APP_0 e APP_1 (FFE0 e FFE1) serão utilizados pelo processador de entrada e saída definido para esta dissertação.

TABELA A1.2 – Marcadores JPEG sem dados

Valor	Símbolo Usado no Padrão JPEG	Descrição
FF01*	TEM	Marcador temporário para codificação aritmética.
FFD0 – FFD7*	RST_0 – RST_7	Marcador de recomeço usado para separar dados codificados independentemente.
FFD8	SOI	Marcador de início de imagem.
FFD9	EOI	Marcador de fim de imagem.

* Marcadores não utilizados pelo processador de entrada e saída definido para esta dissertação

Das tabs. A1.1 e A1.2, pode-se perceber que a maior parte dos marcadores definidos no padrão JPEG não serão usados no modo de operação *baseline*, que é o foco dos compressores desenvolvidos nesta dissertação. Maiores informações sobre os formatos de arquivo criados para os demais modos de operação do padrão JPEG podem ser encontradas em [THE92], [PEN92] e [MIA99].

As estruturas dos marcadores utilizados no modo *baseline* serão detalhadas neste anexo como forma de definir, tão completamente quanto possível, o arquivo JFIF que deverá ser montado pelo processador de entrada e saída.

O arquivo JFIF, por definição, deve começar com o marcador SOI, que indica o início da imagem. A seguir, é necessário que exista o marcador APP_0, que é obrigatório para o formato JFIF.

Os marcadores APP_n são específicos da aplicação e podem ser usados livremente para identificar o codificador que gerou a imagem ou para inserir qualquer outro tipo de informação que seja específica da aplicação. Usualmente os marcadores APP_n são simplesmente desconsiderados no processo de descompressão da imagem. A estrutura dos marcadores APP_n é simples, após o identificador do marcador, que ocupa 2 bytes, os próximos 2 bytes indicam quantos dados existem dentro do marcador, sendo que estes dois bytes também são contados. O arquivo JFIF definido por esta dissertação utilizará apenas dois marcadores específicos da aplicação, que são os marcadores APP0 e APP1. O marcador APP0 é obrigatório no formato de arquivo JFIF e serve de identificação do formato. Este marcador, por definição, deve possuir a seguinte estrutura:

```
FF E0 00 07 4A 46 49 46 00
```

Onde os 2 primeiros bytes (FF E0) são o próprio marcador APP0, os dois bytes seguintes (00 07) indicam que o marcador contém sete bytes associados a ele e os últimos cinco bytes (4A 46 49 46 00) são os códigos ASCII referentes a palavra “JFIF” seguida de zero, como exige o formato JFIF.

O marcador APP1 é especifico da aplicação proposta por esta dissertação e terá a seguinte estrutura:

```
FF E1 00 19 55 46 52 47 53 20 4A 50 45 47 20 43 6F 6D 70 72
      65 73 73 6F 72 00 00
```

Os dois primeiros bytes (FF E1) identificam o marcador APP1, os dois bytes seguintes (00 19) indicam que o marcador possui 25 bytes associados (19 em hexadecimal). Os 23 bytes seguintes são os códigos ASCII da frase *UFRGS JPEG Compressor* seguida de dois bytes zero.

No arquivo JFIF, após os marcadores específicos da aplicação, está o marcador DQT, que define as tabelas de quantização. Como, no modo de operação *baseline*, são usadas somente duas tabelas de quantização, o marcador DQT será usado apenas duas vezes no arquivo definido por este anexo. A tab. A1.3 apresenta a estrutura deste marcador.

TABELA A1.3 – Estrutura do marcador DQT

Parâmetro	**Símbolo**	**Tamanho** (bits)
Marcador FFDB	DQT	16
Definição do tamanho do marcador	*Lq*	16
Precisão dos elementos da tabela	*Pq*	4
Identificador da tabela de quantização	*Tq*	4
Elementos da tabela de quantização	Q_k	(8 ou 16) x 64

Na tab. A1.3, o campo *Lq* indica o número de bytes associados ao marcador. Para o arquivo JFIF definido neste anexo, este campo será sempre fixo e terá o valor 43 em hexadecimal, indicando a presença de 67 bytes, dos quais 2 são usados pelo próprio campo *Lq*, um é usado pelos campos *Pq* e *Tq* e os demais 64 são usados para os elementos da tabela. O campo *Pq* indica qual o número de bits que será usado para cada elemento da tabela de quantização. Se *Pq* for 0, são usados oito bits por elemento da tabela, se *Pq* for 1, são usados 16 bits para cada elemento. No arquivo JFIF definido neste anexo, campo *Pq* será mantido sempre em 0, indicando que cada elemento da tabela de quantização ocupa 8 bits. O campo *Tq* identifica qual tabela de quantização está sendo definida, sendo que, para o modo *baseline*, *Tq* pode ter valor 0 ou 1. Para a compressão de imagens em tons de cinza, apenas uma tabela de quantização é utilizada, portanto *Tq* assumirá, sempre, valor 0. No campo Q_k estão os 64 elementos da tabela de quantização, com uma precisão de 8 bits.

O exemplo abaixo apresenta, em negrito, a definição das tabelas de quantização de luminância e de crominância utilizadas nesta dissertação. Os elementos das tabelas de quantização estão apresentados no anexo 2. No caso do processamento de imagens em tons de cinza, apenas a primeira definição é utilizada.

FF DB 00 43 00 *(64 elementos da tabela de quantização de luminância)*

FF DB 00 43 01 *(64 elementos da tabela de quantização de crominância)*

Após a definição das tabelas de quantização está o marcador SOF que indica o início do *frame*. Como, para o modo *baseline* existe apenas um *frame* por imagem, o marcador SOF aparece apenas uma vez no arquivo. A tab. A1.4 apresenta a estrutura do marcador SOF.

TABELA A1.4 – Estrutura do marcador SOF

Parâmetro	**Símbolo**	**Tamanho** (bits)
Marcador FFC0	SOF	16
Definição do tamanho do marcador	*Lf*	16
Precisão da amostragem	*P*	8
Altura da imagem em pixeis	*Y*	16
Largura da imagem em pixeis	*X*	16
Número de componentes no *frame*	*Nf*	8
Identificador do componente	C_i	8
Fator de amostragem horizontal de C_i	H_i	4
Fator de amostragem vertical de C_i	V_i	4
Tabela de quantização para C_i	Tq_i	8

O tamanho do marcador SOF é indicado pelo campo *Lf* , cujo valor depende do tipo de imagem que está sendo processada: em tons de cinza ou colorida. Se for colorida, este marcador ocupará 17 bytes, se for em tons de cinza, este marcador ocupará 11 bytes. O campo *P* informa qual a precisão das amostras que estão sendo usadas nos componentes do *frame*. Nos modos de compressão com perdas são permitidos 8 ou 12 bits por amostra, sendo que este trabalho utiliza oito bits por amostra. Os campos *Y* e *X* indicam o número de linhas e de colunas da imagem. O campo *Nf* indica o número de componentes contidos no *frame*. Se a imagem for em tons de cinza, há um único componente, mas se a imagem for colorida, existem três componentes (Y, Cb e Cr). Então o campo *Nf*, neste trabalho, poderá assumir o valor 1 ou o valor 3.

Os campos C_i, H_i, V_i e Tq_i são definidos para cada um dos *Nf* componentes contidos no *frame*. O campo C_i identifica o componente, onde 1 identifica o componente Y, 2 identifica o componente Cb e 3 identifica o componente Cr. O formato JFIF exige que a ordem de definição dos componentes seja Y, Cb e Cr. Os campos H_i e V_i definem as amostragens horizontais e verticais para o componente e, segundo o padrão, os valores de H_i e V_i podem ser 1, 2, 3 ou 4. A amostragem diz respeito a como a operação de *downsampling* foi efetuada. Para o processamento de imagens em tons de cinza, H_i e V_i terão valor igual a 1. No caso de imagens coloridas, como a taxa de *downsampling* utilizada neste trabalho é igual a 4:1:1, as amostragens horizontal e vertical de Y são respectivamente 2 e 2, enquanto que as amostragens horizontais e verticais de Cb e Cr são iguais a 1. O campo Tq_i indica a qual tabela de quantização o componente está associado, podendo, neste trabalho, ter valor 0 ou 1.

Abaixo estão apresentados dois possíveis marcadores SOF, o primeiro para imagens em tons de cinza e o segundo para imagens coloridas. O que está em negrito é definido para o arquivo JFIF usado neste trabalho. Os dados em itálico são dependentes da imagem que será comprimida. Nos dois exemplos, a imagem possui 480 x 640 *pixels*. No primeiro exemplo, o *frame* possui um único componente de cor, cujas amostragens horizontal e vertical são iguais a 1 e que está associado à tabela de quantização 0. No segundo exemplo, o *frame* possui três componentes de cor. O componente Y possui amostragens horizontal e vertical iguais a 2 e está associado à tabela de quantização 0. Os componentes Cb e Cr possuem uma amostragem horizontal e vertical igual a 1 e estão associados à tabela de quantização 1.

FF C0 00 0B 00 *01 E0 02 80* **01 01 11 00**

FF C0 00 11 00 *01 E0 02 80* **03 01 22 00 02 11 01 03 11 01**

Após a definição do *frame*, estão dispostas as definições das tabelas de Huffman, que são determinadas pelo marcador DHT. O modo *baseline* prevê o uso de apenas quatro tabelas de Huffman, portanto, serão usados quatro marcadores DHT no arquivo definido por este anexo. A tab. A1.5 apresenta a estrutura do marcador DHT.

TABELA A1.5 – Estrutura do marcador DHT

Parâmetro	**Símbolo**	**Tamanho** (bits)
Marcador FFC4	DHT	16
Definição do tamanho do marcador	*Lh*	16
Classe da tabela	*Tc*	4
Identificador da tabela de Huffman	*Th*	4
Número de códigos com comprimento i	L_i	8
Valor associado com cada código	V_{ij}	8

O tamanho do marcador DHT é indicado pelo campo *Lh*. O campo *Tc* indica qual é a classe da tabela de Huffman. Se *Tc* for igual a 0 a tabela é DC, se for igual a 1 a tabela é AC. O campo *Th* é o identificador da tabela de Huffman dentro da classe e indica se a tabela é referente ao componente de luminância (se possuir valor 0) ou de crominância (se possuir valor 1). Para a compressão de imagens em tons de cinza, *Th* possui sempre valor 0.

O campo *Li* indica quantos códigos de Huffman de comprimento *i* existem na tabela. Como o tamanho máximo de cada código de Huffman é de 16 bits, *i* varia de 1 a 16. A soma de todos *Li* é igual ao número total de códigos de Huffman contidos na tabela e o número máximo de códigos com o mesmo comprimento é 255. Cada código de Huffman é expresso por *Vij*, onde *i* indica o comprimento do código, que pode variar de 1 a 16, e *j* identifica o código entre os códigos de comprimento *i* e pode variar de 1 a *Li*.

O exemplo abaixo apresenta quatro marcadores DHT, o primeiro refere-se à tabela de Huffman para componentes de luminância DC, o segundo refere-se à tabela de Huffman para componentes de luminância AC, o terceiro refere-se à tabela de Huffman para componentes de crominância DC e o quarto refere-se à tabela de Huffman para componentes de crominância AC. Em caso de processamento de imagens em tons de cinza, apenas os dois primeiros marcadores são usados.

FF C4 00 1A 00 00 01 05 01 01 01 01 01 01 00 00 00 00 00 00 00 *(56 bits de dados da tabela de Huffman DC para luminância)*

FF C4 01 33 01 00 02 01 03 03 02 04 03 05 05 04 04 00 00 01 7D *(2300 bits de dados da tabela de Huffman AC para luminância)*

FF C4 00 45 10 00 03 01 01 01 01 01 01 01 01 01 00 00 00 00 00 *(69 bits de dados da tabela de Huffman DC para crominância)*

FF C4 01 2F 11 00 02 01 02 04 04 03 04 07 05 04 04 00 01 02 77 *(2265 bits de dados da tabela de Huffman AC para crominância)*

O último marcador antes dos dados comprimidos é o marcador que define o início do *scan*, identificado como SOS, apresentado na tab. A1.6.

TABELA A1.6 – Estrutura do marcador SOS

Parâmetro	**Símbolo**	**Tamanho** (bits)
Marcador FFDA	SOS	16
Definição do tamanho do marcador	*Ls*	16
Número de componentes no *scan*	*Ns*	8
Identificador do componente	Cs_k	8
Tabela de Huffman DC do componente	Td_k	4
Tabela de Huffman AC do componente	Ta_k	4
Início da seleção espectral (0-63)	*Ss*	8
Fim da seleção espectral (0-63)	*Se*	8
Aproximação sucessiva alta	*Ah*	4
Aproximação sucessiva baixa	*Al*	4

O tamanho do marcador SOS é indicado pelo campo *Ls*. No marcador SOS o componente *Ns* contém o número de componentes dentro do *scan*. Caso o processamento seja de imagens em tons de cinza, *Ns* será igual a 1, uma vez que só existe um componente de cor. Para imagens coloridas intercaladas, *Ns* será igual a 3, uma vez que são utilizados três componentes de cor (Y, Cb e Cr).

Os campos *Csk*, *Tdk* e *Tak* são definidos para os *Ns* componentes do *scan*. O campo *Csk* identifica o componente e esta identificação deve casar com a identificação do componente feita no *frame*. Os campos *Tdk* e *Tak* identificam as tabelas de Huffman DC e AC para o componente identificado por *Csk*. Os campos *Ss* e *Se* identificam o início e o fim da seleção espectral [PEN 92]. Para o modo *baseline*, a seleção espectral identifica os índices inicial e final da matriz de 64 elementos que é processada pela DCT, por isso o valor de *Ss* deve ser sempre zero e o valor de *Se* deve ser sempre 63 [PEN 92]. Os campos *Ah* e *Al* definem a aproximação sucessiva, recurso que não é utilizado no modo *baseline* [PEN 92] devendo, então, serem mantidos em zero.

O exemplo abaixo apresenta dois marcadores SOS, o primeiro considerando uma imagem em tons de cinza e o segundo, uma imagem colorida e intercalada.

No primeiro exemplo existe apenas um elemento, que está associado às tabelas de Huffman AC e DC zero.

No segundo exemplo existem três elementos, onde o elemento 1 (Y) está associado às tabelas de Huffman AC e DC zero e os elementos 2 e 3 (Cb e Cr) estão associados às tabelas de Huffman AC e DC um.

FF DA 00 08 01 01 00 00 3F 00

FF DA 00 0C 03 01 00 02 11 03 11 00 3F 00

Após os dados do marcador SOS estão dispostos os dados da imagem comprimida. Os dados comprimidos são a única parte de um arquivo JPEG que não está dentro de um marcador. No modo *baseline*, dentro dos dados comprimidos não são permitidos marcadores, portanto o identificador de marcador FF não pode aparecer dentro destes dados.

Por fim, após os dados comprimidos a última informação que deve existir no arquivo JFIF é o marcador EOI, que indica o final da imagem.

Unindo todos os exemplos de marcadores que foram utilizados neste anexo é possível montar um arquivo JFIF vazio (sem os elementos das tabelas de quantização, das tabelas de Huffman e dos dados da imagem comprimida). Desta forma, segue um exemplo de arquivo JFIF para uma imagem em tons de cinza, sendo que todos os dados que aparecem em negrito devem ser utilizados literalmente no arquivo gerado pelo processador de entrada e saída.

FF D8

FF E0 00 07 4A 46 49 46 00

FF E1 00 19 55 46 52 47 53 20 4A 50 45 47 20 43 6F 6D 70 72 65 73 73 6F 72 00 00

FF DB 00 43 00 *(64 elementos da tabela de quantização)*

FF C0 00 0B 00 *01 E0 02 80* **01 01 11 00**

FF C4 00 1A 00 00 01 05 01 01 01 01 01 01 00 00 00 00 00 00 00 *(56 bits de dados da tabela de Huffman DC para luminância)*

FF C4 01 33 01 00 02 01 03 03 02 04 03 05 05 04 04 00 00 01 7D *(2300 bits de dados da tabela de Huffman AC para luminância)*

FF DA 00 08 01 01 00 00 3F 00

(dados da imagem comprimida)

FF D9

Anexo 2 Tabelas de Quantização, Deslocamentos e Conteúdo das Memórias

Este anexo apresenta as tabelas de quantização para os componentes de luminância (Y) e crominância (Cb e Cr), que são propostas pelo padrão JPEG [THE 92] e que foram utilizadas nas implementações desenvolvidas nesta dissertação. Também é apresentada a tabela de fatores de escala da DCT 2-D, cuja correção é anexada ao cálculo da quantização, como já explicado no item 3.1.5 desta dissertação. Por fim, são apresentadas algumas informações específicas da implementação do quantizador, como os deslocamentos realizados por cada *barrel shifter* para cada elemento da matriz de entrada, o controle dos quatro *barrel shifters* para cada deslocamento e o conteúdo das memórias utilizadas no quantizador.

É importante ressaltar que, para o compressor de imagens em tons de cinza, apenas a tabela de quantização referente a luminância é utilizada.

As tabelas de quantização para luminância e crominância, propostas pelo padrão, estão apresentadas em Qy_{ij} *e* Qc_{ij}, respectivamente:

$$Qy_{ij} = \begin{bmatrix} 16 & 11 & 10 & 16 & 124 & 140 & 151 & 161 \\ 12 & 12 & 14 & 19 & 126 & 158 & 160 & 155 \\ 14 & 13 & 16 & 24 & 140 & 157 & 169 & 156 \\ 14 & 17 & 22 & 29 & 151 & 187 & 180 & 162 \\ 18 & 22 & 37 & 56 & 168 & 109 & 103 & 177 \\ 24 & 35 & 55 & 64 & 181 & 104 & 113 & 192 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 199 \end{bmatrix}$$

$$Qc_{ij} = \begin{bmatrix} 17 & 18 & 24 & 47 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{bmatrix}$$

A tabela de fatores de escala, referente ao cálculo da DCT 2-D, está apresentada em Fe_{ij} e foi obtida a partir dos cálculos apresentados no item 3.1.5.

$$Fe_{ij} = \begin{bmatrix} 8,00 & 11,10 & 10,45 & 9,41 & 8,00 & 6,29 & 4,33 & 2,21 \\ 11,10 & 15,39 & 14,50 & 13,05 & 11,10 & 8,72 & 6,01 & 3,07 \\ 10,45 & 14,50 & 13,66 & 12,29 & 10,45 & 8,21 & 5,66 & 2,88 \\ 9,41 & 13,05 & 12,29 & 11,06 & 9,41 & 7,39 & 5,09 & 2,60 \\ 8,00 & 11,10 & 10,45 & 9,41 & 8,00 & 6,29 & 4,33 & 2,21 \\ 6,29 & 8,72 & 8,21 & 7,39 & 6,29 & 4,94 & 3,40 & 1,73 \\ 4,33 & 6,01 & 5,66 & 5,09 & 4,33 & 3,40 & 2,34 & 1,20 \\ 2,21 & 3,07 & 2,88 & 2,60 & 2,21 & 1,73 & 1,20 & 0,61 \end{bmatrix}$$

O cálculo realizado pelo quantizador, que foi apresentado no item 3.2, é:

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

onde C_{ij} é o coeficiente gerado pela DCT 2-D sendo, portanto, a entrada do quantizador e Cq_{ij} é o coeficiente quantizado. O quantizador realiza a multiplicação de C_{ij} pela constante $1/(Q_{ij} \times Fe_{ij})$.

É importante ressaltar que a operação do quantizador é realizada coluna a coluna, pois os resultados da DCT 2-D estão organizados desta forma, deste modo, primeiro são processadas todas as linhas da coluna 0, depois todas as linhas da coluna 1 e assim sucessivamente.

O fator $1/(Q_{ij} \times Fe_{ij})$ é utilizado como referência para os deslocamentos realizados no processo de multiplicação, que já foi explicado no item 3.2. O multiplicador utiliza quatro *barrel shifters*, portanto, os valores dos fatores $1/(Q_{ij} \times Fe_{ij})$ são aproximados para que contenham apenas quatro bits em nível alto, respeitando esta restrição. Deslocamentos maiores que 17 bits para a direita também não são realizados pelos deslocadores, exigindo uma nova aproximação dos fatores $1/(Q_{ij} \times Fe_{ij})$.

Como todos os fatores $1/(Q_{ij} \times Fe_{ij})$ são conhecidos e constantes, para *i* e *j* variando entre 0 e 7, é possível prever quais serão os deslocamentos necessários para o cálculo de qualquer C_{ij}.

As tabs. A2.1 e A2.2 apresentam todos os possíveis fatores gerados para o componente de luminância (Y) e para os componentes de crominância (Cb e Cr), respectivamente. Nas tabs. A2.1 e A2.2 estão apresentados, respectivamente, os fatores $1/(Qy_{ij} \times Fe_{ij})$ e $1/(Qc_{ij} \times Fe_{ij})$, em decimal e em binário, bem como os deslocamentos, referentes a estes fatores, para cada um dos quatro *barrel shifters* presentes na arquitetura do quantizador. As tabs. A2.1 e A2.2 estão ordenadas de acordo com a ordem de processamento do quantizador, ou seja, coluna a coluna, onde *i* identifica a linha e *j* identifica a coluna do elemento que está sendo considerado.

TABELA A2.1 – Constantes de quantização e deslocamentos para luminância (Y)

		$1/(Qy_{ij} \times Fe_{ij})$		Barrel Shifter			
i	**j**	**Decimal**	**Binário**	**1**	**2**	**3**	**4**
0	0	0,007812500	0,000 000 100 000 000 000	7	-	-	-
1	0	0,007509998	0,000 000 011 110 000 000	8	9	10	11
2	0	0,006833633	0,000 000 011 011 000 000	8	9	11	12
3	0	0,007593126	0,000 000 011 110 000 000	8	9	10	11
4	0	0,006944444	0,000 000 011 100 010 000	8	9	10	14
5	0	0,006628951	0,000 000 011 011 000 000	8	9	11	12
6	0	0,004713671	0,000 000 010 011 010 000	8	11	12	14
7	0	0,006292552	0,000 000 011 001 100 000	8	9	12	13
0	1	0,008192725	0,000 000 100 001 100 010	7	12	13	17
1	1	0,005414407	0,000 000 010 110 001 000	8	10	11	15
2	1	0,005305757	0,000 000 010 101 100 000	8	10	12	13
3	1	0,004508279	0,000 000 010 010 011 000	8	11	14	15
4	1	0,004096363	0,000 000 010 000 110 000	8	13	14	-
5	1	0,003277171	0,000 000 001 101 011 000	9	10	12	14
6	1	0,002601875	0,000 000 001 010 101 010	9	11	13	15
7	1	0,003550443	0,000 000 001 110 100 000	9	10	11	13
0	2	0,009567086	0,000 000 100 111 000 000	7	10	11	12
1	2	0,004926775	0,000 000 010 100 001 010	8	10	15	17
2	2	0,004576457	0,000 000 010 010 101 000	8	11	13	15
3	2	0,003698244	0,000 000 001 111 000 000	9	10	11	12
4	2	0,002585699	0,000 000 001 010 100 100	9	11	13	16
5	2	0,002213926	0,000 000 001 001 000 100	9	12	16	-
6	2	0,002266368	0,000 000 001 001 010 010	9	12	14	17
7	2	0,003650105	0,000 000 001 110 100 000	9	10	11	13
0	3	0,006643985	0,000 000 011 011 000 000	8	9	11	12
1	3	0,004033723	0,000 000 010 000 100 000	8	13	-	-
2	3	0,003390057	0,000 000 001 101 100 000	9	10	12	13
3	3	0,003117376	0,000 000 001 100 110 000	9	10	13	14
4	3	0,001898281	0,000 000 000 111 100 000	10	11	12	13
5	3	0,002114047	0,000 000 001 000 101 010	9	13	15	17
6	3	0,002257744	0,000 000 001 001 001 100	9	12	15	16
7	3	0,003931623	0,000 000 010 000 000 110	8	16	-	17
0	4	0,005208333	0,000 000 010 101 010 000	8	10	12	14
1	4	0,003466153	0,000 000 001 110 001 000	9	10	11	15
2	4	0,002391771	0,000 000 001 001 110 000	9	12	13	14
3	4	0,002084387	0,000 000 001 000 100 010	9	13	-	17
4	4	0,001838235	0,000 000 000 111 100 000	10	11	12	13
5	4	0,001964134	0,000 000 001 000 000 010	9	-	-	17
6	4	0,002242426	0,000 000 001 001 001 010	9	12	15	17
7	4	0,004045212	0,000 000 010 000 100 100	8	13	16	-
0	5	0,003977371	0,000 000 010 000 010 010	8	14	-	17
1	5	0,001977603	0,000 000 001 000 000 110	9	-	16	17
2	5	0,002136244	0,000 000 001 000 110 000	9	13	14	-
3	5	0,001555161	0,000 000 000 110 010 100	10	11	14	16

		$1/(Qy_{ij} \times Fe_{ij})$		Barrel Shifter			
i	**j**	**Decimal**	**Binário**	**1**	**2**	**3**	**4**
4	5	0,001459586	0,000 000 000 101 110 000	10	12	13	14
5	5	0,001947013	0,000 000 000 111 100 000	10	11	12	13
6	5	0,002429495	0,000 000 001 001 110 000	9	12	13	14
7	5	0,005766407	0,000 000 010 111 000 000	8	10	11	12
0	6	0,004528821	0,000 000 010 010 100 010	8	11	13	17
1	6	0,002775333	0,000 000 001 011 010 000	9	11	12	14
2	6	0,002561981	0,000 000 001 010 011 000	9	11	14	15
3	6	0,002455297	0,000 000 001 010 000 010	9	11	-	17
4	6	0,002242426	0,000 000 001 001 001 010	9	12	15	17
5	6	0,002601495	0,000 000 001 010 101 000	9	11	13	15
6	6	0,003556472	0,000 000 001 110 100 000	9	10	11	13
7	6	0,008127695	0,000 000 100 001 010 010	7	12	14	17
0	7	0,007427274	0,000 000 011 110 000 000	8	9	10	11
1	7	0,005938923	0,000 000 011 000 010 100	8	9	14	16
2	7	0,006192142	0,000 000 011 001 010 000	8	9	12	14
3	7	0,0062145	0,000 000 011 001 010 000	8	9	12	14
4	7	0,005883944	0,000 000 011 000 000 110	8	9	16	17
5	7	0,006267834	0,000 000 011 001 100 000	8	9	12	13
6	7	0,00828864	0,000 000 100 001 110 000	7	12	13	14
7	7	0,016587211	0,000 001 000 011 100 000	6	11	12	13

TABELA A2.2 – Constantes de quantização e deslocamentos para crominância (Cb e Cr)

		$1/(Qy_{ij} \times Fe_{ij})$		Barrel Shifter			
i	**j**	**Decimal**	**Binário**	**1**	**2**	**3**	**4**
0	0	0,007352941	0,000 000 011 110 000 000	8	9	10	11
1	0	0,005006665	0,000 000 010 100 100 000	8	10	13	-
2	0	0,003986286	0,000 000 010 000 010 100	8	14	16	-
3	0	0,001073775	0,000 000 000 100 011 000	10	14	15	-
4	0	0,001262626	0,000 000 000 101 001 010	10	12	15	17
5	0	0,001607018	0,000 000 000 110 100 100	10	11	13	16
6	0	0,002333029	0,000 000 001 001 100 010	9	12	13	17
7	0	0,004576401	0,000 000 010 010 101 000	8	11	13	15
0	1	0,005006665	0,000 000 010 100 100 000	8	10	13	-
1	1	0,003093947	0,000 000 001 100 101 000	9	10	13	15
2	1	0,002652879	0,000 000 001 010 110 000	9	11	13	14
3	1	0,001161223	0,000 000 000 100 110 000	10	13	14	-
4	1	0,000910303	0,000 000 000 011 101 000	11	12	13	15
5	1	0,001158596	0,000 000 000 100 101 100	10	13	15	16
6	1	0,00168202	0,000 000 000 110 110 000	10	11	13	14
7	1	0,003299401	0,000 000 001 101 100 000	9	10	12	13
0	2	0,003986286	0,000 000 010 000 010 100	8	14	16	-
1	2	0,002652879	0,000 000 001 010 110 000	9	11	13	14
2	2	0,001307559	0,000 000 000 101 010 100	10	12	14	16
3	2	0,000821832	0,000 000 000 011 010 100	11	12	14	16

		$1/(Qy_{ij} \times Fe_{ij})$		Barrel Shifter			
i	**j**	**Decimal**	**Binário**	**1**	**2**	**3**	**4**
4	2	0,000966372	0,000 000 000 011 110 000	11	12	13	14
5	2	0,001229959	0,000 000 000 101 000 010	10	12	-	17
6	2	0,001785623	0,000 000 000 111 010 000	10	11	12	14
7	2	0,003502626	0,000 000 001 110 010 000	9	10	11	14
0	3	0,002261782	0,000 000 001 001 010 000	9	12	14	-
1	3	0,001161223	0,000 000 000 100 110 000	10	13	14	-
2	3	0,000821832	0,000 000 000 011 010 100	11	12	14	16
3	3	0,000913171	0,000 000 000 011 101 000	11	12	13	15
4	3	0,001073775	0,000 000 000 100 011 000	10	14	15	-
5	3	0,001366657	0,000 000 000 101 100 100	10	12	13	16
6	3	0,001984078	0,000 000 001 000 001 000	9	15	-	-
7	3	0,003891909	0,000 000 001 111 000 000	9	10	11	12
0	4	0,001262626	0,000 000 000 101 001 010	10	12	15	17
1	4	0,000910303	0,000 000 000 011 101 000	11	12	13	15
2	4	0,000966372	0,000 000 000 011 110 000	11	12	13	14
3	4	0,001073775	0,000 000 000 100 011 000	10	14	15	-
4	4	0,001262626	0,000 000 000 101 001 010	10	12	15	17
5	4	0,001607018	0,000 000 000 110 100 100	10	11	13	16
6	4	0,002333029	0,000 000 001 001 100 010	9	12	13	17
7	4	0,004576401	0,000 000 010 010 101 000	8	11	13	15
0	5	0,001607018	0,000 000 000 110 100 100	10	11	13	16
1	5	0,001158596	0,000 000 000 100 101 100	10	13	15	16
2	5	0,001229959	0,000 000 000 101 000 010	10	12	-	17
3	5	0,001366657	0,000 000 000 101 100 100	10	12	13	16
4	5	0,001607018	0,000 000 000 110 100 100	10	11	13	16
5	5	0,002045346	0,000 000 001 000 011 000	9	14	15	-
6	5	0,002969383	0,000 000 001 100 001 010	9	10	15	17
7	5	0,005824654	0,000 000 010 111 000 000	8	10	11	12
0	6	0,002333029	0,000 000 001 001 100 010	9	12	13	17
1	6	0,00168202	0,000 000 000 110 110 000	10	11	13	14
2	6	0,001785623	0,000 000 000 111 010 000	10	11	12	14
3	6	0,001984078	0,000 000 001 000 001 000	9	15	-	-
4	6	0,002333029	0,000 000 001 001 100 010	9	12	13	17
5	6	0,002969383	0,000 000 001 100 001 010	9	10	15	17
6	6	0,004310876	0,000 000 010 001 101 000	8	12	13	15
7	6	0,008456087	0,000 000 100 010 101 000	7	11	13	15
0	7	0,004576401	0,000 000 010 010 101 000	8	11	13	15
1	7	0,003299401	0,000 000 001 101 100 000	9	10	12	13
2	7	0,003502626	0,000 000 001 110 010 000	9	10	11	14
3	7	0,003891909	0,000 000 001 111 000 000	9	10	11	12
4	7	0,004576401	0,000 000 010 010 101 000	8	11	13	15
5	7	0,005824654	0,000 000 010 111 000 000	8	10	11	12
6	7	0,008456087	0,000 000 100 010 101 000	7	11	13	15
7	7	0,016587211	0,000 001 000 011 100 000	6	11	12	13

Os deslocamentos para cada deslocador, apresentados nas tabs. A2.1 e A2.2 foram organizados para permitir uma minimização no número de deslocamentos realizados por cada deslocador, desta forma, foi possível limitar em oito o número máximo de deslocamentos realizados por deslocador, o que permitiu que fossem utilizados três bits de controle para cada deslocador. Desta forma, a memória que armazena os controles dos deslocamentos para cada cálculo pôde utilizar palavras de apenas 12 bits (três bits para o controle de cada um dos quatro deslocadores). A tab. A2.3 apresenta os bits de controle e os deslocamentos gerados para cada um dos deslocadores, onde *X* significa que não há deslocamento para estes bits de controle e onde o traço significa que são entregues apenas zeros na saída do deslocador.

TABELA A2.3 – Deslocamentos gerados por deslocador

Controle	Barrel Shifter			
	1	2	3	4
000	6	-	-	-
001	7	9	10	11
010	8	10	11	12
011	9	11	12	13
100	10	12	13	14
101	11	13	14	15
110	X	14	15	16
111	X	15	16	17

A última parte deste anexo apresenta as palavras de memória das duas memórias utilizadas pelo quantizador, sendo que cada posição armazena os doze bits referentes ao controle dos quatro deslocadores. A tab. A2.4 apresenta as palavras de memória relativas ao componente de luminância (Y) e a tab. A2.5 apresenta as palavras de memória relativas aos componentes de crominância (Cb e Cr). É importante perceber que, como os cálculos realizados pelo quantizador são orientados por coluna, para facilitar o endereçamento das memórias, a primeira linha de memória contém a primeira coluna de elementos para o cálculo do quantizador, a segunda linha contém a segunda coluna de elementos e assim sucessivamente. Desta forma, o quantizador pode acessar as memórias com endereços incrementais.

TABELA A2.4 – Conteúdos da memória relativa ao componente de luminância (Y)

Endereço	Palavra de Memória			
	BS1	BS2	BS3	BS4
0	001	000	000	000
1	010	001	001	001
2	010	001	010	010
3	010	001	001	001
4	010	001	001	100
5	010	001	010	010
6	010	011	011	100
7	010	001	011	011
8	001	100	100	111
9	010	010	010	101
10	010	010	011	011
11	010	011	101	101

Endereço	Palavra de Memória			
	BS1	BS2	BS3	BS4
12	010	101	101	000
13	011	010	011	100
14	011	011	100	101
15	011	010	010	011
16	001	010	010	010
17	010	010	110	111
18	010	011	100	101
19	011	010	010	010
20	011	011	100	110
21	011	100	111	000
22	011	100	101	111
23	011	010	010	011
24	010	001	010	010
25	010	101	000	000
26	011	010	011	011
27	011	010	100	100
28	100	011	011	011
29	011	101	110	111
30	011	100	110	110
31	010	000	111	111
32	010	010	011	100
33	011	010	010	101
34	011	100	100	100
35	011	101	000	111
36	100	011	011	011
37	011	000	000	111
38	011	100	110	111
39	010	101	111	000
40	010	110	000	111
41	011	000	111	111
42	011	101	101	000
43	100	011	101	110
44	100	100	100	100
45	100	011	011	011
46	011	100	100	100
47	010	010	010	010
48	010	011	100	111
49	011	011	011	100
50	011	011	101	101
51	011	011	000	111
52	011	100	110	111
53	011	011	100	101
54	011	010	010	011
55	001	100	101	111

Endereço	Palavra de Memória			
	BS1	BS2	BS3	BS4
56	010	001	001	001
57	010	001	101	110
58	010	001	011	100
59	010	001	011	100
60	010	001	111	111
61	010	001	011	011
62	001	100	100	100
63	000	011	011	011

TABELA A2.5 – Conteúdos da memória relativa aos componentes de crominância (Cb e Cr)

Endereço	Palavra de Memória			
	BS1	BS2	BS3	BS4
0	010	001	001	001
1	010	010	100	000
2	010	110	111	000
3	100	110	110	000
4	100	100	110	111
5	100	011	100	110
6	011	100	100	111
7	010	011	100	101
8	010	010	100	000
9	011	010	100	101
10	011	011	100	100
11	100	101	101	000
12	101	100	100	101
13	100	101	110	110
14	100	011	100	100
15	011	010	011	011
16	010	110	111	000
17	011	011	100	100
18	100	100	101	110
19	101	100	101	110
20	101	100	100	100
21	100	100	000	111
22	100	011	011	100
23	011	010	010	100
24	011	100	101	000
25	100	101	101	000
26	101	100	101	110
27	101	100	100	101
28	100	110	110	000
29	100	100	100	110
30	011	111	000	000

Endereço	Palavra de Memória			
	BS1	BS2	BS3	BS4
31	011	010	010	010
32	100	100	110	111
33	101	100	100	101
34	101	100	100	100
35	100	110	110	000
36	100	100	110	111
37	100	011	100	110
38	011	100	100	111
39	010	011	100	101
40	100	011	100	110
41	100	101	110	110
42	100	100	000	111
43	100	100	100	110
44	100	011	100	110
45	011	110	110	000
46	011	010	110	111
47	010	010	010	010
48	011	100	100	111
49	100	011	100	100
50	100	011	011	100
51	011	111	000	000
52	011	100	100	111
53	011	010	110	111
54	010	100	100	101
55	001	011	100	101
56	010	011	100	101
57	011	010	011	011
58	011	010	010	100
59	011	010	010	010
60	010	011	100	101
61	010	010	010	010
62	001	011	100	101
63	000	011	011	011

Anexo 3 Tabelas de Huffman e Conteúdo das Memórias

Este anexo apresenta as tabelas de Huffman DC e AC para os componentes de luminância (Y) e crominância (Cb e Cr), que são sugeridas pelo padrão JPEG [THE 92] e que foram utilizadas nas implementações desenvolvidas nesta dissertação. As tabs. A3.1, A3.2, A3.3 e A3.4 apresentam as quatro tabelas de Huffman referentes aos elementos DC do componente de luminância, aos elementos DC dos componentes de crominância, aos elementos AC do componente de luminância e aos elementos AC dos componentes de crominância. Além das tabelas de Huffman propriamente ditas, as tabs. A3.1, A3.2, A3.3 e A3.4 apresentam o endereço de memória utilizado para armazenar cada código de Huffman, bem como o conteúdo da palavra de memória, que possui, além do código de Huffman, o tamanho deste código em número de bits.

Nas tabelas de Huffman DC, apresentadas nas tabs. A3.1 e A3.2, a primeira coluna diz respeito ao tamanho do componente a ser codificado, que é obtido através do cálculo de tamanho, que foi explicado no item 3.4.4. Nas tabelas de Huffman DC, o campo tamanho é utilizado, nesta implementação, como endereço da memória. A tabela de Huffman DC para componentes de luminância utiliza 13 bits de palavra de memória, sendo 9 bits para o código de Huffman e 4 bits para o tamanho deste código. Já a tabela de Huffman DC para componentes de crominância utiliza 15 bits de palavra, sendo 11 bits para o código de Huffman e 4 bits para o tamanho do código.

Nas tabelas de Huffman AC, apresentadas nas tabs. A3.3 e A3.4, a primeira coluna, com o título de Ocorrências/Tamanho, é gerada pelo codificador RLE e pelo cálculo de tamanho, já explicados no item 3.4. O par Ocorrências/Tamanho invertido, isto é, o par Tamanho/Ocorrências é utilizado como endereço para as memórias. As posições com *X* nestas tabelas são endereços de memória não utilizados. As duas tabela de Huffman AC utilizam 21 bits de palavra de memória, sendo 16 bits para o código de Huffman e 5 bits para o tamanho do código.

TABELA A3.1 – Tabela de Huffman DC para componentes de luminância

Tabela de Huffman			Ende-reço	Palavra de memória	
Tamanho	Nº bits	Cód. Huffman		Cód. Huffman	Nº de bits
0	2	00	0	000000000	0010
1	3	010	1	010000000	0011
2	3	011	2	011000000	0011
3	3	100	3	100000000	0011
4	3	101	4	101000000	0011
5	3	110	5	110000000	0011
6	4	1110	6	111000000	0100
7	5	11110	7	111100000	0101
8	6	111110	8	111110000	0110
9	7	1111110	9	111111000	0111
10	8	11111110	10	111111100	1000
11	9	111111110	11	111111110	1001

TABELA A3.2 – Tabela de Huffman DC para componentes de crominância

Tabela de Huffman			Ende-reço	Palavra de memória	
Tamanho	Nº de bits	Cód. Huffman		Cód. Huffman	Nº de bits
0	2	00	0	00000000000	0010
1	2	01	1	01000000000	0010
2	2	10	2	10000000000	0010
3	3	110	3	11000000000	0011
4	4	1110	4	11100000000	0100
5	5	11110	5	11110000000	0101
6	6	111110	6	11111000000	0110
7	7	1111110	7	11111100000	0111
8	8	11111110	8	11111110000	1000
9	9	111111110	9	11111111000	1001
10	10	1111111110	10	11111111100	1010
11	11	11111111110	11	11111111110	1011

TABELA A3.3 – Tabela de Huffman AC para componentes de luminância

Tabela de Huffman			Ende-reço	Palavra de memória	
Ocorrências/ Tamanho	Nº de bits	Código de Huffman		Código de Huffman	Nº de bits
0/0	4	1010	00	1010000000000000	00100
X	0	X	01	0000000000000000	00000
X	0	X	02	0000000000000000	00000
X	0	X	03	0000000000000000	00000
X	0	X	04	0000000000000000	00000
X	0	X	05	0000000000000000	00000
X	0	X	06	0000000000000000	00000
X	0	X	07	0000000000000000	00000
X	0	X	08	0000000000000000	00000
X	0	X	09	0000000000000000	00000
X	0	X	0A	0000000000000000	00000
X	0	X	0B	0000000000000000	00000
X	0	X	0C	0000000000000000	00000
X	0	X	0D	0000000000000000	00000
X	0	X	0E	0000000000000000	00000
F/0	11	11111111001	0F	1111111100100000	01011
0/1	2	00	10	0000000000000000	00010
1/1	4	1100	11	1100000000000000	00100
2/1	5	11100	12	1110000000000000	00101
3/1	6	111010	13	1110100000000000	00110
4/1	6	111011	14	1110110000000000	00110
5/1	7	1111010	15	1111010000000000	00111
6/1	7	1111011	16	1111011000000000	00111
7/1	8	11111010	17	1111101000000000	01000
8/1	9	111111000	18	1111110000000000	01001
9/1	9	111111001	19	1111110010000000	01001
A/1	9	111111010	1A	1111110100000000	01001
B/1	10	1111111001	1B	1111111001000000	01010

Tabela de Huffman			Ende-reço	Palavra de memória	
Ocorrências/ Tamanho	Nº de bits	Código de Huffman		Código de Huffman	Nº de bits
C/1	10	1111111010	1C	1111111010000000	01010
D/1	11	11111111000	1D	1111111100000000	01011
E/1	16	1111111111101011	1E	1111111111101011	10000
F/1	16	1111111111110101	1F	1111111111110101	10000
0/2	2	01	20	0100000000000000	00010
1/2	5	11011	21	1101100000000000	00101
2/2	8	11111001	22	1111100100000000	01000
3/2	9	111110111	23	1111101110000000	01001
4/2	10	1111111000	24	1111111000000000	01010
5/2	11	11111110111	25	1111111011100000	01011
6/2	12	111111110110	26	1111111101100000	01100
7/2	12	111111110111	27	1111111101110000	01100
8/2	15	111111111000000	28	1111111110000000	01111
9/2	16	1111111110111110	29	1111111110111110	10000
A/2	16	1111111111000111	2A	1111111111000111	10000
B/2	16	1111111111010000	2B	1111111111010000	10000
C/2	16	1111111111011001	2C	1111111111011001	10000
D/2	16	1111111111100010	2D	1111111111100010	10000
E/2	16	1111111111101100	2E	1111111111101100	10000
F/2	16	1111111111110110	2F	1111111111110110	10000
0/3	3	100	30	1000000000000000	00011
1/3	7	1111001	31	1111001000000000	00111
2/3	10	1111110111	32	1111110111000000	01010
3/3	12	111111110101	33	1111111101010000	01100
4/3	16	1111111110010110	34	1111111110010110	10000
5/3	16	1111111110011110	35	1111111110011110	10000
6/3	16	1111111110100110	36	1111111110100110	10000
7/3	16	1111111110101110	37	1111111110101110	10000
8/3	16	1111111110110110	38	1111111110110110	10000
9/3	16	1111111110111111	39	1111111110111111	10000
A/3	16	1111111111001000	3A	1111111111001000	10000
B/3	16	1111111111010001	3B	1111111111010001	10000
C/3	16	1111111111011010	3C	1111111111011010	10000
D/3	16	1111111111100011	3D	1111111111100011	10000
E/3	16	1111111111101101	3E	1111111111101101	10000
F/3	16	1111111111110111	3F	1111111111110111	10000
0/4	4	1011	40	1011000000000000	00100
1/4	9	111110110	41	1111101100000000	01001
2/4	12	111111110100	42	1111111101000000	01100
3/4	16	1111111110001111	43	1111111110001111	10000
4/4	16	1111111110010111	44	1111111110010111	10000
5/4	16	1111111110011111	45	1111111110011111	10000
6/4	16	1111111110100111	46	1111111110100111	10000
7/4	16	1111111110101111	47	1111111110101111	10000
8/4	16	1111111110110111	48	1111111110110111	10000
9/4	16	1111111111000000	49	1111111111000000	10000
A/4	16	1111111111001001	4A	1111111111001001	10000
B/4	16	1111111111010010	4B	1111111111010010	10000

Tabela de Huffman			Ende-reço	Palavra de memória	
Ocorrências/ Tamanho	Nº de bits	Código de Huffman		Código de Huffman	Nº de bits
C/4	16	1111111111011011	4C	1111111111011011	10000
D/4	16	1111111111100100	4D	1111111111100100	10000
E/4	16	1111111111101110	4E	1111111111101110	10000
F/4	16	1111111111111000	4F	1111111111111000	10000
0/5	5	11010	50	1101000000000000	00101
1/5	11	11111110110	51	1111111011000000	01011
2/5	16	1111111110001001	52	1111111110001001	10000
3/5	16	1111111110010000	53	1111111110010000	10000
4/5	16	1111111110011000	54	1111111110011000	10000
5/5	16	1111111110100000	55	1111111110100000	10000
6/5	16	1111111110101000	56	1111111110101000	10000
7/5	16	1111111110110000	57	1111111110110000	10000
8/5	16	1111111110111000	58	1111111110111000	10000
9/5	16	1111111111000001	59	1111111111000001	10000
A/5	16	1111111111001010	5A	1111111111001010	10000
B/5	16	1111111111010011	5B	1111111111010011	10000
C/5	16	1111111111011100	5C	1111111111011100	10000
D/5	16	1111111111100101	5D	1111111111100101	10000
E/5	16	1111111111101111	5E	1111111111101111	10000
F/5	16	1111111111111001	5F	1111111111111001	10000
0/6	7	1111000	60	1111000000000000	00111
1/6	16	1111111110000100	61	1111111110000100	10000
2/6	16	1111111110001010	62	1111111110001010	10000
3/6	16	1111111110010001	63	1111111110010001	10000
4/6	16	1111111110011001	64	1111111110011001	10000
5/6	16	1111111110100001	65	1111111110100001	10000
6/6	16	1111111110101001	66	1111111110101001	10000
7/6	16	1111111110110001	67	1111111110110001	10000
8/6	16	1111111110111001	68	1111111110111001	10000
9/6	16	1111111111000010	69	1111111111000010	10000
A/6	16	1111111111001011	6A	1111111111001011	10000
B/6	16	1111111111010100	6B	1111111111010100	10000
C/6	16	1111111111011101	6C	1111111111011101	10000
D/6	16	1111111111100110	6D	1111111111100110	10000
E/6	16	1111111111110000	6E	1111111111110000	10000
F/6	16	1111111111111010	6F	1111111111111010	10000
0/7	8	11111000	70	1111100000000000	01000
1/7	16	1111111110000101	71	1111111110000101	10000
2/7	16	1111111110001011	72	1111111110001011	10000
3/7	16	1111111110010010	73	1111111110010010	10000
4/7	16	1111111110011010	74	1111111110011010	10000
5/7	16	1111111110100010	75	1111111110100010	10000
6/7	16	1111111110101010	76	1111111110101010	10000
7/7	16	1111111110110010	77	1111111110110010	10000
8/7	16	1111111110111010	78	1111111110111010	10000
9/7	16	1111111111000011	79	1111111111000011	10000
A/7	16	1111111111001100	7A	1111111111001100	10000

Tabela de Huffman			Ende-reço	Palavra de memória	
Ocorrências/ Tamanho	Nº de bits	Código de Huffman		Código de Huffman	Nº de bits
B/7	16	1111111111010101	7B	1111111111010101	10000
C/7	16	1111111111011110	7C	1111111111011110	10000
D/7	16	1111111111100111	7D	1111111111100111	10000
E/7	16	1111111111110001	7E	1111111111110001	10000
F/7	16	1111111111111011	7F	1111111111111011	10000
0/8	10	1111110110	80	1111110110000000	01010
1/8	16	1111111110000110	81	1111111110000110	10000
2/8	16	1111111110001100	82	1111111110001100	10000
3/8	16	1111111110010011	83	1111111110010011	10000
4/8	16	1111111110011011	84	1111111110011011	10000
5/8	16	1111111110100011	85	1111111110100011	10000
6/8	16	1111111110101011	86	1111111110101011	10000
7/8	16	1111111110110011	87	1111111110110011	10000
8/8	16	1111111110111011	88	1111111110111011	10000
9/8	16	1111111111000100	89	1111111111000100	10000
A/8	16	1111111111001101	8A	1111111111001101	10000
B/8	16	1111111111010110	8B	1111111111010110	10000
C/8	16	1111111111011111	8C	1111111111011111	10000
D/8	16	1111111111101000	8D	1111111111101000	10000
E/8	16	1111111111110010	8E	1111111111110010	10000
F/8	16	1111111111111100	8F	1111111111111100	10000
0/9	16	1111111110000010	90	1111111110000010	10000
1/9	16	1111111110000111	91	1111111110000111	10000
2/9	16	1111111110001101	92	1111111110001101	10000
3/9	16	1111111110010100	93	1111111110010100	10000
4/9	16	1111111110011100	94	1111111110011100	10000
5/9	16	1111111110100100	95	1111111110100100	10000
6/9	16	1111111110101100	96	1111111110101100	10000
7/9	16	1111111110110100	97	1111111110110100	10000
8/9	16	1111111110111100	98	1111111110111100	10000
9/9	16	1111111111000101	99	1111111111000101	10000
A/9	16	1111111111001110	9A	1111111111001110	10000
B/9	16	1111111111010111	9B	1111111111010111	10000
C/9	16	1111111111100000	9C	1111111111100000	10000
D/9	16	1111111111101001	9D	1111111111101001	10000
E/9	16	1111111111110011	9E	1111111111110011	10000
F/9	16	1111111111111101	9F	1111111111111101	10000
0/A	16	1111111110000011	A0	1111111110000011	10000
1/A	16	1111111110001000	A1	1111111110001000	10000
2/A	16	1111111110001110	A2	1111111110001110	10000
3/A	16	1111111110010101	A3	1111111110010101	10000
4/A	16	1111111110011101	A4	1111111110011101	10000
5/A	16	1111111110100101	A5	1111111110100101	10000
6/A	16	1111111110101101	A6	1111111110101101	10000
7/A	16	1111111110110101	A7	1111111110110101	10000
8/A	16	1111111110111101	A8	1111111110111101	10000
9/A	16	1111111111000110	A9	1111111111000110	10000

Tabela de Huffman			Ende-reço	Palavra de memória	
Ocorrências/ Tamanho	Nº de bits	Código de Huffman		Código de Huffman	Nº de bits
A/A	16	1111111111001111	AA	1111111111001111	10000
B/A	16	1111111111011000	AB	1111111111011000	10000
C/A	16	1111111111100001	AC	1111111111100001	10000
D/A	16	1111111111101010	AD	1111111111101010	10000
E/A	16	1111111111110100	AE	1111111111110100	10000
F/A	16	1111111111111110	AF	1111111111111110	10000

TABELA A3.4 – Tabela de Huffman DC para componentes de crominância

Tabela de Huffman			Ende-reço	Palavra de memória	
Ocorrências/ Tamanho	Nº de bits	Código de Huffman		Código de Huffman	Nº de bits
0/0	2	00	00	0000000000000000	00010
X	0	X	01	0000000000000000	00000
X	0	X	02	0000000000000000	00000
X	0	X	03	0000000000000000	00000
X	0	X	04	0000000000000000	00000
X	0	X	05	0000000000000000	00000
X	0	X	06	0000000000000000	00000
X	0	X	07	0000000000000000	00000
X	0	X	08	0000000000000000	00000
X	0	X	09	0000000000000000	00000
X	0	X	0A	0000000000000000	00000
X	0	X	0B	0000000000000000	00000
X	0	X	0C	0000000000000000	00000
X	0	X	0D	0000000000000000	00000
X	0	X	0E	0000000000000000	00000
F/0	10	1111111010	0F	1111111010000000	01010
0/1	2	01	10	0100000000000000	01100
1/1	4	1011	11	1011000000000000	01110
2/1	5	11010	12	1101000000000000	01111
3/1	5	11011	13	1101100000000000	01111
4/1	6	111010	14	1110100000000000	10000
5/1	6	111011	15	1110110000000000	10000
6/1	7	1111001	16	1111001000000000	10000
7/1	7	1111010	17	1111010000000000	10000
8/1	8	11111001	18	1111100100000000	01000
9/1	9	111110111	19	1111101110000000	01001
A/1	9	111111000	1A	1111110000000000	01001
B/1	9	111111001	1B	1111110010000000	01001
C/1	9	111111010	1C	1111110100000000	01001
D/1	11	11111111001	1D	1111111100100000	01011
E/1	14	11111111100000	1E	1111111110000000	01110
F/1	15	111111111000011	1F	1111111110000110	01111
0/2	3	100	20	1000000000000000	00011
1/2	6	111001	21	1110010000000000	10000
2/2	8	11110111	22	1111011100000000	01000

Tabela de Huffman			Ende-reço	Palavra de memória	
Ocorrências/ Tamanho	Nº de bits	Código de Huffman		Código de Huffman	Nº de bits
3/2	8	11111000	23	1111100000000000	01000
4/2	9	111110110	24	1111101100000000	01001
5/2	10	1111111001	25	1111111001000000	01010
6/2	11	11111110111	26	1111111011100000	01011
7/2	11	11111111000	27	1111111100000000	01011
8/2	16	1111111110110111	28	1111111110110111	10000
9/2	16	1111111111000000	29	1111111111000000	10000
A/2	16	1111111111001001	2A	1111111111001001	10000
B/2	16	1111111111010010	2B	1111111111010010	10000
C/2	16	1111111111011011	2C	1111111111011011	10000
D/2	16	1111111111100100	2D	1111111111100100	10000
E/2	16	1111111111101101	2E	1111111111101101	10000
F/2	16	1111111111110110	2F	1111111111110110	10000
0/3	4	1010	30	1010000000000000	00100
1/3	8	11110110	31	1111011000000000	01000
2/3	10	1111110111	32	1111110111000000	01010
3/3	10	1111111000	33	1111111000000000	01010
4/3	16	1111111110010111	34	1111111110010111	10000
5/3	16	1111111110011111	35	1111111110011111	10000
6/3	16	1111111110100111	36	1111111110100111	10000
7/3	16	1111111110101111	37	1111111110101111	10000
8/3	16	1111111110111000	38	1111111110111000	10000
9/3	16	1111111111000001	39	1111111111000001	10000
A/3	16	1111111111001010	3A	1111111111001010	10000
B/3	16	1111111111010011	3B	1111111111010011	10000
C/3	16	1111111111011100	3C	1111111111011100	10000
D/3	16	1111111111100101	3D	1111111111100101	10000
E/3	16	1111111111101110	3E	1111111111101110	10000
F/3	16	1111111111110111	3F	1111111111110111	10000
0/4	5	11000	40	1100000000000000	00101
1/4	9	111110101	41	1111101010000000	01001
2/4	12	111111110110	42	1111111101100000	01100
3/4	12	111111110111	43	1111111101110000	01100
4/4	16	1111111110011000	44	1111111110011000	10000
5/4	16	1111111110100000	45	1111111110100000	10000
6/4	16	1111111110101000	46	1111111110101000	10000
7/4	16	1111111110110000	47	1111111110110000	10000
8/4	16	1111111110111001	48	1111111110111001	10000
9/4	16	1111111111000010	49	1111111111000010	10000
A/4	16	1111111111001011	4A	1111111111001011	10000
B/4	16	1111111111010100	4B	1111111111010100	10000
C/4	16	1111111111011101	4C	1111111111011101	10000
D/4	16	1111111111100110	4D	1111111111100110	10000
E/4	16	1111111111101111	4E	1111111111101111	10000
F/4	16	1111111111111000	4F	1111111111111000	10000
0/5	5	11001	50	1100100000000000	00101
1/5	11	11111110110	51	1111111011000000	01011

Tabela de Huffman			Ende-reço	Palavra de memória	
Ocorrências/ Tamanho	Nº de bits	Código de Huffman		Código de Huffman	Nº de bits
2/5	15	111111111000010	52	1111111110000100	01111
3/5	16	1111111110010001	53	1111111110010001	10000
4/5	16	1111111110011001	54	1111111110011001	10000
5/5	16	1111111110100001	55	1111111110100001	10000
6/5	16	1111111110101001	56	1111111110101001	10000
7/5	16	1111111110110001	57	1111111110110001	10000
8/5	16	1111111110111010	58	1111111110111010	10000
9/5	16	1111111111000011	59	1111111111000011	10000
A/5	16	1111111111001100	5A	1111111111001100	10000
B/5	16	1111111111010101	5B	1111111111010101	10000
C/5	16	1111111111011110	5C	1111111111011110	10000
D/5	16	1111111111100111	5D	1111111111100111	10000
E/5	16	1111111111110000	5E	1111111111110000	10000
F/5	16	1111111111111001	5F	1111111111111001	10000
0/6	6	111000	60	1110000000000000	10000
1/6	12	111111110101	61	1111111101010000	01100
2/6	16	1111111110001100	62	1111111110001100	10000
3/6	16	1111111110010010	63	1111111110010010	10000
4/6	16	1111111110011010	64	1111111110011010	10000
5/6	16	1111111110100010	65	1111111110100010	10000
6/6	16	1111111110101010	66	1111111110101010	10000
7/6	16	1111111110110010	67	1111111110110010	10000
8/6	16	1111111110111011	68	1111111110111011	10000
9/6	16	1111111111000100	69	1111111111000100	10000
A/6	16	1111111111001101	6A	1111111111001101	10000
B/6	16	1111111111010110	6B	1111111111010110	10000
C/6	16	1111111111011111	6C	1111111111011111	10000
D/6	16	1111111111101000	6D	1111111111101000	10000
E/6	16	1111111111110001	6E	1111111111110001	10000
F/6	16	1111111111111010	6F	1111111111111010	10000
0/7	7	1111000	70	1111000000000000	00111
1/7	16	1111111110001000	71	1111111110001000	10000
2/7	16	1111111110001101	72	1111111110001101	10000
3/7	16	1111111110010011	73	1111111110010011	10000
4/7	16	1111111110011011	74	1111111110011011	10000
5/7	16	1111111110100011	75	1111111110100011	10000
6/7	16	1111111110101011	76	1111111110101011	10000
7/7	16	1111111110110011	77	1111111110110011	10000
8/7	16	1111111110111100	78	1111111110111100	10000
9/7	16	1111111111000101	79	1111111111000101	10000
A/7	16	1111111111001110	7A	1111111111001110	10000
B/7	16	1111111111010111	7B	1111111111010111	10000
C/7	16	1111111111100000	7C	1111111111100000	10000
D/7	16	1111111111101001	7D	1111111111101001	10000
E/7	16	1111111111110010	7E	1111111111110010	10000
F/7	16	1111111111111011	7F	1111111111111011	10000
0/8	9	111110100	80	1111101000000000	01001

Tabela de Huffman			Ende-reço	Palavra de memória	
Ocorrências/ Tamanho	Nº de bits	Código de Huffman		Código de Huffman	Nº de bits
1/8	16	1111111110001001	81	1111111110001001	10000
2/8	16	1111111110001110	82	1111111110001110	10000
3/8	16	1111111110010100	83	1111111110010100	10000
4/8	16	1111111110011100	84	1111111110011100	10000
5/8	16	1111111110100100	85	1111111110100100	10000
6/8	16	1111111110101100	86	1111111110101100	10000
7/8	16	1111111110110100	87	1111111110110100	10000
8/8	16	1111111110111101	88	1111111110111101	10000
9/8	16	1111111111000110	89	1111111111000110	10000
A/8	16	1111111111001111	8A	1111111111001111	10000
B/8	16	1111111111011000	8B	1111111111011000	10000
C/8	16	1111111111100001	8C	1111111111100001	10000
D/8	16	1111111111101010	8D	1111111111101010	10000
E/8	16	1111111111110011	8E	1111111111110011	10000
F/8	16	1111111111111100	8F	1111111111111100	10000
0/9	10	1111110110	90	1111110110000000	01010
1/9	16	1111111110001010	91	1111111110001010	10000
2/9	16	1111111110001111	92	1111111110001111	10000
3/9	16	1111111110010101	93	1111111110010101	10000
4/9	16	1111111110011101	94	1111111110011101	10000
5/9	16	1111111110100101	95	1111111110100101	10000
6/9	16	1111111110101101	96	1111111110101101	10000
7/9	16	1111111110110101	97	1111111110110101	10000
8/9	16	1111111110111110	98	1111111110111110	10000
9/9	16	1111111111000111	99	1111111111000111	10000
A/9	16	1111111111010000	9A	1111111111010000	10000
B/9	16	1111111111011001	9B	1111111111011001	10000
C/9	16	1111111111100010	9C	1111111111100010	10000
D/9	16	1111111111101011	9D	1111111111101011	10000
E/9	16	1111111111110100	9E	1111111111110100	10000
F/9	16	1111111111111101	9F	1111111111111101	10000
0/A	12	111111110100	A0	1111111101000000	01100
1/A	16	1111111110001011	A1	1111111110001011	10000
2/A	16	1111111110010000	A2	1111111110010000	10000
3/A	16	1111111110010110	A3	1111111110010110	10000
4/A	16	1111111110011110	A4	1111111110011110	10000
5/A	16	1111111110100110	A5	1111111110100110	10000
6/A	16	1111111110101110	A6	1111111110101110	10000
7/A	16	1111111110110110	A7	1111111110110110	10000
8/A	16	1111111110111111	A8	1111111110111111	10000
9/A	16	1111111111001000	A9	1111111111001000	10000
A/A	16	1111111111010001	AA	1111111111010001	10000
B/A	16	1111111111011010	AB	1111111111011010	10000
C/A	16	1111111111100011	AC	1111111111100011	10000
D/A	16	1111111111101100	AD	1111111111101100	10000
E/A	16	1111111111110101	AE	1111111111110101	10000
F/A	16	1111111111111110	AF	1111111111111110	10000

Anexo 4 Determinação do Número de Bits nos Estágios da DCT 2-D

Este anexo irá apresentar o resumo dos cálculos desenvolvidos para determinar o número de bits utilizados em cada estágio da arquitetura da DCT 2-D, desenvolvida nesta dissertação. A precisão na determinação do número de bits possibilita uma significativa minimização no uso de recursos, além de garantir que nenhum bit válido seja perdido pelo cálculo da DCT 2-D.

Do ponto de vista do número de bits, não há diferenças entre os compressores para imagens coloridas e para imagens em tons de cinza, pois em todos os casos, a DCT 2-D irá receber como entrada números de 8 bits, que possuirão, em média, um valor igual a zero, isto é, irão estar na faixa de –128 a 127.

Considerando as operações existentes em cada passo do algoritmo da DCT 1-D, o número de bits necessários na saída de cada estágio foi obtido a partir de entradas capazes de gerar o máximo resultado possível para cada cálculo de cada estágio.

Os resultados dos cálculos realizados são apresentados como múltiplos da entrada máxima, no caso, –128 ou 127. De acordo com o fator a que a entrada máxima estiver sendo multiplicada é possível definir quantos bits adicionais serão necessários em cada estágio, em relação aos 8 bits usados na entrada. Para tanto, o fator de multiplicação é posto na base dois para que o seu expoente indique o número de bits adicionais necessários naquele estágio. Quando o fator de multiplicação não possuir um expoente inteiro na base dois, serão apresentados os dois expoentes inteiros que delimitam a faixa de valores onde o resultado está contido. Neste caso, o maior expoente indica o número de bits adicionais necessários para a saída da operação em relação à entrada. Nos estágios onde o número máximo de bits não é constante para todos os cálculos, o cálculo com maior resultado irá definir o número de bits usado pelo estágio.

Os cálculos realizados nas duas DCT 1-D são similares por usarem o mesmo algoritmo, mas diferem exatamente no número de bits usados em cada estágio do cálculo. Por isso, os resultados máximos indicam a qual cálculo da DCT 1-D a saída faz parte, através dos dígitos 1 ou 2, que são acrescentados à identificação da saída.

A entrada máxima será representada nos cálculos pela constante $IN_{máx}$, que pode ter o máximo valor positivo ou negativo da entrada para gerar a máxima saída possível para cada cálculo de cada estágio. Nos cálculos, todas as entradas são substituídas por seu máximo valor, isto é, $IN_{máx}$. Como o máximo valor pode ser negativo ou positivo, considerando-se entradas distintas, uma operação de soma ou subtração das entradas, gera um resultado máximo onde, independentemente da operação, este resultado será a soma das parcelas de $IN_{máx}$ presentes na operação. Como exemplo, para calcular o máximo resultado da operação: $c_2 = a_3 - a_4 + a_2 - a_5$, tem-se que $c_{2máx} = IN_{máx} + IN_{máx} + IN_{máx} + IN_{máx}$ ou $c_{2máx} = 4IN_{máx}$. Os resultados destacados em negrito, nos cálculos apresentados, são as saídas que determinam o número de bits utilizados pelo estágio.

O primeiro estágio do *pipeline* da primeira DCT 1-D é um estágio de somas, que gera um bit a mais em relação à entrada, ou seja, as suas saídas têm nove bits. Como cada soma é efetuada sempre sobre duas entradas distintas, cada operação deste estágio gera um resultado máximo equivalente à duas vezes o valor máximo de entrada, necessitando, para tanto, de um bit adicional. Estes cálculos estão apresentados a seguir:

$b_0 = a_0 + a_7 \rightarrow b_{0máx1} = IN_{máx} + IN_{máx} \rightarrow \mathbf{b_{0máx1} = 2IN_{máx} = 2^1IN_{máx}}$

$b_1 = a_1 + a_6 \rightarrow \mathbf{b_{1máx1} = 2IN_{máx} = 2^1IN_{máx}}$

$b_2 = a_3 - a_4 \rightarrow \mathbf{b_{2máx1} = 2IN_{máx} = 2^1IN_{máx}}$

$b_3 = a_1 - a_6 \rightarrow \mathbf{b_{3máx1} = 2IN_{máx} = 2^1IN_{máx}}$

$b_4 = a_2 + a_5 \rightarrow \mathbf{b_{4máx1} = 2IN_{máx} = 2^1IN_{máx}}$

$b_5 = a_3 + a_4 \rightarrow \mathbf{b_{5máx1} = 2IN_{máx} = 2^1IN_{máx}}$

$b_6 = a_2 - a_5 \rightarrow \mathbf{b_{6máx1} = 2IN_{máx} = 2^1IN_{máx}}$

$b_7 = a_0 - a_7 \rightarrow \mathbf{b_{7máx1} = 2IN_{máx} = 2^1IN_{máx}}$

No segundo estágio temos, novamente, somas, que também irão gerar um bit adicional em relação às suas entradas, portanto as saídas possuem dez bits. Neste estágio existe uma operação de *bypass* em uma das saídas do estágio anterior, cuja saída poderia ser representada apenas com nove bits. Os cálculos que nos conduzem a estas conclusões estão apresentados a seguir:

$c_0 = b_0 + b_5 = a_0 + a_7 + a_3 + a_4 \rightarrow \mathbf{c_{0máx1} = 4IN_{máx} = 2^2IN_{máx}}$

$c_1 = b_1 - b_4 = a_0 + a_7 - a_2 - a_5 \rightarrow \mathbf{c_{1máx1} = 4IN_{máx} = 2^2IN_{máx}}$

$c_2 = b_2 + b_6 = a_3 - a_4 + a_2 - a_5 \rightarrow \mathbf{c_{2máx1} = 4IN_{máx} = 2^2IN_{máx}}$

$c_3 = b_1 + b_4 = a_1 + a_6 + a_2 + a_5 \rightarrow \mathbf{c_{3máx1} = 4IN_{máx} = 2^2IN_{máx}}$

$c_4 = b_0 - b_5 = a_0 + a_7 - a_3 - a_4 \rightarrow \mathbf{c_{4máx1} = 4IN_{máx} = 2^2IN_{máx}}$

$c_5 = b_3 + b_7 = a_1 - a_6 + a_0 - a_7 \rightarrow \mathbf{c_{5máx1} = 4IN_{máx} = 2^2IN_{máx}}$

$c_6 = b_3 + b_6 = a_1 - a_6 + a_2 - a_5 \rightarrow \mathbf{c_{6máx1} = 4IN_{máx} = 2^2IN_{máx}}$

$c_7 = b_7 \rightarrow c_{7máx1} = 2IN_{máx} = 2^1IN_{máx}$

O terceiro estágio também realiza operações de soma e de *bypass*. Este estágio, do mesmo modo que os estágios anteriores, irá gerar um bit adicional na saída, que irá possuir onze bits. Com a existência de operações de *bypass*, algumas das saídas poderiam ser representadas apenas com nove ou dez bits. Os cálculos estão apresentados a seguir:

$d_0 = c_0 + c_3 = a_0 + a_7 + a_3 + a_4 + a_1 + a_6 + a_2 + a_5 \rightarrow \mathbf{d_{0máx1} = 8IN_{máx} = 2^3IN_{máx}}$

$d_1 = c_0 - c_3 = a_0 + a_7 + a_3 + a_4 - a_1 - a_6 - a_2 - a_5 \rightarrow \mathbf{d_{1máx1} = 8IN_{máx} = 2^3IN_{máx}}$

$d_2 = c_2 \rightarrow d_{2máx1} = 4IN_{má} = 2^2IN_{máx\ x}$

$d_3 = c_1 + c_4 = a_0 + a_7 - a_2 - a_5 + a_0 + a_7 - a_3 - a_4 \rightarrow \mathbf{d_{3máx1} = 8IN_{máx} = 2^3IN_{máx}}$

$d_4 = c_2 - c_5 = a_3 - a_4 + a_2 - a_5 - a_1 + a_6 - a_0 + a_7 \rightarrow \mathbf{d_{4máx1} = 8IN_{máx} = 2^3IN_{máx}}$

$d_5 = c_4 \rightarrow d_{5máx1} = 4IN_{máx} = 2^2IN_{máx}$

$d_6 = c_5 \rightarrow d_{6máx1} = 4IN_{máx} = 2^2IN_{máx}$

$d_7 = c_6 \rightarrow d_{7máx1} = 4IN_{máx} = 2^2IN_{máx}$

$d_8 = c_7 \rightarrow d_{8máx1} = 2IN_{máx} = 2^1IN_{máx}$

O quarto estágio da primeira DCT 1-D é de multiplicações. O multiplicador tem como operandos os resultados da soma do terceiro estágio e as constantes *m*1, *m*2, *m*3 e *m*4 que foram apresentadas na tab. 3.5, no item 3.1.3 da dissertação.

Neste estágio não existe a geração de um bit extra em relação ao estágio anterior, como pode ser observado nos cálculos a seguir, onde também podem ser percebidas as operações de *bypass* existentes.

$e_0 = d_0 \rightarrow \mathbf{e_{0máx1} = 8IN_{máx} = 2^3IN_{máx}}$

$e_1 = d_1 \rightarrow \mathbf{e_{1máx1} = 8IN_{máx} = 2^3IN_{máx}}$

$e_2 = m_3 \, x \, d_2 = m_3 \, x \, (a_3 - a_4 + a_2 - a_5) \rightarrow$
$e_{2máx1} = 0{,}541016 \, x \, 4IN_{máx} \rightarrow$
$e_{2máx1} = 2{,}16406IN_{máx} \rightarrow 2^1IN_{máx} < e_{2máx1} < 2^2IN_{máx}$

$e_3 = m_1 \, x \, d_7 = m_1 \, x \, (a_1 - a_6 + a_2 - a_5) \rightarrow$
$e_{3máx1} = 0{,}703125 \, x \, 4IN_{máx} \rightarrow$
$e_{3máx1} = 2{,}8125IN_{máx} \rightarrow 2^1IN_{máx} < e_{3máx1} < 2^2IN_{máx}$

$e_4 = m_4 \, x \, d_6 = m_4 \, x \, (a_1 - a_6 + a_0 - a_7) \rightarrow$
$e_{4máx1} = 1{,}296875 \, x \, 4IN_{máx} \rightarrow$
$e_{4máx1} = 5{,}1875IN_{máx} \rightarrow 2^1IN_{máx} < e_{4máx1} < 2^2IN_{máx}$

$e_5 = d_5 \rightarrow e_{5máx1} = 4IN_{máx} = 2^2IN_{máx}$

$e_6 = m_1 \, x \, d_3 = m_1 \, x \, (a_0 + a_7 - a_2 - a_5 + a_0 + a_7 - a_3 - a_4) \rightarrow$
$e_{6máx1} = 0{,}703125 \, x \, 8IN_{máx} \rightarrow$
$\mathbf{e_{6máx1} = 5{,}625IN_{máx} \rightarrow 2^2IN_{máx} < e_{6máx1} < 2^3IN_{máx}}$

$e_7 = m_2 \, x \, d_4 = m_2 \, x \, (a_3 - a_4 + a_2 - a_5 - a_1 + a_6 - a_0 + a_7) \rightarrow$
$e_{7máx1} = 0{,}384766 \, x \, 8IN_{máx} \rightarrow$
$e_{7máx1} = 3{,}07813IN_{máx} \rightarrow 2^1IN_{máx} < e_{7máx1} < 2^2IN_{máx}$

$e_8 = d_8 \rightarrow e_{8máx1} = 2IN_{máx} = 2^1IN_{máx}$

O quinto estágio também envolve operações de soma e de *bypass*. Como pode ser observado a partir dos cálculos abaixo, este estágio gera um bit adicional em relação ao estágio anterior e, portanto, são necessários doze bits para suas saídas.

$f_0 = e_0 \rightarrow f_{0máx1} = 8IN_{máx} = 2^3IN_{máx}$

$f_1 = e_1 \rightarrow f_{1máx1} = 8IN_{máx} = 2^3IN_{máx}$

$f_2 = e_5 + e_6 = a_0 + a_7 - a_3 - a_4 + m_1 \, x \, (a_0 + a_7 - a_2 - a_5 + a_0 + a_7 - a_3 - a_4) \rightarrow$
$f_{2máx1} = 4IN_{máx} + 0{,}703125 \, x \, 8IN_{máx} \rightarrow$
$\mathbf{f_{2máx1} = 9{,}625IN_{máx} \rightarrow 2^3IN_{máx} < f_{2máx1} < 2^4IN_{máx}}$

$f_3 = e_5 - e_6 = a_0 + a_7 - a_3 - a_4 - m_1 \, x \, (a_0 + a_7 - a_2 - a_5 + a_0 + a_7 - a_3 - a_4) =$
$(1 - m_1) \, x \, (a_0 + a_7) - a_3 - a_4 - m_1 \, x \, (- a_2 - a_5 + a_0 + a_7 - a_3 - a_4) \rightarrow$
$f_{3máx1} = (2IN_{máx} - 0{,}703125 \, x \, 2IN_{máx}) + 2IN_{máx} + 0{,}703125 \, x \, 6IN_{máx} \rightarrow$
$f_{3máx1} = 6{,}8125IN_{máx} \rightarrow 2^2IN_{máx} < f_{3máx1} < 2^3IN_{máx}$

$f_4 = e_3 + e_8 = m_1 \, x \, (a_1 - a_6 + a_2 - a_5) + a_0 - a_7 \rightarrow$
$f_{4máx1} = 0{,}703125 \, x \, 4IN_{máx} + 2IN_{máx} \rightarrow$
$f_{4máx1} = 4{,}8125IN_{máx} \rightarrow 2^2IN_{máx} < f_{4máx1} < 2^3IN_{máx}$

$f_5 = e_8 - e_3 = a_0 - a_7 - m_1 \, x \, (a_1 - a_6 + a_2 - a_5) \rightarrow$
$f_{5máx1} = 2IN_{máx} + 0{,}703125 \, x \, 4IN_{máx} \rightarrow$
$f_{5máx1} = 4{,}8125IN_{máx} \rightarrow 2^2IN_{máx} < f_{5máx1} < 2^3IN_{máx}$

$f_6 = e_2 + e_7 = m_3\,x(a_3 - a_4 + a_2 - a_5) + m_2\,x(a_3 - a_4 + a_2 - a_5 - a_1 + a_6 - a_0 + a_7) \rightarrow$
$f_{6máx1} = 0{,}541016 \times 4IN_{máx} + 0{,}384766 \times 8IN_{máx} \rightarrow$
$f_{6máx1} = 5{,}24219IN_{máx} \rightarrow 2^2IN_{máx} < f_{6máx1} < 2^3IN_{máx}$

$f_7 = e_4 + e_7 = m_4\,x(a_1 - a_6 + a_0 - a_7) + m_2\,x(a_3 - a_4 + a_2 - a_5 - a_1 + a_6 - a_0 + a_7) =$
$(m_4 - m_2)\,x\,(a_1 - a_6 + a_0 - a_7) + a_3 - a_4 + a_2 - a_5 \rightarrow$
$f_{7máx1} = (1{,}296875 \times 4IN_{máx} - 0{,}384766 \times 4IN_{máx}) + 4IN_{máx} \rightarrow$
$f_{7máx1} = 7{,}64844IN_{máx} \rightarrow 2^2IN_{máx} < f_{7máx1} < 2^3IN_{máx}$

O sexto e último estágio da primeira DCT 1-D também possui operações de soma e de *bypass*, sendo que neste estágio não há a geração de bit adicional. Os cálculos que conduzem a esta conclusão são apresentados a seguir:

$S_0 = f_0 \rightarrow S_{0máx1} = 8IN_{máx} = 2^3IN_{máx}$

$S_1 = f_4 + f_7 = m_1\,x\,(a_1 - a_6 + a_2 - a_5) + a_0 - a_7 + m_4\,x(a_1 - a_6 + a_0 - a_7) +$
$m_2\,x(a_3 - a_4 + a_2 - a_5 - a_1 + a_6 - a_0 + a_7) =$
$(m_1 + m_4 - m_2)\,x\,(a_1 - a_6) + (1 + m_4 - m_2)\,x\,(a_0 - a_7) + m_1\,x\,(a_2 - a_5) +$
$m_2\,x\,(a_3 - a_4 + a_2 - a_5) \rightarrow$
$S_{1máx1} = (0{,}703125 \times 2IN_{máx} + 1{,}296875 \times 2IN_{máx} - 0{,}384766 \times 2IN_{máx}) +$
$(2IN_{máx} + 1{,}296875 \times 2IN_{máx} - 0{,}384766 \times 2IN_{máx}) + 0{,}703125 \times 2IN_{máx} +$
$0{,}384766 \times 4IN_{máx} \rightarrow$
$\mathbf{S_{1máx1} = 10IN_{máx} \rightarrow 2^3IN_{máx} < S_{1máx1} < 2^4IN_{máx}}$

$S_2 = f_2 \rightarrow \mathbf{S_{2máx1} = 9{,}625IN_{máx} \rightarrow 2^3IN_{máx} < S_{2máx1} < 2^4IN_{máx}}$

$S_3 = f_5 - f_6 = a_0 - a_7 + m_1\,x\,(-a_1 + a_6 - a_2 + a_5) + m_3\,x(-a_3 + a_4 - a_2 + a_5) +$
$m_2\,x\,(-a_3 + a_4 - a_2 + a_5 + a_1 - a_6 + a_0 - a_7) =$
$(m_1 - m_2)\,x\,(-a_1 + a_6) + a_0 - a_7 + m_1\,x\,(-a_2 + a_5) +$
$m_3\,x\,(-a_3 + a_4 - a_2 + a_5) + m_2\,x\,(-a_3 + a_4 - a_2 + a_5 + a_0 - a_7) \rightarrow$
$S_{3máx1} = (0{,}703125 \times 2INmáx - 0{,}384766 \times 2IN_{máx}) + 2IN_{máx} +$
$0{,}703125 \times 2INmáx + 0{,}541016 \times 4IN_{máx} + 0{,}384766 \times 6IN_{máx} \rightarrow$
$\mathbf{S_{3máx1} = 8{,}51563IN_{máx} \rightarrow 2^3IN_{máx} < S_{3máx1} < 2^4IN_{máx}}$

$S_4 = f_1 \rightarrow S_{4máx1} = 8IN_{máx} = 2^3IN_{máx}$

$S_5 = f_5 + f_6 = a_0 - a_7 + m_1\,x\,(-a_1 + a_6 - a_2 + a_5) + m_3\,x(a_3 - a_4 + a_2 - a_5) +$
$m_2\,x\,(a_3 - a_4 + a_2 - a_5 - a_1 + a_6 - a_0 + a_7) =$
$(1 - m_2)\,x\,(a_0 - a_7) + (-m_1 + m_3 + m_2)\,x\,(a_2 - a_5) + m_1\,x\,(-a_1 + a_6) +$
$m_3\,x\,(a_3 - a_4) + m_2\,x\,(a_3 - a_4 - a_1 + a_6) \rightarrow$
$S_{5máx1} = (2IN_{máx} - 0{,}384766 \times 2IN_{máx}) + (-0{,}703125 \times 2INmáx +$
$0{,}541016 \times 2IN_{máx} + 0{,}384766 \times 2IN_{máx}) + 0{,}703125 \times 2INmáx +$
$0{,}541016 \times 2IN_{máx} + 0{,}384766 \times 4IN_{máx} \rightarrow$
$S_{5máx1} = 5{,}70313IN_{máx} \rightarrow 2^2IN_{máx} < S_{5máx1} < 2^3IN_{máx}$

$S_6 = f_3 \rightarrow S_{6máx1} = 4{,}8125IN_{máx} \rightarrow 2^2IN_{máx} < S_{6máx1} < 2^3IN_{máx}$

$S_7 = f_4 - f_7 = m_1\,x\,(a_1 - a_6 + a_2 - a_5) + a_0 - a_7 - m_4\,x(a_1 - a_6 + a_0 - a_7) -$
$m_2\,x(a_3 - a_4 + a_2 - a_5 - a_1 + a_6 - a_0 + a_7) =$
$(-m_1 + m_4 - m_2)\,x\,(-a_1 + a_6) + (1 - m_4 - m_2)\,x\,(a_0 - a_7) + m_1\,x\,(a_2 - a_5) -$
$m_2\,x(a_3 - a_4 + a_2 - a_5) \rightarrow$
$S_{7máx1} = (-0{,}703125 \times 2IN_{máx} + 1{,}296875 \times 2IN_{máx} - 0{,}384766 \times 2IN_{máx}) +$
$(2IN_{máx} - 1{,}296875 \times 2IN_{máx} - 0{,}384766 \times 2IN_{máx}) + 0{,}703125 \times 2IN_{máx} +$
$0{,}384766 \times 4IN_{máx} \rightarrow$
$S_{7máx1} = 2IN_{máx} = 2^1IN_{máx}$

Como todos os valores das saídas da primeira DCT 1-D são menores que $2^4 \times IN_{máx}$, estas saídas podem ser representadas, sem perdas, com quatro bits a mais em relação à entrada $IN_{máx}$, ou seja, as saídas da primeira DCT 1-D necessitam de doze bits para uma representação sem perdas.

Os valores da saída da primeira DCT 1-D são calculados e armazenados no *buffer* de transposição de forma seqüencial e ordenada linha a linha. Estes valores são consumidos coluna a coluna pela segunda DCT 1-D. Como as operações envolvidas no *buffer* de transposição envolvem apenas leituras e escritas em memória, é possível determinar que não haverá acréscimo ou decréscimo no número de bits das palavras de entrada.

Será considerado o mais alto valor de coluna na saída da primeira DCT 1-D para utilizar como entrada máxima na segunda DCT 1-D. Então, tomaremos os valores da coluna gerada por $S_{1máx1}$, com valor de $10IN_{máx}$, como máximo valor das entradas da segunda DCT 1-D.

Como a entrada máxima para a segunda DCT 1-D é dez vezes maior que a entrada máxima da primeira DCT 1-D e como o algoritmo utilizado é exatamente o mesmo para os dois cálculos, é possível concluir que todas as saídas da segunda DCT 1-D serão dez vezes superiores às mesmas saídas da primeira DCT 1-D. Por isso, os cálculos para a determinação do número de bits de cada estágio da segunda DCT 1-D não foram novamente desenvolvidos, sendo apenas apresentados os resultados finais.

Iniciando a análise da segunda DCT 1-D, no primeiro estágio temos a geração de um bit adicional em relação à sua entrada, então as saídas deste estágio terão treze bits, cinco a mais que a entrada $IN_{máx}$, como pode ser observado a seguir:

$$b_{0máx2} = 20IN_{máx} \rightarrow 2^4 IN_{máx} < b_{0máx2} < 2^5 IN_{máx}$$

$$b_{1máx2} = 20IN_{máx} \rightarrow 2^4 IN_{máx} < b_{1máx2} < 2^5 IN_{máx}$$

$$b_{2máx2} = 20IN_{máx} \rightarrow 2^4 IN_{máx} < b_{2máx2} < 2^5 IN_{máx}$$

$$b_{3máx2} = 20IN_{máx} \rightarrow 2^4 IN_{máx} < b_{3máx2} < 2^5 IN_{máx}$$

$$b_{4máx2} = 20IN_{máx} \rightarrow 2^4 IN_{máx} < b_{4máx2} < 2^5 IN_{máx}$$

$$b_{5máx2} = 20IN_{máx} \rightarrow 2^4 IN_{máx} < b_{5máx2} < 2^5 IN_{máx}$$

$$b_{6máx2} = 20IN_{máx} \rightarrow 2^4 IN_{máx} < b_{6máx2} < 2^5 IN_{máx}$$

$$b_{7máx2} = 20IN_{máx} \rightarrow 2^4 IN_{máx} < b_{7máx2} < 2^5 IN_{máx}$$

No segundo estágio temos novamente a geração de um bit adicional em relação às suas entradas, sendo necessários 14 bits para representar suas saídas. Os dados abaixo subsidiam esta afirmação.

$$c_{0máx2} = 40IN_{máx} \rightarrow 2^5 IN_{máx} < c_{0máx2} < 2^6 IN_{máx}$$

$$c_{1máx2} = 40IN_{máx} \rightarrow 2^5 IN_{máx} < c_{1máx2} < 2^6 IN_{máx}$$

$$c_{2máx2} = 40IN_{máx} \rightarrow 2^5 IN_{máx} < c_{2máx2} < 2^6 IN_{máx}$$

$$c_{3máx2} = 40IN_{máx} \rightarrow 2^5 IN_{máx} < c_{3máx2} < 2^6 IN_{máx}$$

$$c_{4máx2} = 40IN_{máx} \rightarrow 2^5 IN_{máx} < c_{4máx2} < 2^6 IN_{máx}$$

$$c_{5máx2} = 40IN_{máx} \rightarrow 2^5 IN_{máx} < c_{5máx2} < 2^6 IN_{máx}$$

$$c_{6máx2} = 40IN_{máx} \rightarrow 2^5 IN_{máx} < c_{6máx2} < 2^6 IN_{máx}$$

$$c_{7máx2} = 20IN_{máx} \rightarrow 2^4 IN_{máx} < c_{7máx2} < 2^5 IN_{máx}$$

O terceiro estágio, como pode ser observado nos cálculos a seguir, também gera um bit a mais em relação à sua entrada, então são necessários 15 bits para representar as saídas deste estágio da segunda DCT 1-D.

$$\boldsymbol{d_{0máx2} = 80IN_{máx} \rightarrow 2^6 N_{máx} < d_{0máx2} < 2^7 IN_{máx}}$$

$$\boldsymbol{d_{1máx2} = 80IN_{máx} \rightarrow 2^6 IN_{máx} < d_{1máx2} < 2^7 IN_{máx}}$$

$$d_{2máx2} = 40IN_{máx} \rightarrow 2^5 IN_{máx} < d_{2máx2} < 2^6 IN_{máx}$$

$$\boldsymbol{d_{3máx2} = 80IN_{máx} \rightarrow 2^6 IN_{máx} < d_{3máx2} < 2^7 IN_{máx}}$$

$$\boldsymbol{d_{4máx2} = 80IN_{máx} \rightarrow 2^6 IN_{máx} < d_{4máx2} < 2^7 IN_{máx}}$$

$$d_{5máx2} = 40IN_{máx} \rightarrow 2^5 IN_{máx} < d_{5máx2} < 2^6 IN_{máx}$$

$$d_{6máx2} = 40IN_{máx} \rightarrow 2^5 IN_{máx} < d_{6máx2} < 2^6 IN_{máx}$$

$$d_{7máx2} = 40IN_{máx} \rightarrow 2^5 IN_{máx} < d_{7máx2} < 2^6 IN_{máx}$$

$$d_{8máx2} = 20IN_{máx} \rightarrow 2^4 IN_{máx} < d_{8máx2} < 2^5 IN_{máx}$$

O quarto estágio não gera bit adicional, sendo usados 15 bits para representar as suas saídas. Os cálculos são apresentados abaixo.

$$\boldsymbol{e_{0máx2} = 80IN_{máx} \rightarrow 2^6 IN_{máx} < e_{0máx2} < 2^7 IN_{máx}}$$

$$\boldsymbol{e_{1máx2} = 80IN_{máx} \rightarrow 2^6 IN_{máx} < e_{1máx2} < 2^7 IN_{máx}}$$

$$e_{2máx2} = 21{,}6406IN_{máx} \rightarrow 2^4 IN_{máx} < e_{2máx2} < 2^5 IN_{máx}$$

$$e_{3máx2} = 28{,}125IN_{máx} \rightarrow 2^4 IN_{máx} < e_{3máx2} < 2^5 IN_{máx}$$

$$e_{4máx2} = 51{,}875IN_{máx} \rightarrow 2^5 IN_{máx} < e_{4máx} < 2^6 IN_{máx}$$

$$e_{5máx2} = 40IN_{máx} \rightarrow 2^5 IN_{máx} < e_{5máx2} < 2^6 IN_{máx}$$

$$e_{6máx2} = 56{,}25IN_{máx} \rightarrow 2^5 IN_{máx} < e_{6máx2} < 2^6 IN_{máx}$$

$$e_{7máx2} = 30{,}7813IN_{máx} \rightarrow 2^4 IN_{máx} < e_{7máx2} < 2^5 IN_{máx}$$

$$e_{8máx2} = 20IN_{máx} \rightarrow 2^4 IN_{máx} < e_{8máx2} < 2^5 IN_{máx}$$

O quinto estágio da segunda DCT 1-D, diferentemente do que ocorre no quinto estágio da primeira DCT 1-D, não gera um bit adicional em relação às suas entradas. Desta forma este estágio usa os mesmos 15 bits das entradas para representar as saídas, conforme está apresentado nos cálculos que seguem.

$$\boldsymbol{f_{0máx2} = 80IN_{máx} \rightarrow 2^6 IN_{máx} < f_{0máx2} < 2^7 IN_{máx}}$$

$$\boldsymbol{f_{1máx2} = 80IN_{máx} \rightarrow 2^6 IN_{máx} < f_{1máx2} < 2^7 IN_{máx}}$$

$$\boldsymbol{f_{2máx2} = 96{,}25IN_{máx} \rightarrow 2^6 IN_{máx} < f_{2máx2} < 2^7 IN_{máx}}$$

$$\boldsymbol{f_{3máx2} = 68{,}125IN_{máx} \rightarrow 2^6 IN_{máx} < f_{3máx2} < 2^7 IN_{máx}}$$

$$f_{4máx2} = 48{,}125IN_{máx} \rightarrow 2^5 IN_{máx} < f_{4máx2} < 2^6 IN_{máx}$$

$$f_{5máx2} = 48{,}125IN_{máx} \rightarrow 2^5 IN_{máx} < f_{5máx2} < 2^6 IN_{máx}$$

$$f_{6máx2} = 52{,}4219IN_{máx} \rightarrow 2^5 IN_{máx} < f_{6máx2} < 2^6 IN_{máx}$$

$$\boldsymbol{f_{7máx2} = 76{,}4844IN_{máx} \rightarrow 2^6 IN_{máx} < f_{7máx2} < 2^7 IN_{máx}}$$

O sexto e último estágio do *pipeline* da segunda DCT 1-D é também o último estágio no cálculo da DCT 2-D. Os cálculos para determinar qual é o máximo valor da saída é apresentado abaixo.

$$\boldsymbol{S_{0máx2} = 80IN_{máx} \rightarrow 2^6IN_{máx} < S_{0máx2} < 2^7IN_{máx}}$$

$$\boldsymbol{S_{1máx2} = 100IN_{máx} \rightarrow 2^6IN_{máx} < S_{1máx2} < 2^7IN_{máx}}$$

$$\boldsymbol{S_{2máx2} = 96{,}25IN_{máx} \rightarrow 2^6IN_{máx} < S_{2máx2} < 2^7IN_{máx}}$$

$$\boldsymbol{S_{3máx2} = 85{,}1563IN_{máx} \rightarrow 2^6IN_{máx} < S_{3máx2} < 2^7IN_{máx}}$$

$$\boldsymbol{S_{4máx2} = 80IN_{máx} \rightarrow 2^6IN_{máx} < S_{4máx2} < 2^7IN_{máx}}$$

$$S_{5máx2} = 57{,}0313IN_{máx} \rightarrow 2^5IN_{máx} < S_{5máx2} < 2^6IN_{máx}$$

$$S_{6máx2} = 48{,}125IN_{máx} \rightarrow 2^5IN_{máxv} < S_{6máx2} < 2^6IN_{máx}$$

$$S_{7máx2} = 20IN_{máx} \rightarrow 2^4IN_{máx} < S_{7máx2} < 2^5IN_{máx}$$

O máximo valor nas saídas da segunda DCT 1-D e, por conseqüência, da própria DCT 2-D é atingido com $S_1 = 100 \times IN_{máx}$. Como este valor é menor que $2^7 \times IN_{máx}$, esta saída, bem como todas as demais saídas da DCT 2-D, podem ser representadas, sem perdas, com sete bits adicionais em relação à entrada INmáx, ou seja, as saídas da DCT 2-D usam 15 bits.

A tab. A4.1 apresenta um resumo de todos os cálculos realizados para a determinação do número de bits dos estágios do cálculo da DCT 2-D.

TABELA A4.1 – Resumo dos cálculos dos números de bits por estágio

Estágio	Operação	Primeira DCT 1-D			Segunda DCT 1-D		
		Saída Máxima	Nº de Bits Usados		Saída Máxima	Nº de Bits Usados	
			Operação	Estágio		Operação	Estágio
1	b0	$2IN_{máx}$	9	9	$20IN_{máx}$	13	13
	b1	$2IN_{máx}$	9		$20IN_{máx}$	13	
	b2	$2IN_{máx}$	9		$20IN_{máx}$	13	
	b3	$2IN_{máx}$	9		$20IN_{máx}$	13	
	b4	$2IN_{máx}$	9		$20IN_{máx}$	13	
	b5	$2IN_{máx}$	9		$20IN_{máx}$	13	
	b6	$2IN_{máx}$	9		$20IN_{máx}$	13	
	b7	$2IN_{máx}$	9		$20IN_{máx}$	13	
2	c0	$4IN_{máx}$	10	10	$40IN_{máx}$	14	14
	c1	$4IN_{máx}$	10		$40IN_{máx}$	14	
	c2	$4IN_{máx}$	10		$40IN_{máx}$	14	
	c3	$4IN_{máx}$	10		$40IN_{máx}$	14	
	c4	$4IN_{máx}$	10		$40IN_{máx}$	14	
	c5	$4IN_{máx}$	10		$40IN_{máx}$	14	
	c6	$4IN_{máx}$	10		$40IN_{máx}$	14	
	c7	$2IN_{máx}$	9		$20IN_{máx}$	13	
3	d0	$8IN_{máx}$	11	11	$80IN_{máx}$	15	15
	d1	$8IN_{máx}$	11		$80IN_{máx}$	15	
	d2	$4IN_{máx}$	10		$40IN_{máx}$	14	
	d3	$8IN_{máx}$	11		$80IN_{máx}$	15	
	d4	$8IN_{máx}$	11		$80IN_{máx}$	15	
	d5	$4IN_{máx}$	10		$40IN_{máx}$	14	
	d6	$4IN_{máx}$	10		$40IN_{máx}$	14	
	d7	$4IN_{máx}$	10		$40IN_{máx}$	14	
	d8	$2IN_{máx}$	9		$20IN_{máx}$	13	
4	e0	$8IN_{máx}$	11	11	$80IN_{máx}$	15	15
	e1	$8IN_{máx}$	11		$80IN_{máx}$	15	
	e2	$2,16IN_{máx}$	10		$21,64IN_{máx}$	13	
	e3	$2,81IN_{máx}$	10		$28,13IN_{máx}$	13	
	e4	$5,19IN_{máx}$	11		$51,88IN_{máx}$	14	
	e5	$4IN_{máx}$	10		$40IN_{máx}$	14	
	e6	$5,63IN_{máx}$	11		$56,25IN_{máx}$	14	
	e7	$3,08IN_{máx}$	10		$30,78IN_{máx}$	13	
	e8	$2IN_{máx}$	9		$20IN_{máx}$	13	
5	f0	$8IN_{máx}$	11	12	$80IN_{máx}$	15	15
	f1	$8IN_{máx}$	11		$80IN_{máx}$	15	
	f2	$9,63IN_{máx}$	12		$96,25IN_{máx}$	15	
	f3	$6,81IN_{máx}$	11		$68,13IN_{máx}$	15	
	f4	$4,81IN_{máx}$	11		$48,13IN_{máx}$	14	
	f5	$4,81IN_{máx}$	11		$48,13IN_{máx}$	14	
	f6	$5,24IN_{máx}$	11		$52,42IN_{máx}$	14	
	f7	$7,65IN_{máx}$	11		$76,48IN_{máx}$	15	
6	S0	$8IN_{máx}$	11	12	$80IN_{máx}$	15	15
	S1	$10IN_{máx}$	12		$100IN_{máx}$	15	
	S2	$9,63IN_{máx}$	12		$96,25IN_{máx}$	15	
	S3	$8,52IN_{máx}$	12		$85,16IN_{máx}$	15	
	S4	$8IN_{máx}$	11		$80IN_{máx}$	15	
	S5	$5,7IN_{máx}$	11		$57,03IN_{máx}$	14	
	S6	$4,81IN_{máx}$	11		$48,13IN_{máx}$	14	
	S7	$2IN_{máx}$	9		$20IN_{máx}$	13	

Anexo 5 Simulação do Compressor JPEG para Imagens em Tons de Cinza

Este anexo apresenta uma das simulações da arquitetura desenvolvida para o compressor JPEG de imagens em tons de cinza. A partir das formas de onda apresentadas neste anexo, é possível perceber o funcionamento dos quatro blocos principais do compressor: DCT 2-D, quantizador, *buffer* ziguezague e codificador de entropia. Antes de serem apresentadas as formas de onda extraídas da simulação do compressor, são apresentadas tabelas que resumem a simulação, tomando como base os quatro blocos principais da arquitetura.

A matriz 8x8 de entrada para o compressor e, por conseqüência, para a DCT 2-D, foi a seguinte:

168	161	161	150	154	168	164	154
171	154	161	150	157	171	150	164
171	168	147	164	164	161	143	154
164	171	154	161	157	157	147	132
161	161	157	154	143	161	154	132
164	161	161	154	150	157	154	140
161	168	157	154	161	140	140	132
154	161	157	150	140	132	136	128

As saídas da arquitetura da DCT 2-D estão apresentadas a seguir. É importante lembrar que o resultado da DCT 2-D está ordenado coluna a coluna e que é uma escala do resultado real da DCT 2-D.

1716	533	-31	193	-82	-3	5	-11
377	-118	149	152	50	-13	90	-30
-63	-54	108	-110	27	-24	27	31
82	-128	53	36	-141	78	30	15
-100	57	-15	-15	-118	53	-23	1
28	80	-63	20	27	-32	-46	9
7	-8	14	-8	5	10	-5	-5
1	-2	1	4	8	-5	-6	-2

A arquitetura do quantizador recebe como entrada a saída do cálculo da DCT 2-D, gerando a matriz de resultados apresentada abaixo, onde novamente os dados estão organizados coluna a coluna.

13	4	0	1	0	0	0	0
3	-2	1	1	0	0	0	0
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

A matriz gerada pelo quantizador é entregue ao *buffer* ziguezague, que recebe os dados ordenados coluna a coluna e os entrega organizados em ziguezague. A matriz abaixo apresenta as saídas do *buffer* ziguezague para a matriz gerada pelo quantizador.

13	4	3	0	-2	0	1	1
0	1	-1	0	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Por fim, a matriz gerada pelo *buffer* ziguezague é entregue ao codificador de entropia, que gera o resultado apresentado abaixo, onde é possível perceber a palavra JPEG sendo montada incrementalmente a cada nova saída do codificador de entropia. A parte válida da palavra JPEG está em negrito. Estão destacadas, com fundo cinza, as palavras JPEG montadas para esta simulação.

Palavra JPEG
101110100000000000000000000000000
101110110010000000000000000000000
1011101100100011100000000000000
10111011001000111110110100000000
1011101100100011111011011001000
10111011001000111110110111001001
1100100000000000000000000000000
1100100000000000000000000000000
1100100011100100000000000000000
11001000111001111111110010000000
11001000111001111111110011111111
10011111111100100000000000000000
10011111111100110100000000000000*

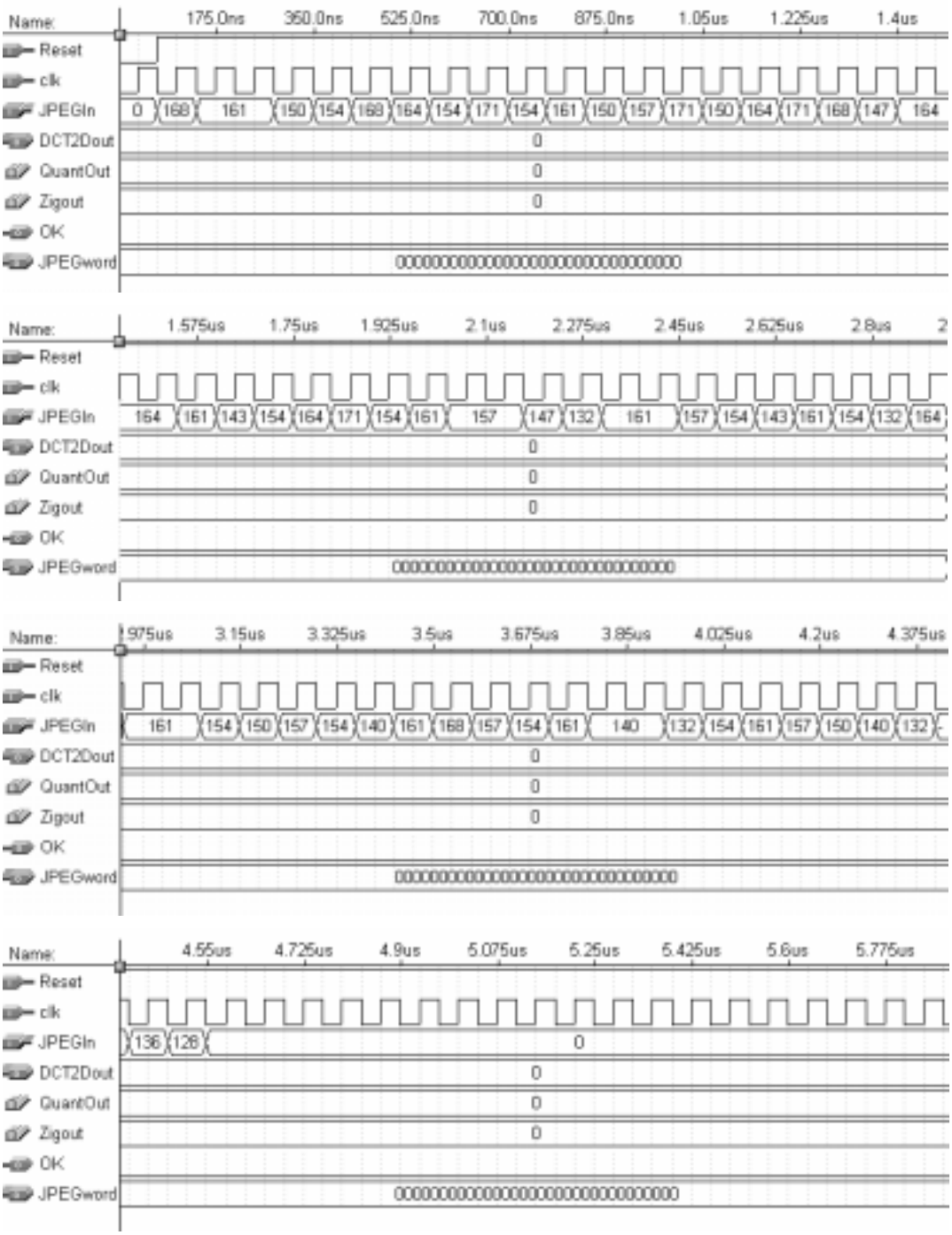
* palavra incompleta

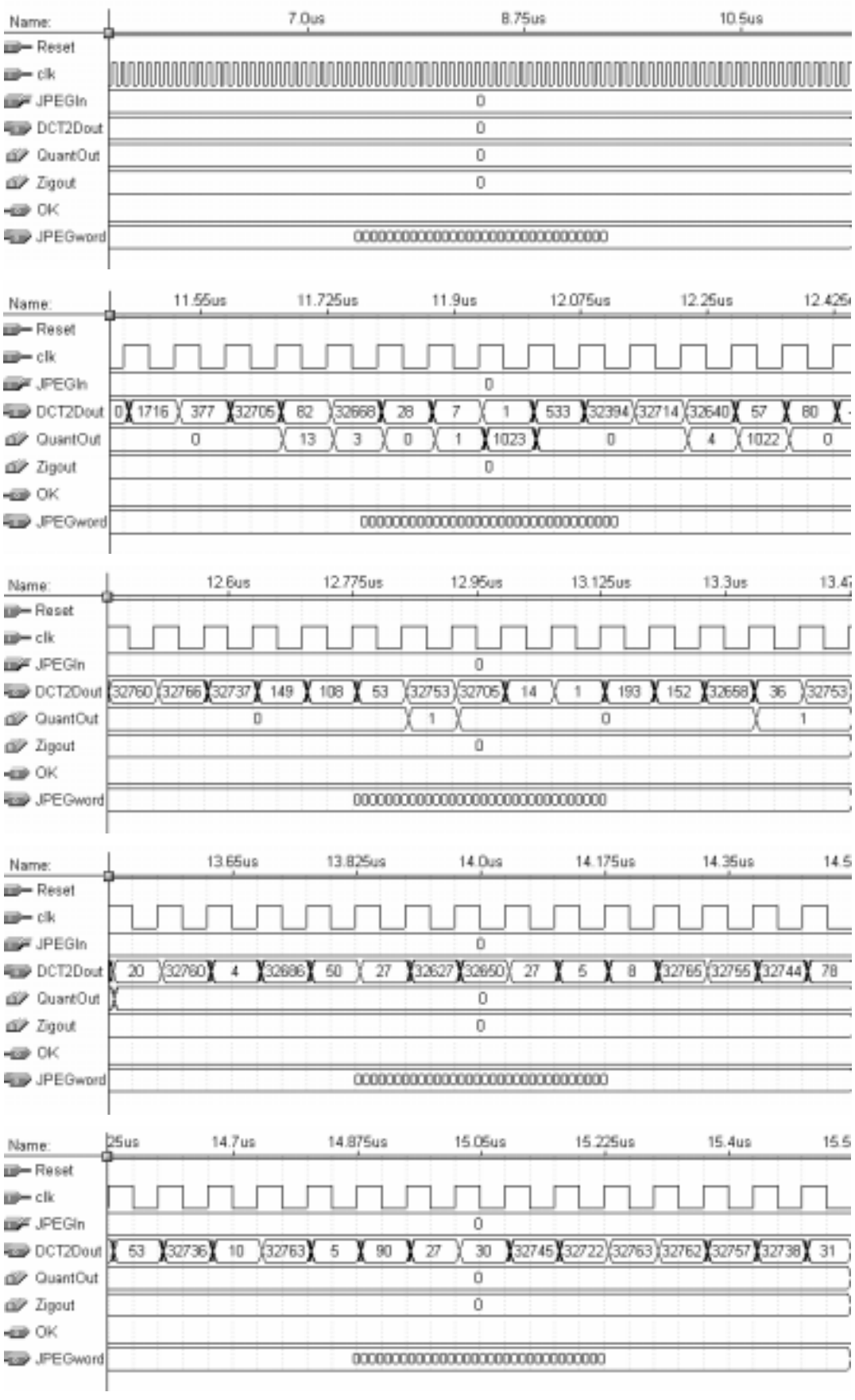
As formas de onda resultantes desta simulação estão apresentadas abaixo, onde *JPEGIn* é a entrada do compressor, *DCT2Dout* é a saída da arquitetura da DCT 2-D, *QuantOut* é a saída do quantizador, *Zigout* é a saída do *buffer* ziguezague e *JPEGword* é a saída do codificador de entropia e, por conseqüência, do compressor JPEG para imagens em tons de cinza. A saída *OK* indica que uma nova palavra JPEG está pronta na saída. As imagens extraídas da simulação buscaram apresentar todos os detalhes relevantes na operação do compressor, por isso algumas delas tiveram o *zoom* aumentado, para trechos críticos da simulação, e outras tiveram *zoom* diminuído, para trechos não relevantes.

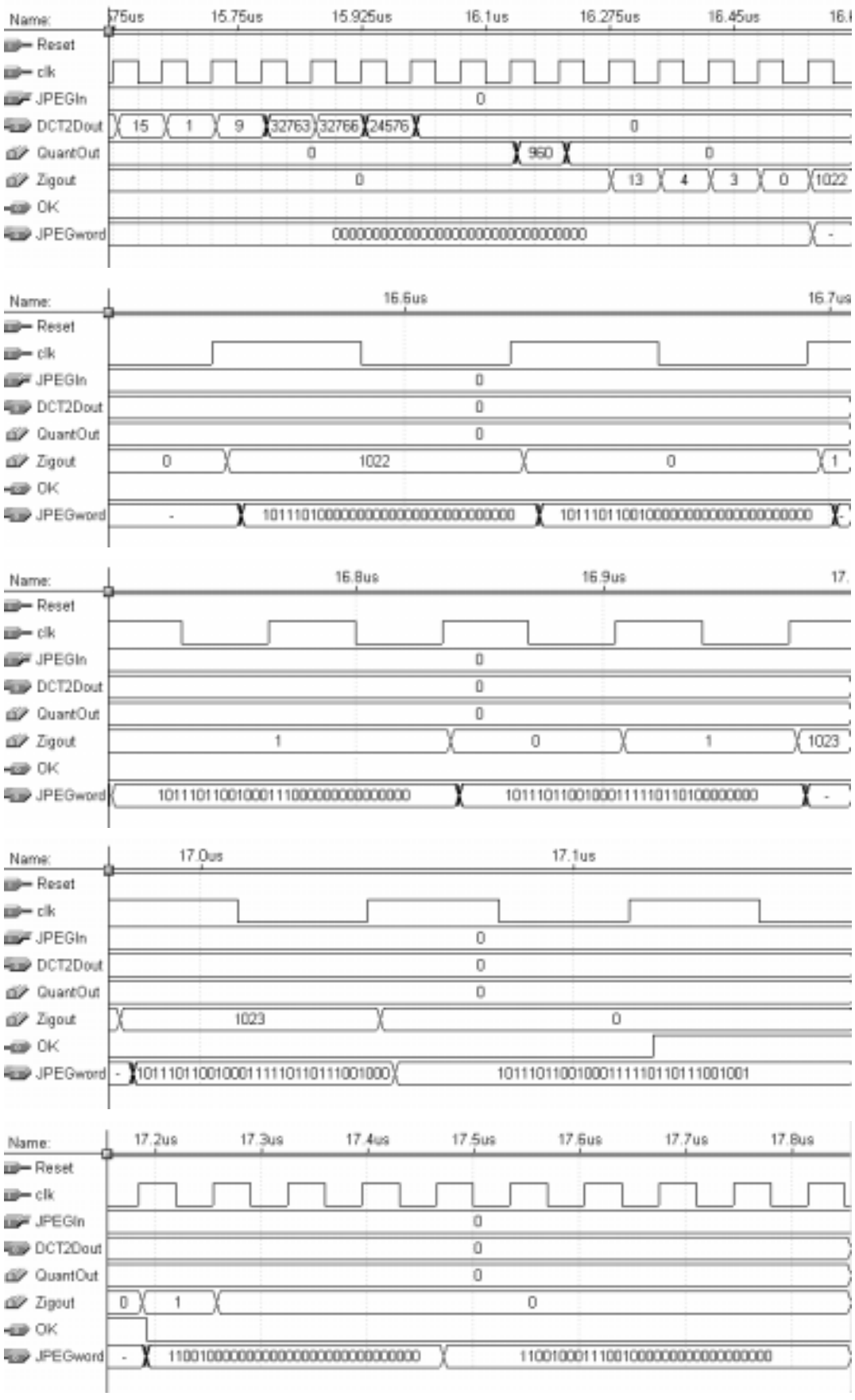
O visualizador de formas de onda utilizado não considera números negativos, portanto, para se chegar aos resultados das matrizes acima apresentadas, os números

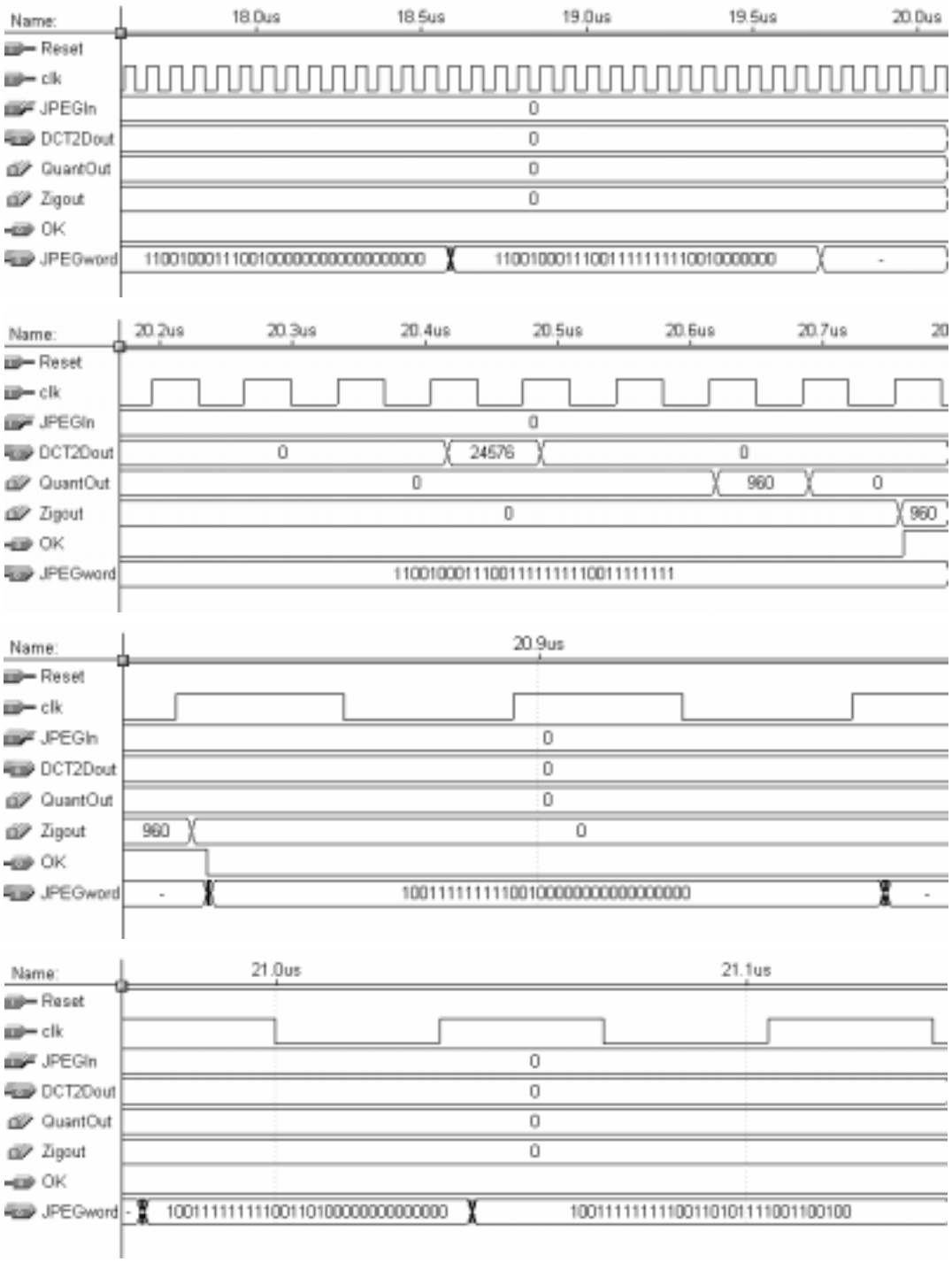
negativos foram transformados a partir da operação do complemento de dois. Então, um dado importante para a análise das formas de onda apresentadas abaixo, é o número de bits na saída de cada bloco do compressor. A saída da DCT 2-D possui 15 bits, enquanto que a saída do quantizador utiliza 10 bits, mesmo número utilizado pela saída do *buffer* ziguezague. A saída do compressor é formada por palavras de 32 bits.

Observando a saída *JPEGword* é possível observar o processo de montagem das palavras JPEG, com os seus resultados intermediários. Somente quando o *flag OK* recebe nível alto é que uma nova palavra JPEG está pronta.




Name:
7.0us
8.75us
10.5us
Reset
clk
JPEGIn
DCT2Dout
QuantOut
Zigout
OK
JPEGword
0
11.55us
11.725us
11.9us
12.075us
12.25us
0 1716 377 32705 82 32668 28 7 1 533 32394 32714 32640 57 80
0 13 3 0 1 1023 0 4 1022 0
12.6us
12.775us
12.95us
13.125us
13.3us
32760 32766 32737 149 108 53 32753 32705 14 1 193 152 32668 36 32753
0 1 0 1
13.65us
13.825us
14.0us
14.175us
14.35us
20 32760 4 32686 50 27 32627 32650 27 5 8 32765 32755 32744 78
14.7us
14.875us
15.06us
15.225us
15.4us
53 32736 10 32763 5 90 27 30 32745 32722 32763 32762 32757 32738 31


Name:
15.75us
15.925us
16.1us
16.275us
16.45us
Reset
clk
JPEGIn
DCT2Dout
QuantOut
Zigout
OK
JPEGword
0
15
1
9
32763
32766
24576
960
13
4
3
1022
16.6us
16.7us
1011101000000000000000000000000
1011101100100000000000000000000
16.8us
16.9us
17.
1011101100100011100000000000000
1011101100100011111011010000000
17.0us
17.1us
1011101100100011111011011001000
1011101100100011111011011001001
17.2us
17.3us
17.4us
17.5us
17.6us
17.7us
17.8us
1100100000000000000000000000000
1100100011100100000000000000000


Name:
18.0us
18.5us
19.0us
19.5us
20.0us
Reset
clk
JPEGIn
DCT2Dout
QuantOut
Zigout
OK
JPEGword
0
11001000111001000000000000000000
11001000111001111111110010000000
-
Name:
20.2us
20.3us
20.4us
20.5us
20.6us
20.7us
Reset
clk
JPEGIn
DCT2Dout
QuantOut
Zigout
OK
JPEGword
0
24576
960
11001000111001111111110011111111
Name:
20.9us
Reset
clk
JPEGIn
DCT2Dout
QuantOut
Zigout
OK
JPEGword
960
100111111111001000000000000000
Name:
21.0us
21.1us
Reset
clk
JPEGIn
DCT2Dout
QuantOut
Zigout
OK
JPEGword
100111111111001101000000000000
100111111111001101011110011001

Anexo 6 Simulação do Conversor de Espaço de Cores

Este anexo apresenta uma das simulações da arquitetura desenvolvida para o conversor de espaço de cores de RGB para YCbCr. A simulação apresentada neste anexo irá considerar a geração de uma matriz 8x8 para cada um dos componentes Y, Cb e Cr, apenas para ilustrar o funcionamento da arquitetura. Se usada no compressor JPEG para imagens coloridas, conforme proposto nesta dissertação, a arquitetura geraria quatro matrizes Y, uma Cb e uma Cr, para respeitar a taxa de *downsampling* proposta e explicada no item 2.3.2.

As matrizes R, G e B utilizadas como entrada para esta simulação estão apresentadas abaixo, sendo que as matrizes R e B são iguais e possuem todos os componentes iguais a 160. A matriz G é a mesma utilizada para a simulação do compressor JPEG para imagens em tons de cinza, apresentada no anexo 5.

Matriz G

168	161	161	150	154	168	164	154
171	154	161	150	157	171	150	164
171	168	147	164	164	161	143	154
164	171	154	161	157	157	147	132
161	161	157	154	143	161	154	132
164	161	161	154	150	157	154	140
161	168	157	154	161	140	140	132
154	161	157	150	140	132	136	128

Matrizes R e B

160	160	160	160	160	160	160	160
160	160	160	160	160	160	160	160
160	160	160	160	160	160	160	160
160	160	160	160	160	160	160	160
160	160	160	160	160	160	160	160
160	160	160	160	160	160	160	160
160	160	160	160	160	160	160	160
160	160	160	160	160	160	160	160

A síntese do resultado da simulação, para os componentes Y, Cb e Cr, estão apresentados nas matrizes abaixo.

Matriz Y

165	161	161	154	156	165	162	156
166	156	161	154	158	166	154	162
166	165	152	162	162	161	150	156
162	166	156	161	158	158	152	144
161	161	158	156	150	161	156	144
162	161	161	156	154	158	156	148
161	165	158	156	161	148	148	144
156	161	158	154	148	144	146	141

Matriz Cb

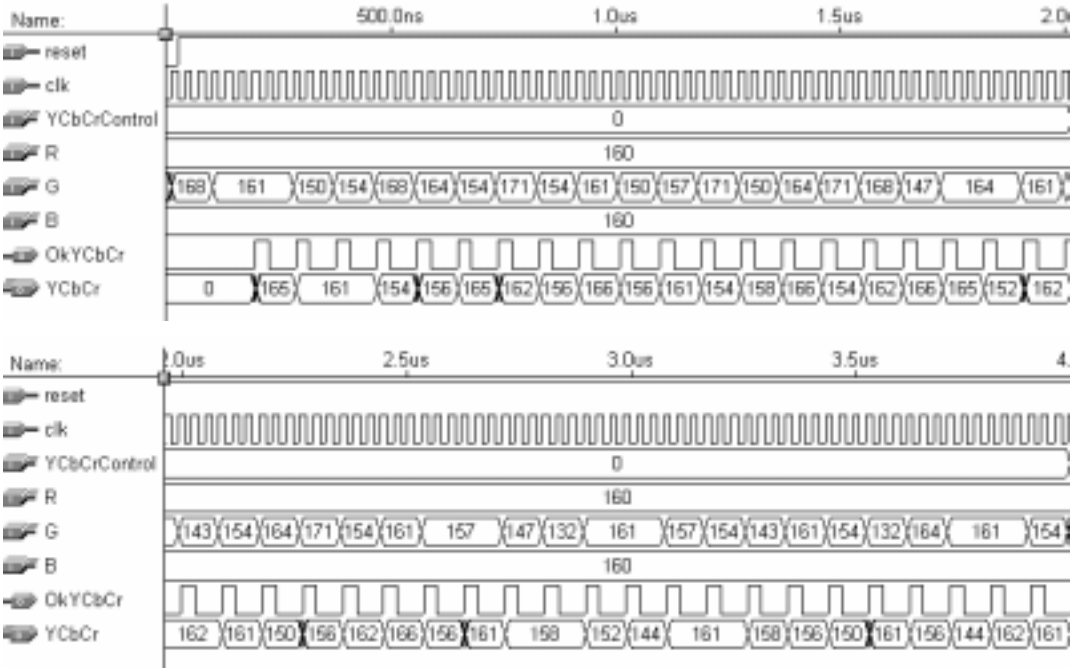
-2	0	0	4	2	-2	-1	2
-3	2	0	4	1	-3	4	-1
-3	-2	5	-1	-1	0	6	2
-1	-3	2	0	1	1	5	10
0	0	1	2	6	0	2	10
-1	0	0	2	4	1	2	7
0	-2	1	2	0	7	7	10
2	0	1	4	7	10	8	11

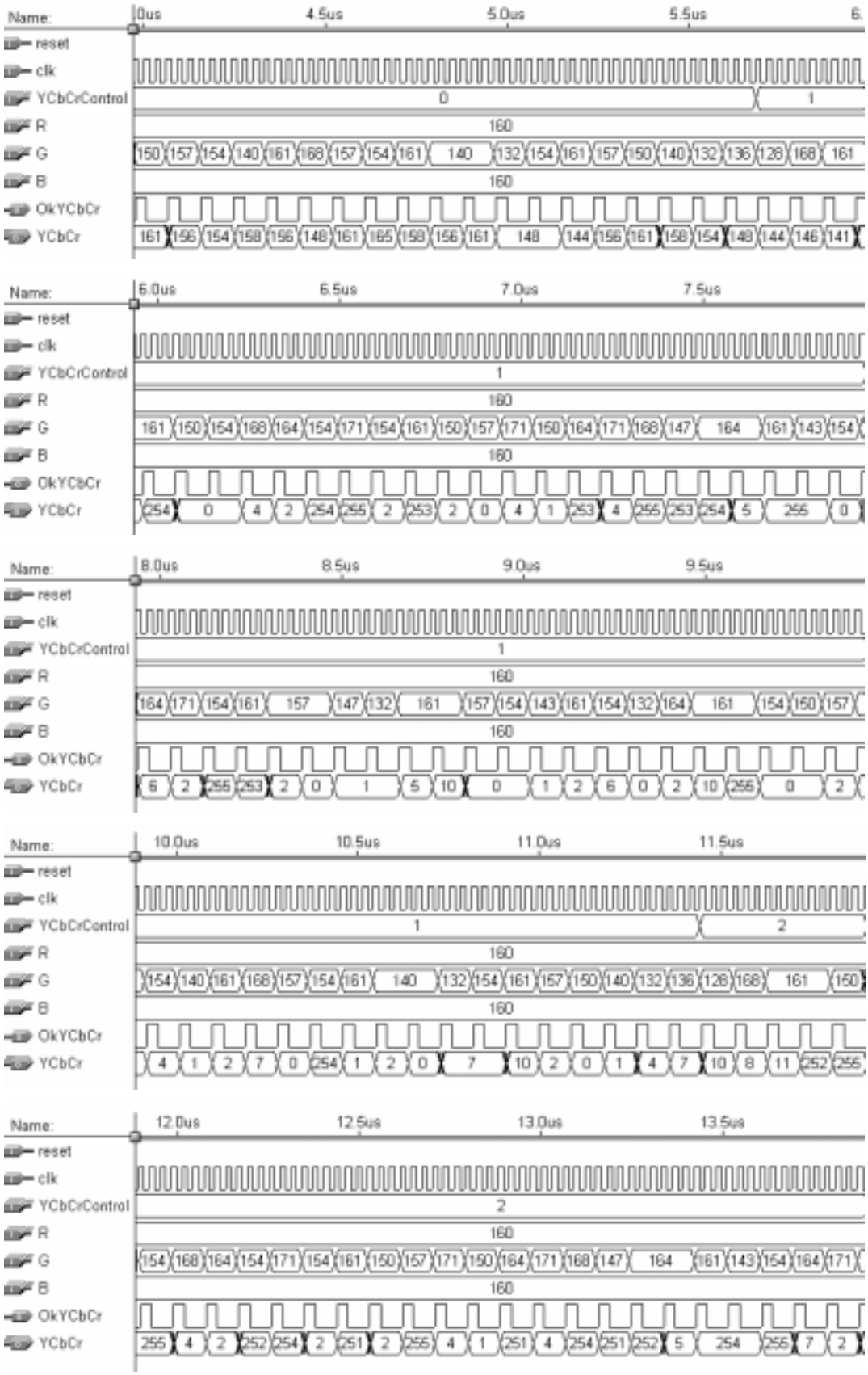
Matriz Cr

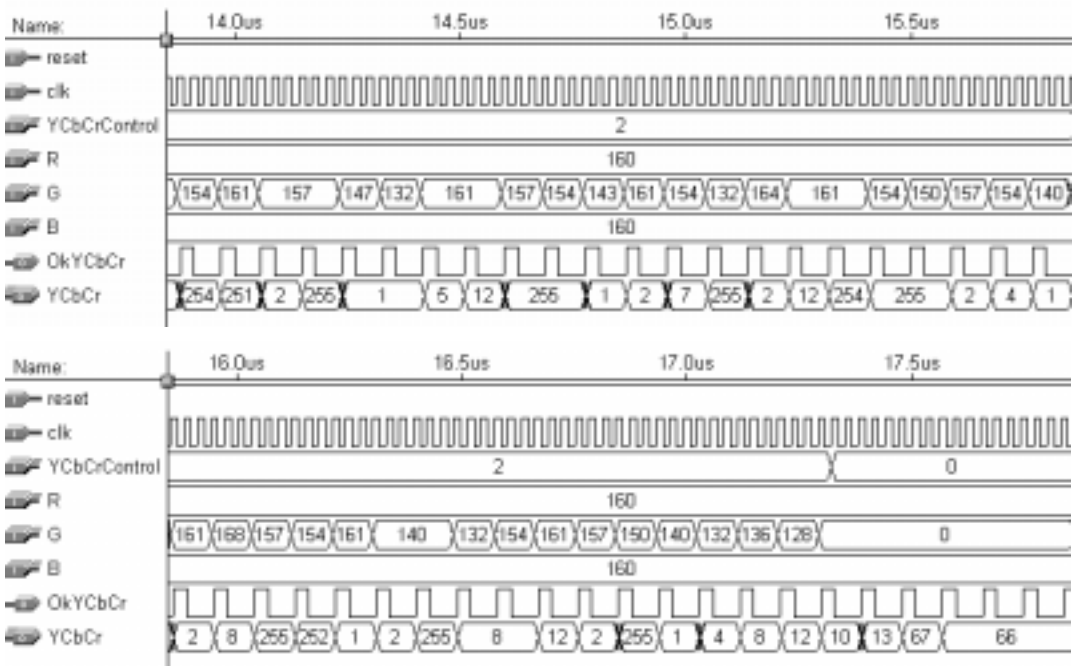
-3	-1	-1	4	2	-3	-2	2
-4	2	-1	4	1	-4	4	-2
-4	-3	5	-2	-2	-1	7	2
-2	-4	2	-1	1	1	5	12
-1	-1	1	2	7	-1	2	12
-2	-1	-1	2	4	1	2	8
-1	-3	1	2	-1	8	8	12
2	-1	1	4	8	12	10	13

O visualizador de formas de onda utilizado, do mesmo modo que para o anexo 5, não considera números negativos, portanto, para se chegar aos resultados das matrizes acima apresentadas, os números negativos foram transformados a partir da operação do complemento de dois.

Nas formas de onda, estão apresentadas as três entradas: *R*, *G* e *B*, e a saída *YCbCr*. O sinal *YCbCrControl* indica qual componente está sendo processado (0 = Y, 1 = Cb e 2 = Cr) e o sinal *OkYCbCr* indica que uma nova saída válida está disponível. Tanto as entradas *R, G* e *B*, quanto a saída *YCbCr*, possuem oito bits.




Name:
reset
clk
YCbCrControl
R
G
B
OkYCbCr
YCbCr
4.5us
5.0us
5.5us
6.0us
6.5us
7.0us
7.5us
8.0us
8.5us
9.0us
9.5us
10.0us
10.5us
11.0us
11.5us
12.0us
12.5us
13.0us
13.5us


Name:
14.0us
14.5us
15.0us
15.5us
reset
clk
YCbCrControl
2
R
160
G
154 161 157 147 132 161 157 154 143 161 154 132 164 161 154 150 157 154 140
B
160
OkYCbCr
YCbCr
254 251 2 256 1 5 12 255 1 2 7 255 2 12 254 255 2 4 1
Name:
16.0us
16.5us
17.0us
17.5us
reset
clk
YCbCrControl
2
0
R
160
G
161 168 157 154 161 140 132 154 161 157 150 140 132 136 128 0
B
160
OkYCbCr
YCbCr
2 8 256 252 1 2 255 8 12 2 255 1 4 8 12 10 13 67 66

Bibliografia

[AGO 2001] AGOSTINI, Luciano; BAMPI, Sergio. Integrated Digital Architecture for JPEG Image Compression. In: EUROPEAN CONFERENCE ON CIRCUIT THEORY AND DESIGN, ECCTD, 2001, Epoo, Finlândia. **Proceedings...** Espoo: ECS, 2001. v. 3, p. 181-184.

[AGO 2001a] AGOSTINI, Luciano; SILVA, Ivan; BAMPI, Sergio. Pipelined Fast 2-D DCT Architecture for JPEG Image Compression. In: SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEM DESIGN, SBCCI, 13., 2001, Pirinópolis, Brasil. **Proceedings...** Los Alamitos: IEEE, 2001. 237p. p.226-231.

[AGO 2001b] AGOSTINI, Luciano; BAMPI, Sergio. Projeto de uma Arquitetura de DCT 1D para a Compressão de Imagens JPEG. In: WORKSHOP IBERCHIP, 7., 2001, Montevideo, Uruguai. **[Memórias].** Montevideo: Universidad de la Republica, 2001. 1 CD.

[AGO 2001c] AGOSTINI, Luciano; BAMPI, Sergio. Arquitetura Integrada para Conversor de Espaço de Cores e Downsampler para a Compressão de Imagens JPEG. WORKSHOP IBERCHIP, 7., 2001, Montevideo, Uruguai. **[Memórias].** Montevideo: Universidad de la Republica, 2001. 1 CD.

[AGO 2000] AGOSTINI, Luciano; BAMPI, Sergio. Projeto de Arquitetura Integrada de um Compressor de Imagens para JPEG. In: SEMANA ACADÊMICA DO PPGC, 5., 2000, Porto Alegre, Brasil. **Anais...** Porto Alegre: PPGC da UFRGS, 2000. 219p. p.215-218.

[ALT 2001] ALTERA CORPORATION. **Altera:** The Programmable Solutions Company. San Jose: Altera Corporation, 2002. Disponível em: <http://www.altera.com>. Acesso em: 15 fev. 2002.

[ALT 2001a] ALTERA CORPORATION. **FLEX 10KE - Embedded Programmable Logic Devices Data Sheet - version 2.3.** San Jose: Altera Corporation, 2001. 1 CD.

[ARA 88] ARAI, Y.; AGUI, T.; NAKAJIMA, M. A Fast DCT-SQ Scheme for Images. **Transactions of IEICE**, [S.l.], v.E71, n.11, p. 1095-1097, 1988.

[BHA 99] BHASKARAN, Vasudev; KONSTANTINIDES, Konstantinides. **Image and Video Compression Standards Algorithms and Architectures.** 2nd ed. Massachusetts: Kluwer Academic Publishers, 1999, 454p.

[CHE 77] CHEN, W.; SMITH, C.; FRALICK, S. A Fast Computational Algorithm for the Discrete Cosine Transform. **IEEE Transactions on Communications**, New York, v.25, n.9, p.1004-1009, 1977.

[COM 87] COMPUSERVE INCORPORATED. **GIF – Graphics Interchange Format (tm) – A standard defining a mechanism for the storage and transmission of raster-based graphics information.** [S.l.]: CompuServe Incorporated, 1987. Disponível em: <http://www.w3.org/Graphics/GIF/spec-gif87.txt>. Acesso em: 14 fev. 2002.

[FEI 92] FEIG, Ephraim.; WINOGRAD, Shmuel. Fast Algorithms for the Discrete Cosine Transform. **IEEE Transactions on Signal Processing**, New York, v.40, n.9, p.2174-2193, Sept. 1992.

[GON 93] GONZALES, Rafael; WOODS, Richard. **Digital Image Processing**. Massachusetts: Addison Wesley, 1993. 716p.

[HAM 92] HAMILTON, Eric. **JPEG File Interchange Format - version 1.02**. [S.l.]: C-Cube Microsystems, 1992. Disponível em: <http://www.w3.org/Graphics/JPEG/jfif.txt>. Acesso em: 13 jul. 2001.

[HWA 79] HWANG, Kai. **Computer Arithmetic:** Principles, Architecture and Design. New York: John Wiley & Sons, 1979. 423p.

[HOF 2001] HOFFMANN, Gustavo; FABRIS, Eric; ZANDONAI, Diogo; BAMPI, Sergio. The BinDCT processor. In: WORKSHOP IBERCHIP, 7., 2001, Montevideo, Uruguai. **[Memórias].** Montevideo: Universidad de la Republica, 2001. 1 CD.

[JAI 89] JAIN, Anil. **Fundamentals of Digital Image Processing**. London: Prentice Hall, 1989. 569p.

[JPE 2001] JPEG AND JBIG COMMITTEES. **Home Site of the JPEG and JBIG Committees**. Disponível em: <http://www.jpeg.org>. Acesso em: 21 abr. 2001.

[KOV 95] KOVAC, Mario; RANGANATHAN, N. JAGUAR: A Fully Pipelined VLSI Architecture for JPEG Image Compression Standard. **Proceedings of the IEEE**, New York, v.83, n.2, p. 247-258, Feb. 1995.

[LEE 97] LEE, Yung-Pin; CHEN, Thou-Ho et al. A Cost-Effective Architecture for 8 x 8 Two-Dimensional DCT/IDCT Using Direct Method. **IEEE Transactions on Circuits and Systems for Video Technology**, New York, v.7, n.3, p.459-467, June 1997.

[LEE 84] LEE, B. A New Algorithm to Compute the Discrete Cossine Transform. **IEEE Transactions on ASSP**, [S.l.], v.32, n.6, p.1243-1245, Dec. 1984.

[LEI 91] LEI, Shaw-Min; SUN, Ming-Ting. An Entropy Coding System for Digital HDTV Applications. **IEEE Transactions on Circuits and Systems for Video Technology**, New York, v.1, n.1, p.147-155, Mar. 1991.

[LI 2002] LI, Ze-Nian. **Color in Image and Video.** [S.l.]: Simon Fraser University, 2002. Disponível em: <http://www.cs.sfu.ca/CourseCentral /365/li/material/notes/Chap3/Chap3.3/Chap3.3.html>. Acesso em: 10 jan. 2002.

[MAD 95] MADISETTI, Avanindra; WILLSON JR, Alan. A 100 MHz 2-D 8 x 8 DCT/IDCT Processor for HDTV Applications. **IEEE Transactions on Circuits and Systems for Video Technology**, New York, v.5, n.2, p.158-165, Apr. 1995.

[MIA 99] MIANO, John. **Compressed Image File Formats – JPEG, PNG, GIF, XBM, BMP**. New York: Addison Wesley, 1999. 264p.

[MEL 2000] MELO, Marco; NEVE, Alessandro. Processador Reconfigurável PDCT para Compressão de Imagens de Vídeo Digital em Tempo Real. In: COMPUTAÇÃO RECONFIGURÁVEL – EXPERIÊNCIAS E PERSPECTIVAS, 2000, Marília, Brasil. **Anais...** Marília: Fundação de Ensino Eurípides Soares da Rocha, 2000. 314p. p.192-201.

[MUR 96] MURRAY, J. D., VANRYPER, W**. Encyclopedia of Graphics File Formats.** 2nd ed. Sebastopol: O'Reilly & Associates, 1996. 1116p.

[PEN 92] PENNEBAKER, Willian; MITCHELL, Joan. **JPEG Still Image Data Compression Standard**. New York: Van Nostrand Reinhold, 1992. 638p.

[ROE 2002] ROELOFS, Greg. **PNG - Portable Network Graphics - A Turbo-Studly Image Format with Lossless Compression.** 2002. Disponível em: <http://www.libpng.org/pub/png/>. Acesso em: 14 fev. 2002.

[THE 92] THE INTERNATIONAL TELEGRAPH AND TELEPHONE CONSULTATIVE COMMITTEE (CCITT). **Information Technology – Digital Compression and Coding of Continuous-Tone Still Images – Requirements and Guidelines - Recommendation T.81**. Geneva: International Telecommunication Union, 1992. 182p.

[TRA 2000] TRAN, Trac. The BinDCT: Fast Multiplierless Approximation of the DCT. **IEEE Signal Processing Letters**, New York, v.7, n.6, p.141-144, June 2000.

[WAN 95] WANG, Chin-Liang; CHEN, Chang-Yu. High-Throughput VLSI Architectures for the 1-D and 2-D Discrete Cosine Transforms. **IEEE Transactions on Circuits and Systems for Video Technology**, New York, v.5, n.1, p.31-40, Feb. 1995.

[WES 95] WESTE, Neil; ESHRAGHIAN, Kamran. **Principles of CMOS VLSI Design.** 2nd Ed. Massachusetts: Addison-Wesley, 1995. 531p.