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

SANDRO VILELA DA SILVA

Projeto de uma Arquitetura Dedicada à Compressão de Imagens no Padrão JPEG2000

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Sergio Bampi
Orientador

Porto Alegre, novembro de 2005.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Silva, Sandro Vilela da

Arquiteturas de Processadores Dedicados à Compressão de Imagens no Padrão JPEG2000 / Sandro Vilela da Silva – Porto Alegre: Programa de Pós-Graduação em Computação, 2005.

124 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2005. Orientador: Sergio Bampi.

1. JPEG2000. 2. Compressão de Imagens 3. Transformada Wavelet Discreta. I. Bampi, Sergio. II. Título.

AGRADECIMENTOS

Ao término deste trabalho, quero prestar o meu agradecimento a todas as pessoas, que colaboraram direta ou indiretamente neste trabalho.

Inicialmente quero agradecer ao Centro Federal de Educação Tecnológica de Pelotas (CEFET-RS) por me proporcionar a oportunidade de desenvolver este trabalho em tempo integral.

Ao Intituto de Informática da Universidade Federal do Rio Grande do Sul por me aceitar dentro de seu quadro discente.

Ao CNPQ e seu programa nacional de microeletrônica, sem o qual esta caminhada teria sido mais difícil.

Aos meus colegas do curso de Eletrônica do CEFET-RS por me apoiarem no desenvolvimento deste trabalho.

Aos meus amigos e parentes, que me incentivaram nesta jornada.

Aos professores do PPGC por sua dedicação e incentivo.

Aos novos amigos que conheci no GME, pelo companheirismo criado durante todo este tempo e que tiveram contribuição fundamental neste trabalho.

Ao meu orientador, prof Sérgio Bampi, que inspirou e me ajudou a trilhar todo o caminho para a realização deste trabalho.

Aos meus pais, Francisco, que foi a inspiração para toda a minha caminhada e Ceci por me ajudar a enxergar o mundo de outras maneiras.

À minha filha Nathalie, por me permitir ver a beleza do mundo.

Ao meu filho Lorenzzo, que me trouxe muitas alegrias durante o mestrado.

À minha esposa e grande companheira Mara, que durante todo o tempo que esteve a meu lado sempre me apoiou em todas as realizações, trazendo felicidade, carinho e amor para a minha vida.

E a todas as pessoas que, de alguma forma, contribuíram para a minha formação e que, portanto, indiretamente, contribuiram para o sucesso deste trabalho.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AMA	*Adder - Multiplier - Adder*
ASIC	*Application Specific Integrated Circuit* ou Circuito Integrado de Aplicação Específica
BAC	*Binary Arithmetic Coding* ou Codificação Aritmética Binária
BCP	*Bit-Plane Coding* ou Codificação de plano de bits
Bitmap	mapa de bits
CCD	*charge-coupled device*
CCITT	*International Telegraph and Telephone Consultative Committee*
CUP	*CleanUp Pass*
DCT	*Discrete Cosine Transform* ou Transformada Discreta do Cosseno
1D-DCT	*Uni dimensional Discrete Cosine Transform* ou Transformada Discreta do Cosseno unidimensional
2D-DCT	*Bidimensional Discrete Cosine Transform* ou Transformada Discreta do Cosseno bidimensional
DFT	*Discrete Fourier Transform* ou Transformada Discreta de Fourier
DMA	*Direct Memory Access* ou Acesso Direto à Memória
DWT	*Discrete Wavelet Transform* ou Transformada Wavelet Discreta
1D-DWT	*Unidimensional Discrete Wavelet Transform* ou Transformada Wavelet Discreta Unidimensional
2D-DWT	*Bidimensional Discrete Wavelet Transform* ou Transformada Wavelet Discreta Bidimensional
EAB	*Embedded Array Block* ou Blocos de Arranjo Embarcado
EBCOT	*Embedded Block Coding with Optimized Truncation*
FDCT	*Forward Discrete Cosine Transform* ou Transformada Discreta do Cosseno Direta
FFT	*Fast Fourier Transform* ou Transformada Rápida de Fourier
FIR	*Finite Impulse Response*

FPGA	*Field Programmable Gate Array* ou Arranjo de Portas Programáveis por Campo
FSM	*Finite State Machine* ou Máquina de Estados Finitos
ICT	*Irreversible color transformation* ou Transformação de Cores Irreversível
IDCT	*Inverse Discrete Cosine Transform* ou Transformada Discreta de Fourier Inversa
IEC	*International Electrotechnical Commission*
ISO	*International Organization for Standardization* ou Organização Internacional para Padronização
ITU	*International Telecommunication Union*
JBIG	*Joint Bi-level Image Processing Groupt*
JFIF	*JPEG File Interchange Format*
JPEG	*Joint Photographic Expert Group*
KLT	*Karhunen-Loeve Transform* ou Transformada Karhunen-Loeve
LE	*Logic Element* ou Elemento Lógico
LPS	*Less probable Symbol* ou Símbolo de menor probabilidade
LSB	*Least Significant Bit* ou Bit menos significativo
MER	*Mars Exploration Rover* ou Veículo de Exploração de Marte
MP3	*Moving Photographic Expert Group*
MPS	*More probable Symbol* ou Símbolo de maior probabilidade
MPEG	*Moving Picture Experts Group*
MRC	*Magnitude Refinement Coding* ou Codificação por Refinamento de Magnitude.
MRP	*Magnitude Refinement Pass* ou Passagem para Refinamento da Magnitude
MSB	*More Significant Bit* ou Bit mais significativo
MSE	*Mean Squared Error* ou erro médio quadrático
PAL	*Phase Alternating Line*
PE	*Processing Element* ou Elemento de Processamento
pixel	*picture element* ou elemento da imagem
PSNR	*peak signal to noise ratio* ou relação sinal ruído de pico
RAM	*Random Access Memory* ou Memória de Acesso Randômico
RCT	*Reversible color transformation* ou Transformação de Cores Reversível
RGB	*Red-Green-Blue* (padrão de formação de cores em imagem de vídeo)
RLE	*Run Length Encoder*

RMSE	*Root Mean Squared Error* ou raiz quadrada do erro médio quadrático
RPA	*Recursive Pyramid Algorithm* ou Algoritmo da Pirâmide Recursiva
SPP	*Significance Propagation Pass* ou Passagem para Propagação da Significância
SNR	*signal to noise ratio* ou relação sinal ruído
VLC	*Variable Length Coder* ou codificador de comprimento variável
VLSI	*Very Large Scale Integration* ou Integração em muito Larga Escala

LISTA DE SÍMBOLOS

ψ	função Wavelet
ϕ	função escala
Wf(u,s)	Transformada Wavelet
Sf(u,ξ)	Transformada de Fourier janelada
W	função escala
Wh	função wavelet
#	número de elementos

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O incremento das taxas de transmissão e de armazenamento demanda o desenvolvimento de técnicas para aumentar a taxa de compressão de imagens e ao mesmo tempo mantenha a qualidade destas imagens. O padrão JPEG2000 propõe a utilização da transformada *wavelet* discreta e codificação aritmética para alcançar altos graus de compressão, proporcionando que a imagem resultante tenha qualidade razoável. Este padrão permite tanto compressão com perdas como compressão sem perdas, dependendo apenas do tipo de transformada *wavelet* utilizada.

Este trabalho propõe a implementação de blocos internos em *hardware* para compor um compressor de imagens com perdas seguindo o padrão JPEG2000. O principal componente deste compressor de imagens é a transformada *wavelet* discreta irreversível em duas dimensões, que é implementada utilizando um esquema *lifting* a partir dos coeficientes Daubechies 9/7 descritos na literatura. Para proporcionar altas taxas de compressão para a transformada irreversível, são utilizados coeficientes reais – que são originalmente propostos em representação de ponto-flutuante. Neste trabalho, estes coeficientes foram implementados em formato de ponto-fixo arredondado, o que resulta erros que foram estimados e controlados.

Neste trabalho, várias arquiteturas em *hardware* para a descrição da transformada *wavelet* discreta irreversível em duas dimensões foram implementadas para avaliar a relação entre tipo de descrição, consumo de área e atraso de propagação. A arquitetura de melhor relação custo benefício requer 2.090 células de um dispositivo FPGA, podendo operar a até 78,72 MHz, proporcionando uma taxa de processamento de 28,2 milhões de amostras por segundo. Esta arquitetura resultou em um nível de erro médio quadrático de 0,41% para cada nível de transformada. A arquitetura implementada para o bloco do codificador de entropia foi sintetizada a partir de uma descrição comportamental, gerando um *hardware* capaz de processar até 843 mil coeficientes de entrada por segundo.

Os resultados indicam que o compressor de imagens com perdas seguindo o padrão JPEG2000, utilizando os blocos implementados nesta dissertação e operando na máxima freqüência de operação definida, pode codificar em média 1,8 milhões de coeficientes por segundo, ou seja, até 27 *frames* de 256x256 *pixels* por segundo. Esta limitação na taxa de codificação é definida pelo codificador de entropia, que possui um algoritmo mais complexo, necessitando de um trabalho complementar para melhorar sua taxa de codificação aumentando o paralelismo do *hardware*.

Palavras Chave: Compressão de Imagens, JPEG2000, Transformada Wavelet, DWT.

Design of a dedicated architecture to Image compression in the JPEG2000 Standard

ABSTRACT

The increasing demands for higher data transmission rates and higher data storage capacity call for the development of techniques to increase the compression rate of images while at the same time keeping the image quality. The JPEG2000 Standard proposes the use of the discrete wavelet transform and of arithmetic coding to reach high compression rates, providing reasonable quality to the resulting compressed image. This standard allows lossy as well as loss-less compression, dependent on the type of wavelet transform used.

This work considers the implementation of the internal hardware blocks that comprise a lossy image compressor in hardware following the JPEG2000 standard. The main component of this image compressor is the two dimensional irreversible discrete wavelet transform, that is implemented using a lifting scheme with the Daubechies 9/7 coefficients presented in the literature. To provide high compression rates for the irreversible transform, these coefficients – originally proposed in their floating-point representation – are used. In this work, they are implemented as fixed-point rounded coefficients, incurring in errors that we estimate and control.

In this work, various hardware architectures for the two dimensional irreversible discrete wavelet transform were implemented to evaluate the tradeoff between the type of description, area consumption and delay. The architecture for the best trade-off requires 2,090 logic cells of a FPGA device, being able to operate up to 78.72 MHz, providing a processing rate of 28.2 million of samples per second. This architecture resulted in 0.41% of mean quadratic error for each transformed octave. The architecture implemented for the block of the entropy encoder was synthesized from a behavioral description, generating the hardware able to process up to 843 thousands of input coefficients per second.

The results indicate that the lossy image compressor following JPEG2000 standard, using the blocks implemented in this dissertation and operating in the maximum clock frequency can codify, in average, 1.8 million coefficients per second, or conversely, up to 27 frames of 256x256 pixels per second. The rate-limiting step in this case is the entropy encoder, which has a more complex algorithm that needs further work to be sped up with more parallel hardware.

Keywords: Image compression, JPEG2000, Wavelet Transform, DWT.

1 INTRODUÇÃO

Desde a antigüidade, o homem busca incessantemente a criação de novos meios para tornar a vida cada vez mais facilitada. Um dos primeiros grandes avanços da humanidade foi a criação e utilização de ferramentas. Desde então a criação de novas idéias, ferramentas e tecnologias se tornaram constante. Porém a cada idéia e a cada ferramenta descoberta, novas possibilidades surgiam, exigindo a criação de tecnologias ainda mais novas para facilitar o desenvolvimento e o uso das ferramentas criadas.

Atualmente, vivemos em um mundo onde a tecnologia cresce a uma taxa elevada, potencializada pelos diversos meios para o intercâmbio das informações. Assim, a troca de informações é uma necessidade constante e a quantidade no tráfego de dados cresce continuamente. Deste modo, faz-se cada vez mais necessário aumentar esta taxa de intercâmbio de informações, ou seja, que a transmissão, recepção e armazenamento de documentos eletrônicos ocorram com a maior rapidez possível para quantidades cada vez maiores de dados, mantendo uma confiabilidade em níveis aceitáveis para o usuário.

Considerando que a demanda por transmissão e processamento de informações aumenta constantemente, é necessário que novas tecnologias sejam incorporadas aos processos de transmissão, recepção e armazenamento dos dados. Estas novas tecnologias devem alinhar-se em duas estratégias básicas:

- Aumentar a velocidade de transmissão (utilizando redes de comunicação de dados de alta velocidade);
- Reduzir o volume dos dados a serem transmitidos (utilizando compressão de dados).

Para o aumento da velocidade de transmissão é necessário o desenvolvimento das tecnologias dos meios de transmissão, sejam físicos ou não, como fibras óticas, dispositivos *wireless*, modems de alta velocidade, roteadores, protocolos dedicados, etc. Porém o aumento da velocidade de transmissão proporcionado pelo desenvolvimento das tecnologias dos meios de transmissão não atende totalmente a demanda do crescimento da taxa de intercâmbio de informações. Para a redução do tráfego e do requisito de armazenamento dos dados é necessária também uma compressão dos dados originais.

De um modo geral a compressão de dados é feita a partir da retirada ou substituição de porções redundantes do conjunto total de dados, ou seja, é possível substituir grandes cadeias de bits por cadeias menores, sem perda da informação. A partir desta idéia geral, é possível utilizar dois processos distintos.

O processo de compressão sem perdas, que visa obter um conjunto de dados comprimidos que pode ser reconstruído exatamente igual ao conjunto original de bits. O processo de compressão com perdas, que visa obter um conjunto de dados comprimidos que pode ser reconstruído de modo semelhante ao conjunto original de bits, porém nunca exatamente igual. A compressão com perdas possibilita a obtenção de grandes taxas de compressão, podendo alcançar até 100 vezes ou mais, dependendo do algoritmo utilizado e do grau de fidelidade requerido.

As técnicas tradicionais para compressão de imagens exploram a redundância estatística presente em imagens reais (ex: representação digitalizada de uma fotografia), que também podem ser definidas como imagens em tons contínuos. Esta redundância em imagens também é conhecida como correlação, pode ser traduzida como a semelhança entre dois *pixels* adjacentes (*pixel correlation*). Os principais processos para compressão de imagens alteram a representação digital da imagem do domínio espacial para o domínio das freqüências, de modo que na representação no domínio das freqüências as informações de correlação entre *pixels* podem ser reduzidas e até mesmo eliminadas (compressão ideal) (SALOMON, 2000).

A eficiência de um compressor de imagens depende da quantidade de energia da imagem que a transformada pode concentrar em um número reduzido de coeficientes. De modo geral, a aplicação de uma transformada matemática a uma imagem em tons contínuos resulta em uma representação com grandes amplitudes nos valores próximos à origem e com baixas amplitudes em valores distantes da origem.

Para proporcionar uma maior redução da quantidade total de dados, é possível descartar algumas informações sem ocasionar perda significativa na qualidade da imagem reconstruída. Nos métodos de compressão com perdas, a quantização dos coeficientes é o principal método para redução da quantidade de bits necessária para representar os coeficientes resultantes da transformada.

Considerando que a técnica de compressão com perdas introduz alterações irreversíveis em uma imagem, é comum imaginarmos que "a compreensão da imagem ficaria comprometida". Porém, ao processar informações sensoriais, o cérebro humano realiza interpolação entre estas informações, de modo que não é necessária a observação de todos os pontos de uma imagem para compreendê-la, provando a existência de uma redundância natural nas informações sensoriais. Assim, a eliminação de algumas porções de informações do conjunto de dados que compõe uma imagem pode ser compensada pela análise mental do cérebro humano, então estas informações podem ser desprezadas sem prejuízo de compreensão (GONZALES, 1992).

Alguns tipos de imagens permitem que determinado nível de perda possa ser considerado aceitável. As transformadas matemáticas utilizadas nos algoritmos de compressão podem ter sua precisão reduzida sem ocorrer perda significativa da informação da imagem, uma vez que esta perda de informação pode ser suprimida pelo erro previsto nos processos de amostragem e quantização. O erro inserido no processo de compressão pode ser medido por meio do PSNR (*Peak Signal to Noise Ratio*) da imagem reconstruída.

Em 2000 surgiu um dos mais eficientes padrões de compressão de imagens em tons contínuos, conhecido como JPEG2000 (ITU-T, 2000). Este padrão de compressão de imagens define a transformada *Wavelet* Discreta (DWT) em duas dimensões como o algoritmo padrão para a transformação do domínio espacial para o domínio das freqüências para reduzir a correlação entre os elementos da imagem. O padrão

JPEG2000 permite compressão com ou sem perdas, bastando para isso modificar os coeficientes da transformada e os coeficientes de quantização.

O padrão anterior ao padrão JPEG2000 para a compressão de imagens com e sem perdas é o padrão JPEG, descrito em ITU-T (1992). Uma das principais diferenças entre estes dois padrões é o tipo de transformada utilizada. Como já citado anteriormente, o padrão JPEG2000 utiliza a transformada *wavelet* enquanto o padrão JPEG utiliza a transformada do cosseno. Esta mudança do tipo de transformada permite que o padrão JPEG2000 alcance taxas de compressão maiores que o padrão JPEG.

A implementação da transformada *Wavelet* Discreta (DWT) em duas dimensões (2D-DWT) pode ser obtida por aplicações sucessivas de uma mesma 1D-DWT com rotação de 90^o, conforma demonstrado em SALOMON (2000), logo uma arquitetura em hardware para a implementação da 2D-DWT pode ter por base uma implementação em hardware da 1D-DWT.

Diversas arquiteturas para a implementação da 1D-DWT podem ser encontradas na literatura, como arquiteturas utilizando banco de filtros, como apresentado em MASUD (2004) e MOTRA (2003), arquiteturas utilizando o Algoritmo da Pirâmide Recursivo (RPA) descrito em VISHWANATH (1994) e implementado em FRIDMAN (1994), KNOWLES (1990) e SYED (1995) e arquiteturas utilizando o esquema *lifting*, como apresentado em DILLEN (2003) e LAKSHMINARAYANAN (2003).

Quanto à etapa de quantização, no padrão JPEG2000 sem perdas, o valor do coeficiente de quantização é sempre igual a um e no padrão com perdas, este coeficiente depende da qualidade requerida para a imagem reconstruída.

O codificador de entropia definido no padrão JPEG2000 é implementado a partir de uma variação do algoritmo QM *coder* (ACHARYA, 2005) que realiza a codificação aritmética dos coeficientes provenientes dos blocos de transformada wavelet e quantização.

Este trabalho propõe a implementação de um compressor de imagens seguindo o padrão JPEG2000 com perdas. Este compressor foi projetado em VHDL utilizando a ferramenta QUARTUS II (ALTERA, 2005-b) para ser inicialmente sintetizado para FPGAs da ALTERA (2005-a), e posteriormente ser prototipada para ASIC utilizando o pacote de ferramentas da CADENCE (CADENCE, 2005). Para isso os blocos internos do compressor foram projetados inicialmente segundo uma descrição comportamental para permitir que a ferramenta de síntese utilize os recursos internos dos FPGAs e posteriormente foram projetados segundo uma descrição estrutural para permitir a prototipação para ASIC.

Este trabalho é dividido nos seguintes capítulos:

No capítulo 2 são apresentadas as propriedades e elementos envolvidos na visualização de imagens, os principais padrões para compressão de imagens e as métricas de como medir a qualidade de uma imagem em uma compressão com perdas.

No capítulo 3 são apresentadas as principais transformadas utilizadas no processamento de imagens, dando destaque especial para a transformada Wavelet e o esquema de *lifting*.

No capítulo 4 são mostradas algumas arquiteturas encontradas na literatura para a implementação de uma DWT de uma dimensão.

No capítulo 5 apresenta a implementação do compressor de imagens proposto neste trabalho, bem como a implementação dos blocos constituintes do compressor de imagens: DWT de duas dimensões, bloco quantizador e codificador de entropia.

No capítulo 6 são apresentadas as conclusões do trabalho e as propostas para desenvolvimento futuro.

2 COMPRESSÃO DE IMAGENS

Genericamente, compressão de dados é um processo para converter um conjunto de dados de uma origem qualquer em uma representação reduzida e equivalente destes dados, de modo a:

- aumentar a capacidade de armazenamento de grandes quantidades de informações, independente do espaço disponível que pode ser usado para este fim;
- reduzir o tempo de transferência da informação, devido à redução da quantidade de dados a ser transmitida.

Estas duas idéias representam o ponto de partida para o desenvolvimento dos métodos de compressão de dados.

Todos os métodos para compressão de dados são baseados em um mesmo princípio, que é a remoção da redundância do conjunto de dados de entrada.

A menor quantidade de bits necessária para representar um conjunto de bits de entrada é chamada de entropia (TAUBMAN, 2002). A entropia de um símbolo a_i é definida como $-\boldsymbol{P}_i \log_2 \boldsymbol{P}_i$, onde P_i é a probabilidade de ocorrência do símbolo a_i em um conjunto de dados. A entropia de um conjunto de dados depende das probabilidades individuais de cada elemento individualmente. Assim:

$$E = \sum_{i=1}^{n} Pi \log 2 \, Pi$$

A eficiência de um compressor de dados pode ser medida pelo tamanho da representação resultante. Assim, quanto mais próxima esta representação estiver do valor da entropia, mais eficiente será o compressor.

Para proporcionar alta eficiência de compressão, cada tipo de dados (texto, imagem, códigos executáveis, etc...) que um arquivo pode conter, requer um método de compressão específico e cada método de compressão se adapta melhor a um determinado tipo de dados. De modo geral, os métodos de compressão de dados podem ser divididos em dois grupos, compressão sem perdas, quando os dados após descompressão são idênticos aos dados originais e compressão com perdas, quando os dados após descompressão são diferentes dos dados originais.

Para fins de compressão de imagens, uma imagem digital pode ser definida de modo simplificado como um arranjo bidimensional de *pixels*, dispostos em *m* linhas e *n* colunas, onde a relação *m* x *n* é chamada de resolução da imagem. Cada *pixel* é

amostrado com uma quantidade finita de bits, que define sua precisão (ex: 8-bits de precisão). Na compressão de imagens podem ser utilizados os métodos de compressão com ou sem perdas.

2.1 Amostragem do Sinal

O processo de amostragem da imagem é feito por uma varredura linear da esquerda para a direita e de cima para baixo de um conjunto de fotorreceptores que são sensibilizados pela reflexão ou emissão de fótons em um objeto real. A amplitude de saída de cada foto-receptor de imagem define o valor de cada *pixel*. A leitura dos *pixels* ocorre por amostragem do sinal dos fotorreceptores cadenciado pelo sinal de *clock* do sistema. Assim, a informação da imagem, que é organizada espacialmente, é convertida para uma representação da informação organizada serialmente. Esta representação serial pode ser armazenada linearmente em um arquivo, representando um mapa de bits (*bitmap*).

Porém a digitalização aproxima o valor real do *pixel* para um valor discreto que pode é representado por um conjunto de bits, por exemplo:

- 1 bit, onde cada *pixel* pode ser preto ou branco;
- n bits, onde cada *pixel* pode assumir 2^n tons ou cores (n pode assumir os valores 2, 4, 8, 12, 16, 24 e 32).

Devido à quantidade de bits para representar cada *pixel* ser um valor finito, o processo de amostragem necessariamente descarta uma grande quantidade de informação (PENNEBAKER, 1992). Logo, o processo de amostragem insere intrinsecamente erros na imagem digitalizada.

2.2 Sistema Visual Humano

Uma câmera eletrônica possui sua estrutura e operação similar ao olho. Ambos são baseados em dois componentes principais: um conjunto de lentes e um sensor de imagem. O conjunto de lentes captura uma porção da luminosidade emanada de um objeto e foca o sensor de imagens. Os fótons emanados do objeto incidem sobre a superfície do sensor de imagem, onde o padrão de luminosidade é transformado em um sinal de vídeo, eletrônico ou neural. A quantidade de fótons que incidem sobre o sensor de imagem pode ser regulada pela íris. Este ajuste é importante devido à faixa de intensidade luminosa do ambiente ser maior que a faixa de sensibilidade do sensor de imagens. Os fótons que penetram pela íris e passam pelas lentes incidem sobre o sensor de imagens.

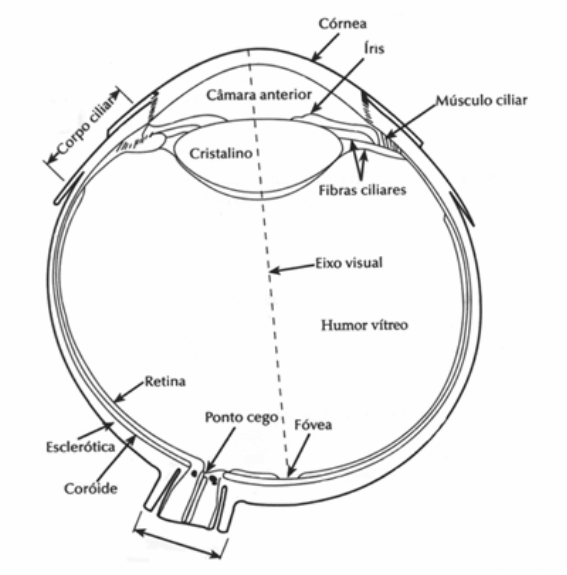

Figura 2.1: Diagrama do olho humano.
(WIKIPEDIA, 2005-a).

Conforme a Figura 2.1, que mostra a estrutura interna do olho humano, o sensor de imagens é chamado de retina. Na retina existem dois tipos de células de detecção de luminosidade no olho humano: os cones e os bastonetes. Os cones são células especializadas na detecção de cores, porém necessitam de uma razoável quantidade de luminosidade, enquanto os bastonetes, que são células especializadas na detecção de tons (claro e escuro), funcionam com uma quantidade mínima de luminosidade (próximo da escuridão absoluta).

Cada célula de detecção de luminosidade, seja cone ou bastonete, possui aproximadamente 3μm de diâmetro e a retina possui uma área de aproximadamente 9 cm^2. Isso significa que a retina tem aproximadamente 100 milhões de receptores. Diretamente no centro da retina existe uma pequena região formada principalmente de cones, chamada *fovea* (poço em Latim) responsável pela alta resolução do ponto central da visão. Apesar da *fovea* possuir uma área muito pequena, um movimento espasmódico chamado *saccades*, permite que este ponto faça uma varredura em toda região de informação pertinente.

O limite superior da capacidade de perceber detalhes em uma imagem é determinado pela sensibilidade do sistema visual a alterações na intensidade da imagem. De acordo com PENNEBAKER (1992), o pico de contraste na sensibilidade ocorre próximo a 5 ciclos/grau e a sensitividade cai a zero para freqüências acima de 100 ciclos/grau. Isto significa que a sensibilidade do olho é maior para os objetos com aproximadamente 0,2 cm a 1m de distância. Considerando na distância de 1m, podem-se distinguir objetos de até 0,01cm e neste caso o tamanho ideal de imagem fica em torno de 40 cm.

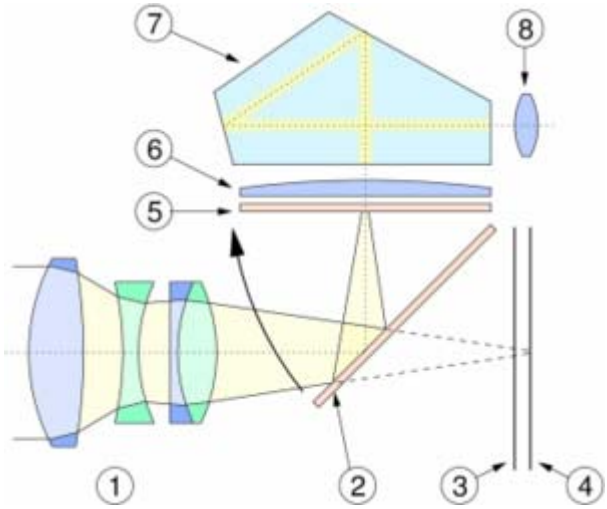

Figura 2.2: Diagrama de uma câmera fotográfica.
(WIKIPEDIA, 2005-c)

A Figura 2.2 mostra a estrutura interna de uma câmera fotográfica, onde mostra o percurso da luz passando através das múltiplas lentes (1), refletida por um conjunto com espelho (2), lentes para ajuste do foco (5 e 6) e um prisma (7) para ser observada no visor (8). Quando ocorre o disparo da foto, o espelho se move na direção da seta e o obturador (3) se abre permitindo que a luz atinja o filme fotográfico (4). Neste tipo de câmera, a resolução da foto depende principalmente da quantidade de pontos que o filme fotográfico permite.

Nas câmeras fotográficas digitais, o obturador e o filme fotográfico foram substituídos por um *charge-coupled device* (CCD), que é um dispositivo de estado sólido, formado por um arranjo de sensores que convertem a energia luminosa em energia elétrica, organizada na forma de um código binário. Cada sensor do arranjo converte uma região luminosa em um *pixel* e devido a isso insere dois tipos de erro na imagem resultante. Cada *pixel* possui um único valor de luminosidade, logo considerando o ponto central de cada *pixel* e que cada *pixel* apresenta uma área, ocorre um efeito de descontinuidade entre *pixels* adjacentes, além disso, cada *pixel* é representado por um conjunto limitado de bits, requerendo que a representação do sinal luminoso em binário seja arredondada.

Estes dois erros são minimizados com o aumento da resolução dos CCDs. Atualmente as câmeras digitais possuem CCDs com resolução de 8Mpixels e com o passar do tempo são produzidos CCDs com resolução cada vez maior. Porém apesar da tecnologia propor que em um futuro próximo possam existir sensores eletrônicos que alcancem a resolução do olho humano (aproximadamente 100 milhões de receptores), as células de detecção de luminosidade no olho humano ainda apresentam uma resposta linear (contínua) às variações de luminosidade, ao contrário dos receptores eletrônicos que apresentam resposta descontínua (dependente da quantidade de bits de resolução do sensor).

Segundo GONZALES (1992), o cérebro humano, quando realiza a interpretação das informações sensoriais, apresenta a característica de ignorar informações constantes, ou seja, ao olhar para uma cena já observado anteriormente, o cérebro ignora a maioria dos elementos, procurando apenas os elementos novos. Esta idéia pode ser expandida

para imagens individuais, de modo que uma imagem em tons contínuos não requer que todos os seus pontos sejam analisados individualmente, ocorrendo assim uma interpolação de forma inconsciente, proporcionando uma rápida interpretação da imagem. Esta interpolação pode ser explorada de modo que imagens com resoluções menores possam ser interpretadas sem grande prejuízo para a compreensão.

2.3 Classificação das imagens

Existem vários tipos de arquivos de computador, dentre estes existem vários padrões para o armazenamento de imagens (no Anexo são apresentados os padrões mais conhecidos). Independentemente destes padrões para armazenamento, as imagens podem ter uma classificação relativa ao processo de geração da própria imagem. De acordo com SALOMON (2000), as imagens quanto ao processo de geração podem se classificar em:

- Imagens *bi-level*
- Imagens *grayscale*
- Imagens em tons contínuos
- Imagens em tons descontínuos
- Imagens *cartoon-like*

As imagens *bi-level*, também conhecidas como monocromáticas, são as imagens em que cada *pixel* pode assumir somente dois valores, normalmente preto e branco. Cada *pixel* da imagem é representado por um bit.

Imagens em tons contínuos podem conter muitos tons de cores próximas e *pixels* adjacentes podem apresentar diferença de apenas uma unidade (tornando difícil a visualização da mudança de tom). Uma imagem pode conter áreas onde os tons variam continuamente permitindo a visualização da variação do tom. Este tipo de imagem é normalmente originário de uma imagem natural, adquirida por fotografia digital ou por digitalização de uma fotografia.

Nas imagens em *grayscale*, cada *pixel* pode assumir qualquer valor em uma escala de 2^n tons de cinza (ou outra cor), ou seja, cada *pixel* é representado por um conjunto de *n* bits. O valor de *n* normalmente é 4, 8, 12, 16 ou 24.

Imagem em tons descontínuos, também chamadas imagens gráficas ou sintéticas, são normalmente imagens artificiais, podendo ter muitas cores e tons, porém não possui o ruído e borrados característicos de imagens naturais. Nestes tipos de imagens, *pixels* adjacentes normalmente apresentam o mesmo valor, ou variam significantemente.

Imagem *cartoon-like* é aquela que apresenta grandes porções uniformes, onde os *pixels* de cada área apresentam o mesmo valor, porém áreas adjacentes podem possuir cores muito diferente.

2.4 Redundância em imagens

Como já foi explicitada anteriormente, a compressão de dados se baseia na remoção de redundâncias da imagem original. Esta redundância pode ocorrer de formas diferentes:

Redundância estatística: Quando existe um conjunto de valores pertencentes à faixa dinâmica da imagem original, que não são utilizados, ocorre uma situação de redundância estatística. Este tipo de redundância pode ser eliminado pela representação do conjunto de amostras originais por um conjunto de valores com número reduzido de bits.

Irrelevância visual: Ocorre quando a resolução da imagem excede os limites da acuidade visual humana, ou quando o dispositivo para a visualização da imagem possui capacidade de resolução menor que a imagem original. Neste caso a redução da resolução da imagem não acarreta nenhuma alteração aparente na visualização da imagem.

Irrelevância para aplicações específicas: Em imagens que possuem porções irrelevantes para aplicação específica da imagem, como imagens para diagnóstico médico, reconhecimento de alvos, etc..., pode ocorrer a redução significativa na resolução, ou até mesmo a eliminação destas porções da imagem sem causar nenhum tipo de perda na interpretação da imagem.

Irrelevância na resolução de cores da imagem: O sistema visual humano possui menor sensibilidade a cores do que a tons (TAUBMAN, 2002). Assim a transformação do padrão de representação de cores da imagem, do padrão RGB para o padrão YCbCr, permite tratar separadamente os componentes de cor da imagem. Estes componentes de cor (Cb e Cr) podem sofrer uma sub-amostragem, a assim uma redução significativa na representação da imagem com pouco impacto na compreensão da imagem.

Redundância espacial: De um modo geral, imagens em tons contínuos apresentam a característica de correlacionamento entre *pixels* vizinhos. Isso significa que o espectro de freqüência deste tipo de imagem apresenta coeficientes de grandes amplitudes nas baixas freqüências e coeficientes de baixa amplitude nas freqüências elevadas. Esta característica permite a exploração de transformadas matemáticas para transformar a representação da imagem do domínio espacial para o domínio das freqüências. De modo geral, o sistema visual humano é menos sensível a coeficientes de freqüências elevadas, assim estes coeficientes podem ser reduzidos, ou até mesmo eliminados, sem causar perda significativa na compreensão da imagem.

2.5 Métodos de Compressão sem perdas

Os métodos de compressão sem perdas somente eliminam as informações redundantes presentes na representação original, de modo a possibilitar a reconstrução da imagem original por um processo de descompressão. Os métodos de compressão sem perdas são aplicáveis a dados de textos, programas executáveis, imagens que requerem alta definição (como imagens médicas digitais), arquivos de banco de dados e outros arquivos que não admitem a perda de dados.

Os métodos de compressão sem perdas se baseiam em técnicas como codificação RLE, codificações estatísticas (como codificação *Huffman*, codificação aritmética, etc...) e codificações baseadas em dicionário.

O método *Run Length Encoding* (RLE) segue a premissa de que se um determinado símbolo ***d*** é repetido ***n*** vezes consecutivamente, todas as ***n*** ocorrências consecutivas podem ser substituídos por um par ***nd***. Neste método as cadeias de *pixels* contendo a mesma informação de cor são substituídas por um contador seguido de uma amostra do *pixel* codificado.

Os métodos estatísticos se baseiam na idéia de substituir por uma representação compacta os símbolos ou grupo de símbolos de maior probabilidade de ocorrência no conjunto de dados de entrada. A qualidade da compressão resultante deste método depende da qualidade do modelo estatístico que cada método utiliza.

Os métodos baseados em dicionário selecionam conjuntos de símbolos e codificam cada conjunto individualmente usando um dicionário. Um dicionário mantém as equivalências dos conjuntos de símbolos e permite que novas entradas ocorram. As principais vantagens destes métodos é a facilidade de programação (utilizando algoritmos de procura e substituição) e a simplicidade de decodificação.

2.6 Métodos de Compressão com perdas

Os métodos de compressão com perdas somente podem ser utilizados onde perda de dados não gera perda significativa de informação. Neste conceito se enquadram arquivos de imagens, áudio e vídeo. Este tipo de compressão é utilizado considerando as seguintes razões (TAUBMAN, 2002):

- O sistema visual (ou auditivo) humano pode tolerar alguma perda de informação, sem interferir com a compreensão da imagem (sinal de áudio);
- O processo de digitalização (amostragem) da imagem (do som) introduz erro entre a imagem digitalizada e a imagem real;
- Compressão sem perdas é normalmente incapaz de proporcionar a taxa de compressão requerida pelos sistemas de armazenamento e transmissão.

Uma boa imagem reconstruída deve se assemelhar à imagem original, de modo que para medir a qualidade dos métodos de compressão com perdas e reconstrução das imagens, a padronização de algumas métricas se faz necessário.

Uma boa medida deve produzir um valor adimensional que não seja sensível a pequenas diferenças entre a imagem reconstruída e a imagem original.

A medida de distorção mais comumente utilizada é *mean squared error* (MSE) (TAUBMAN, 2002), definida por:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=-\infty}^{+\infty}(a_i - \hat{a}\ )^2$$

Onde:

n define a quantidade de amostras que a imagem possui.

a_i representa as amostras da imagem original.

$\hat{a}_i$ representa as amostras da imagem reconstruída.

Para compressão de imagens, uma medida baseada no MSE é o *peak signal to noise ratio* (PSNR). Altos valores de PSNR significam que a imagem reconstruída se aproxima da imagem original.

O cálculo de PSNR é dado por:

$$PSNR = \frac{20 \log_{10} \max|P_i|}{MSE}$$

$\max|P_i|$ define o maior valor que representa o tom de um *pixel*. Como os *pixels* P_i podem possuir valores positivos e negativos, o cálculo utiliza seu valor em módulo.

Os resultados de PSNR são utilizados somente para comparar a performance de diferentes métodos de compressão com perdas ou o efeito de diferentes valores paramétricos na performance do algoritmo. PSNR é um valor adimensional, desde que ambos numerador e denominador são valores de *pixels*, porém devido ao uso do logaritmo, é dito que PSNR é expresso em decibel (dB). Boas imagens reconstruídas possuem, tipicamente o valor de PSNR iguais ou maiores que 30dB (TAUBMAN, 2002).

Outro método eficiente de mensurar a qualidade de uma imagem reconstruída é gerar uma imagem diferença entre a imagem reconstruída e a imagem original e realizar um julgamento visual. Porém neste caso espera-se que os valores sejam pequenos e próximos a um extremo (preto ou branco) se tornando de difícil julgamento. Para melhorar o julgamento é permitido ampliar a diferença e deslocar os valores para a faixa de tons médios, conforme cálculo abaixo:

$D_i = a(a_i - \hat{a}_i) + b$

onde:

- a é o parâmetro de magnificação (tipicamente, é um valor baixo como 2).
- b é o valor médio dos *pixels* (para deslocar a imagem para um cinza médio).

2.7 Padrões para Compressão de imagens

Em imagens em tons contínuos, a informação amostrada apresenta um alto grau de correlação entre os valores de *pixels* vizinhos (*pixel correlation*).

As técnicas tradicionais para compressão de imagens (conjunto de dados que possuem alta correlação) foram idealizadas para explorar a redundância estatística presentes em imagens reais (aqui definidas como imagens em tons contínuos).

Os padrões que apresentam as maiores taxas de compressão são os padrões JPEG e JPEG2000.

2.7.1 Padrão JPEG

O padrão JPEG surgiu na década de 80 como uma tentativa para padronizar a compressão de imagens. Para isso membros do ITU, ISO e IEC se uniram para formar o *Joint Photographic Expert Group*. Este padrão prevê quatro modos de operação, como apresentado em ITU-T (1992), o modo seqüencial baseado na DCT, o modo progressivo baseado na DCT, o modo seqüencial sem perdas e o modo hierárquico.

Todo processo de compressão que se basear na utilização de uma DCT será considerado com perdas, mas estas perdas não ocorrem na etapa da DCT e sim na etapa de quantização, que sempre ocorre após a DCT. No compressor de imagens coloridas, existe outro ponto de perda de informação, que é o bloco de sub-amostragem (*downsampling*), que reduz a taxa de amostragem dos sinais de crominância.

Uma unidade de dados é a menor unidade lógica dos dados da unidade original que pode ser processada em determinado modo de operação (PENNEBAKER, 1992) e é de 8x8 *pixels* sempre que for usada a DCT (ITU-T, 1992). Em imagens coloridas cada conjunto de amostras é composto por três unidades de dados, uma para o sinal de luminância e duas para os sinais de crominância, enquanto imagens em tons de cinza possuem apenas uma unidade de dados.

Um *frame* contém um ou mais *scans*, que consiste em uma varredura dos *pixels* da imagem, nos sentidos da esquerda para a direita e do topo para a base, a quantidade de *scans* depende do modo de operação utilizado. Em processos seqüenciais um *scan* contém todos os *pixels* da imagem e em processos progressivos contém parte da codificação de todas as unidades de dados de um ou mais componentes de imagem.

No modo **seqüencial**, a imagem é codificada, processando totalmente uma unidade de dados, ou um grupo de unidade de dados, de cada vez, da esquerda para a direita, linha por linha, do topo até a base, ou seja, realizando a codificação um único *scan*. Cada *pixel* pode ter 8 bits ou 12 bits, ou seja, 256 ou 4096 tons. A compressão pode ser aritmética ou pelo algoritmo de compressão de *Huffman*. Cada unidade de dados tipicamente é composta por uma matriz de 8x8 *pixels*. Este formato possui as seguintes definições (ITU-T, 1992):

- processo baseado na DCT;

- cada *pixel* é representado por 8 bits (256 tons) ou 12 bits (4096 tons), independentemente se a imagem for colorida ou não;

- utiliza o modo seqüencial;

- a codificação realizada pelo algoritmo de compressão de *Huffman* ou codificação aritmética, possuindo 4 tabelas de codificação AC e 4 tabelas de codificação DC.

Existe uma forma restrita particular do modo Seqüencial, que é chamado de modo ***baseline***. Este formato possui as características mínimas necessárias para qualquer sistema de compressão baseado em uma DCT, e quaisquer processos baseados em DCT que requeiram mais que estes mínimos necessários são conhecidos por processos estendidos baseados em DCT. Este formato possui as seguintes definições (ITU-T, 1992):

- processo baseado na DCT;

- cada *pixel* é representado por 8 bits (256 tons), independentemente se a imagem é colorida ou não;

- utiliza o modo seqüencial;

- a codificação realizada pelo algoritmo de compressão de *Huffman*, possuindo 2 tabelas de codificação AC e 2 tabelas de codificação DC.

Existe ainda um modo com perdas chamado *baseline*, que é um submodo do modo seqüencial, por apresentar grande simplicidade de implementação, devido a possuir os requisitos mínimos necessários para qualquer sistema de compressão baseado em uma DCT. A Figura 2.3 apresenta a estrutura básica de um compressor JPEG no modo *baseline*.

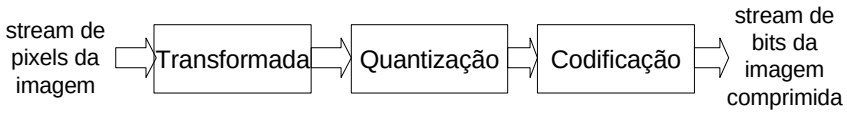

Figura 2.3: Compressor padrão JPEG baseline.

No modo **progressivo**, a imagem é codificada em múltiplos *scans*, onde o *scan* inicial cria uma versão com um reduzido número de detalhes, e os *scans* seguintes vão refinando a imagem, até que ela fique completa. Assim como no modo seqüencial cada *pixel* pode ter 8 bits ou 12 bits, e a compressão pode ser aritmética ou por *Huffman*. Por utilizar a DCT cada unidade de dados tipicamente é composta por uma matriz de 8x8 *pixels*.

A cada bloco transformado pela FDCT e quantizado, seus coeficientes são armazenados em um *buffer*. Os coeficientes da DCT são parcialmente codificados a cada *scan*.

Este formato possui as seguintes definições (ITU-T, 1992):

- processo baseado na DCT;

- cada *pixel* é representado por 8 bits (256 tons) ou 12 bits (4096 tons), independentemente se a imagem é colorida ou não;

- utiliza o modo progressivo;

- a codificação realizada pelo algoritmo de compressão de *Huffman* ou codificação aritmética, possuindo 4 tabelas de codificação AC e 4 tabelas de codificação DC.

No modo **sem perdas** não é utilizada a DCT no processo de codificação, e sim um preditor, porém este modo é pouco utilizado porque atinge taxas de compressão menores que o de outros formatos sem perdas já desenvolvidos, como o GIF e o PNG (PENNEBAKER, 1992).

Este formato possui as seguintes definições (ITU-T, 1992):

- processo preditivo não baseado na DCT;

- cada amostra da imagem origem deve ter de 2 a 16 bits;

- utiliza o modo seqüencial;

- a codificação realizada pelo algoritmo de compressão de *Huffman* ou codificação aritmética, possuindo 4 tabelas de codificação DC.

No modo **hierárquico** a imagem é codificada como uma seqüência de *frames* (campos), de modo a permitir um estágio final sem perdas. Devido à sua complexidade elevada, este método é pouco utilizado.

Este formato possui as seguintes definições (ITU-T, 1992):

- o processo pode ser baseado na DCT ou usar algoritmo preditivo;

- cada *pixel* é representado por 8 bits (256 tons) ou 12 bits (4096 tons), independentemente se a imagem é colorida ou não;

- pode utilizar o modo seqüencial ou progressivo;

- a codificação realizada pelo algoritmo de compressão de *Huffman* ou codificação aritmética, possuindo 4 tabelas de codificação DC e 4 tabelas de codificação AC, ou apenas 4 tabelas de codificação DC.

A utilização de uma etapa de quantização para reduzir a quantidade de bits de informação é realizada em todos os métodos que utilizam a DCT como algoritmo para realizar o decorrelacionamento entre os dados. Este processo é irreversível e a qualidade da imagem reconstruída é fortemente dependente dele. Com exceção do método de compressão sem perdas, todos os outros métodos permitem o uso da DCT, logo este método é o único que não permite o processo de quantização.

A última etapa de processamento dos dados é a codificação de entropia. Esta etapa não apresenta perda de informação, assim pode ser usado tanto nos métodos de compressão com perdas como os métodos de compressão sem perdas. Internamente no compressor de entropia dois algoritmos básicos são utilizados, a codificação RLE para os coeficientes DC (coeficiente da posição 0,0 da matriz 8x8 pós-transformada e pós-quantização) e a codificação *Huffman* para os coeficientes AC (outras 63 posições da matriz 8x8).

2.7.2 Padrão JPEG2000

A contínua expansão da utilização de imagens em aplicações como multimídia e Internet, necessitou de uma evolução nas tecnologias utilizadas no processamento de imagens. Assim em março de 1997 foi iniciado o desenvolvimento de um novo padrão de compressão de imagens que permitisse maior desempenho e flexibilidade.

Apesar de ter sido definida em 2000, devido à necessidade de mais informações complementares, o projeto, especificação e teste do novo padrão JPEG2000 somente foram concluídos em 2001. Algumas das mais importantes características do padrão JPEG2000 (ITU-T, 2000) são:

- Superior performance a baixas taxas de bits: o padrão oferece performance superior ao padrão atual a baixas taxas de bits, permitindo menos de 0.25 bits por *pixel* para imagens em tons de cinza detalhadas.

- Compressão de imagens em tons contínuos e imagens *bi-level*: O sistema pode comprimir imagens com várias faixas dinâmicas (de 1 bit a 16 bits) para cada componente de cor.
- Compressão com perdas e sem perdas: permite que seja possível que a ocorrência de reconstrução da imagem sem perdas.
- Transmissão progressiva por qualidade e resolução dos *pixels*: permite que a reconstrução da imagem ocorra por qualidade ou por resolução.
- Codificação por regiões de interesse: permite que algumas partes da imagem (de localização pré-definidas) sejam comprimidas com menor distorção que o resto da imagem.
- Robustez a erros de bits: minimização de erros na transmissão e posterior reconstrução da imagem.
- Arquitetura aberta: permite que o sistema seja otimizado para diferentes tipos de imagens e aplicações.
- Proteção de imagem: permite a utilização de métodos de proteção de imagem como *watermark*, *labeling*, *stamping fingerprinting*, *encryption*, *scrambling*, etc.

A estrutura básica do compressor JPEG2000 segue a estrutura básica introduzida na compressão *baseline* JPEG.

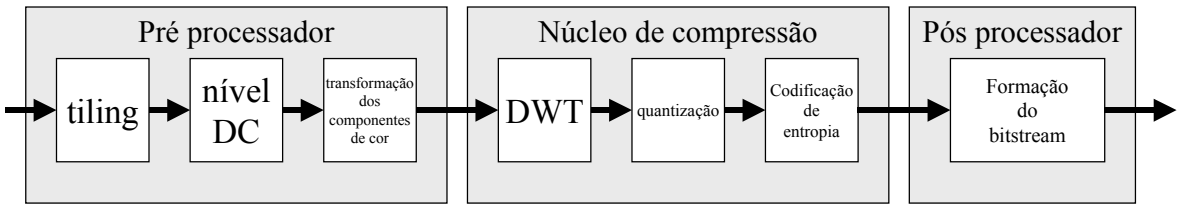

Figura 2.4: Diagrama geral do compressor padrão JPEG2000.

Conforme a Figura 2.4, o compressor JPEG2000 se divide em três blocos básicos, pré-processador, núcleo de compressão e pós processador.

O pré-processador é dividido em três partes:

- Etapa de *tiling*;
- Etapa de deslocamento do nível DC;
- Etapa de transformação de cores.

A etapa de *tiling* particiona a imagem em blocos de *pixels* dispostos de forma retangular não sobreposta, conhecidos como *tiles*. O tamanho do *tile* é arbitrário e pode corresponde a toda a imagem ou apenas um *pixel*. O *tile* é a unidade básica para a compressão onde todas as operações ocorrem, independentemente de *tiles* adjacentes.

A etapa de deslocamento do nível DC é similar ao deslocamento do nível DC no padrão JPEG, os valores de cada componente de cor são deslocados em 2^{B-1}, onde cada *pixel* possui B bits de cor. O valor dos *pixels* de uma imagem normalmente são valores

positivos, nestes casos, deslocar o nível DC representa permitir a computação utilizando valores positivos e negativos. Além disso, imagens não apresentam somente valores próximos aos extremos, assim deslocar o nível de DC dos componentes de uma imagem, representa colocar o valor médio dos *pixels* próximo ao valor zero.

A etapa de transformação de cores realiza a decorrelação entre os componentes de cor RGB. Esta decorrelação pode ser feita de duas maneiras:

- Transformação de cores irreversível (ICT);
- Transformação de cores reversível (RCT).

A transformação de cores irreversível apresenta melhor decorrelação que a transformação de cores reversível, porém somente pode ser utilizado no método de compressão com perdas.

A ICT transforma do padrão RGB para o padrão YCbCr e a RCT transforma do padrão RGB para o padrão YUV.

O padrão JPEG2000 define que a ICT é definida pela seguinte matriz:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.16875 & -0.33126 & 0.500 \\ 0.500 & -0.41869 & -0.08131 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

e a RCT é definida da seguinte maneira:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.25 & 0.5 & 0.25 \\ 1 & -1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Esta separação entre os componentes de cor é necessária, pois o núcleo de compressão codifica cada componente YCbCr ou YUV independentemente.

O núcleo de compressão é dividido em três partes:

- Transformada *Wavelet* Discreta (DWT);
- Quantização;
- Codificação de Entropia.

A Transformada *Wavelet* Discreta é utilizada como transformação linear para reduzia a correlação entre os *pixels* do componente da imagem processado. A DWT foi escolhida como transformada deste, devido pelos seguintes motivos:

- A representação da imagem por multiresolução proporciona um mecanismo simples de promover a escalabilidade espacial e por qualidade (SNR) sem sacrificar a eficiência da compressão. Este tipo de representação é uma propriedade inerente das transformada *wavelet*.

- Para imagens de grandes dimensões, o número de níveis de decomposição pode ser aumentado, permitindo que altos ganhos de compressão possam ser alcançados pela exploração da correlação entre os *pixels* de grandes porções da imagem.
- A DWT com coeficientes inteiros pode ser utilizada para compressão sem perdas, ao contrário da DCT que não pode ser utilizada em compressão sem perdas por ter sua matriz de transformação aproximada.

Uma das principais alterações observadas com a utilização do padrão JPEG2000 é o tipo de transformada utilizada. Na compressão com perdas, o padrão JPEG original usa a DCT para transformar porções de 8x8 *pixels* da imagem (ITU-T, 1992). No padrão JPEG2000 recomenda dois tipos de banco de filtros: para compressão com perdas e sem perdas.

A transformada irreversível sugerida no padrão (utilizada na compressão com perdas) requer banco de filtros com os coeficientes em ponto flutuante, definidos em Daubechies 9tap/7tap, enquanto a transformada reversível (utilizada na compressão sem perdas) utiliza banco de filtros com os coeficientes inteiros, definidos em Daubechies 5tap/3tap. A operação da transformada, independentemente se for reversível ou irreversível, ocorre para todas as linhas e após para todas as colunas. Para cada linha/coluna computada ocorre a extensão simétrica periódica dos coeficientes das bordas, de modo a não permitir a ocorrência de *alias* na computação, gerando a mesma quantidade de coeficientes após a computação de cada linha/coluna.

No padrão JPEG2000, a etapa de quantização consiste em uma divisão entre cada coeficiente proveniente da etapa da transformada e um valor constante, chamado coeficiente de quantização. Cada sub-banda pode ter um coeficiente de quantização diferente e seu valor afeta diretamente a qualidade da imagem reconstruída. No padrão sem perdas, o coeficiente de quantização é sempre igual a um.

Após a etapa de quantização, os coeficientes que estavam originalmente divididos em *tiles*, ficaram divididos em sub-bandas (SALOMON, 2000). Cada sub-banda é dividida em grades retangulares chamadas *precincts*. Cada *precinct* tem suas dimensões divididas em potências de 2. A primeira *precinct* é sempre posicionada na posição esquerda superior do *array* resultante da etapa de quantização. Cada *precinct* por sua vez é dividido em *code-blocks*, onde cada *code-block* possui 2^x de largura por 2^y de altura e $x \geq 2$, $y \leq 10$ e $x+y \leq 12$. Em outras palavras, um *code-block* pode ter no máximo 4096 coeficientes. A diferença entre *tiles*, *precincts* e *code-blocks* pode ser definida, respectivamente, como particionamento grosso, médio e fino de uma imagem. Particionar a imagem em pequenas porções permite a criação de implementações eficientes no que diz respeito à memória, que permitam o acesso facilitado a qualquer ponto da imagem comprimida. Os *code-blocks* são separados em planos de bits e estes planos de bits são codificados através do codificador de entropia.

O codificador de entropia opera a partir do plano dos bits mais significativos (MSB) para o plano dos bits menos significativos (LSB). Um contexto é definido para cada bit de acordo com a significância de seus oito bits vizinhos (utilizando o algoritmo *MQ-Coder*) e a probabilidade de cada bit é estimada a partir deste valor de contexto.

No bloco pós processador, todos os bits codificados no núcleo de compressão, oriundos de um *code-block* são empacotados em um reduzido *bitstream*, ao qual é

acrescentado um cabeçalho contendo todas as informações necessárias para o processo de descompactação. Os conjuntos de bits sob esta definição são chamados de *packets*. Os *packets* são agrupados em *layers* assim como os *code-blocks* são agrupados em *precincts*.

O padrão JPEG 2000 permite que os *packets* sejam agrupados de tal maneira a permitir que a transmissão e/ou descompressão ocorra de forma progressiva por resolução, qualidade, localização espacial ou por componente.

Outra importante característica incorporada ao padrão JPEG 2000 é a possibilidade de definir regiões de interesse. Como uma imagem pode ser dividida em vários *tiles*, e cada *tile* pode ser comprimido individualmente independente das características dos *tiles* vizinhos, logo é possível definir grau de quantização diferente para cada um dos tiles. Deste modo, pode ser definido baixo nível de perda nas porções da imagem que possuam alto grau de interesse e alto nível de perda nas porções da imagem que possuam baixo grau de interesse.

3 ELEMENTOS PARA A COMPRESSÃO DE IMAGENS SEGUNDO O PADRÃO JPEG2000

Conforme apresentado na seção 2.7.2, o núcleo de compressão do padrão JPEG2000 apresenta três operações básicas:

- etapa de transformada, para realizar o decorrelacionamento dos *pixels*;
- etapa de quantização, para reduzir a quantidade de informação, de modo a permitir alta taxa de compactação;
- etapa de codificação de entropia, para reduzir a quantidade de bits que representa o conjunto de dados resultante do processo de quantização.

3.1 Transformadas aplicadas no processamento de imagens

Segundo MALLAT (1998), somente as ocorrências de variações nas percepções são consideradas informações significativas para os sentidos humanos, independente da quantidade e diversidade destas percepções. Isto significa que um conjunto de percepções estáticas apresenta inicialmente uma grande quantidade de informação, porém se estas informações não variam com o passar do tempo, se tornam não significativas.

A ocorrência de variações em uma representação estática significa a introdução de novas informações em um conjunto de dados estáticos, logo estas variações têm grande significância. Em um conjunto de informações amostradas, estas variações podem ser traduzidas na forma de descontinuidades. Especificamente em imagens amostradas, estas descontinuidades são chamadas de bordas, significando que a amplitude relativa entre dois *pixels* adjacentes varia significativamente.

Um conjunto de dados amostrado é definido no domínio temporal caso cada representação do sinal tenha sido amostrado em um ciclo de uma unidade de tempo padronizada, ou é definido no domínio espacial caso várias amostras tenham sido obtidas em um mesmo ciclo da uma unidade de tempo padrão.

Devido a esta característica de posicionamento espacial (ou temporal) do conjunto de informações amostradas, as transformadas matemáticas são largamente utilizadas para o processamento de imagens, proporcionando um processamento simplificado. Assim para a solução de um problema, genericamente podemos percorrer dois caminhos:

i) Realizar uma análise complexa do problema;

ii) Transformar o domínio do problema, realizar uma análise simples do problema, transformar o problema para o domínio original.

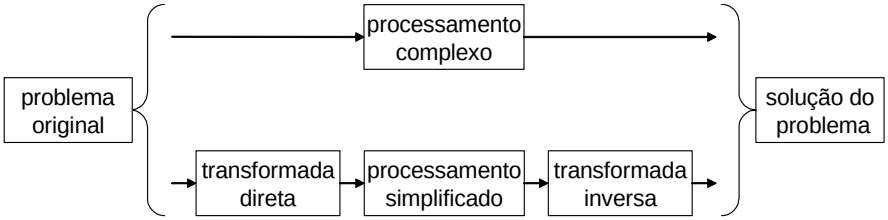

Figura 3.1: Fluxograma de análise e resolução de problemas.

Utilizando uma transformada podemos trocar o domínio de uma área de aplicação, como Sistemas Lineares, Irradiação de Antenas, Ótica, Processos randômicos, Probabilidade, Física Quântica, Equações diferenciais, etc... e assim permitir uma análise simplificada.

As principais transformadas utilizadas no processamento de imagens são:

- Transformada de Fourier;
- Transformada Karhunen-Loeve
- Transformada do seno
- Transformada do cosseno
- Transformada *wavelet*

Segundo MALLAT (1998), apesar da grande quantidade e diversidade de percepções que temos, no mundo dos sentidos as variações nas percepções é que são significativas. As variações observadas nos domínios espacial e temporal são representadas por componentes de freqüências no domínio das freqüências, assim a análise de sinais amostrados a partir de informações físicas pode ser feita através de procedimentos como a transformada de Fourier.

A transformada de Fourier é uma poderosa ferramenta para manipulação de dados amostrados e contínuos, pois permite examinar o relacionamento de uma determinada função amostrada, sob diferentes pontos de vista. Ela pode ser utilizada principalmente em processos que usem filtragem, segmentação, reconhecimento de padrões, descrição, compressão e reconstrução de sinais amostrados, de forma a permitir, normalmente, uma análise simplificada de um problema.

A transformada de Fourier, entretanto, possui um inconveniente: a representação no domínio das freqüências não especifica onde estão localizados, no domínio do tempo, os pontos de altas e baixas freqüências (SALOMON, 2000). Além disso, o princípio da incerteza de Heisenberg define que não é possível que a cobertura de energia e a transformada de Fourier de uma mesma função sejam infinitamente pequenas, simultaneamente. Com isto, as funções amostradas são analisadas por

decomposição do sinal a partir de átomos tempo-freqüência, que são formas de onda que representam a menor cobertura em um plano tempo-freqüência.

Com esta definição de átomos tempo-freqüência, a análise da função pode ser feita através de uma transformada de Fourier janelada. Porém este procedimento gera uma representação infinita, o que é uma desvantagem significativa. Com isso, é requerida a utilização de uma transformada com características semelhantes, porém sem esta desvantagem.

3.1.1 Transformada Wavelet

Nos últimos anos ocorreu um grande interesse em Transformadas *Wavelet*, principalmente após os estudos de MALLAT (1998) e sua posterior aplicação em padrões internacionais, como o JPEG2000 (ITU-T, 2000) e o MPEG-4. Porém de acordo com MALLAT (1998) a utilização de *wavelets* para o processamento de sinais não se baseou em uma idéia nova, mas em conceitos que já existiam sob várias formas em diferentes campos de estudo, principalmente na matemática, na física e na engenharia, onde os pesquisadores estavam desenvolvendo independentemente idéias semelhantes.

Assim como a Transformada de Fourier Janelada, a Transformada *Wavelet* pode medir variações tempo-freqüência de componentes espectrais, mas com uma resolução tempo-freqüência diferente.

Algumas das mais importantes considerações a respeito das *wavelet* são:

Casamento tempo-freqüência: Informações importantes surgem da análise simultânea das propriedades de tempo e freqüência do sinal. Isto motiva a decomposição em porções elementares que estão concentradas no tempo e/ou em freqüência. A energia de uma função e sua transformada de Fourier não podem ser reduzidas a porções elementares simultaneamente (princípio da incerteza de Heisenberg), com isto podemos compreender a limitação da flexibilidade da transformada de Fourier na análise tempo e freqüência.

Escala para aproximação (zoom): *wavelet* são formas de onda escalar que medem variações de sinal. Com isso procedimentos de aproximação permitem a observação de estruturas do sinal, como as singularidades.

Mais e mais bases: Muitas bases ortogonais podem ser implementadas com algoritmos rápidos de computação. A utilização de banco de filtros e bases *wavelet* tem gerado uma procura por bases, com isso novas famílias de bases são criadas regularmente.

Representações esparsas: Uma base ortogonal é útil se puder definir uma representação onde sinais são bem aproximados com poucos coeficientes diferentes de zero.

Por definição, uma *wavelet* ψ é uma função com média zero:

$$\int_{-\infty}^{+\infty} \psi(t)dt = 0$$

e média quadrática finita:

$$\int_{-\infty}^{+\infty} |\psi(t)|^2 dt < +\infty$$

que é dilatada com um parâmetro de escala *s* e transladada no seu domínio por *u*:

$$\psi_{u,s}(t) = \frac{1}{\sqrt{s}} \psi(\frac{t-u}{s})$$

Assim a Transformada *Wavelet* de uma f(t) arbitrária na escala *s* e posição *u* é computada correlacionando a função f com um átomo *wavelet*.

$$Wf(u,s) = \int_{-\infty}^{+\infty} f(t) \frac{1}{\sqrt{s}} \psi(\frac{t-u}{s}) \, dt$$

$$Wf(u,s) = \int_{-\infty}^{+\infty} f(t) \ \psi_{u,s}(t) \, dt$$

Um átomo wavelet $\phi_\gamma(t)$ é uma dilatação por *s* e uma translação por *u* de uma *wavelet* mãe ψ:

$$\phi_\gamma(t) = \psi_{u,s}(t) = \frac{1}{\sqrt{s}} \psi(\frac{t-u}{s})$$

Um átomo de Fourier janelado é construído com uma janela *g* transladada por *u* e modulado por uma freqüência ξ:

$$\phi_\gamma(t) = g_{u,\xi}(t) = e^{i\xi t} g(t-u)$$

Na análise tempo-freqüência, a Transformada *Wavelet* Wf(u,s) é comparável à transformada de Fourier janelada Sf(u,ξ) para medidas dos componentes espectrais de tempo-freqüência, porém com resoluções diferentes.

$$Sf(u,\xi) = \int_{-\infty}^{+\infty} f(t) \, g_{u,\xi} \, dt$$

A escalabilidade da Transformada *Wavelet* pode ser usada para a detecção e caracterização de singularidades. Singularidade e estruturas irregulares carregam informações essenciais do sinal. Um coeficiente *wavelet* Wf(u, s) mede a variação de *f* em uma vizinhança de *u* com tamanho de vizinhança proporcional a *s*. Em imagens, os contornos são indicados pelos coeficientes *wavelet* de grande amplitude.

A transformada de Fourier Contínua janelada Sf(u,ξ) e a Transformada *Wavelet* Wf(u,s) são representações bidimensionais de um sinal unidimensional f. Isto indica a existência de redundâncias entre os coeficientes resultantes da transformada. Estas redundâncias podem ser reduzidas ou mesmo removidas por meio de uma sub-amostragem que consistem na remoção de um a cada dois coeficientes gerados. O resultado sub-amostrado destas transformadas pode definir uma representação completa

do sinal se o mesmo puder ser reconstruído por uma combinação linear de famílias discretas de átomos de Fourier janelados ou átomos *wavelet*.

A teoria de *frames* analisa a completude, estabilidade e redundâncias da representação de um sinal discreto linear. Um *frame* é uma família de vetores $\{\phi_n\}_{n\in\mathbb{Z}}$ que são caracterizados por qualquer sinal f de seu produto interno $\{<f,\phi_n>\}_{n\in\mathbb{Z}}$.

A Transformada de Fourier janelada Discreta e a Transformada *Wavelet* Discreta geram representações do sinal que seguem este o formalismo da teoria de *frames*.

Uma Transformada *Wavelet* Rápida é calculada com algoritmos de banco de filtros e são utilizadas principalmente em imagens para discriminação de texturas e detecção de bordas.

Pode-se construir wavelets ψ tais que a família dilatada e transladada de:

$$\left\{ \psi_{j,n}(t) = \frac{1}{\sqrt{2}^{j}} \psi\left(\frac{t - 2^{j} n}{2^{j}}\right) \right\}_{(j,n)\in\mathbb{Z}^2}$$

Onde $\psi_{j,n}(t)$ é uma base ortonormal em L^2 ($\mathbb{R}$). *Wavelet* ortogonais dilatadas por 2^j carregam variações do sinal na resolução 2^{-j}.

A aproximação em multiresolução é totalmente caracterizada pela função de escala ϕ que gera uma base ortogonal para cada espaço V_j. A função escala é especificada por um filtro discreto chamado *filtro espelho conjugado*.

A propriedade de causalidade de multiresolução que impõe:

$\forall j \in \mathbb{Z}$, $V_{j+1} \subset V_j$

Em particular $2^{-1/2}\phi(t/2) \in V_1 \subset V_0$. Se $\{\phi(t-n)\}_{n\in\mathbb{Z}}$ é uma base ortogonal de V0, podemos decompor:

$$\frac{1}{\sqrt{2}} \phi\left(\frac{t}{2}\right) = \sum_{n=-\infty}^{+\infty} h[n]\, \phi(t-n),$$

utilizando:

$$h[n] = < \frac{1}{\sqrt{2}} \phi\left(\frac{t}{2}\right) , \phi(t-n) > .$$

A equação de escala define uma dilatação de ϕ por 2. A seqüência h[n] será interpretado como um filtro discreto.

Muitas aplicações de bases *wavelet* exploram a habilidade de aproximar eficientemente classes particulares de funções com poucos coeficientes *wavelet* não zero. A definição da ψ precisa ser otimizada para produzir o máximo número de valores próximos de zero. Isto depende grandemente da regularidade da função f, o número de momentos nulos de ψ e o tamanho de seu suporte.

Para uma função ψ com p momentos nulos temos:

$$\int_{-\infty}^{+\infty} t^k \psi(t)\, dt = 0 \qquad \text{para } 0 \le k < p$$

significando que ψ é ortogonal a qualquer polinômio de grau p-1.

Se f tem uma singularidade isolada em t0 e se t0 está dentro do suporte de

$$\psi_{j,n}(t) = \frac{2 - j}{2\psi(2\text{-}j(t\text{-}n))},$$

então <f, $\psi_{j,n}$ > poderá ter uma grande amplitude. Para minimizar a quantidade de coeficientes de grandes amplitudes é preciso reduzir o tamanho do suporte de ψ. Para minimizar o tamanho do suporte, é preciso sintetizar filtros espelho conjugado com a menor quantidade de coeficientes não-zero quanto possível.

O tamanho do suporte e o número de momentos nulos são, a princípio, independentes, entretanto, se ψ tem p momentos nulos então seu suporte deve ter ao menos o tamanho 2p-1. Assim, quando escolhemos uma wavelet em particular é necessário fazer um trade-off entre o número de momentos nulos e o tamanho de seu suporte.

A regularidade de ψ tem grande influência na ocorrência de erros na quantização dos coeficientes wavelet. Quando um sinal for reconstruído a partir dos coeficientes wavelets, um erro adicionado ao coeficiente (inserido pela quantização) irá gerar um erro no sinal reconstruído, onde a regularidade da *wavelet* ψ é proporcional a regularidade dos erros gerados. Em aplicações de codificação de imagens, erros regulares são menos visíveis que erros irregulares, devido ao seu menor nível de energia. Assim, as imagens reconstruídas terão melhor qualidade caso sejam utilizadas *wavelets* com maior regularidade.

Em uma base ortogonal, os componentes de decomposição são computados com um algoritmo rápido de banco de filtros.

Uma Transformada Wavelet rápida decompõe sucessivamente cada aproximação Pvjf em uma aproximação Pvj+1f e coeficientes wavelet presentes em Pwj+1f. Para a reconstrução dos coeficientes wavelet originais cada Pvjf é reconstruído a partir de Pvj+1f e Pwj+1f.

De acordo com MALLAT (1998), a decomposição *wavelet* segue as seguintes equações:

$$a_{j+1}[p] = \sum_{n=-\infty}^{+\infty} h[n-2p]\, a_j[n] = a_j * h[2p]$$

$$d_{j+1}[p] = \sum_{n=-\infty}^{+\infty} g[n-2p]\, a_j[n] = a_j * g[2p]$$

e a sub-amostragem:

$$x[n] = \begin{cases} x[p] & \text{para } n = 2p \\ 0 & \text{para } n = 2p+1 \end{cases}$$

Assim, a reconstrução descrita por MALLAT segue a seguinte equação:

$$\tilde{a}_j[p] = \sum_{n=-\infty}^{+\infty} h[p-2n]\, a_{j+1}[n] + \sum_{n=-\infty}^{+\infty} g[p-2n]\, d_{j+1}[n] = a_{j+1} * h[p] + d_{j+1} * g[p]$$

Na reconstrução é necessária a inserção de zeros para retirar o *alias* inserido na decomposição, como segue:

$$x[n] = \begin{cases} x[p] & \text{se } n = 2p \\ 0 & \text{se } n = 2p+1 \end{cases}$$

A Figura 3.2 mostra o processo de decomposição Wavelet e posterior reconstrução do sinal original. Na etapa de decomposição temos a convolução entre o conjunto de dados de entrada $a_j[n]$ e o conjunto de coeficientes da função escala g sub-amostrados por um fator 2, gerando os coeficientes $d_{j+1}[n]$ do nível de detalhes j+1 e a convolução entre o conjunto de dados de entrada $a_j[n]$ e o conjunto de coeficientes da função wavelet h sub-amostrados por um fator 2, gerando os coeficientes $a_{j+1}[n]$ do nível de aproximação j+1.

Na etapa de reconstrução temos, inicialmente, uma etapa de interpolação por zero nos sinais de detalhes $d_{j+1}[n]$ e aproximação $a_{j+1}[n]$, seguida pela convolução entre o conjunto de dados de detalhes $d_{j+1}[n]$ e o conjunto de coeficientes da função escala inversa g, somado com a convolução entre o conjunto de dados de aproximação $a_{j+1}[n]$ e o conjunto de coeficientes da função wavelet inversa h, gerando o sinal $\tilde{a}_j[n]$.

A diferença entre os sinais $\tilde{a}_j[n]$ e $a_j[n]$ deve ser a menor possível, preferencialmente nula, significando que o processo de decomposição e reconstrução não alterou o conjunto de dados originais, ou alterou de forma não significativa em sistemas onde alguma perda de qualidade é aceitável.

O sistema visual humano é capaz de realizar a interpolação entre as informações visuais, de modo que apenas algumas informações sensoriais são necessárias para compreensão da imagem. Nestes casos algum nível de perda pode ser tolerado no sinal reconstruído, de modo a validar o processo de compactação e reconstrução.

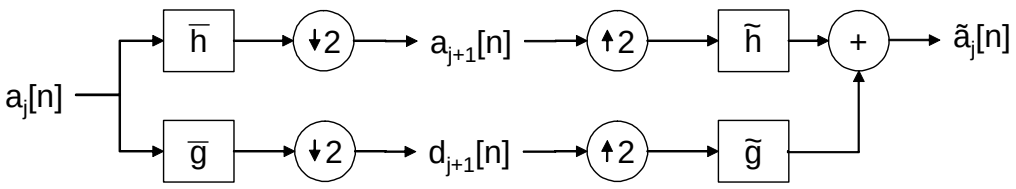

Figura 3.2: Decomposição e Reconstrução Wavelet.

Em uma decomposição *wavelet*, o sinal de entrada é decomposto em um ou mais níveis de resolução, também chamados de oitavas. Esta decomposição é realizada por meio de filtros passa-baixas e passa-altas, onde o filtro passa-baixas produz o sinal de aproximação, enquanto o filtro passa-altas produz o sinal de detalhes. A Figura 3.3 mostra a estrutura básica de um sistema onde um sinal unidimensional é decomposto por uma DWT de três oitavas, gerando três níveis de detalhes e um nível de aproximação.

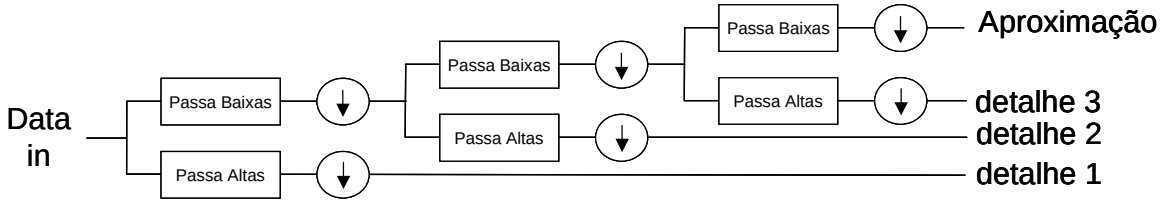

Figura 3.3: Decomposição Wavelet.

A saída de cada oitava possui a mesma quantidade de valores que a entrada, sendo metade representando o sinal de detalhes e a outra metade o sinal de aproximação. Isto se dá devido a decimação por 2 que deve ocorrer após a decomposição de cada oitava, em decorrência da redundância existente na representação resultante previsto na teoria Wavelet. A Figura 3.4 ilustra a relação existente entre os dados de uma oitava e os da oitava anterior.

De acordo com a Figura 3.4, o nível de detalhe 1 é o conjunto de valores que representa as freqüências mais altas existentes no sinal de entrada, o nível de detalhe 2 é o conjunto de valores que representa as freqüências mais altas existentes na aproximação proveniente da 1ª oitava. Assim cada nível de detalhe conterá as freqüências mais altas existentes na aproximação resultante da oitava anterior. Cada nível de detalhes terá N/(2n) valores, onde N é a quantidade de dados que compõem o vetor original e n é o número da oitava onde o detalhe foi gerado e a aproximação final sempre terá N/(2n) valores. Com isso podemos afirmar que independente do número de oitavas, o conjunto de saída terá sempre a mesma quantidade de amostras que o sinal de entrada.

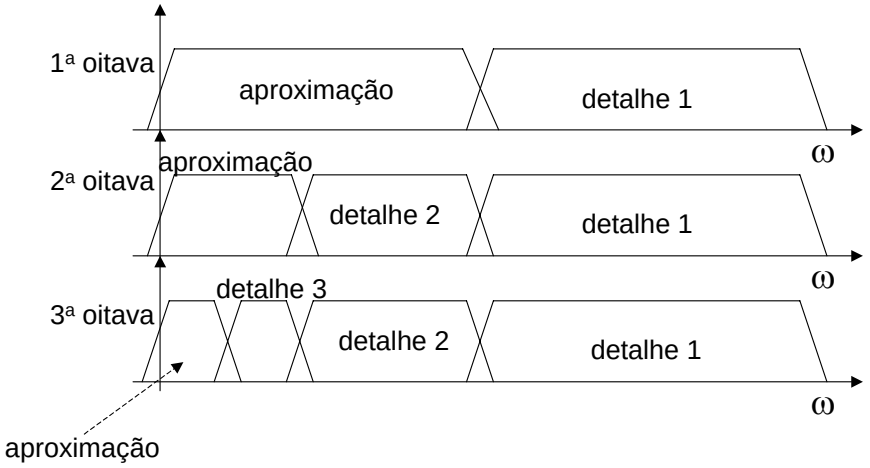

Figura 3.4: Decomposição em multiresolução de 3 oitavas.

Uma DWT para duas dimensões opera de forma semelhante a uma DWT para uma dimensão com a diferença que em cada oitava ocorre um nível de decomposição vertical e um nível de decomposição horizontal, gerando quatro conjuntos de dados, onde três são níveis de detalhe (LH, HL e HH) e um nível de aproximação (LL), conforme observado na Figura 3.5. Uma idéia semelhante pode ser aplicada para transformadas Wavelet de qualquer número de dimensões.

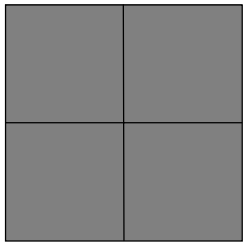

Figura 3.5: Decomposição 2D.

A decomposição em duas dimensões de um frame em várias oitavas ocorre de forma semelhante à transformada em uma dimensão, ou seja, em cada oitava ocorre um nível da decomposição do sub-frame que corresponde ao sinal de aproximação (sinal de baixas freqüências). A Figura 3.6 mostra cada passo da decomposição em quatro oitavas.

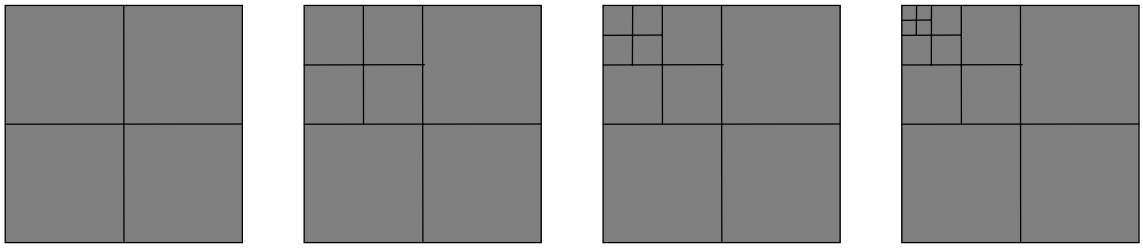

Figura 3.6: Decomposição 2D em 4 oitavas.

Como apresentado em ACHARYA (2005), o filtro passa-baixas é implementado pela convolução do sinal de entrada e uma função Wavelet, enquanto o filtro passa-altas é implementado pela convolução do sinal de entrada e uma função de escala.

g representa os coeficientes da função de escala

h representa os coeficientes da função wavelet

W representa os valores resultantes da aplicação da função wavelet (sinal de aproximação)

Wh representa os valores resultantes da aplicação da função escala (sinal de detalhe)

$$W(j,n) = \sum_{m=0}^{2n} W(j-1,m) * h(2n-m)$$

$$Wh(j,n) = \sum_{m=0}^{2n} W(j-1,m) * g(2n-m)$$

Matematicamente, a transformada wavelet aproxima uma função representando-a como uma combinação linear de duas funções, a função wavelet ψ e sua função escala associada ϕ, pertencentes a uma família de funções $\{\psi j,n(t)\}$, onde:

$$\{\psi_{j,n}(t) = \frac{1}{\sqrt{2}^{j}} \psi\left(\frac{t - 2^{j} n}{2^{j}}\right) \}(j,n)\in\mathbb{Z}2$$

sendo ψ uma função wavelet mãe, t uma translação da função ψ e 2j seu coeficiente de dilatação. Resultando na família de funções $\psi j,n(t)$, que representam uma base ortonormal em $\mathbb{R}2$. Com isso a transformada produzida proporciona uma transformação linear que pode ser aplicada nas formas direta e inversa.

3.1.1.1 Esquema Lifting

Um conjunto de amostras provenientes mundo real apresentam alto grau de correlacionamento entre os *pixels* e a idéia básica do esquema *lifting* consiste em decorrelacionar estas informações, de modo a produzir uma representação compacta do conjunto de amostras original. O esquema *lifting* é um processo que pode ser dividido em três estágios (DAUBECHIES, 1998) (SWELDENS 1995) (SWELDENS 1996) (SWELDENS 1997), os quais são conhecidos como: *split*, *predict* e *update*.

No primeiro estágio do processo de *lifting*, estágio de *split*, o conjunto original de amostras é separado em dois subconjuntos. Não existe nenhuma regra formal ou restrição de como deve ocorrer o *split* das amostras, existe apenas a necessidade de existir um procedimento para restaurar o conjunto original a partir dos dois subconjuntos resultantes. Para proporcionar que na saída do processo seja produzida a menor quantidade de informação possível, o processo de *split*, aplicado a um conjunto de amostras de entrada altamente correlacionado, deve separar estes dados mantendo a correlação entre os dois subconjuntos resultantes. Os melhores resultados são obtidos quando os dois conjuntos de dados são entrelaçados (DAUBECHIES, 1998).

Em DAUBECHIES (1998) é mostrada uma maneira simples e intuitiva de realizar o processo de *split*. Considerando um conjunto de amostras de entrada definida como x, é gerado um primeiro subconjunto chamado x_e, que corresponde às amostras de índice par e um segundo subconjunto chamado de x_o, que corresponde às amostras de índice ímpar, de modo a resultar na separação do conjunto de amostras de entrada em duas partes disjuntas. Este método também é conhecido como *Lazy Wavelet*.

Considerado que os conjuntos de amostras x_e e x_o apresentam forte correlação, a partir de um dos subconjuntos é possível construir um bom preditor (P) para o outro subconjunto de amostras, ou seja:

$$x_o \approx P(x_e)$$

Como o preditor P não produz um valor exato, ocorre o surgimento de um conjunto diferença ou detalhe *d*:

$$d = x_o - P(x_e)$$

Considerando que P seja um bom preditor, *d* será um conjunto esparso e decorrelacionado. Na prática, pode não ser possível prever exatamente x_o baseado em x_e, porém se a predição for razoável, a substituição de x_o pela diferença entre x_o e $P(x_e)$ resulta em uma quantidade menor de informação que o conjunto x_o original.

Como resultado do passo *predict*, temos que os dois subconjuntos x_e e x_o são transformados nos subconjuntos x_e e *d*. Como x_e representa uma sub-amostragem do conjunto original de amostras *x*, um problema de *aliasing* ocorre. Este *aliasing* pode ser eliminado com a introdução do segundo passo de *lifting*, conhecido como *update*.

No estágio de *update*, o subconjunto x_e é substituído por uma versão do subconjunto x_e atenuada pelo valor resultante da aplicação de um operador *update* (U) ao conjunto *d*, ou seja:

$$s = x_e + U(d)$$

Como resultado do passo *update*, temos a geração dos subconjuntos *s* e *d*, onde o subconjunto x_e foi substituído pelo subconjunto *s* para evitar *aliasing*.

A Figura 3.7 mostra a estrutura do esquema lifting, onde é possível observar a disposição dos três estágios *split*, *predict* (P) e *update* (U) e o fluxo de geração dos subconjuntos x_e e x_o a partir do conjunto de amostras de entrada *x* e a computação dos subconjuntos de saída *d* e *s*.

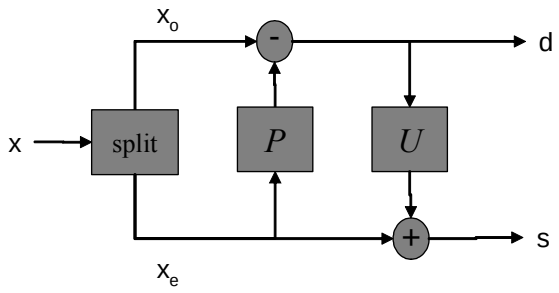

Figura 3.7: Esquema *Lifting*.

Uma importante consideração a respeito do esquema *lifting* é que independente de como ocorre a escolha de P e U, esta arquitetura é inversível para todos os casos e sempre permite a reconstrução perfeita das amostras originais. Entretanto, a escolha de P e U afetam diretamente o grau de decorrelacionamento entre as amostras. Assim a escolha dos coeficientes de P e U é um ponto importante para a implementação de uma transformada wavelet.

Os operadores aritméticos de circuitos digitais possuem o tamanho da palavra limitado, de modo que a utilização de operações com números reais apresenta precisão limitada. Assim, a utilização de operadores aritméticos limitados para o cálculo de P e U onde é requerido um alto grau de decorrelacionamento, resulta em um esquema lifting não inversível, ou também chamado irreversível. Logo, surgem dois tipos diferentes de transformada wavelet: transformadas reversíveis e transformadas irreversíveis.

Em métodos de compressão de imagens sem perdas, a imagem deve ser reconstruída exatamente como a original, logo a transformada utilizada deve ser

reversível. O padrão JPEG2000 sugere a implementação desta transformada com coeficientes inteiros. A grande desvantagem da utilização deste tipo de transformada é o reduzido grau de decorrelacionamento resultante.

Em métodos de compressão de imagens com perdas, é permitido certo nível de erro entre a imagem original e a imagem reconstruída, permitindo assim, a utilização de operadores aritméticos limitados, que são utilizados em cálculos envolvendo valores reais aproximados. Na Figura 3.8 é mostrada a estrutura *lifting* sugerida no padrão JPEG2000 para o cálculo da transformada *wavelet* irreversível.

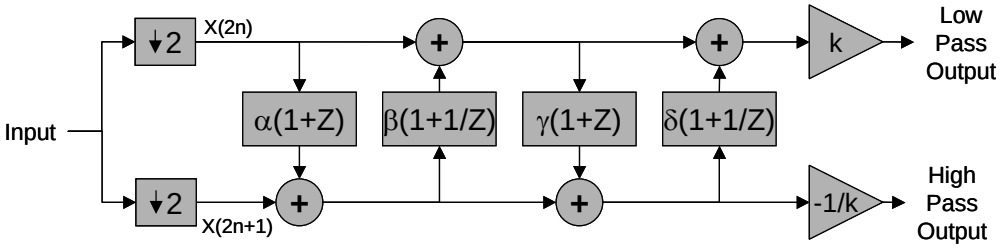

Figura 3.8: Estrutura *lifting utilizada* no compressor com perdas

(ITU-T, 2000)

A arquitetura, sugerida em ITU-T (2000) apresenta um estágio de *split*, dois estágios de *predict* (P) e *update* (U) e um estágio de escalonamento final.

3.2 Quantização

Para proporcionar o alto ganho em área de armazenamento, nos modos de compressão com perdas do padrão JPEG 2000, ocorre um passo de quantização após a DWT. Esta quantização tem por objetivo reduzir a precisão dos coeficientes, de modo a reduzir a quantidade de bits necessária para representar a imagem.

A quantização é um dos passos que proporciona a maior perda de informação de todo compressor. Como esta perda é irreversível, o nível de quantização deve estar de acordo com fidelidade requerida para a imagem reconstruída.

Para compressão irreversível, não é definido pelo padrão JPEG2000 (ITU-T, 2000) o valor do passo de quantização, permitindo que o aplicativo selecione o melhor passo de quantização, de acordo com as características da imagem a ser comprimida. Para compressão sem perdas o passo de quantização é sempre definido como 1.

A quantização é implementada divisão escalar uniforme de acordo com a equação abaixo:

$$q_b(i,j) = sign\ (y_b(i,j)) \left\lfloor \frac{|y_b(i,j)|}{\Delta_b} \right\rfloor$$

onde $y_b(i,j)$ representa o coeficiente da DWT da sub-banda b e Δ_b é o passo de quantização da sub-banda b.

3.3 Codificação Aritmética

A codificação aritmética é uma das muitas técnicas de compressão com comprimento de palavra variável (ACHARYA, 2005).

Neste tipo de codificação, a seqüência de símbolos de entrada é representada por um intervalo de números reais entre 0,0 e 1,0.

Este tipo de codificação oferece maior eficiência e flexibilidade, quando comparado com outros métodos de codificação, como Huffman. Porém, a codificação aritmética apresenta maior complexidade e menor possibilidade de recuperação de dados em caso de erros.

Pode-se considerar que a codificação aritmética apresenta inicialmente, um intervalo normalizado e cada símbolo de um alfabeto A possui uma probabilidade inerente. A cada novo símbolo na entrada, a faixa de valores normalizada é modificada.

A partir do intervalo de entrada [0,1), é lido um símbolo da entrada e de acordo com a sua probabilidade esta faixa é modificada para [0,X) ou [X,1). No passo seguinte a faixa de valores é novamente normalizada, porém mantendo os valores extremos que representa.

A Figura 3.9 representa o processo de codificação no qual foi inserida em sua entrada a seqüência de símbolos ***abbaa***, pertencentes a um alfabeto A e a probabilidade do símbolo *a* é 0,6 e a probabilidade do símbolo *b* é 0,4. Assim, o intervalo inicial é [0,0 ; 1,0) e a faixa de intervalos de cada símbolo de entrada pertencente ao alfabeto A é: *b* = [0,0 ; 0,4) e *a* = [0,4 ; 1,0). Com a entrada do primeiro símbolo (símbolo *a*), o intervalo é reduzido para [0,4 ; 1,0). Cada novo sub-intervalo é considerado um intervalo normalizado para fins de cálculo da probabilidade de cada símbolo, de modo que o símbolo *a* (que possui probabilidade 0,6) do primeiro sub-intervalo é representado pelo intervalo [0,64 ; 1,00) e o símbolo *b* (que possui probabilidade 0,4) é representado pelo intervalo [0,40 ; 0,64). O segundo símbolo (símbolo *b*), posiciona o novo sub-intervalo em [0,40 ; 0,64). O terceiro símbolo (símbolo *b*), posiciona o novo sub-intervalo em [0,400 ; 0,496). O quarto símbolo (símbolo *a*), posiciona o novo sub-intervalo em [0,4384 ; 0,4960). O quinto símbolo (símbolo *a*), posiciona o novo sub-intervalo em [0,46144 ; 0,49600). Deste modo, a seqüência ***abbaa*** pode ser representada por qualquer valor pertencente ao intervalo [0,46144 ; 0,49600), por exemplo 0,48.

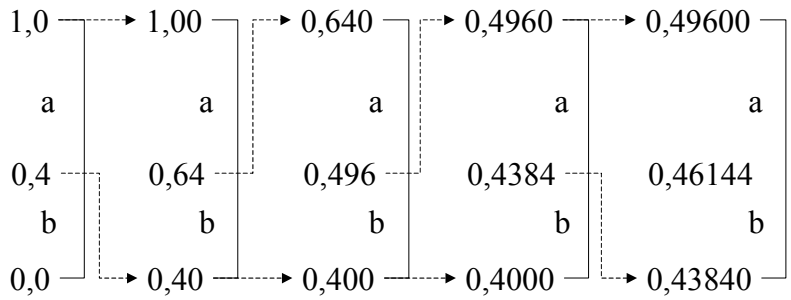

Figura 3.9: Processo de codificação aritmética.

Segundo ACHARYA (2005), o algoritmo de codificação aritmética apresenta as seguintes limitações:

- O valor codificado não é único. Como a codificação resulta em um intervalo de valores [x,y), a decodificação pode apresentar uma seqüência com um número infinito de símbolos.
- Não ocorre a transmissão de nenhuma informação até o final da codificação. Logo o decodificador deve ser capaz de armazenar toda a mensagem antes de realizar a decodificação.
- A precisão requerida para representar o intervalo resultante cresce com a quantidade de símbolos codificados.
- A utilização de multiplicadores para recalcular a nova faixa dinâmica, a cada passo, pode se tornar proibitiva para aplicações de tempo real.
- O algoritmo é extremamente sensível a ocorrência de erros. O menor erro pode representar a decodificação de um conjunto de símbolos completamente diferente.

3.3.1 Codificação Aritmética Binária

A técnica de codificação aritmética binária soluciona alguns dos problemas que a codificação aritmética original apresenta.

Uma das principais vantagens é a não necessidade de aguardar o final da codificação para transmitir a informação, logo o decodificador não necessita armazenar toda a informação codificada antes de proceder a decodificação.

O problema da precisão no cálculo das faixas dinâmicas pode ser resolvido com a utilização de uma aritmética de ponto fixo e o escalonamento do intervalo resultante. Para a utilização de aritmética em ponto fixo, a faixa inicial é substituída de [0,0 ; 1,0) para [0, MAX_VALUE) onde MAX_VALUE representa o maior número inteiro que pode ser representado e vale 2^n-1. Sempre que a faixa dinâmica ficar reduzida à metade da faixa original, ocorre um re-escalonamento da faixa dinâmica e conseqüente transmissão de um bit, que possui valor Lo ou Hi dependendo da situação que causou o re-escalonamento.

Considerando o intervalo inicial [*MIN_VALUE*, *MAX_VALUE*), a computação de cada passo de codificação gera um novo intervalo definido como [*NEW_MIN_VALUE*, *NEW_MAX_VALUE*) e pode produzir um bit codificado de acordo com as regras abaixo:

- Caso *NEW_MAX_VALUE* < (*MAX_VALUE*+1)/2 ocorre a geração de um bit de valor **Lo** e re-escalonamento. O novo intervalo será [2**NEW_MIN_VALUE*, 2**NEW_MAX_VALUE*). Caso *SP_COUNT* > 0 gera *n* bits em nível Hi, onde n representa o conteúdo de *SP_COUNT*;
- Caso *NEW_MIN_VALUE* ≥ (*MAX_VALUE*+1)/2 ocorre a geração de um bit de valor **Hi** e re-escalonamento. O novo intervalo será [2*(*NEW_MIN_VALUE*-(*MAX_VALUE*+1)/2), 2*(*NEW_MAX_VALUE*-(*MAX_VALUE*+1)/2)). Caso *SP_COUNT* > 0 gera *n* bits em nível Lo, onde n representa o conteúdo de *SP_COUNT*;

- Caso *NEW_MAX_VALUE* < (*MAX_VALUE*+1)*3/4 e *NEW_MIN_VALUE* ≥ (*MAX_VALUE*+1)/4 não ocorre a geração de bit algum, ocorre re-escalonamento e incremento de um contador *SP_COUNT*. O novo intervalo será [2*(*NEW_MIN_VALUE*-(*MAX_VALUE*+1)/4), 2*(*NEW_MAX_VALUE*-(*MAX_VALUE*+1)/4)).

3.3.2 QM-Coder

O QM-coder é um algoritmo de codificação aritmética adaptado para utilização com valores binários, também é utilizado no processo de compressão de imagens *bi-level* do padrão JBIG (ACHARYA, 2005).

As principais características do algoritmo QM-coder são simplicidade e velocidade. Este algoritmo é livre de multiplicações e realiza as aproximações dos intervalos utilizando somente operações de soma, subtração e deslocamento com precisão fixa.

O QM-coder opera segundo um mapeamento dos bits de entrada em símbolos mais prováveis (MPS) ou símbolos menos prováveis (LPS). O mapeamento em função de MPS e LPS é vantajoso em relação a um mapeamento de bits “0” e bits ”1”, pois permite a obtenção de uma melhor compressão.

Como somente existem dois símbolos no alfabeto (MPS e LPS), se a probabilidade do LPS é Q, então a probabilidade do MPS será (1-Q). Considerando que A seja o intervalo unitário, então o intervalo designado para LPS será AQ e o intervalo designado para MPS será A(1-Q).

O algoritmo considera que o intervalo A sempre possui seu valor aproximado a 1 (na faixa de 0,75 a 1,5), logo os intervalos podem ser aproximados para:

- Intervalo designado para LPS => $AQ \approx Q$ => [A-Q, A);
- Intervalo designado para MPS => $A(1-Q) \approx A-Q$ => [0, A-Q).

e deste modo, as multiplicações são evitadas.

As duas principais variáveis do codificador são A e C, onde A representa o sub-intervalo considerado e C representa código de saída.

Cada codificação de um intervalo MPS ou LPS afeta as variáveis A e C. O valor da variável C deve ser somado ao valor inicial do sub-intervalo, 0 para o caso de uma codificação MPS e A - Q para o caso de uma codificação LPS. O valor de A é substituido pelo sub-intervalo MPS, que vale A - Q ou pelo sub-intervalo LPS, que vale Q. Caso o valor de A fique abaixo de 0,75, as variáveis A e C sofrem renormalização. A renormalização consiste em um deslocamento à esquerda das variáveis A e C, até que o valor de A volte a pertencer à faixa (1,5 , 0,75]. A variável C deve ser deslocada para manter o sincronismo entre as duas variáveis.

O valor de Q, que representa a estimação da probabilidade, é dado por uma tabela pré-definida que é atualizada cada vez que ocorre renormalização.

Considerando que Q representa o intervalo do LPS, seu valor sempre será menor que 0,5, pois a probabilidade do LPS será sempre menor que 50%. Neste caso, sempre que ocorrer a codificação de um intervalo LPS, ocorrerá renormalização.

4 ARQUITETURAS PARA A DWT DE UMA DIMENSÃO

A implementação da transformada *Wavelet* Discreta (DWT) em *n* dimensões pode ser obtida por aplicações sucessivas de uma 1D-DWT, de acordo com CHAKRABARTI (1995), logo a importância da implementação em hardware da 1D-DWT.

De acordo com a seção 3.1.1, a transformação wavelet é realizada por um processo de filtragem de um sinal de entrada, gerando dois sinais distintos, onde o primeiro representa a componente de saída de um filtro passa-baixas e o segundo representa a componente de saída de um filtro passa-altas.

Considerando que o sinal de entrada é um conjunto de dados amostrado a partir de um sinal de entrada analógico, estes filtros passa-altas e passa-baixas podem ser implementados através de dois filtros digitais, que podem ser organizados como um banco de filtros. Entretanto é possível utilizar outras arquiteturas, visando melhorias de desempenho. A utilização de uma arquitetura *lifting* proporciona uma redução na quantidade de operadores aritméticos, porém aumenta a latência da arquitetura.

A implementação de uma DWT com mais de uma oitava é realizada utilizando o conceito de recursividade, ou seja, os coeficientes gerados pela saída passa-baixas são reinseridos para serem computados no bloco de filtragem. Nestes casos, o uso do RPA proporciona um ganho significativo no tempo de computação e na redução do uso da memória.

4.1 Banco de filtros

Em SALOMON (2000) é definido que uma filtragem digital é uma operação linear entre os coeficientes do filtro h(0), h(1), .., h(n) e um vetor de entrada x, produzindo um vetor de saída y, de acordo com:

$$y(n) = \sum_{k=-\infty}^{+\infty} h(k)\, x(n-k) \quad = h * x$$

Onde $*$ indica uma operação de convolução. Como o somatório acima não apresenta explícitamente um limite na quantidade de operandos, a quantidade de operações realizadas depende do número de amostras dos vetores x e h. Apesar da equação apresentar o índice n variando de $-\infty$ a $+\infty$, o valor das amostras para $n<0$ e $n>N-1$ terá sempre valor zero, considerando N igual ao número de amostras de entrada.

A idéia básica de um banco de filtros é realizar um banco de análise composto de dois filtros e um banco de síntese também composto de dois filtros.

O banco de filtros de análise, que é utilizado no processo de transformação direta, realiza a separação do sinal de entrada em duas bandas bem definidas, uma banda composta de valores de alta freqüência gerada pela convolução do sinal de entrada com os coeficientes de um filtro passa-altas h1 e uma banda composta de valores de baixa freqüência gerada pela convolução do sinal de entrada com os coeficientes de um filtro passa-baixas h0.

Os filtros de análise são alimentados com as amostras do vetor x, uma a cada ciclo de clock, gerando a cada amostra x(n) uma saída yL(n) e uma saída yH(n). Com isso o número de coeficientes de saída corresponde ao dobro do número de amostras de entrada (considerando apenas dois filtros), significando um resultado negativo para uma compressão de dados. Para corrigir esta situação, cada filtro é seguido por uma etapa de *downsampling* que descarta todas as saídas de índice 2n+1.

O banco de filtros de síntese, que é utilizado no processo de transformação inversa, realiza a união do sinal proveniente de cada uma das duas sub-bandas gerando um vetor de saída reconstruído $\bar{x}(n)$ equivalente ao vetor x. Os filtros de síntese são alimentados com os sinais gerados pelas saídas de análise yL(n) e yH(n), que passaram pela etapa de *downsampling*, logo para gerar o vetor $\bar{x}(n)$ com uma quantidade de valores equivalente ao do vetor x, se faz necessário um processo de *upsampling* antes dos filtros de síntese. Este processo consiste na inserção de coeficientes nulos (valor 0) entre cada um dos coeficientes dos vetores yL e yH.

Em processos de transformação sem perdas os vetores x e $\bar{x}$ são idênticos, porém em processos de transformação que permitam algum grau de perda, os dois vetores podem ser ligeiramente diferentes, devido a limitações do número de casas significativas nos coeficientes h.

A profundidade de cada filtro influencia na quantidade de coeficientes das saídas yL e yH. Considerando que o vetor de entrada tenha *n* amostras e o filtro tenha *m* coeficientes, o vetor de saída terá *m*+*n*-1 coeficientes. Após o *downsampling*, a quantidade de coeficientes é reduzida para (*m*+*n*-1)/2, proporcionando uma diferença com a quantidade esperada de coeficientes na saída do filtro, que deveria ser *n*/2. A truncagem dos coeficientes excedentes (coeficientes após índice *n*/2) não é uma estratégia válida por ocasionar a perda irreversível das informações da borda da imagem. A solução para este problema é a adoção de uma estratégia de espelhamento das bordas do vetor de entrada e posterior truncagem dos valores de índices excedentes. Com isso não ocorre perda de informação de borda nem quantidade excedente de coeficientes no vetor de saída. A Figura 4.1 ilustra a estratégia de espelhamento das bordas.

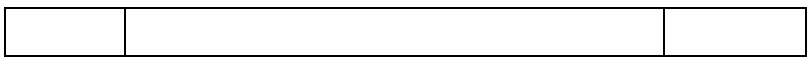

Figura 4.1: Espelhamento das bordas do vetor de entrada.

O processo de espelhamento das bordas requer a computação extra de coeficientes, porém permite a geração dos *n*/2 coeficientes válidos, truncando os coeficientes excedentes.

4.2 RPA

O algoritmo RPA descrito em VISHWANATH (1994) é uma importante estratégia para implementar a arquitetura de uma DWT de uma dimensão, por permitir a computação dos coeficientes de saída de todas as oitavas de ordem superior a primeira de modo intercalado com a computação dos coeficientes de saída da primeira oitava. Deste modo, o tempo total de computação de todas as oitavas é igual ao tempo de computação da primeira oitava. Uma segunda vantagem do uso do RPA é a redução da quantidade de memória para o armazenamento dos dados para a computação de todas as oitavas.

No RPA o requisito de memória para o armazenamento dos coeficientes de saída é de apenas um conjunto de J*K registradores, onde J representa a quantidade de oitavas da DWT e K representa a profundidade do filtro com maior quantidade de TAPs.

Por exemplo: uma transformada de 3 oitavas utilizando filtros de 4 TAPs requer 12 registradores, sendo 4 registradores para a computação de cada oitava.

A Figura 4.2 mostra a seqüência de saída do RPA para 3 oitavas. No primeiro ciclo é computado o primeiro coeficiente da primeira oitava.

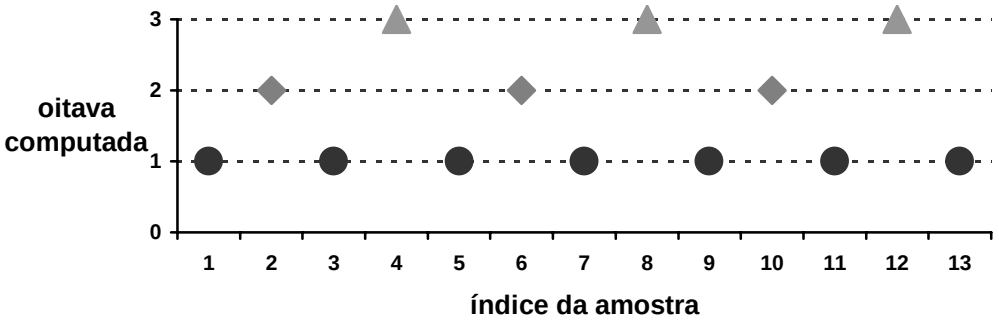

Figura 4.2: Seqüência do RPA para 3 oitavas.

O algoritmo abaixo apresenta o RPA de modo comportamental, onde **N** define a quantidade de amostras de entrada, ***i*** é o contador do número da amostra a ser computada e ***j*** é o contador da oitava a ser computada.

```
main program
begin{recursive Pyramid}
input: x[i:1,N]
for (i=1 to N)
    rdwt(i,1)
end {recursive Pyramid}
```

```
rdwt(i,j)
begin{rdwt}
    if (i is odd)
        comput output numbered ((i+1)/2) of octave j
    else
        rdwt(i/2, j+1)
end{rdwt}
```

A cada ciclo do laço "*for*", apenas um coeficiente de saída é computado. Neste algoritmo, a computação ocorre somente quando a amostra for de índice impar. Quando ***i*** tiver índice par, o procedimento *rdwt*(i,j) é chamado recursivamente com i valendo metade de seu valor no ponto de chamado.

A computação do coeficiente de aproximação deverá ser considerada pelo algoritmo acima como uma exceção, pois deverá ser computada para as amostras de índice par da última oitava. A maneira de fazer isso é computar como o coeficiente de índice par da ultima oitava. A Tabela 4.1 mostra o valor de ***i*** a cada nível de recursão.

Tabela 4.1: Seqüência de computação do RPA para 3 oitavas

i	i/2	i/4	operação
1			computação do 1º coeficiente de detalhe da 1ª oitava
2	1		computação do 1º coeficiente de detalhe da 2ª oitava
3			computação do 2º coeficiente de detalhe da 1ª oitava
4	2	1	computação do 1º coeficiente de detalhe da 3ª oitava
5			computação do 3º coeficiente de detalhe da 1ª oitava
6	3		computação do 2º coeficiente de detalhe da 2ª oitava
7			computação do 4º coeficiente de detalhe da 1ª oitava
8	4	2	computação do 1º coeficiente de aproximação
9			computação do 5º coeficiente de detalhe da 1ª oitava
10	5		computação do 3º coeficiente de detalhe da 2ª oitava
11			computação do 6º coeficiente de detalhe da 1ª oitava
12	6	3	computação do 2º coeficiente de detalhe da 3ª oitava
13			computação do 7º coeficiente de detalhe da 1ª oitava
14	7		computação do 4º coeficiente de detalhe da 2ª oitava
15			computação do 8º coeficiente de detalhe da 1ª oitava
16	8	4	computação do 2º coeficiente de aproximação
17			computação do 9º coeficiente de detalhe da 1ª oitava
18	9		computação do 5º coeficiente de detalhe da 2ª oitava

A Figura 4.3 apresenta a seqüência dos coeficientes de saída do bloco da 1D-DWT. Considerando n = [0..N-1], N igual ao número de amostras de entrada (ex: uma linha completa da imagem), X o número de oitavas a ser computada e t o índice inteiro para definir o momento da computação de um coeficiente, a geração dos coeficientes de saída da DWT segue as seguintes regras:

- Gera os coeficientes de saída do nível X de detalhes quando $t = 2^X n+2^{X-1}-1$;
- Gera os coeficientes de saída do nível de aproximação quando $t = 2^X n+2^X-1$.

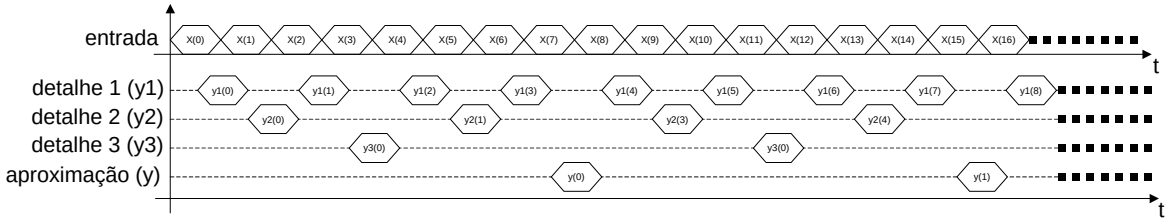

Figura 4.3: Seqüência do RPA para 3 oitavas.

A partir da Figura 4.3, observa-se que a principal desvantagem do uso RPA é que os dados de saída se apresentam de forma intercalada, requerendo um pós-processamento para rearranjar cada uma das bandas resultantes.

A seqüência de saída do RPA para o processamento de uma DWT de 3 oitavas, considerando n = [0..N-1] e N igual ao número de amostras de entrada:

- os valores de ordem 2n correspondem ao nível de detalhe 1;
- os valores de ordem 4n+1 correspondem ao nível de detalhe 2;
- os valores de ordem 8n+3 correspondem ao nível de detalhe 3;
- os valores de ordem 8n+7 correspondem ao nível de aproximação.

Assim, de acordo com a Figura 4.4, no início da linha devem ser copiados os N/8 valores referentes ao nível de aproximação, a seguir os N/8 valores referentes ao 3º nível de detalhes, a seguir os N/4 valores referentes ao 2º nível de detalhes e por último os N/2 valores referentes ao 1º nível de detalhes.

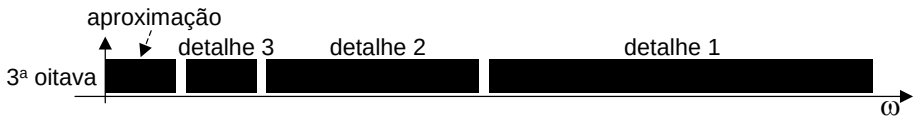

Figura 4.4: Ordenação de uma linha da saída da 1D-DWT de 3 oitavas.

4.3 Arquiteturas para a DWT de uma dimensão

O cálculo da DWT de uma dimensão (1D-DWT) é feito por decomposição em sub-bandas, assim a implementação da arquitetura VLSI pode ser naturalmente feita por banco de filtros. Onde temos basicamente um filtro passa-altas e um filtro passa-baixas para o processamento de cada oitava.

O processamento da 1D-DWT é feito por oitavas e cada oitava tem uma quantidade diferente de amostras de entrada, logo o tempo de computação é diferente para cada oitava. O processamento de cada oitava requer $n/2^{b-1}$ computações, onde n representa a quantidade total de amostras de entrada e b representa a oitava a ser computada. Assim o tempo de computação é proporcional à quantidade de computações requeridas.

Após o processamento no banco de filtros, os coeficientes de saída de cada oitava são armazenados em uma memória, requerendo assim:

$$\text{Quantidade de posições de memória} = \sum_{b=0}^{\#oitavas} \frac{n}{2^{b-1}}$$

Neste método o processamento da DWT de J oitavas requer $(2^J\text{-}1)N/2^{J-1}$ ciclos de *clock*. Porém, considerando o espelhamento das bordas a cada oitava, como apresentado na seção 4.1, deve ser considerada o acréscimo de 2m ciclos de *clock* para a

computação das bordas. Deste modo, o processamento de uma DWT de J oitavas e borda de tamanho m requer ($(2^J - 1)\frac{N}{2^{J-1}} + 2Jm$) ciclos de *clock*.

Para otimizar este procedimento é utilizado o algoritmo da pirâmide recursivo (RPA), descrito na seção 4.2. Assim a cada oitava ocorre a geração de duas bandas a partir de um único conjunto amostras de entrada, uma banda de aproximação que é realimentada no sistema para a geração das oitavas posteriores e outra banda de detalhes.

As primeiras arquiteturas dedicadas para a computação da DWT utilizando o RPA, encontradas na literatura, são as arquiteturas paralelas e as arquiteturas sistólicas, onde as arquiteturas paralelas utilizam a técnica de multiplexação no tempo, enquanto as arquiteturas sistólicas utilizam arranjos regulares baseadas em elementos de processamento pré-definidos.

Além destas arquiteturas, Processadores Digitais de sinais (DSPs) também foram produzidos especificamente para a computação da DWT. Na seção 4.3.3 são mostradas diversas arquitetura desenvolvida comercialmente.

4.3.1 Arquiteturas de filtros paralelos

A implementação de uma arquitetura paralela multiplexada para a computação da DWT foi implementada a partir da descrição de KNOWLES (1990). A idéia básica da arquitetura é utilizar somente um conjunto de filtros para realizar a computação de todas as oitavas, multiplexando a computação dos coeficientes, seguindo a idéia do RPA apresentada na seção 4.2. A arquitetura apresenta as seguintes características:

- banco de filtros com dois filtros de profundidade K1 e K2.
- J*K registradores de m-bits, onde J representa a quantidade de oitavas, K a maior quantidade de TAPs dos filtros K1 e K2 e m representa o tamanho da palavra para armazenar o maior coeficiente possível;
- um multiplexador (chamado Mux1) para selecionar uma entre J entradas dos 2*K registradores de m-bits;
- um multiplexador (chamado Mux2) para fazer a seleção entre as duas saídas dos filtros (saídas de m-bits).

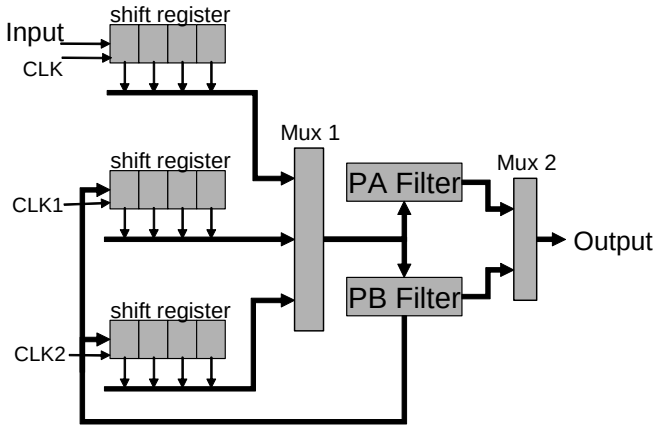

Figura 4.5: Arquitetura Paralela de três oitavas.
(KNOWLES, 1990)

A arquitetura paralela para a computação da 1D-DWT apresentado na Figura 4.5 mostra uma arquitetura para computar 3 oitavas (J = 3) utilizando 4 TAPs em cada filtro (K1 = K2 = 4). As amostras de entrada são inseridas diretamente no primeiro conjunto de registradores com taxa de cadenciamento idêntica ao do *clock* global do sistema. O multiplexador Mux1 deve habilitar a conexão do primeiro conjunto de registradores com os dois filtros a cada dois ciclos do sinal de *clock*. A saída do filtro passa-altas representa o sinal de detalhes da primeira oitava. A saída do filtro passa-baixas deve ser direcionada para o segundo conjunto de registradores com taxa de cadenciamento de metade do *clock* global do sistema. O multiplexador Mux1 deve habilitar a conexão do segundo conjunto de registradores com os dois filtros a cada quatro ciclos do sinal de *clock* (a partir do segundo ciclo). A saída do filtro passa-altas representa o sinal de detalhes da segunda oitava. A saída do filtro passa-baixas deve ser direcionada para o terceiro conjunto de registradores com taxa de cadenciamento de um quarto do *clock* global do sistema. O multiplexador Mux1 deve habilitar a conexão do terceiro conjunto de registradores com os dois filtros a cada quatro ciclos do sinal de *clock* (a partir do quarto ciclo). A cada quatro ciclos de *clock* são gerados de modo intercalado, o sinal de detalhes da terceira oitava, vindo da saída do filtro passa-altas, e o sinal de aproximação vindo da saída do filtro passa-baixas. O multiplexador Mux2 apenas muda de estado quando da seleção do sinal de aproximação, que ocorre a cada 8 ciclos.

A síntese desta arquitetura utilizando coeficientes Daubechies 3TAPs direcionada para um FPGA da família APEX da ALTERA resultou em uma área de 551 LEs e uma freqüência máxima de 40MHz, considerando que os registradores foram mapeados em memória. Para aumentar a quantidade de oitavas nesta arquitetura basta acrescentar um conjunto de registradores para cada oitava extra e o número de estados da máquina de controle. Com isso a síntese da nova arquitetura resultou em uma área de 562 LEs e uma freqüência máxima de 40MHz. Como os registradores são mapeados na memória interna do FPGA seu acréscimo não influenciou na área total em LEs, assim presume-se que o acréscimo de 11 LEs ocorreu na etapa de controle.

4.3.2 Arquiteturas de Arranjos Sistólicos

A implementação da arquitetura da 1D-DWT utilizando um arranjo sistólico, apresenta as características de regularidade, localidade e escalabilidade. A localidade é proporcionada pela utilização de um elemento de processamento (PE) pré-definido e fixo, a regularidade é proporcionada pela utilização do mesmo PE em todo o arranjo e escalabilidade é resultante da simplicidade de inserção de mais oitavas no arranjo.

O primeiro arranjo sistólico implementado segue a descrição de SYED (1995). Todos os PE (mostrado na Figura 4.6) possuem internamente um registrador de m-bits, dois multiplicadores por constantes (G e H), que devido a isso, podem se multiplicadores simplificados e dois somadores.

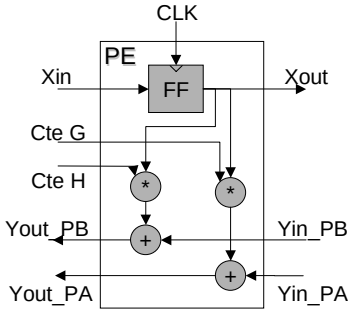

Figura 4.6: Estrutura do Elemento de Processamento (SYED, 1995).

Um arranjo sistólico para 3 oitavas e 4 TAPs em cada filtro é mostrado na Figura 4.7. A quantidade de PEs em cada oitava deve ser igual à quantidade de TAPs apresentada pelo filtro de maior quantidade de coeficientes. Uma etapa de controle foi implementada para gerar o sinal de *clock* diferenciado para cada oitava e a seleção do multiplexador de saída, de modo que o conjunto de dados de saída obedeça às definições do RPA.

Devido a esta arquitetura não necessitar de qualquer controle para os PEs com exceção do sinal de *clock*, esta arquitetura possui uma regularidade muito boa.

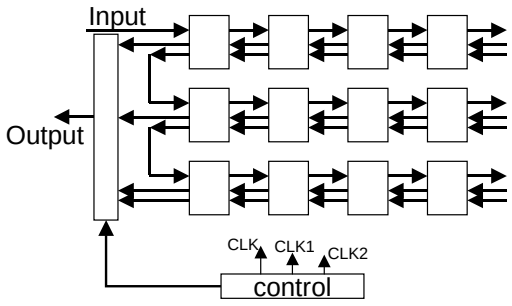

Figura 4.7: Arquitetura por Arranjo sistólico de três oitavas e 4 TAPs.

A síntese desta arquitetura para coeficientes Daubechies 3TAPs direcionada para um FPGA da família APEX da ALTERA resultou em uma área de 988 LEs e uma freqüência máxima de 44MHz, considerando que os registradores foram mapeados em memória.

A inserção de oitavas extras nesta arquitetura demanda a inserção de um conjunto de PEs proporcional à quantidade de TAPs necessários em cada oitava, bem como a alteração da etapa de controle para a geração dos sinais de *clock* intercalados e seleção do multiplexador de saída. Com isso a síntese da nova arquitetura resultou em uma área de 1295 LEs e uma freqüência máxima de 44,4MHz. Comparado à arquitetura da seção 4.3.1, que não apresentou acréscimo significativo em área com o acréscimo de uma oitava, esta arquitetura possui operadores aritméticos internamente ao PE, logo o acréscimo de uma oitava resulta em aumento proporcional de área. Em outras palavras cada oitava requer aproximadamente 320 LEs.

A segunda arquitetura sistólica implementada segue a descrição feita por FRIDMAN (1994). Esta arquitetura se baseia no arranjo de elementos de processamento (PE), possuindo uma quantidade de PEs igual ao número de TAPs do maior dos filtros. Como os registradores que definem o número de oitavas ficam colocados internamente aos PEs, o circuito de controle necessita apenas gerar os sinais para o acesso aos registradores internos dos PEs.

A Figura 4.8 apresenta a estrutura interna do PE utilizado nesta arquitetura. Nela podemos observar os elementos presentes nas arquiteturas anteriores, como dois multiplicadores, dois somadores, as duas constantes (h e g) e os elementos de armazenamento.

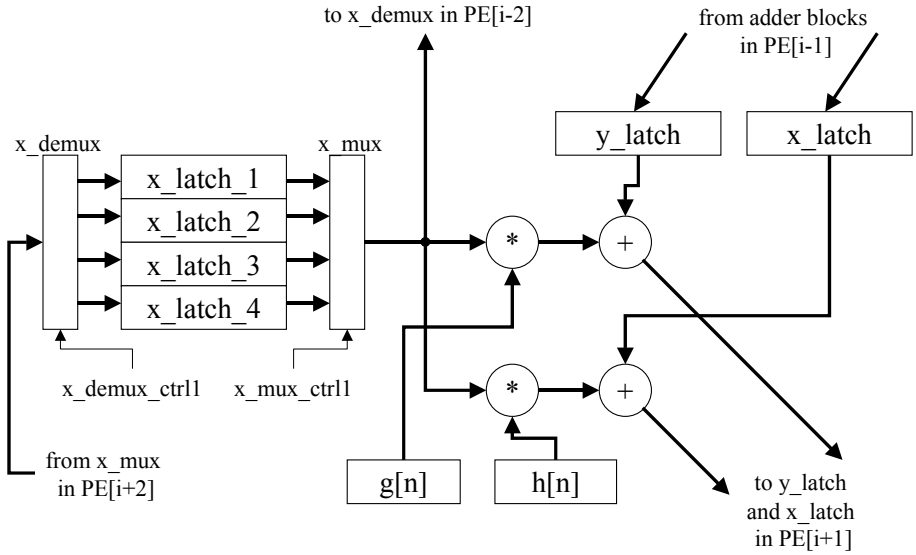

Figura 4.8: Estrutura do Elemento de Processamento (PE) para quatro oitavas.

(FRIDMAN, 1994)

Nesta arquitetura a localidade e a regularidade ficam bem definidas, porém a escalabilidade no que diz respeito ao número de oitavas é baixa por necessitar de uma complexa alteração no controle quando da inserção de oitavas extras. Entretanto, a escalabilidade no que diz respeito ao número de TAPs do filtro é alta, pois para isso basta conectar novos PEs em cascata.

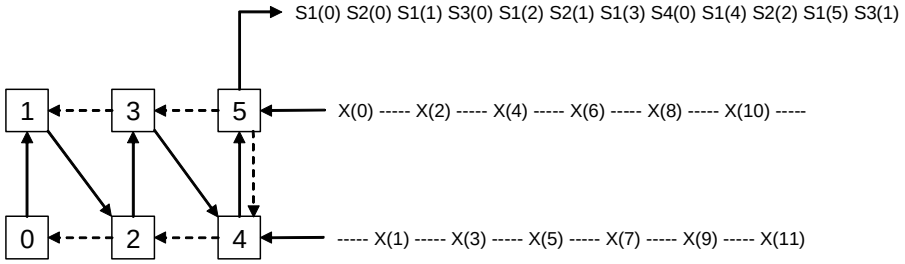

Figura 4.9: Arquitetura por Arranjo sistólico com 6 TAPs.

O fluxo de dados entre os diversos PEs da arquitetura é ilustrado por um arranjo de 6 TAPs mostrado na Figura 4.9. Nesta figura podemos observar que as amostras de ordem ímpar seguem pelas PEs de ordem par e as amostras de ordem par seguem pelas PEs de ordem ímpar. Os coeficientes de saída são gerados seguindo a ordem descrita no RPA e o fluxo de computação segue a ordem crescente dos PEs, conforme Figura 4.9. Nesta arquitetura, os coeficientes de oitavas de ordem menores são realimentados de volta para o PE 4 e PE 5 (de acordo se amostra de índice par ou impar) a partir da saída do PE 5.

A síntese desta arquitetura para coeficientes Daubechies 4TAPs direcionada para um FPGA da família APEX da ALTERA resultou em uma área de 568 LEs e uma freqüência máxima de 25,3MHz, desconsiderando os registradores internos, que foram mapeados em memória.

4.3.3 Arquiteturas Comerciais

Atualmente já existem IPs comerciais e equipamentos dedicados que implementam a DWT. Desde IPs que podem ser utilizados em protótipos de arquiteturas microprocessadas até circuitos que compõem sondas espaciais.

A Analog Devices produz diversos dispositivos para realizar a transformada wavelet e a compressão/descompressão segundo o padrão JPEG2000. Para realizar a transformada Wavelet a Analog Devices possui os dispositivos ADV601 (ANALOG, 2005-a) e ADV611 (ANALOG, 2005-b) que são codificadores de vídeo, que operam a 27MHz e o dispositivo ADV202 (ANALOG, 2005-c), que realiza a compressão e a descompressão segundo o padrão JPEG2000 (ITU-T, 2000).

O principal objetivo das missões de exploração espacial feita pela NASA é coletar e transmitir para uma estação receptora na Terra, imagens e informações de diversos tipos de sensores. Atualmente existem em marte dois MERs (chamados Spirit e Opportunity) equipados com nove câmeras cada um. Para transmitir as imagens captadas pelas câmeras para a Terra, é utilizado um processo de compressão de imagens semelhante ao padrão JPEG2000, chamado ICER (KIELY, 2002) (KIELY, 2003). O ICER é capar de tratar as imagens com resolução de 1024x1024 pixels de 12 bits.

Empresas como a Barco Silex (BARCO, 2005-a) e a CAST Inc. (CAST, 2005) mantém parcerias com os fabricantes de FPGAs para o desenvolvimento de soft IPs diversos.

A Tabela 4.2 mostra os soft IPs para a implementação da 2D-DWT desenvolvido pela Barco Silex, chamado 113FDWT (BARCO, 2005-b) e pela CAST Inc, chamado C_2DDWT (CAST, 2005-c).

Tabela 4.2: Soft IPs para a implementação da 2D-DWT

IP	Frame	Área	fmáx	Taxa de operação	Dispositivo	Empresa
BA113FDWT	128X128	4291	130,0	-	EP1S20F484C5	Barco Silex
BA113FDWT	128X128	3381	135,0	-	XC2V1000-6	Barco Silex
RC_2DDWT	512x512	3280	65,3	23,6	EPF10K200SRC240	CAST Inc
RC_2DDWT	512x512	3280	68,7	24,9	EP1K100QC208	CAST Inc
RC_2DDWT	512x512	3239	84,6	30,6	EP20K160EQC208	CAST Inc
RC_2DDWT	512x512	1698	77,0	-	Spartan-II 2S200-6	CAST Inc
RC_2DDWT	512x512	1698	72,0	-	Virtex-E V300-6	CAST Inc
RC_2DDWT	512x512	1698	90,0	-	Virtex-E V300E-8	CAST Inc
RC_2DDWT	512x512	1695	95,0	-	Virtex-II 2V500-5	CAST Inc
	pixels	células	MHz	milhões de amostras/s		

O soft IP para a implementação do Compressor JPEG2000 desenvolvido pela Barco Silex chama-se BA112JPEG2000E Encoder (BARCO, 2005-c) e seus resultados de síntese para diversos dispositivos é mostrado na Tabela 4.3.

O soft IP para a implementação do Compressor JPEG2000 desenvolvido pela CAST Inc chama-se JPEG2K_E (CAST, 2005-b) e seus resultados de síntese para diversos dispositivos é mostrado na Tabela 4.4.

Tabela 4.3: Soft IPs para a implementação do compressor JPEG2000 da Barco Silex

IP	frame	área	fmáx	Número de Oitavas	dispositivo
BA112JPEG2000E	128X128	23.372	114	5	EP1S30FC780C5
BA112JPEG2000E	128X128	9.445	118	5	EP1S20F484C5
	pixels	células	MHz		

Tabela 4.4: Soft IPs para a implementação do compressor JPEG2000 da CAST Inc

IP	frame	área	fmáx	Taxa de operação	dispositivo
JPEG2K_E	256X256	13.432	74	7,4M	Virtex-E V1600E-8
JPEG2K_E	256X256	12.935	106	10,6M	Virtex-II 2V3000-6
JPEG2K_E	256X256	12.885	107	10,7M	Virtex-II Pro 2V30-7
	pixels	células	MHz	milhões de amostras/s	

5 ARQUITETURA DO COMPRESSOR DE IMAGENS COM PERDAS

A implementação da arquitetura para um compressor de imagens completo com perdas, foi dividida em dois blocos. O primeiro bloco representa a arquitetura 2D-DWT irreversível, a qual foi desenvolvida utilizando coeficientes Daubechies 9/7 conforme o padrão e uma etapa de quantização agregada ao bloco da transformada.

O segundo bloco representa a arquitetura desenvolvida para o codificador de entropia. Este codificador foi implementado em duas partes, um bloco para a geração da informação de probabilidade de cada bit e um bloco para realizar a codificação aritmética.

O desenvolvimento da 2D-DWT se baseia na aplicação da 1D-DWT na primeira dimensão da imagem, seguida da aplicação na segunda dimensão da imagem. Assim, a implementação da 2D-DWT requer o desenvolvimento de uma arquitetura eficiente para a computação da 1D-DWT. Este capítulo apresenta a implementação de várias arquiteturas para a computação da 1D-DWT, bem como a síntese de uma arquitetura para a 2D-DWT. O bloco da transformada poderia, a princípio, ser implementado por meio de um banco de filtros, porém a arquitetura *lifting* proporciona maior eficiência, devido à redução da quantidade de operadores aritméticos.

O bloco quantizador se resume em uma divisão inteira dos coeficientes da transformada e como a última etapa do bloco da DWT (na arquitetura *lifting*) é uma multiplicação por constante, a incorporação da etapa de quantização à multiplicação final do bloco da DWT proporciona redução de hardware, mantendo as mesmas características de atraso global, logo proporciona o aumento na eficiência da arquitetura global.

Na arquitetura do codificador de entropia, o gerador de probabilidades, também chamado de gerador de contextos, foi desenvolvido a partir do algoritmo EBCOT, enquanto o codificador aritmético é uma variação do algoritmo QM *coder*.

5.1 Implementação da Transformada Wavelet Discreta

Seguindo a propriedade da separabilidade (USEVITCH, 2001) (GONZALES, 1992), uma 2D-DWT pode ser implementada pela aplicação consecutiva da 1D-DWT a todas as linhas (1ª dimensão) seguida da aplicação da 1D-DWT a todas as colunas (2ª dimensão) de um *tile* da imagem, conforme definido no padrão JPEG2000 (ITU-T, 2000).

O padrão define um *tile* como um arranjo retangular de pontos em uma grade de referência, orientados a partir do *offset* da origem da grade de referência e definidos por um valor de largura e altura. Ou seja, é uma porção bidimensional da imagem original, podendo conter toda a imagem original.

A utilização de técnicas para acelerar a computação da DWT, como o RPA mostrado na seção 4.2, proporciona uma redução significativa no tempo de computação de um DWT com mais de uma oitava. A utilização desta técnica produz o resultado mostrado na Figura 5.1 (SALOMON, 2000) e a arquitetura necessária para produzir este resultado possui uma estrutura simplificada, devido à utilização do RPA. Porém este tipo de arranjo não está previsto no padrão JPEG2000 (no padrão JPEG2000, cada sub-banda possui metade da resolução verticar e metade da resolução horizontal da sub-banda de ordem n-1).

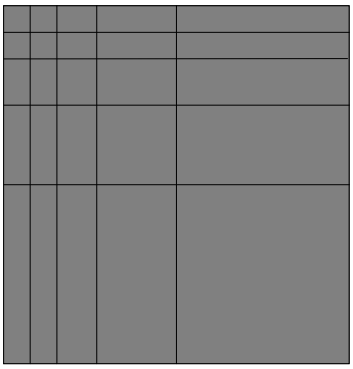

Figura 5.1: 2D-DWT de 4 oitavas utilizando o RPA.

Com a não utilização de técnicas como o RPA na computação da 2D-DWT, é necessária a inclusão de memória suficiente para armazenar todo o *tile* da imagem a ser compactado, deste modo, a computação de cada coeficiente de cada oitava segue a seguinte seqüência:

- as amostras são lidas continuamente da memória para a etapa da transformada;
- a cada passo a etapa da transformada (1D-DWT) gera um valor de aproximação (saída passa-baixas) e um valor de detalhes (saída passa-altas);
- os valores computados de aproximação e detalhe são armazenados na memória.

Em compressão de dados é requerida a ocorrência de várias oitavas de uma DWT, a fim de proporcionar uma maior redução na magnitude dos coeficientes. A cada oitava computada, o sinal de aproximação é re-inserido na entrada da arquitetura, gerando dois novos sinais de aproximação e detalhes.

A arquitetura implementada para a 1D-DWT que apresenta menor complexidade, em nível de descrição de hardware, utiliza banco de filtros, onde a implementação dos dois filtros digitais (passa-altas e passa-baixas) é feita através de registradores, multiplicadores e somadores.

Com o objetivo de reduzir os requisitos de hardware da arquitetura, a fatorização dos coeficientes em um esquema de *lifting* é recomendada no padrão JPEG2000 (ITU-T, 2000).

5.1.1 Banco de filtros

Um banco de filtros, segundo SALOMON (2000), pode ser definido como um conjunto de vários filtros digitais operando sobre os mesmos dados de entrada.

A 1D-DWT irreversível utilizada no padrão JPEG2000 é descrita por um conjunto de dois filtros que utilizam os coeficientes de Daubechies 9/7 (ITU-T, 2000), que são mostrados na Tabela 5.1.

Tabela 5.1: Coeficientes Daubechies 9/7 para banco de filtros FIR

i	h_i	g_i
0	0,6029490182363579	0,115087052456994
±1	-0,2668641184428723	0,5912717631142470
±2	-0,07822326652898785	-0,05754352622849957
±3	0,01686411844287495	-0,09127176311424948
±4	0,02674875741080976	

De acordo com SALOMON (2000), vários conjuntos de coeficientes podem ser utilizados para a computação de uma DWT, porém para proporcionar uma compressão de dados de alta eficiência, apenas poucos conjuntos de dados podem ser utilizados. Desta forma, Para implementação da arquitetura da 1D-DWT irreversível, foi utilizado um banco de filtros utilizando os coeficientes Daubechies 9/7 (ITU-T, 2000). Este conjunto de coeficientes requer um filtro passa-baixas com 9 TAPs e um filtro passa-altas com 7 TAPs, conforme mostrado na Figura 5.2.

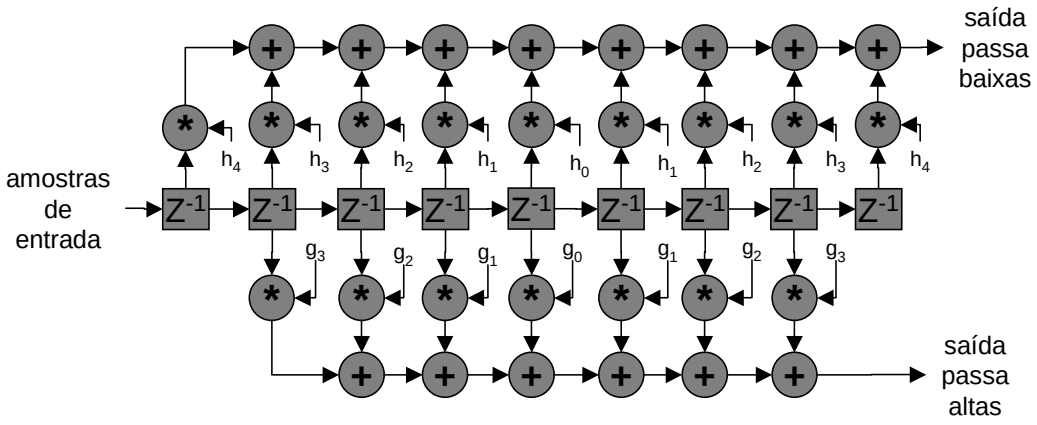

Figura 5.2: 1D-DWT por banco de filtros FIR.

A arquitetura da 1D-DWT mostrada na Figura 5.2 apresenta:

- 8 registradores de n-bits;
- 16 multiplicadores de n bits;
- 14 somadores.

Esta arquitetura apresenta latência proporcional a quantidade de TAPs do filtro, ou seja, latência de 8 ciclos do sinal de *clock*. Considerando que em sua entrada temos um *stream* de k amostras, na saída passa-baixas são gerados k+#h-1 coeficientes e na saída passa-altas são gerados k+#g-1 coeficientes, onde #h e #g representam a profundidade de cada filtro. Neste caso temos um acréscimo de coeficientes, sinalizando um condição que deve ser evitada.

Para evitar o acréscimo de coeficientes, a técnica de espelhamento das bordas com subseqüente eliminação dos coeficientes irrelevantes, mostrado na seção 4.1, se mostra eficiente. Com a utilização do espelhamento das bordas de tamanho igual a m, para cada stream de k amostras são gerados k+2m+#h-1 coeficientes na saída passa-baixas e k+2m+#g-1 coeficientes na saída passa-baixas, porém os coeficientes extras, à esquerda e à direita do *stream* de coeficientes de saída podem ser descartados sem perda de informação.

5.1.2 Lifting

O padrão JPEG2000 (ITU-T, 2000) recomenda o uso da técnica de *lifting* descrevendo o processo e os coeficientes utilizados em sua implementação.

A utilização desta técnica proporciona a redução do custo em hardware dos filtros digitais ao custo do aumento da latência total da arquitetura. Esta técnica consiste na fatorização dos coeficientes da transformada (SWELDENS, 1995), de modo a permitir que seja produzido o mesmo resultado utilizando um conjunto menor de operadores aritméticos, porém causa o aumento da latência dos dados de saída.

A Tabela 5.2 apresenta os coeficientes em formato de ponto flutuante recomendados no padrão JPEG2000 (ITU-T, 2000).

Tabela 5.2: Coeficientes Daubechies 9/7 para *Lifting*

Símbolo	*valor*
α	-1,586134342
β	-0,052980118
γ	0,882911075
δ	0,443506852
K	1,230174105

O padrão JPEG2000 (ITU-T, 2000) apresenta o cálculo da 1D-DWT a partir da seguinte seqüência de operações:

$Y(2n+1)$	$\leftarrow$	$X_{ext}(2n+1) + (\alpha (X_{ext}(2n) + X_{ext}(2n+2)))$	[STEP1]
$Y(2n)$	$\leftarrow$	$X_{ext}(2n) + (\beta (Y(2n-1) + Y(2n+1)))$	[STEP2]
$Y(2n+1)$	$\leftarrow$	$Y(2n+1) + (\gamma (Y(2n) + Y(2n+2)))$	[STEP3]
$Y(2n)$	$\leftarrow$	$Y(2n) + (\delta (Y(2n-1) + Y(2n+1)))$	[STEP4]
$Y(2n+1)$	$\leftarrow$	$-K\ Y(2n+1)$	[STEP5]
$Y(2n)$	$\leftarrow$	$(1/K)\ Y(2n)$	[STEP6]

Do modo como está apresentada em ITU-T (2000), esta seqüência de operações deve ser executada em ordem para cada vetor de dados de entrada. O vetor de entrada é um vetor unidimensional definido por X_{ext} e o vetor de saída Y apresenta os resultados corretos após o passo 6.

Para permitir que o processamento das operações ocorra em paralelo, e assim aumentar a performance do sistema, foram acrescentados vetores de dados extras, de modo que a cada dois passos os dados de saída são carregados em um vetor diferente. A seguir é mostrada a alteração da seqüência original utilizando um vetor de dados intermediários Y' e um vetor de dados finais Y'':

$Y(2n+1)$	$\leftarrow$	$X_{ext}(2n+1) + (\alpha (X_{ext}(2n) + X_{ext}(2n+2)))$	[STEP1]
$Y(2n)$	$\leftarrow$	$X_{ext}(2n) + (\beta (Y(2n-1) + Y(2n+1)))$	[STEP2]
$Y'(2n+1)$	$\leftarrow$	$Y(2n+1) + (\gamma (Y(2n) + Y(2n+2)))$	[STEP3]
$Y'(2n)$	$\leftarrow$	$Y(2n) + (\delta (Y'(2n-1) + Y'(2n+1)))$	[STEP4]
$Y''(2n+1)$	$\leftarrow$	$-K\ Y'(2n+1)$	[STEP5]
$Y''(2n)$	$\leftarrow$	$(1/K)\ Y'(2n)$	[STEP6]

Esta seqüência de operações foi transcrita para o gráfico da Figura 5.3.

O primeiro passo do algoritmo tem por entrada o vetor de dados Xext, este vetor apresenta todos os dados de uma linha ou coluna de um *tile* da imagem. Cada duas amostras pares são somadas e multiplicadas pelo coeficiente alfa e após são somadas com a amostra ímpar de Xext (amostra de índice intermediário às amostras pares). O resultado passa a ser o vetor de amostras intermediárias ímpares de primeiro nível (Y).

O segundo passo do algoritmo tem por entrada os coeficientes ímpares do vetor Y gerado no primeiro passo e os coeficientes pares do vetor de dados Xext. Cada duas amostras ímpares do vetor Y são somadas e multiplicadas pelo coeficiente beta e após são somadas com a amostra par do vetor de dados Xext (amostra de índice intermediário às amostras ímpares). O resultado passa a ser a amostra intermediária par de primeiro nível (Y).

As amostras intermediárias pares e ímpares de primeiro nível compõem o vetor intermediário Y.

O terceiro passo do algoritmo tem por entrada o vetor intermediário Y. Cada duas amostras pares são somadas e multiplicadas pelo coeficiente gama e após são somadas com a amostra ímpar de Y (amostra de índice intermediário às amostras pares). O resultado passa a ser o vetor de amostras intermediárias ímpares de segundo nível (Y').

O quarto passo do algoritmo tem por entrada os coeficientes ímpares do vetor Y' gerado no terceiro passo e os coeficientes pares do vetor intermediário Y. Cada duas amostras ímpares do vetor Y' são somadas e multiplicadas pelo coeficiente delta e após são somadas com a amostra par do vetor intermediário Y (amostra de índice intermediário às amostras ímpares). O resultado passa a ser a amostra intermediária par de segundo nível (Y').

As amostras intermediárias pares e ímpares de segundo nível compõem o vetor intermediário de segundo nível Y'.

O quinto passo corresponde a uma multiplicação das amostras ímpares do vetor Y' pelo coeficiente K multiplicado por -1. Resultando nas amostras ímpares do vetor de saída Y'' e o sexto passo corresponde a uma divisão das amostras pares do vetor Y' pelo coeficiente K. Resultando nas amostras pares do vetor de saída Y''.

O vetor Y'' corresponde ao vetor de saída da transformada.

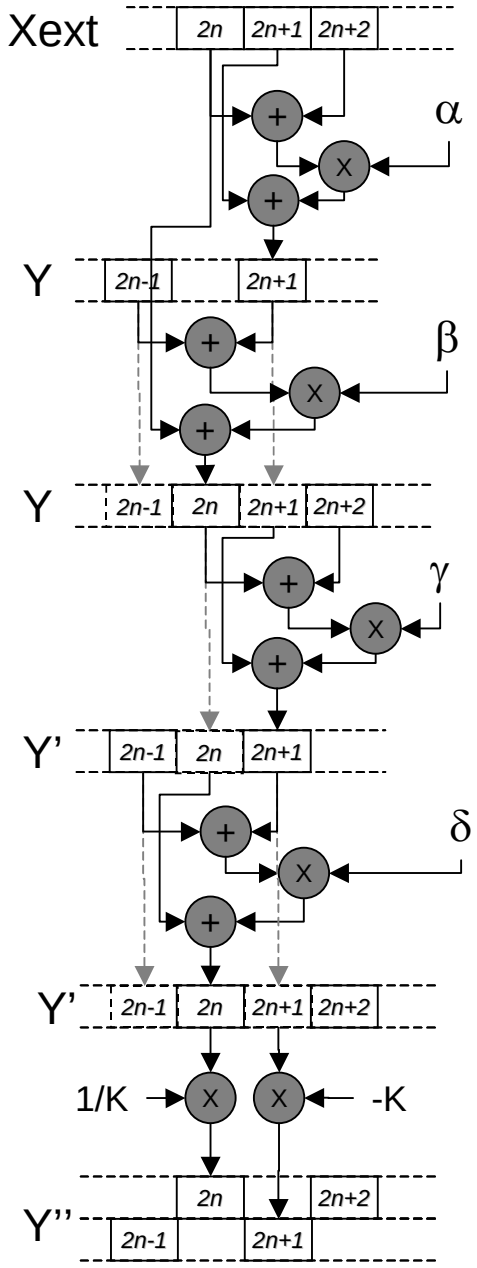

Figura 5.3: Gráfico de operações para o cálculo da 1D-DWT.

A partir do algoritmo descrito na Figura 5.3 nota-se que a seqüência:

- soma de dois registros (de índices x e x+2);
- multiplicação da resultante por um coeficiente constante;
- soma da resultante com o registro de índice intermediário (de índice x+1).

se repete nos quatro primeiros passos do algoritmo, logo cada passo pode ser reduzido ao operador AMA (*Adder-Multiplier-Adder*) mostrado na Figura 5.4. Os dois últimos passos do algoritmo podem ser substituídos por uma multiplicação simples.

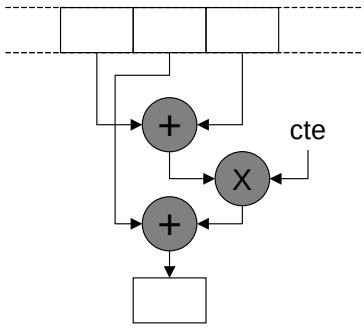

Figura 5.4: Operador AMA para a Arquitetura *lifting*.

A implementação direta do algoritmo da Figura 5.3 é apresentado na arquitetura *pipeline* da Figura 5.5. Esta arquitetura possui latência de 7 ciclos e a cada dois ciclos um dado é gerado nas saídas passa-altas e passa-baixas.

Esta arquitetura apresenta apenas 4 operadores AMA descritos na Figura 5.4 (com constantes alfa, beta gama e delta) e dois operadores de multiplicação (por -K e por 1/K). Os registradores S0, S1, S2, S3, S4, S5 e S6 são utilizados para isolar cada um dos operadores AMA, reduzindo assim o atraso geral da arquitetura da transformada 1D, de modo que o ponto de maior atraso ocorre no interior de um dos operadores AMA.

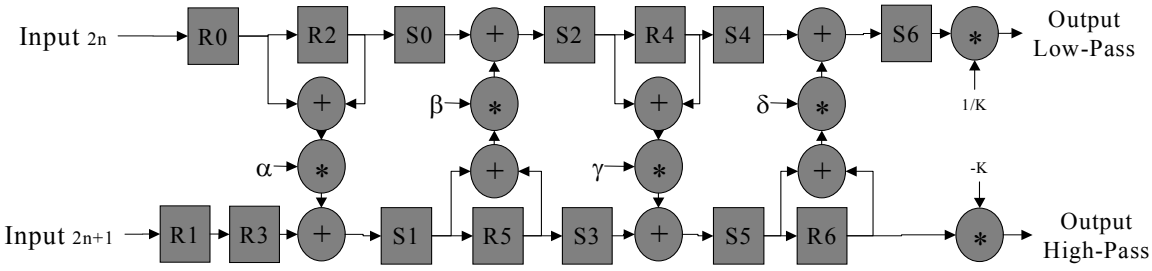

Figura 5.5: Arquitetura *lifting* 1D-DWT.

Nesta arquitetura, o Operador AMA que apresentar o maior atraso produz o pior caso para o atraso geral da arquitetura, isso significa que a otimização deste operador AMA representa a otimização do tempo de atraso da arquitetura.

A Tabela 5.3 apresenta os 6 operadores previstos no padrão JPEG2000 (ITU-T, 2000) bem como a representação inteira arredondada dos valores.

Uma transformada irreversível é aquela na qual seu resultado aplicado a uma transformada inversa não reproduz o valor original, com isso temos a geração de um sinal de erro. A utilização das constantes da transformada com valor em ponto flutuante insere automaticamente certo nível de erro no resultado da transformada impedindo a obtenção dos valores originais por meio da transformada inversa e o arredondamento destas constantes apenas aumenta o valor deste erro, porém como a transformada utilizada na compressão com perdas prevê a existência de algum erro no resultado, o erro inserido nas constantes pode ser admissível. O processo seguinte à transformada é a

etapa de quantização, que aproxima os coeficientes transformados a valores pré-definidos, ocorrendo então a inserção de outro nível de erro no valor resultante.

Tabela 5.3: Constantes *Lifting*

Coeficiente	Valor em Ponto Flutuante	Valor inteiro arredondado	Representação em Binário
alpha	-1.586 134 342	-406/256	10.01101010
beta	-0.052 980 118	-14/256	11.11110010
gamma	0.882 911 075	226/256	00.11100010
delta	0.443 506 852	114/256	00.01110001
-k	-1.230 174 105	-314/256	10.11000101
1/k	0.812 893 066	208/256	00.11010000

As constantes inteiras arredondadas da Tabela 5.3 foram obtidos a partir de uma multiplicação por 256. A escolha desta constante de multiplicação vem pela facilidade de implementação de uma arquitetura para dividir o resultado das operações básicas por 256, ou seja, um deslocamento de 8 bits para a direita.

A partir das constantes apresentadas na Tabela 5.3 diversas arquiteturas de Operadores Básicos foram desenvolvidas. Cada uma das arquiteturas apresenta vantagens e desvantagens que são discutidas nas seções seguintes.

5.1.2.1 Operador AMA por descrição comportamental utilizando multiplicação de inteiros

A utilização de operadores inteiros proporciona a redução do hardware da arquitetura implementada, em relação à utilização de operadores em ponto flutuante. Sua precisão pode ser definida pela taxa de erro entre o valor da constante inteira dividida por 256 (deslocamento de 8 bits para a direita) e o valor da constante em ponto flutuante.

A descrição em VHDL comportamental é, a princípio, genérica para qualquer tipo de FPGA, porém a síntese é totalmente dependente dos recursos de hardware do dispositivo FPGA no qual será mapeada a arquitetura. Assim, a ferramenta de síntese acrescenta as funções de bibliotecas específicas para o FPGA utilizado.

O quadro abaixo mostra parte da descrição comportamental em VHDL do Operador AMA_ALFA, que é utilizado no passo 1 do algoritmo da transformada. De acordo com a Tabela 5.3, o valor de alfa é –406/256, assim a arquitetura deve realizar a operação de soma entre R2 e R3 seguida da multiplicação pela constante –406 e conseqüente divisão por 256 e por último realizar a soma com R1. As operações de multiplicação e divisão inteiras ocorrem utilizando a definição de números inteiros da linguagem.

```
CONSTANT ALFA  : INTEGER := -406;
BEGIN
Output <=      R1 + ((ALFA * (R2 + R3))/256);
END
```

Conforme o quadro acima, a principal vantagem de uma descrição comportamental é a simplicidade na descrição do algoritmo, pois a operação do operador AMA pode ser descrita em uma única linha. Entretanto, por utilizar a operação de multiplicação, esta arquitetura requer a utilização de macro-funções para a execução da operação descrita, de modo que a síntese será específica para um tipo de dispositivo FPGA. Outra desvantagem deste tipo de implementação é que os operadores possuem formato inteiro, de modo que a macro-função que realizar a multiplicação consome uma quantidade significativa de recursos de hardware e com isso apresenta o maior atraso entre todas as arquiteturas implementadas.

Nesta arquitetura não está prevista a redução da quantidade de bits dos operadores envolvidos, resultando assim em uma arquitetura com uma maior utilização de hardware. Esta maior utilização de hardware juntamente com o maior tamanho dos operadores, resulta em maior quantidade de conexões dentro do FPGA, resultando em maior atraso na propagação dos sinais, devido à limitação de conexões rápidas do FPGA.

A síntese de arquitetura do bloco 1D-DWT utilizando esta técnica com a ferramenta ALTERA QUARTUS II direcionada para a família APEX 20KE requer 781 LEs e pode operar a até 16,6MHz, significando um processamento de 16,6 milhões de *pixels* por segundo com o *pipeline* cheio.

5.1.2.2 Operador AMA por descrição comportamental utilizando multiplicação por soma de inteiros deslocados

Uma multiplicação binária pode ser descrita como uma soma deslocada, assim uma maneira simples de reduzir a quantidade de hardware do Operador AMA é limitar a quantidade de somas, arredondando a constante de multiplicação.

O Operador AMA que realiza a multiplicação por alfa implementa a seguinte operação:

Output = R1+alfa(R2+R3)*

O Operador AMA utilizando multiplicação por somas deslocadas utiliza apenas operandos inteiros, logo todas as operações devem ser deslocadas para a esquerda e ao final arredondadas para uma quantidade de bits predefinida. O valor arredondado de alfa em binário, segundo a Tabela 5.3, é 10.01101010, ao ocorrer deslocamento de 8 bits para a esquerda, o valor se torna 1001101010. A quantidade de operadores deslocados (deslocamento do multiplicando) de um Operador AMA é igual ao número de bits em nível alto (Hi) do coeficiente da transformada (multiplicador), ou seja, para o coeficiente alfa são necessários 5 operadores deslocados e cada operador deve ser deslocado de acordo com a posição do respectivo bit em nível alto do coeficiente alfa, conforme a seguinte relação:

Operador AMA-alfa:

$S = (R2+R3)$

$Output = R1 + (S*2^1 + S*2^3 + S*2^5 + S*2^6 - S*2^9)/2^8$

Operador AMA-beta:

S = (R2+R3)

Output = R1 + $(S*2^1 + S*2^4 + S*2^5 + S*2^6 + S*2^7 + S*2^8 - S*2^9)/2^8$

Operador AMA-gama:

S = (R2+R3)

Output = R1 + $(S*2^1 + S*2^5 + S*2^6 + S*2^7)/2^8$

Operador AMA-delta:

S = (R2+R3)

Output = R1 + $(S*2^1 + S*2^4 + S*2^5 + S*2^6)/2^8$

Multiplicação por -K:

Output = $(Input*2^0 + Input*2^2 + Input*2^6 + Input*2^7 - Input*2^9 +)/2^8$

Multiplicação por 1/K:

Output = $(Input*2^4 + Input*2^6 + Input*2^7)/2^8$

Conforme a relação dada, a Figura 5.6 mostra a arquitetura do Operador AMA-Alfa utilizando somas deslocadas.

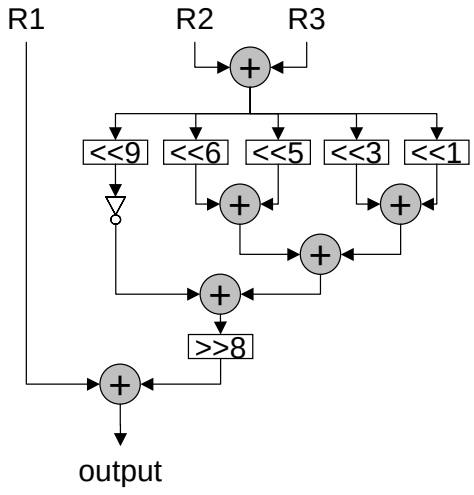

Figura 5.6: Operador AMA para multiplicação por alfa.

Nesta abordagem comportamental, o procedimento utilizado para deslocar os operadores foi realizar uma multiplicação por 2^X, onde X representa a quantidade de bits que o operador deve ser deslocado. O Operador AMA-alfa mostrado na Figura 5.6, requer 6 operações de adição. Com isso, em sua descrição em VHDL comportamental

foi requerida a utilização de 5 operadores intermediários (X1, X2, X3, X4 e SOMA) e um operador de saída (Output) conforme o algoritmo abaixo.

```
SIGNAL  X1, X2, X3, X4, SOMA  : INTEGER ;
BEGIN
SOMA    <=      R2+R3;

X1      <=      SOMA*2+SOMA*8;
X2      <=      SOMA*32+SOMA*64;
X3      <=      X1+X2;
X4      <=      -SOMA*512+X3)/256;

Output  <=      X4+R1;

END
```

Conforme o quadro acima, esta técnica requer 5 linhas de código VHDL para descrever arquitetura do Operador AMA-alfa. Todas as multiplicações possuem o formato "*operando**2^X" que pode ser implementado por meio de um simples deslocamento do operando. Considerando que o compilador VHDL é capaz de eliminar todos os possíveis *full-adders* que possuem suas entradas fixas, temos que:

Primeiro somador (SOMA): 8 = 8 somadores;

Segundo somador (X1): 32-3 = 29 somadores;

Terceiro somador (X2): 32-6 = 26 somadores;

Quarto somador (X3): 32-6 = 26 somadores;

Quinto somador (X4): 32-9 = 23 somadores;

Sexto somador (Output): 32 = 32 somadores;

Total: 144 somadores.

Os operadores de entrada R1, R2 e R3 estão definidos como operadores de 8 bits, logo o primeiro somador (que realiza o cálculo do sinal SOMA) requer somente 8 somadores (7 *full adders* e 1 *half adder*). O resultado da primeira soma pode resultar em um valor positivo ou negativo, logo, para as somas seguintes, o compilador VHDL aloca todos os possíveis bits a esquerda. Considerando que os dois operandos da soma são deslocados para a esquerda antes da soma e o maior deslocamento é de 3 bits, a quantidade necessária de somadores será o número total de bits da variável (variável inteira requer 32-bits) menos o número de bits deslocados, ou seja, o segundo somador requer 28 *full adders* e 1 *half adder*, o terceiro somador e o quarto somador requerem 25 *full adders* e 1 *half adder* cada um, o quinto somador requer 22 *full adders* e 1 *half adder*, e devido ao deslocamento para a direita que ocorre após o quinto somador (soma de X4) o sexto somador requer 31 *full adders* e 1 *half adder*. Assim são necessários 138 *full adders* e 6 *half adders* para implementar o bloco do Operados AMA-alfa. Porém, descrições comportamentais de somadores, quando sintetizadas para FPGAs, sejam somadores completo ou meio somadores, ocupam apenas uma célula lógica para cada bit somado, pois apesar de cada somador possuir duas saídas (saída SOMA e saída CARRY-OUT), a ferramenta de síntese utiliza as cadeias de *carry* rápido previstas nos FPGAs.

A arquitetura do bloco 1D-DWT sintetizada para um dispositivo da família APEX20KE, utilizando esta técnica utiliza 480 LEs e pode operar a até 44MHz. Utilizando a ferramenta de simulação do QUARTUS II, a partir de uma porção da imagem "Lena" foi estimado um consumo de potência de 248mW a uma taxa de operação de 15MHz.

Em comparação com a arquitetura da seção 5.1.2.1, esta arquitetura apresenta 39% menos área, freqüência máxima de operação 2,6 vezes maior e consumo médio de potência 205 menor. Assim, a arquitetura da seção 5.1.2.1 possui somente a vantagem de rapidez na descrição em VHDL.

5.1.2.3 Operador AMA por descrição estrutural utilizando multiplicação por soma de inteiros deslocados

Um dos inconvenientes da utilização da abordagem comportamental diz respeito à quantidade de bits definidos para os operadores. A seção 5.1.2.2 mostra a implementação de um Operador AMA utilizando operadores inteiros, como em VHDL o tipo inteiro corresponde à faixa compreendida entre -2147483648 e 2147483647, ou seja, valores de 32 bits, todos os operadores requerem desnecessariamente 32 bits, utilizando uma quantidade elevada de área.

Uma implementação estrutural requer maior definição nas operações realizadas, porém permite que nesta definição sejam previstas todas as estruturas utilizadas, ou seja, permite controle completo sobre a síntese da arquitetura. Considerando que a quantidade de bits das amostras de entrada é conhecida é possível prever a quantidade de bits para cada operador, bem como a quantidade de bits de saída de cada Operador AMA.

O Operador AMA utilizando o coeficiente alfa, mostrada na Figura 5.6, para amostras de entrada de 8 bits requer 6 operações de soma, porém utilizando uma quantidade otimizada de bits.

A estrutura mostrada na Figura 5.7 mostra uma representação onde os operadores de entrada R1, R2 e R3 possuem 8 bits, o operador de saída (chamado simplesmente de R) possui 11 bits e os operadores internos possuem 18 bits. Esta estrutura prevê o deslocamento de cada operador interno a partir do valor em binário de alfa, conforme a Figura 5.6.

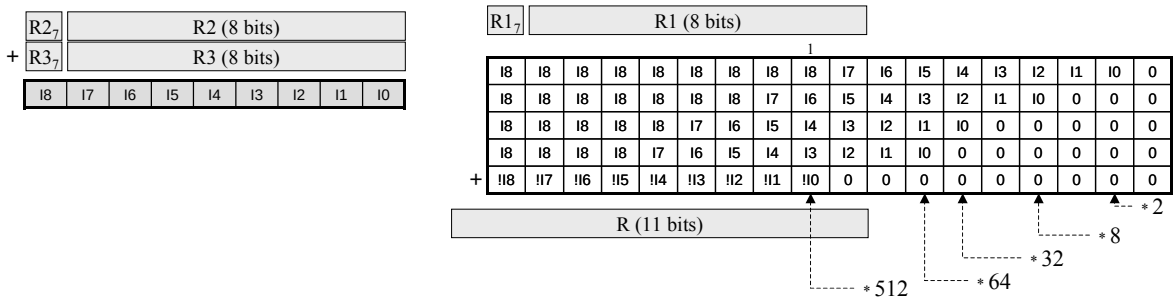

Figura 5.7: Descrição da operação do Operador AMA alfa.

Os operadores internos da estrutura mostrada na Figura 5.7 possuem muitos bits zerados à direita e muitos bits repetidos (I8) à esquerda, gerando hardware desnecessário. Como citado acima, o Operador AMA alfa requer 6 operações de soma, porém essas operações podem ser realizadas com menos de 18 bits (considerando amostras de entrada de 8 bits). A Figura 5.8 mostra a estrutura de alguns dos somadores necessários para o Operador AMA alfa, onde é possível notar que o número de *full-adders* necessários para a operação de soma entre dois operadores internos é sempre igual ao número de bits do operando de maior magnitude.

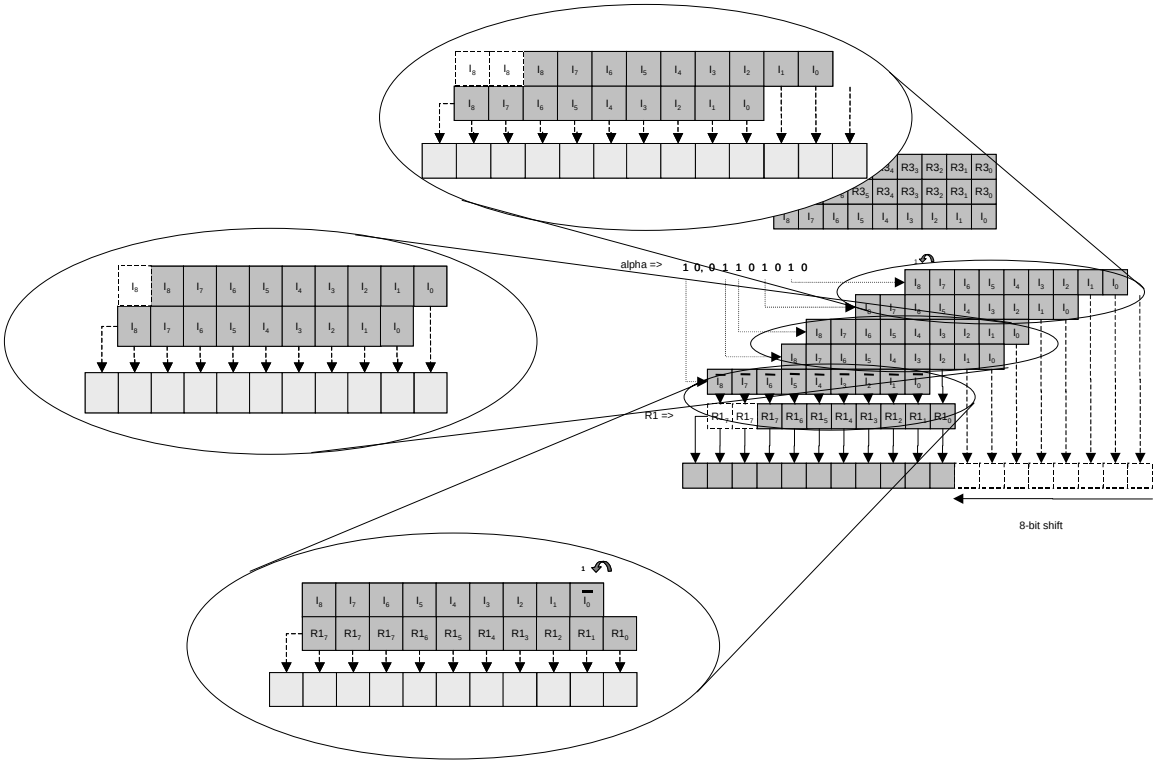

Figura 5.8: Estrutura de somas internas para o Operador AMA alfa.

Assim, considerando a redução do número de bits, temos que:

Primeiro somador:	8 *full-adders* + 1 *half-adder*;
Segundo somador:	8 *full-adders* + 1 *half-adder*;
Terceiro somador:	8 *full-adders* + 1 *half-adder*;
Quarto somador:	9 *full-adders* (requer a soma de +1, conforme a Figura 5.8);
Quinto somador:	10 *full-adders* + 1 *half-adder*;
Sexto somador:	12 *full-adders* + 1 *half-adder*;

Total: 55 full-adders e 5 half-adders.

Esta arquitetura representa a utilização de somente 40% dos recursos de hardware utilizados na arquitetura da seção 5.1.2.2. Esta redução de hardware afeta diretamente o atraso global da arquitetura e o consumo de potência. Entretanto, quando sintetizada para FPGAs, a arquitetura estrutural não utiliza as vantagens estruturais do FPGA. Assim um bloco de soma de 1-bit (*full-adder* ou *half-adder*) requer uma célula lógica quando sintetizado a partir de uma descrição comportamental, enquanto o mesmo bloco sintetizado a partir de uma descrição estrutural requer duas células lógicas. Duas das principais vantagens das descrições comportamentais são simplicidade e rapidez de prototipação, porém, dificultam a síntese para ASIC.

Uma maneira de reduzir o hardware deste tipo de arquitetura é utilizar operações de soma com número reduzido de deslocamentos, proporcionando operadores intermediários menores. Um Operador AMA simplificado segundo este método é mostrado na Figura 5.9.

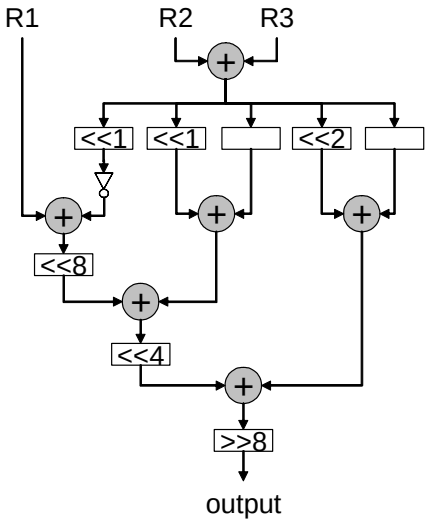

Figura 5.9: Operador AMA alfa simplificado.

Diferentemente de VHDL comportamental, onde todas as operações aritméticas são feitas de maneira automática pela ferramenta de síntese, a implementação da arquitetura em VHDL estrutural requer a descrição detalhada de todos os operadores de soma.

Cada operador de soma (considerando aritmética com sinal) possui dois vetores para as parcelas de entrada, um vetor para o resultado da soma, chamados `Xn` e um vetor para armazenar o valor dos *carrys* intermediários, chamado `CARRY_n`.

Apesar da soma total requerer um vetor de 19 bits, considerados antes do truncamento, neste algoritmo, para simplicidade de implementação automática utilizando o comando GENERATE, todos os sinais (variáveis) possuem 20 bits, e o 20º bit é um bit de *carry* que será descartado na síntese. Assim, a ferramenta de síntese realiza uma eliminação dos bits não utilizados, ou seja, nas somas onde, pelo menos um dos bits de entrada possui valor fixo (nível lógico alto ou nível lógico baixo).

```
SIGNAL  X1, X2, X3, X4, X5, X6, X7,
       CARRY_1, CARRY_2, CARRY_3, CARRY_4, CARRY_5, CARRY_6 :STD_LOGIC_VECTOR(19 DOWNTO 0);
BEGIN
------------------------------------ X1 ------------------------------------------------
aX1a:FOR X IN 0 TO 7 GENERATE
       X1(X)  <=     R1(X);
END GENERATE;
aX1b:FOR X IN 8 TO 18 GENERATE
       X1(X)  <=     R1(7);
END GENERATE;
------------------------------------ X2 ------------------------------------------------
CARRY_1(0)     <=     '0';
aX2a:FOR X IN 0 TO 7 GENERATE
  ADDER: FULL_ADDER
    PORT MAP(CLK=>CLK,A=>R2(X),B=>R3(X),Cin=>CARRY_1(X),S=>X2(X),Cout=>CARRY_1(X+1));
END GENERATE;
aX2b:FOR X IN 8 TO 18 GENERATE
       X2(X)  <=     CARRY_1(8);
END GENERATE;
------------------------------------ X3 ------------------------------------------------
CARRY_2(1)     <=     '1';
X3(0)          <=     X1(0);
aX3a:FOR X IN 1 TO 18 GENERATE
  ADDER: FULL_ADDER
    PORT MAP(CLK=>CLK,A=>X1(X),B=>NOT X2(X-1),Cin=>CARRY_2(X),S=>X3(X),Cout=>CARRY_2(X+1));
END GENERATE;
------------------------------------ X4 ------------------------------------------------
CARRY_3(1)     <=     '0';
X4(0)          <=     X2(0);
aX4:FOR X IN 1 TO 18 GENERATE
  ADDER: FULL_ADDER
    PORT MAP(CLK=>CLK,A=>X2(X),B=>X2(X-1),Cin=>CARRY_3(X),S=>X4(X),Cout=>CARRY_3(X+1));
END GENERATE;
------------------------------------ X5 ------------------------------------------------
CARRY_4(2)     <=     '0';
aX5a:FOR X IN 0 TO 1 GENERATE
       X5(X)  <=     X2(X);
END GENERATE;
aX5b:FOR X IN 2 TO 18 GENERATE
  ADDER: FULL_ADDER
    PORT MAP(CLK=>CLK,A=>X2(X),B=>X2(X-2),Cin=>CARRY_4(X),S=>X5(X),Cout=>CARRY_4(X+1));
END GENERATE;
------------------------------------ X6 ------------------------------------------------
CARRY_5(3)     <=     '0';
aX6a:FOR X IN 0 TO 2 GENERATE
       X6(X)  <=     X4(X);
END GENERATE;
aX6b:FOR X IN 3 TO 18 GENERATE
  ADDER: FULL_ADDER
    PORT MAP(CLK=>CLK,A=>X3(X-3),B=>X4(X),Cin=>CARRY_5(X),S=>X6(X),Cout=>CARRY_5(X+1));
END GENERATE;
------------------------------------ X7 ------------------------------------------------
CARRY_6(4)     <=     '0';
aX8a:FOR X IN 0 TO 3 GENERATE
       X7(X)  <=     X5(X);
END GENERATE;
aX8b:FOR X IN 4 TO 18 GENERATE
  ADDER: FULL_ADDER
    PORT MAP(CLK=>CLK,A=>X6(X-4),B=>X5(X),Cin=>CARRY_6(X),S=>X7(X),Cout=>CARRY_6(X+1));
END GENERATE;
------------------------------------ Output ------------------------------------------
aOUT:FOR X IN 0 TO 10 GENERATE
       Output(X)      <=     X7(X+7);
END GENERATE;
--------------------------------------------------------------------------------------
END a;
```

Conforme quadro anterior, o vetor X1 é utilizado para estender o tamanho do registrador de entrada R1 de 8-bits para os 19-bits requeridos. O vetor X2 armazena a soma de 8-bits entre os registradores de entrada R2 e R3 e estende o bit de carry, neste caso armazenando o sinal da operação, até o 19º bit.

O vetor X3 armazena a subtração entre X1 e X2, através de uma soma com o segundo elemento invertido e somando 1 ao *carry in* (X1 + not X2 +1). A partir de X3 ocorre as somas deslocadas dos vetores e é implementado pela cópia dos bits menos significativos (esta operação não requer hardware) e somas com índices diferentes (ex: o deslocamento de 1-bit para o cálculo de X4: `X4(X) = X2(X) + X2(X-1)`). Os vetores X3, X4 e X5 realizam as somas deslocadas para implementar os fatores da multiplicação original e os vetores X6 e X7 acumulam os resultados dos vetores anteriores. Na saída deste Operador AMA-alfa ocorre a truncagem do valor do vetor X7 nos 11-bits a partir do 8º bit.

A arquitetura do bloco 1D-DWT sintetizada para um dispositivo da família APEX20KE, utilizando esta técnica ocupa 701 LEs e pode operar a até 54,4 MHz. Utilizando a ferramenta de simulação do QUARTUS II, a partir de uma porção da imagem "Lena" foi estimado um consumo de potência de 232mW a uma taxa de operação de 15MHz.

Comparativamente à arquitetura da seção 5.1.2.2, esta arquitetura apresenta cerca de 45% mais área, entretanto, apesar de ocupar mais área, esta arquitetura apresenta um atraso global menor, possuindo uma melhoria de cerca de 23%, bem como um consumo 7% menor sob a mesma taxa de *clock*.

5.1.2.4 Operador AMA por descrição comportamental utilizando multiplicação de inteiros deslocados em uma arquitetura de Pipeline

A arquitetura lifting 1D-DWT é por concepção em forma de *pipeline* e os pontos de maior atraso do circuito se encontram na estrutura interna dos operadores AMA. Uma maneira de reduzir o atraso destes blocos é particioná-los, ou seja, realizar a inserção de barreiras temporais no interior dos operadores AMA.

Analisando a estrutura interna do operador AMA alfa mostrado na figura 31(a), nota-se que no caminho crítico ocorrem 4 operações de soma. Colocando-se barreiras temporais entre as operações de soma, conforme figura 31(b), é possível reduzir o atraso global da arquitetura.

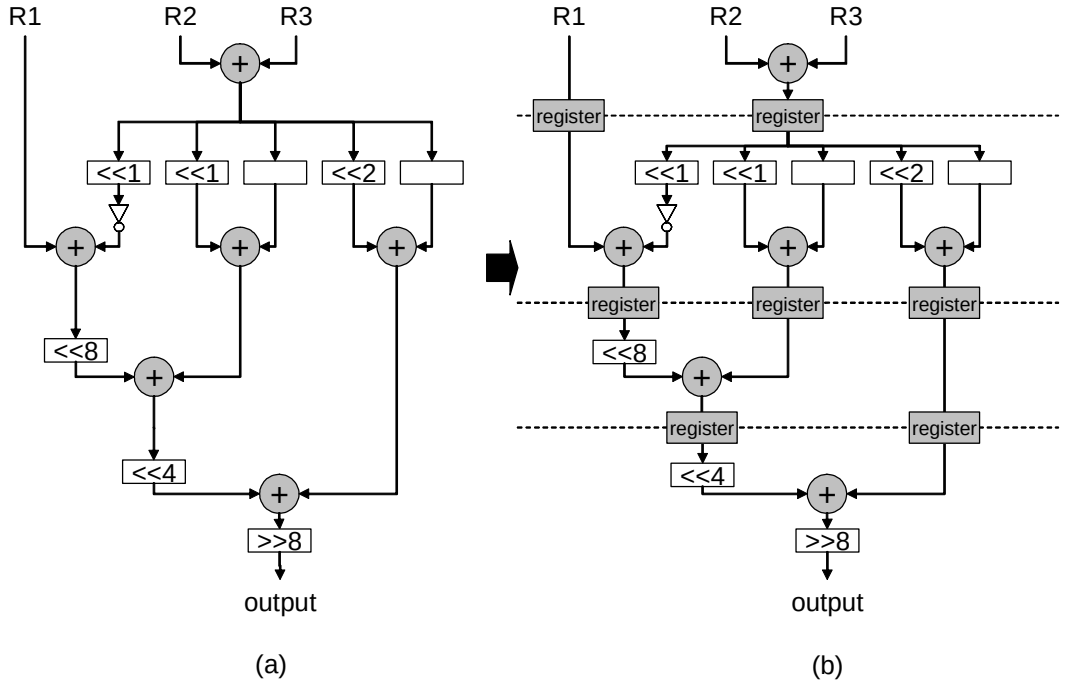

Figura 5.10: Pipelinezação do operador AMA alfa.

A fim de manter a coerência entre os cálculos da 1D-DWT, se faz necessário a inserção de registradores entre os operadores AMA a fim de balancear e ajustar cada um dos fluxos de dados da 1D-DWT. Como observado na figura 32, com o acréscimo de três estágios de *pipeline* nos operadores AMA-alfa, AMA-beta AMA-gama e AMA-delta, os registradores S0, S3, S4 e S7 foram substituídos, cada um, por conjuntos de quatro registradores em serie.

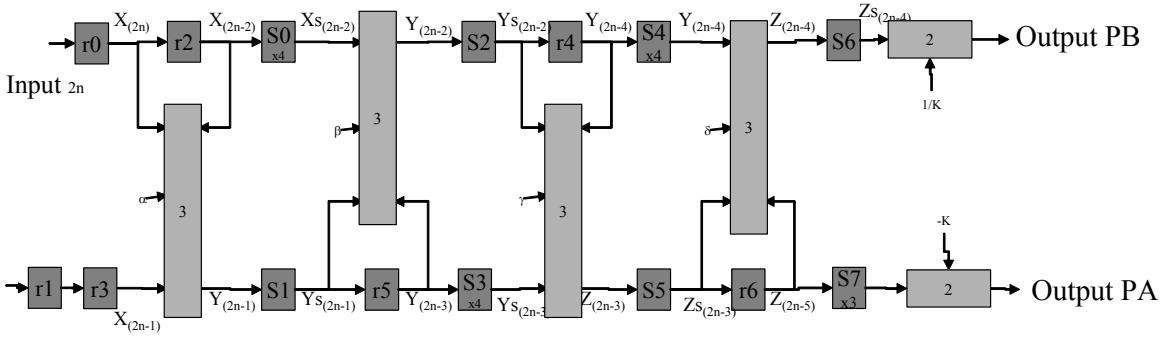

Figura 5.11: Arquitetura *lifting* 1D-DWT *pipeline*.

A alteração requerida na descrição VHDL é mostrada no quadro abaixo. Foi inserida uma barreira temporal, controlada pelo sinal de CLK, entre cada uma das operações de soma.

```
SIGNAL  X1, X2, X3, X4, X5, X6, X7    : INTEGER ;

BEGIN
PROCESS
BEGIN
       WAIT UNTIL CLK = '1';
       X1      <=      R1;
       X2      <=      R2+R3;
       X3      <=      X1+X2*(-2);
       X4      <=      X2*2+X2;
       X5      <=      X2*4+X2;
       X6      <=      X3*8+X4;
       X7      <=      X5;
END PROCESS;

       Output  <=      (X6*16+X7)/128;
END
```

A inserção de registradores como barreira temporal resultou em uma síntese com área de 766 LEs em um dispositivo APEX 20KE, conforme mostrado na Tabela 5.4, significando um aumento de área de 60% em relação à arquitetura com operadores AMA sem *pipeline* utilizando descrição comportamental, mostrado na seção 5.1.2.2. Porém a redução do atraso entre registradores resultou em uma freqüência máxima de operação de 157MHz, representando um ganho de 257%.

Quanto à simulação de potência, a injeção de uma porção da imagem "Lena" resultou em uma potência média de 105mW, representando uma redução de 42% no consumo total em relação à arquitetura da seção 5.1.2.2.

5.1.2.5 Operador AMA por descrição estrutural utilizando multiplicação de inteiros deslocados em uma arquitetura de Pipeline

Como observado na seção 5.1.2.4, a utilização de *pipeline* internamente aos operadores AMA proporciona um aumento na freqüência máxima de operação e uma redução na potência média global da arquitetura, ao custo de aumento na área total.

Assim como na arquitetura descrita na seção 5.1.2.3, cada operador AMA desta arquitetura foi descrita utilizando geração automática de somadores, juntamente com a inserção das barreiras temporais conforme descrito na seção 5.1.2.4.

O quadro abaixo mostra a descrição em VHDL estrutural do operador AMA-alfa utilizando multiplicação de inteiros deslocados em uma arquitetura de *pipeline*. Os sinais X1 a X7 apresentam o uso semelhante ao descrito na seção 5.1.2.4, o sinal X8 é utilizado para realizar a truncagem do valor resultante e os sinais X1a a X7a são os registradores utilizados como barreira temporal cadenciados pelo sinal CLK.

```
SIGNAL  X1, X2, X3, X4, X5, X6, X7, X8, X1a, X2a, X3a, X4a, X5a, X6a, X7a,
       CARRY_1, CARRY_2, CARRY_3, CARRY_4, CARRY_5, CARRY_6: STD_LOGIC_VECTOR(19 DOWNTO 0);
BEGIN
PROCESS
BEGIN
       WAIT UNTIL CLK = '1';
       X1a     <=      X1;
       X2a     <=      X2;
       X3a     <=      X3;
       X4a     <=      X4;
       X5a     <=      X5;
       X6a     <=      X6;
       X7a     <=      X7;
END PROCESS;
```

```
------------------------------------ X1 ------------------------------------------------------------------
aX1a:FOR X IN 0 TO 7 GENERATE
       X1(X)   <=      R1(X);
END GENERATE;
aX1b:FOR X IN 8 TO 18 GENERATE
       X1(X)   <=      '0';
END GENERATE;
------------------------------------ X2 ------------------------------------------------------------------
CARRY_1(0)     <=      '0';
aX2a:FOR X IN 0 TO 7 GENERATE
  ADDER: FULL_ADDER
    PORT MAP(CLK=>CLK,A=>R2(X),B=>R3(X),Cin=>CARRY_1(X),S=>X2(X),Cout=>CARRY_1(X+1));
END GENERATE;
X2(8)   <=      CARRY_1(8);
aX2b:FOR X IN 9 TO 18 GENERATE
       X2(X)   <=      '0';
END GENERATE;
------------------------------------ X3 ------------------------------------------------------------------
CARRY_2(1)     <=      '1';
X3(0)   <=      X1a(0);
aX3a:FOR X IN 1 TO 18 GENERATE
  ADDER: FULL_ADDER
   PORT MAP(CLK=>CLK,A=>X1a(X),B=>NOT X2a(X-1),Cin=>CARRY_2(X),S=>X3(X),Cout=>CARRY_2(X+1));
END GENERATE;
------------------------------------ X4 ------------------------------------------------------------------
CARRY_3(1)     <=      '0';
X4(0)   <=      X2a(0);
aX4:FOR X IN 1 TO 18 GENERATE
  ADDER: FULL_ADDER
   PORT MAP(CLK=>CLK,A=>X2a(X),B=>X2a(X-1),Cin=>CARRY_3(X),S=>X4(X),Cout=>CARRY_3(X+1));
END GENERATE;
------------------------------------ X5 ------------------------------------------------------------------
CARRY_4(2)     <=      '0';
aX5a:FOR X IN 0 TO 1 GENERATE
       X5(X)   <=      X2a(X);
END GENERATE;
aX5b:FOR X IN 2 TO 18 GENERATE
  ADDER: FULL_ADDER
    PORT MAP(CLK=>CLK,A=>X2a(X),B=>X2a(X-2),Cin=>CARRY_4(X),S=>X5(X),Cout=>CARRY_4(X+1));
END GENERATE;
------------------------------------ X6 ------------------------------------------------------------------
CARRY_5(3)     <=      '0';
aX6a:FOR X IN 0 TO 2 GENERATE
       X6(X)   <=      X4a(X);
END GENERATE;
aX6b:FOR X IN 3 TO 18 GENERATE
  ADDER: FULL_ADDER
    PORT MAP(CLK=>CLK,A=>X3a(X-3),B=>X4a(X),Cin=>CARRY_5(X),S=>X6(X),Cout=>CARRY_5(X+1));
END GENERATE;
------------------------------------ X7 ------------------------------------------------------------------
aX7:FOR X IN 0 TO 18 GENERATE
       X7(X)   <=      X5a(X);
END GENERATE;
------------------------------------ X8 ------------------------------------------------------------------
CARRY_6(4)     <=      '0';
aX8a:FOR X IN 0 TO 3 GENERATE
       X8(X)   <=      X7a(X);
END GENERATE;
aX8b:FOR X IN 4 TO 18 GENERATE
  ADDER: FULL_ADDER
    PORT MAP(CLK=>CLK,A=>X6a(X-4),B=>X7a(X),Cin=>CARRY_6(X),S=>X8(X),Cout=>CARRY_6(X+1));
END GENERATE;
------------------------------------ Out -----------------------------------------------------------------
aOUT:FOR X IN 0 TO 10 GENERATE
       Output(X)      <=      X8(X+7);
END GENERATE;
-----------------------------------------------------------------------------------------------------------
END a;
```

A arquitetura do bloco 1D-DWT sintetizada para um dispositivo da família APEX20KE, utilizando esta técnica ocupa 1002 LEs e pode operar a até 105 MHz. Utilizando a ferramenta de simulação do QUARTUS II, a partir de uma porção da

imagem “Lena” foi estimado um consumo de potência de 91,4mW a uma taxa de operação de 15MHz.

Comparativamente à arquitetura da seção 5.1.2.4, esta arquitetura apresenta cerca de 30% mais área e um atraso global de cerca de 93% maior, porém sob mesma taxa de *clock*, apresenta um consumo 13% menor.

5.1.2.6 Resultados das implementações da 1D-DWT

Nesta seção são apresentados os resultados comparativos de síntese e simulação entre as arquiteturas para a 1D-DWT apresentadas neste capítulo. Para permitir uma comparação direta entre os resultados, todas as arquiteturas implementadas na seção 5.1.2 foram descritas, sintetizadas e simuladas para os dispositivos da família APEX 20KE da ALTERA (2004) utilizando a ferramenta QUARTUS II. Todas as simulações de consumo médias realizadas foram feitas sob os mesmos dados de entrada e sob a mesma taxa de *clock*. A Tabela 5.4 apresenta os resultados de síntese e simulação das arquiteturas descritas na seção 5.1.2.

Tabela 5.4: Resultados das Implementações (APEX 20KE).

Arquitetura do Operador Básico	Área utilizada (LEs)	Freqüência Máxima de Operação (MHz)	Consumo médio @15MHz (mW)	Número de Estágios de *Pipeline*
Comportamental com multiplicador inteiro (seção 5.1.2.1)	781	16.6	310	8
Comportamental com multiplicações por somas deslocadas (seção 5.1.2.2)	480	44.0	248	8
Comportamental com operadores com *pipeline* e multiplicações por somas deslocadas (seção 5.1.2.4)	766	157	105	21
Estrutural com multiplicações por somas deslocadas (seção 5.1.2.3)	701	54.4	232	8
Estrutural com operadores com *pipeline* e multiplicações por somas deslocadas (seção 5.1.2.5)	1002	105	91.4	21

De acordo com os resultados apresentados na Tabela 5.4, as arquiteturas com operadores AMA pipelinezados apresentam maior freqüência máxima de operação, proporcionando maior taxa de processamento dos dados, considerando que a cada ciclo do sinal de *clock*, um novo coeficiente é gerado em uma das saídas do bloco da 1D-DWT. Porém o aumento na freqüência máxima de operação ocorre às custas de maior requisito de células lógicas (LEs).

A simulação de fração da imagem Lena resultou que as arquiteturas com Operador AMA pipelinezados apresentaram menor consumo médio de potência a 15MHz, ou seja, para o mesmo valor do sinal de *clock*.

A arquitetura com Operadores Básicos por descrição Comportamental de multiplicador inteiro (seção 5.1.2.1) é a mais simples das descrições realizadas para implementar a 1D-DWT, porém a síntese desta arquitetura se mostrou a de maior consumo de energia.

A arquitetura de menor custo em área foi a descrita de forma comportamental utilizando somas deslocadas de inteiros (arquitetura da seção 5.1.2.2). Esta situação se tornou válida somente pelo fato da ferramenta de síntese prever a utilização de cadeias de carry previstas na estrutura dos elementos lógicos dos FPGAs. Em simulação a 40MHz, esta arquitetura consome 626mW, ou seja, um incremento de 2,7 vezes na freqüência de operação resultou em um aumento de 2,5 vezes no consumo de potência.

A arquitetura da seção 5.1.2.4 é uma implementação comportamental com Operadores AMA descritos como pipeline de somadores deslocados, resultando uma arquitetura 1D-DWT com 21 estágios de pipeline. Conforme a Tabela 5.4, comparado com a arquitetura da seção 5.1.2.2, esta arquitetura apresenta 1,6 vezes mais área e freqüência máxima de operação 3 vezes maior. Quando em simulação à taxa de 128MHz, esta arquitetura consome 808mW, representando 1,3 vezes o consumo médio da arquitetura da seção 5.1.2.2 operando a 40MHz, significando que a arquitetura da seção 5.1.2.4 possui melhor compromisso entre área e consumo por MHz que a arquitetura da seção 5.1.2.2.

A arquitetura da seção 5.1.2.3 é uma implementação estrutural utilizando somas deslocadas de inteiros. Neste tipo de descrição, cada operação deve ser definida em nível de bit, de modo que a síntese não utiliza as cadeias de propagação rápida do *carry*, logo um somador de n-bits requer 2n células lógicas. A princípio seria esperado que a arquitetura da seção 5.1.2.3 necessite duas vezes a área da arquitetura da seção 5.1.2.2, entretanto a arquitetura da seção 5.1.2.3 requer apenas aproximadamente 1,5 vezes a área da arquitetura da seção 5.1.2.2 e ainda possui a freqüência máxima de operação 24% maior. Simulações entre as arquiteturas das seções 5.1.2.2 e 5.1.2.3 sugerem que o consumo de potência se mantém em valores próximos mesmo a taxas de operação maiores.

A arquitetura da seção 5.1.2.5 é uma implementação estrutural com *pipeline* e multiplicações por somas deslocadas nos operadores AMA. Sob mesma freqüência de operação, esta arquitetura apresenta o menor consumo médio de todas as arquiteturas. Comparando com a arquitetura da seção 5.1.2.3, a inserção de pipeline resultou em 43% a mais em hardware, entretanto a freqüência máxima de operação é 93% maior e o consumo médio de potência é 61% menor. Deste modo, desde que exista *hardware* disponível, a inserção de *pipeline* proporciona aumento da eficiência da arquitetura.

Comparando com a arquitetura da seção 5.1.2.4, a utilização de descrição estrutural proporciona 31% de acréscimo em hardware, 32% de redução na freqüência máxima de operação e menor consumo médio de potência de somente 13%. Em simulação na freqüência de 95MHZ esta arquitetura apresenta consumo médio de potência de 476mW, significando que em uma freqüência operação 25% menor (75% da freqüência de 128MHz simulada na arquitetura da seção 5.1.2.4), apresenta um consumo médio de potência 41% menor. Assim a utilização de descrição estrutural somente apresenta vantagens se a implementação é direcionada para construção de um ASIC ou se reduções de até 40% no consumo médio de potência são requeridas.

Em MASUD (2004) é apresentada uma arquitetura que implementa a 1D-DWT por meio de banco de filtros, utilizando 785LEs a uma freqüência máxima de 85,5MHz.

A arquitetura descrita na seção 5.1.2.2 (Operador AMA por descrição comportamental utilizando multiplicação por soma de inteiros deslocados) apresenta área de 61% menor, consumo médio de potência 61% menor que a arquitetura de MASUD (2004), porém apresenta freqüência máxima de operação 51% menor que freqüência máxima de operação de MASUD (2004), ou seja aproximadamente metade. Deste modo pode-se considerar que a arquitetura de MASUD (2004) apresenta uma relação *área x freqüência máxima de operação* levemente inferior à arquitetura da seção 5.1.2.2.

Todas as arquiteturas possuem *pipeline* de no mínimo 8 estágios ou 21 estágios para as arquiteturas que possuem *pipeline* internamente no operador AMA. Assim muitos elementos lógicos (LE) do FPGA espalhados ao longo da arquitetura requerem cadenciamento com um sinal de *clock* global, logo é necessário que o dispositivo FPGA já possua previamente todo o roteamento da árvore de *clock*, de modo a não permitir *clock skew*. Os FPGAs apresentados em ALTERA (2004) e ALTERA (2005-c) apresentam uma árvore de *clock* pré-roteada e de alta velocidade.

5.1.3 2D-DWT

A 2D-DWT proposta opera pela aplicação recursiva da 1D-DWT, na primeira dimensão a seguir na segunda dimensão, nas amostras da imagem armazenadas em uma memória interna ao sistema. Deste modo, a 1D-DWT é aplicada individualmente a todas as linhas e após é aplicada a todas as colunas para cada oitava computada. Para a computação das oitavas de ordem superior à primeira, os coeficientes relativos ao *frame* LL de saída da oitava n são utilizados como entrada para a computação da oitava n+1.

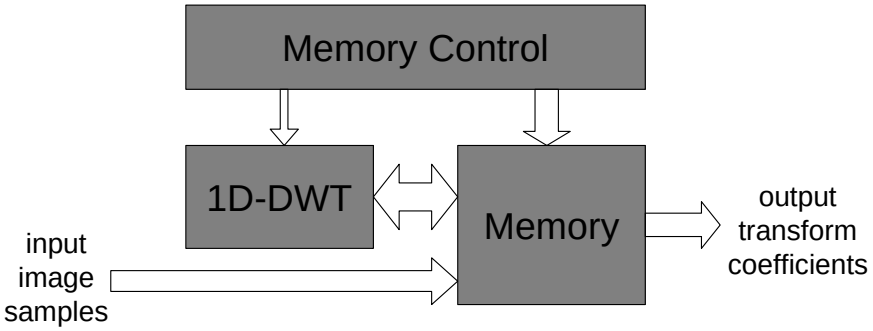

Figura 5.12: 2D-DWT em blocos.

A arquitetura para a computação da 2D-DWT seguindo o padrão JPEG2000, mostrada na Figura 5.12, requer:

- um bloco de memória com capacidade para armazenar todo o *tile*;
- um bloco para o controle dos endereços de leitura e escrita na memória e transferência para o bloco 1D-DWT;
- um bloco para a computação da transformada (1D-DWT) em uma dimensão.

Como os bloco da 1D-DWT e de memória são totalmente passivos, cabe ao bloco de controle definir a seqüência de operações necessárias para a computação da transformada wavelet.

Após inicialização, os dados da imagem são inseridos na arquitetura a taxa de um *pixel* por ciclo de *clock*. O controlador de memória gera os endereços para escrita em memória de forma que a cada n posições é armazenada uma linha inteira da imagem. Após todos os dados serem armazenados, o controlador de memória lê os dados da memória e os envia para o bloco 1D-DWT onde a transformada de uma dimensão é realizada. Esta computação ocorre em todas as linhas e todas as colunas para cada oitava requerida. Considerando que cada par de coeficientes s_n (sinal de aproximação) e d_n (sinal de detalhes) é computado a partir de um par de coeficientes x_{2n} e x_{2n+1}, lidos das posições 2n e 2n+1 da memória, então estes coeficientes s_n e d_n podem ser armazenados novamente nas mesmas posições 2n e 2n+1 da memória, e assim requerer somente a quantidade de memória necessária para armazenar um único *frame* da imagem. Por último o controlador de memória lê os dados transformados e envia para a saída, de forma a serem armazenados em uma memória externa ou computados por um próximo estágio.

A 2D-DWT requer a utilização de uma memória de tamanho n x m, onde n representa o número de linhas da imagem e m representa o numero de colunas da imagem.

A utilização de memória interna permite que o sistema se torne compacto e de maior eficiência. O acesso para leitura e escrita em uma memória interna pode ser feito sob taxas de acesso menores que uma arquitetura utilizando memória externa, com isso, os sistemas embarcados podem operar com maior eficiência.

A arquitetura da 1D-DWT é uma arquitetura sistólica descrita para a compressão JPEG2000 com perdas, ou seja, é uma transformada irreversível que segue a recomendações descritas no padrão JPEG2000. Esta arquitetura foi sintetizada para todas as descrições da seção 5.1.2. A figura 34 apresenta os sinais para realizar as ligações entre os blocos de memória, controlador de memória e transformada wavelet.

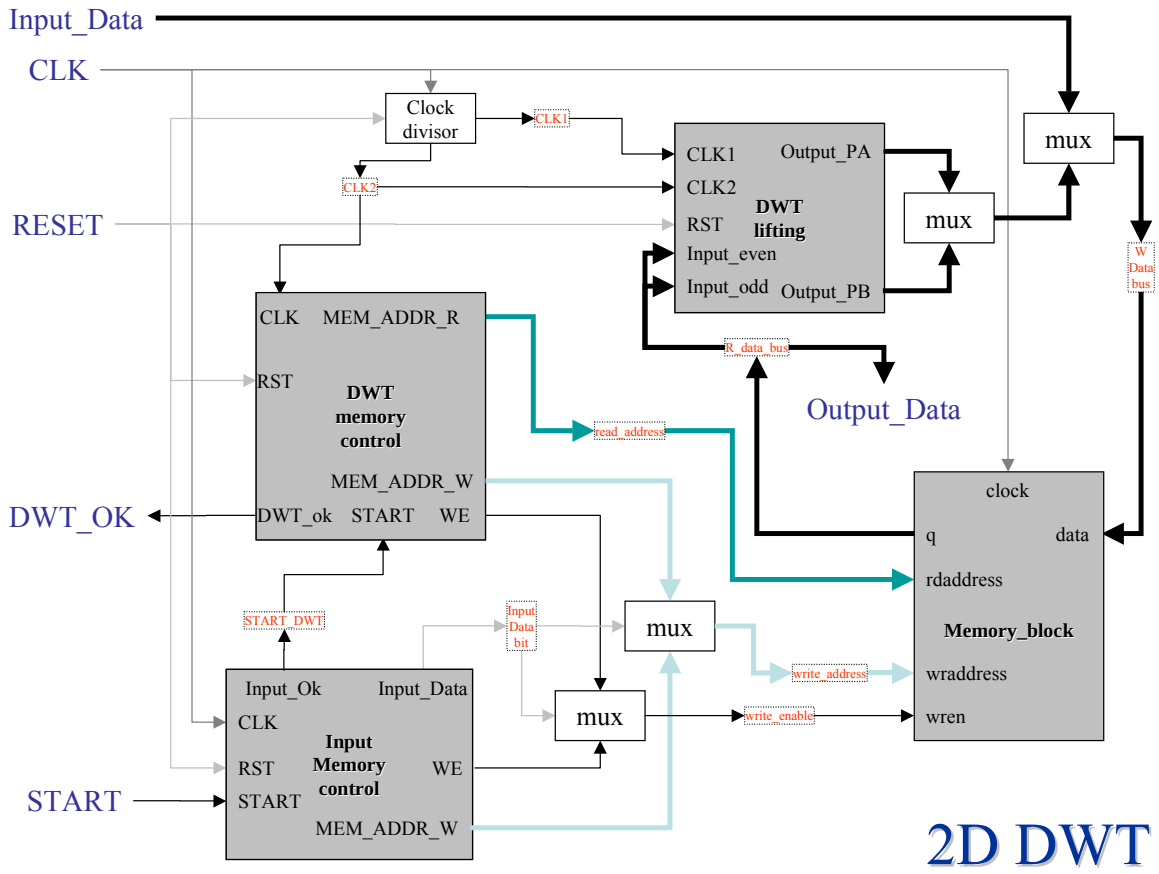

Figura 5.13: Arquitetura 2D-DWT.

Conforme observado na Figura 5.13, o controlador de memória foi dividido em dois blocos, o primeiro bloco (Input_Memory_Control da Figura 5.13) é utilizado para gerar os endereços para o armazenamento da imagem e o segundo bloco (DWT_memory_control da Figura 5.13) é utilizado para gerenciar a leitura e escrita dos coeficientes entre a memória e o bloco da transformada.

A operação desta arquitetura ocorre da seguinte forma:

Ao receber borda de subida no sinal START os dados dos *pixels* são lidos serialmente da entrada pelo sinal Input_Data e direcionados para a memória por um multiplexador através do sinal W_Data_bus. Os endereços de memória são gerados pelo bloco Input_Memory_Control.

O bloco Input_Memory_Control gera n endereços consecutivos de acordo com o sinal MEM_ADDR_W, ou seja, a cada ciclo de *clock* o Input_Memory_Control gera um endereço consecutivo de 0 até n-1, considerando n o valor predefinido como o tamanho do *tile*. O tamanho do *tile* é o produto do número de linhas pelo número de colunas do *tile*.

Após ocorrer a leitura de todo o *tile*, o Input_Memory_Control ativa o sinal START_DWT, para que o bloco DWT_memory_control assuma o controle dos sinais write_address e write_enable.

O bloco DWT_memory_control gera os endereços de leitura da memória dos coeficientes para a transformada obedecendo a dimensão da transformada (se horizontal ou vertical), o nível de oitava e o espelhamento das bordas. A cada linha ou coluna lida da memória o espelhamento das bordas deve ser considerando, assim a seqüência de endereços segue a ordem a(5) a(4) a(3) a(2) a(1) a(0) a(1).... Este espelhamento das bordas é feito à esquerda e à direita nas linhas e acima e abaixo nas colunas.

O bloco DWT_memory_control gera os endereços de escrita na memória dos coeficientes calculados pelo bloco da transformada. Este endereço é gerado por retardo do valor do endereço de leitura. Esta rede de retardo é implementada por um conjunto de *k* registradores ligados em série e *k* corresponde à profundidade do *pipeline* do bloco da transformada.

Para a geração do endereço de escrita ainda é considerado o espelhamento das bordas requerido para a computação da transformada *wavelet*. Assim nos ciclos que correspondem às posições espelhadas, não ocorre armazenamento dos coeficientes na memória.

Para permitir a utilização de um único bloco de memória de dimensões m x n, os coeficientes de saída são armazenados na mesma memória utilizada para armazenar as amostras de entrada. Na computação da primeira oitava, para cada par de amostras na entrada do bloco da transformada (a_i e a_j), é computado um coeficiente passa-baixas e um coeficiente passa-altas e para que não ocorra a sobreposição dos coeficientes na memória, o coeficiente passa-baixas é armazenado na posição *i* e o coeficiente passa-altas é armazenado na posição *j*.

A Figura 5.14 mostra o algoritmo para a geração dos endereços de leitura das amostras de entrada do bloco da transformada 1D. O endereço de escrita é obtido por uma rede de atraso de tamanho corresponde à profundidade do *pipeline* do bloco da transformada 1D.

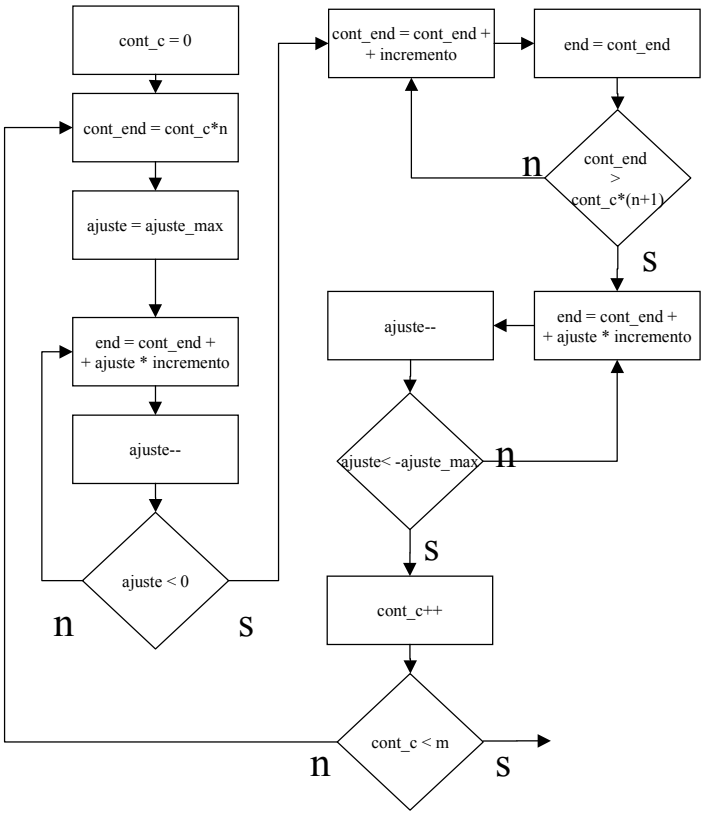

Figura 5.14: Algoritmo para a geração do endereçamento das linhas.

Este algoritmo utiliza:

- O registrador *end* é utilizado para endereçar memória;
- O registrador *incremento* é utilizado para ajustar o próximo endereço a ser acessado
    - No cálculo das linhas na oitava k, *incremento* = k;
    - No cálculo das colunas na oitava k *incremento* = k * n° de colunas;
- O contador de linhas (ou colunas) é chamado *cont_c*. Ao final da varredura de um *frame* (*cont_c* = *m* ou *cont_c* = *n*) é zerado enquanto o valor do registrador *incremento* é alterado (de acordo com regras do item anterior);
- O registrador *cont_end* percorre o vetor dos coeficientes de entrada, é atualizado a cada ciclo de leitura de memória pelo registrador *incremento*;
- O registrador *ajuste* é utilizado para contar o espelhamento das bordas. A variável *ajuste_max* define o tamanho do espelhamento das bordas.

O algoritmo apresentado na Figura 5.14 pode ser utilizado tanto para a geração do endereçamento das linhas como para o endereçamento das colunas, bastando trocar

os valores de *m* e *n*. Os coeficientes são lidos e armazenados na memória de forma intercalada, de acordo com o valor da variável *incremento*. A Figura 5.15 mostra a organização da memória após a computação de três oitavas da transformada *wavelet* de um *frame* de resolução 32x32.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
2	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
3	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
4	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
5	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
6	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
7	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
8	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287
9	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319
10	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351
11	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383
12	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415
13	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447
14	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479
15	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511
16	512	513	514	515	516	517	518	519	520	521	522	523	524	525	526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543
17	544	545	546	547	548	549	550	551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567	568	569	570	571	572	573	574	575
18	576	577	578	579	580	581	582	583	584	585	586	587	588	589	590	591	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607
19	608	609	610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625	626	627	628	629	630	631	632	633	634	635	636	637	638	639
20	640	641	642	643	644	645	646	647	648	649	650	651	652	653	654	655	656	657	658	659	660	661	662	663	664	665	666	667	668	669	670	671
21	672	673	674	675	676	677	678	679	680	681	682	683	684	685	686	687	688	689	690	691	692	693	694	695	696	697	698	699	700	701	702	703
22	704	705	706	707	708	709	710	711	712	713	714	715	716	717	718	719	720	721	722	723	724	725	726	727	728	729	730	731	732	733	734	735
23	736	737	738	739	740	741	742	743	744	745	746	747	748	749	750	751	752	753	754	755	756	757	758	759	760	761	762	763	764	765	766	767
24	768	769	770	771	772	773	774	775	776	777	778	779	780	781	782	783	784	785	786	787	788	789	790	791	792	793	794	795	796	797	798	799
25	800	801	802	803	804	805	806	807	808	809	810	811	812	813	814	815	816	817	818	819	820	821	822	823	824	825	826	827	828	829	830	831
26	832	833	834	835	836	837	838	839	840	841	842	843	844	845	846	847	848	849	850	851	852	853	854	855	856	857	858	859	860	861	862	863
27	864	865	866	867	868	869	870	871	872	873	874	875	876	877	878	879	880	881	882	883	884	885	886	887	888	889	890	891	892	893	894	895
28	896	897	898	899	900	901	902	903	904	905	906	907	908	909	910	911	912	913	914	915	916	917	918	919	920	921	922	923	924	925	926	927
29	928	929	930	931	932	933	934	935	936	937	938	939	940	941	942	943	944	945	946	947	948	949	950	951	952	953	954	955	956	957	958	959
30	960	961	962	963	964	965	966	967	968	969	970	971	972	973	974	975	976	977	978	979	980	981	982	983	984	985	986	987	988	989	990	991
31	992	993	994	995	996	997	998	999	1000	1001	1002	1003	1004	1005	1006	1007	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

LL0 LL0 LL0 entrada da primeira oitava
LL1 LL1 entrada da segunda oitava
LL2 entrada da terceira oitava

Figura 5.15: Organização de memória de uma imagem 32x32 pós DWT de 3 oitavas.

Segundo a Figura 5.15 os endereços de armazenamento dos coeficientes de saída da primeira oitava de um *frame* 32x32 são representados por z, onde:

subframe LL: $\forall x,y \in \mathbb{Z} \mid y \in [0,15]$ e $x \in [0,15] : z = 2(32y+x)$
subframe HL: $\forall x,y \in \mathbb{Z} \mid y \in [0,15]$ e $x \in [0,15] : z = 2(32y+x)+1$

subframe LH: $\forall x,y \in \mathbb{Z} \mid y \in [0,15]$ e $x \in [0,15] : z = 2(32y+x+16)$
subframe HH: $\forall x,y \in \mathbb{Z} \mid y \in [0,15]$ e $x \in [0,15] : z = 2(32y+x+16)+1$

O *subframe* LL da primeira oitava representa as componentes de baixa freqüência da imagem original, assim seus coeficientes são os valores de entrada da segunda oitava. Os endereços de armazenamento dos coeficientes de saída da segunda oitava do *frame* 32x32 são representados por z, onde:

subframe LL: $\forall x,y \in \mathbb{Z} \mid y \in [0,7]$ e $x \in [0,7] : z = 4(32y+x)$
subframe HL: $\forall x,y \in \mathbb{Z} \mid y \in [0,7]$ e $x \in [0,7] : z = 4(32y+x)+2$

subframe LH: $\forall x,y \in \mathbb{Z} \mid y \in [0,7]$ e $x \in [0,7] : z = 4(32y+x+16)$
subframe HH: $\forall x,y \in \mathbb{Z} \mid y \in [0,7]$ e $x \in [0,7] : z = 4(32y+x+16)+2$

Genericamente, os endereços de armazenamento dos coeficientes de saída da oitava *k* de qualquer *frame m* x *n* são representados por z, onde:

subframe LL: $\forall x,y \in \mathbb{Z} \mid y \in [0,(m/(k+1))-1]$ e $x \in [0,(n/(k+1))-1] : z = 2k(ky+x)$

subframe HL: $\forall x,y \in \mathbb{Z} \mid y \in [0,(m/(k+1))-1]$ e $x \in [0,(n/(k+1))-1] : z = 2k(ky+x)+2^{k-1}$

subframe LH: $\forall x,y \in \mathbb{Z} \mid y \in [0,(m/(k+1))-1]$ e $x \in [0,(n/(k+1))-1] : z = 2k(ky+x+m/2)$

subframe HH:

$\forall x,y \in \mathbb{Z} \mid y \in [0,(m/(k+1))-1]$ e $x \in [0,(n/(k+1))-1] : z = 2k(ky+x+m/2)+2^{k-1}$

5.1.3.1 Resultados da arquitetura da 2D-DWT

A arquitetura 2D-DWT apresentada na seção 5.1.3 foi implementada tendo como núcleo, as diversas arquiteturas discutidas na seção 5.1.2. Entretanto, a implementação da 2D-DWT apresenta parâmetros de entrada que independem da arquitetura interna utilizada.

Os principais parâmetros de entrada são:

- Número de linhas do *frame* de entrada: n
- Número de colunas do *frame* de entrada: m
- Tamanho do espelhamento das bordas: b
- Número de estágios de *pipeline* (latência) do bloco da transformada 1D:
- Largura dos dados de cada *pixel*:

A partir dos parâmetros de entrada da arquitetura várias informações do sistema podem ser definidas:

Tamanho do *frame* de entrada: $n_x m$

Número de ciclos para armazenamento do *frame*: nm

Número de ciclos para a DWT de todas as linhas (1^a oitava): (n+2b)m

Número de ciclos para a DWT de todas as colunas (1^a oitava): (m+2b)n

Número de ciclos para a 1^a oitava da DWT: 2mn+2b(n+m)

Número de ciclos para a DWT de todas as linhas (2^a oitava): (n/2+2b)m/2

Número de ciclos para a DWT de todas as colunas (2^a oitava): (m/2+2b)n/2

Número de ciclos para a 2^a oitava da DWT: mn/2+b(n+ m)

Número de ciclos para a DWT de todas as linhas (oitava X): $(n/2^X + 2^X b)m/2^X$

Número de ciclos para a DWT de todas as colunas (oitava X): $(m/2^X + 2^X b)n/2^X$

Número de ciclos para a oitava X da DWT: $mn/2^X + b(n+ m)/ 2^{X-1}$

Independentemente do tipo de bloco da transformada *wavelet* utilizado, considerando um espelhamento das bordas da imagem, a síntese da arquitetura 2D DWT utilizando um *frame* de entrada de 256x256 *pixels* de 8 bits e bordas de 6 *pixels* requer:

- 786.636 bits de memória;
- 65.536 ciclos de leitura para armazenar todo o *frame*;
- 268 ciclos de *clock* para a computação de cada linha da imagem na primeira oitava;
- 68.608 ciclos de *clock* para a computação da transformada de todas as linhas na primeira oitava;
- 137.216 ciclos de *clock* para a computação da primeira oitava;
- 35.840 ciclos de *clock* para a computação da segunda oitava;
- 9.728 ciclos de *clock* para a computação da terceira oitava;

De acordo com o tipo de implementação do bloco da transformada *wavelet* utilizado, os resultados de área, freqüência máxima de operação e potência média de saída são apresentados na Tabela 5.5.

Tabela 5.5: Resultados das Implementações da 2D-DWT (STRATIX EP1S20).

Arquitetura com Operadores Básicos com descrição:	Área utilizada (LEs)	Freqüência Máxima de Operação (MHz)	Consumo médio @15MHz (mW)
Comportamental com multiplicações inteiras (com variáveis do tipo *integer*)	2.276	8,5	-
Comportamental com multiplicações inteiras (com variáveis do tipo *standard logic vector*)	1.580	40,44	62,52
Comportamental com multiplicações por somas deslocadas	1.565	40,89	62,54
Comportamental com operadores com *pipeline* e multiplicações por somas deslocadas	1.786	57,95	66,21
Estrutural com multiplicações por somas deslocadas	1.794	30,45	62,34
Estrutural com operadores com *pipeline* e multiplicações por somas deslocadas	2.090	78,72	67,00

Os valores da Tabela 5.5 foram obtidos a partir de síntese e simulação utilizando a ferramenta ALTERA QUARTUS II (ALTERA, 2005-b) e todas as arquiteturas foram descritas para o dispositivo ALTERA STRATIX EP1S20F484C5 (ALTERA, 2005-c). Para permitir uma comparação direta entre os resultados, todas as simulações utilizaram

uma taxa de *clock* de 15MHz e foram feitas sobre o mesmo arquivo de vetores de teste, onde estes vetores correspondem à imagem "Lena" na resolução de 256x256 *pixels* com 255 tons de cinza e.

Conforme os resultados apresentados na Tabela 5.5, duas implementações utilizando Operadores Básicos em descrição comportamental com multiplicadores inteiros foram realizadas. Apesar de ambas as descrições utilizarem a mesma faixa dinâmica, a primeira descrição, que utiliza variáveis definidas como do tipo *integer*, apresentou área 44% maior que a segunda descrição, que utiliza variáveis definidas como do tipo *standard logic vector*. Além disso, a segunda descrição apresenta freqüência máxima de operação mais de 3,5 vezes maior. Esta diferença significativa em duas descrições aparentemente idênticas é atribuída à ferramenta de síntese não conseguir reduzir a área da arquitetura que utiliza descrição com variáveis do tipo *integer*. Como a freqüência máxima de operação prevista para a arquitetura que utiliza Operadores Básicos em descrição comportamental com multiplicadores inteiros utilizando variáveis do tipo *integer* foi extremamente baixa, a simulação do consumo médio de potência para esta arquitetura não foi realizada por não trazer nenhuma vantagem aparente para o desenvolvimento da arquitetura.

Para síntese em FPGAs, uma descrição comportamental se mostra a melhor opção, devido a permitir que a ferramenta de síntese utilize melhor os recursos internos do FPGA, como cadeias rápidas para propagação de *carry*, multiplicadores otimizados, etc... Porém uma descrição estrutural é a melhor opção para ser utilizada na prototipação de ASICs, por requerer uma descrição detalhada de todas as estruturas internas. Assim as diversas descrições da arquitetura seguiram esta linha de desenvolvimento, inicialmente descrição comportamental prevendo a síntese para FPGA e posteriormente descrição estrutural prevendo uma futura síntese para ASIC.

Conforme a Tabela 5.5, as descrições estruturais da 2D-DWT apresentam cerca 15% mais área que as descrições funcionais, considerando tipos semelhantes de estruturas, ou seja, comparando entre si as arquiteturas com Operadores Básicos não pipelinezados e também entre si as arquiteturas com Operadores Básicos pipelinezados.

Genericamente todas as arquiteturas apresentam *pipeline* no bloco da transformada *wavelet*, porém algumas arquiteturas possuem o Operador Básico pipelinezado internamente. Estas arquiteturas com Operadores Básicos pipelinezados apresentam aproximadamente 15% mais área que arquiteturas com Operadores Básicos não pipelinezados, entretanto possuem maior freqüência máxima de operação.

Em arquitetura pipelinezadas podemos dizer que de processamento (*throughput*) fica igual a freqüência máxima de operação quando temos o *pipeline* cheio, pois a cada ciclo do sinal de *clock* um novo dado é gerado na saída.

Esta afirmação é verdadeira para arquiteturas como a 1D-DWT, entretanto para computar três oitavas de uma imagem com resolução 256x256 *pixels* na arquitetura de 2D-DWT, é necessário:

- para computar a primeira oitava: 137.216 ciclos de *clock*, gerando três sub-bandas (LH, HL e HH) de 128x128;
- para computar a segunda oitava: 35.840 ciclos de *clock*, gerando três sub-bandas (LH, HL e HH) de 64x64;

- para computar a segunda oitava: 9.728 ciclos de *clock*, gerando quatro sub-bandas (LL, LH, HL e HH) de 32x32.

Ao todo na arquitetura da 2D-DWT com operadores AMA por descrição estrutural utilizando multiplicação de inteiros deslocados em uma arquitetura de *Pipeline* são computados 65.536 coeficientes em 182.784 ciclos de *clock*. Assim cada coeficiente é computado em 2,79 ciclos de *clock*, resultando em uma taxa de processamento (*throughput*) de **28,2 milhões de coeficientes por segundo**.

Conforme informações da seção 4.3.3, a CAST Inc., que pertence ao Programa de Parceria para produção de Mega-funções da Altera (AMPP) possui um IP para a implementação da 2D-DWT, que requer 3239 LEs de um dispositivo FPGA da Altera operando a uma freqüência máxima de operação de 84,6MHz. Quando em operação nesta freqüência, a arquitetura apresenta uma taxa de processamento (*throughput*) de **30,6 milhões de coeficientes por segundo**. A arquitetura desenvolvida na seção 5.1.3 apresenta redução de área de 35% e redução na freqüência máxima de operação para 93% comparado com a arquitetura da CAST Inc. Em ambos os casos a taxa de processamento se situa em torno de 35% da freqüência de operação.

A Barco Silex, que também pertence ao Programa de Parceria para produção de Mega-funções da Altera (AMPP), possui um IP para a implementação da 2D-DWT, que requer 4291 LEs de um dispositivo FPGA da Altera operando a uma freqüência máxima de operação de 130MHz. A arquitetura desenvolvida na seção 5.1.3 apresenta redução de área de 51% e redução na freqüência máxima de operação para 40% comparado com a arquitetura da Barco Silex. Caso este IP também possua taxa de processamento situada em torno de 35% da freqüência de operação, neste caso esta arquitetura poderia ter taxa de processamento de **46,4 milhões de coeficientes por segundo**.

5.1.3.2 Validação da arquitetura da 2D-DWT

A validação de todas as versões da arquitetura da 2D-DWT foi realizada por comparação entre a representação resultante após a aplicação transformada na imagem "Lena" utilizando a ferramentas MATLAB e o resultado de simulação da transformada da imagem "Lena" feita na ferramenta ALTERA QUARTUS II (ALTERA, 2005-b). O *frame* de entrada utilizado é uma imagem completa de "Lena" na resolução de 256x256 *pixels* com 255 tons de cinza Figura 5.16.

Figura 5.16: Lena: resolução 256x256 pixels com 255 tons de cinza.

O primeiro passo para a validação da arquitetura foi validar o algoritmo do filtro Daubechies 9/7. Para isso foi desenvolvido um programa para MATLAB, para calcular a transformada *wavelet* de uma dimensão, seguindo a especificação de ITU-T (2000). Aplicando o *stream* de dados correspondente às amostras da imagem Lena Figura 5.16, foi aplicado ao programa em MATLAB e foi obtida a imagem da Figura 5.17.

O resultado de uma simulação da 2D-DWT apresenta seqüências de valores inválidos e coeficientes de borda que deves ser desprezados entre cada linha de coeficientes válidos. Deste modo cada oitava deve ser recuperada separadamente. A partir dos dados da transformada das linhas foi montada a imagem apresentada na Figura 5.18.

O segundo passo para validação consiste em comparar o erro entre cada posição do *frame* da imagem computada pelo MATLAB e cada posição do *frame* da imagem simulada pelo Quartus II. Erros de baixa magnitude significam que os arredondamentos dos coeficientes e truncagens dos resultados nos operadores AMA podem ser admitidos.

Para calcular o erro entre os dois *frames*, foi utilizada a métrica MSE, que realiza a média dos erros quadráticos. Este cálculo resultou em MSE de **1,045837** em uma representação de 8 bits, significando que o erro médio quadrático é próximo a uma unidade na faixa dinâmica de 0 a 255, ou 0,41% do maior valor possível para um *pixel*.

A partir do valor de MSE pode-se inferir o valor de PSNR entre os dois *frames* a partir da fórmula (conforme seção 2.6):

$$PSNR = \frac{20 \log_{10} \max|P_i|}{MSE}$$

O valor de PSNR encontrado na computação da primeira dimensão da primeira oitava é igual a 46,02dB. Extrapolando para o cálculo de 3 oitavas, o valor de MSE pode subir para 2,74, gerando um PSNR de 17,5dB. Este valor de PSNR ainda significa que o *frame* simulado possui suficiente qualidade para considerar a arquitetura válida.

Figura 5.17: Lena: 2D-DWT aplicada às linhas (computado no MATLAB).

Figura 5.18: Lena: 2D-DWT aplicada às linhas (simulado no Quartus II).

A partir da semelhança visual entre as figuras e os resultados das métricas de comparação, pode-se considerar que o algoritmo implementado para a transformada wavelet seguindo o esquema *lifting* é válido.

A metodologia prevista na seção 2.6 para a validação da arquitetura consiste na comparação entre a representação descompactada gerada a partir dos coeficientes computados e a representação original da imagem, porém este procedimento não foi possível devido à falta da arquitetura para a descompressão.

5.2 Quantização

De acordo com a seção 3.2, a etapa de quantização é uma divisão escalar uniforme para todos os coeficientes de uma determinada sub-banda.

Uma maneira efetiva de definir o passo de quantização de uma determinada sub-banda é calculá-lo a partir do valor dos filtros de síntese vertical e horizontal. A relação entre o erro no coeficiente da transformada gerada pela quantização e o correspondente erro no valor das amostras de um componente da imagem é expressa em quantidade de energia (γ_b).

A quantidade de energia de uma determinada sub-banda é o produto da quantidade de energia da linha e da quantidade de energia da coluna. A quantidade de energia da coluna (ou linha) é a função de filtragem aplicada em uma coluna (ou linha) durante o processo de transformada inversa. Para calcular a quantidade de energia da coluna (ou linha), inicialmente realiza-se o *up-sampling* (inserção de um valor zero entre cada coeficiente do filtro) nos coeficientes h_i (de acordo com a Tabela 5.1) e a

seguir convoluciona-se com os coeficientes g_i. Esta convolução deve ser repetida tantas vezes quanto a quantidade de computações da transformada de uma dimensão. Finalmente a quantidade de energia da coluna (ou linha) é obtida pela soma do quadrado de todas as amostras resultantes da convolução final.

Um outro método para a escolha do passo de quantização é seguindo a equação (ITU-T, 2000):

$$\Delta_b = \frac{\Delta_d 2^{Rb}}{\sqrt{\gamma_b}}$$

Onde Δd é dado por 2^{1-RI}, RI representa a profundidade do sub-frame do componente original da imagem, Rb representa a faixa nominal dinâmica da sub-banda e γb representa a quantidade de energia do componente da imagem.

Com o objetivo de simplificar um passo na seqüência de compressão, a etapa de quantização ficou agregada no final da etapa de transformada.

O último passo no cálculo da 1D-DWT é o escalonamento dos coeficientes e este escalonamento é implementado por uma multiplicação simples. Os coeficientes desta multiplicação são -k, para o filtro passa-baixas e 1/k, para o filtro passa-altas. Deste modo, a multiplicação entre cada coeficiente de quantização requerido e os dois coeficientes da etapa de escalonamento da 1D-DWT produz dois novos conjuntos de coeficientes para compor a etapa de saída da 1D-DWT.

A etapa de escalonamento/quantização continua sendo implementada por uma multiplicação simples que pode ser comutada de acordo com o subframe a ser computado e o nível de perda da etapa de quantização requerido. A seleção do novo coeficiente de escalonamento/quantização é definida pelo controlador da etapa da transformada, onde é definido qual sub-frame está sendo processado.

5.3 Implementação do codificador de entropia

Após a etapa de transformada 2D e quantização, os coeficientes devem ser transformados do formato de complemento de 2 (utilizado para os cálculos durante as etapas de transformada e quantização) para o formato sinal-magnitude (utilizado na etapa de codificação de entropia). Esta transformação é feita por uma arquitetura que realiza uma inversão controlada de acordo com o bit de sinal, que permanece inalterado na transformação de um formato para o outro.

Após ser transformado para o formato de sinal-magnitude, o conjunto de dados é armazenado em um arranjo *m* x *n* de memória, chamado *code-block*. Onde m e n podem assumir valores entre 4 e 1024 (ACHARYA, 2005). Normalmente os valores de *n* e *m* assumem potências de 2, ou seja, $n = 2^x$ e $m = 2^y$ e x + y são limitados em 12.

O codificador de entropia utilizado no compressor JPEG 2000 é dividido em duas partes, um gerador de contextos e o codificador aritmético binário.

Para implementação do gerador de contextos (BPC) do padrão JPEG 2000, foi utilizada uma adaptação do algoritmo *Embedded Block Coding with Optimized Truncation* (EBCOT) (ACHARYA, 2005).

5.3.1 BPC - Bit-Plane Coding

Nesta etapa, cada banda de coeficientes gerada pela etapa de DWT é dividida em ***code-blocks***, que são arranjos bidimensionais de coeficientes inteiros, representados em formato sinal-magnitude. Os *code-block*s são divididos em planos de bits, onde existe o plano do bit de sinal e os planos do bit *n*-2 até o bit-0, considerando coeficientes com *n* bits.

Cada *code-block* é processado individualmente pelo bloco BPC (*Bit-Plane Coding*) ou codificador por plano de bits, que realiza a geração dos contextos para cada bit do *code-block*.

O BPC processa o *code-block* no padrão mostrado na Figura 5.19 em um deslocamento vertical de 4 posições para a seguir um deslocamento horizontal. Quando todas as colunas de 4 posições estiverem varridas, passa para o próximo conjunto com colunas de até 4 posições.

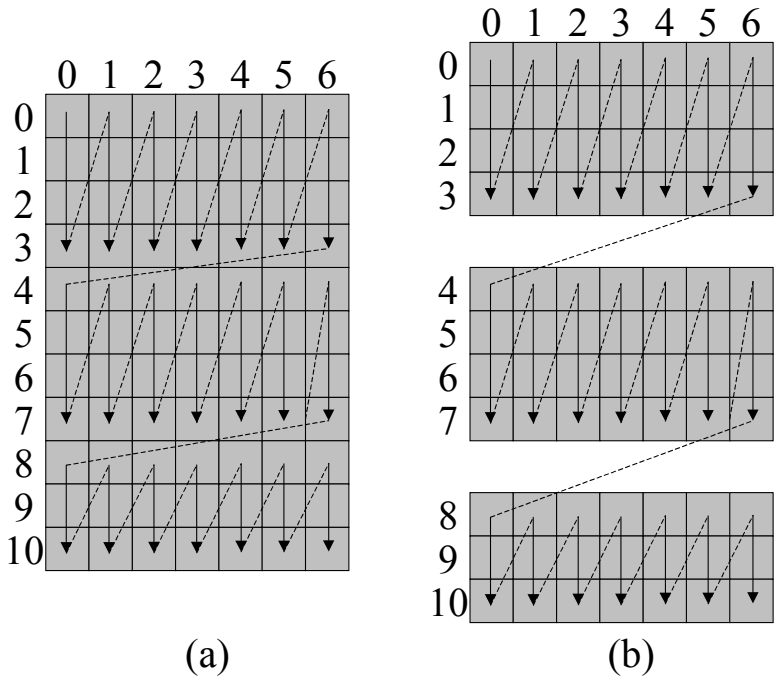

Figura 5.19: Padrões de varredura no processamento do *code-block* 7 x 11.

Na saída do BPC é gerado um conjunto de informações, onde cada informação é composta do bit codificado e do contexto do bit. A informação de contexto pode ser um dos 19 contextos possíveis, de acordo com a Tabela 5.6. Cada conjunto de contextos é gerado por um algoritmo diferente, de acordo com as características do bit a ser codificado, dentro do plano de bits.

Tabela 5.6: Estimação de Probabilidade para o BPC.

Operação	Contexto (CX)
Zero Coding	0
	1
	2
	3
	4
	5
	6
	7
	8
Sign Coding	9
	10
	11
	12
	13
Magnitude Refinement Coding	14
	15
	16
Run-Length Coding	17
UNIFORM	18

Conforme observado na Tabela 5.6, cada bit do *code-block* é classificado para um determinado contexto dos 19 contextos possíveis. O algoritmo BPC realiza esta classificação por contexto mediante a execução de três algoritmos diferentes. Cada algoritmo executa uma varredura completa por todo o plano de bits corrente e para cada plano de bits, com exceção do plano de bit de sinal e o plano dos bits mais significativos.

Conforme a Figura 5.20, no plano dos bits mais significativos ocorre apenas a varredura do algoritmo CUP e para os planos de bits seguintes ocorre a varredura dos algoritmos SPP, MRP e CUP (nesta ordem).

O algoritmo SPP (*Significance Propagation Pass*) codifica os bits em nível Hi, posicionados no plano de bits que corresponde ao bit de maior significância do coeficiente.

O algoritmo MRP (*Magnitude Refinement Pass*) codifica todos os bits com magnitude 1 que não sejam MSB e não foram codificados pelo SPP.

O algoritmo CUP (*CleanUp Pass*) codifica todos os bits não codificados pelos outros dois algoritmos anteriores.

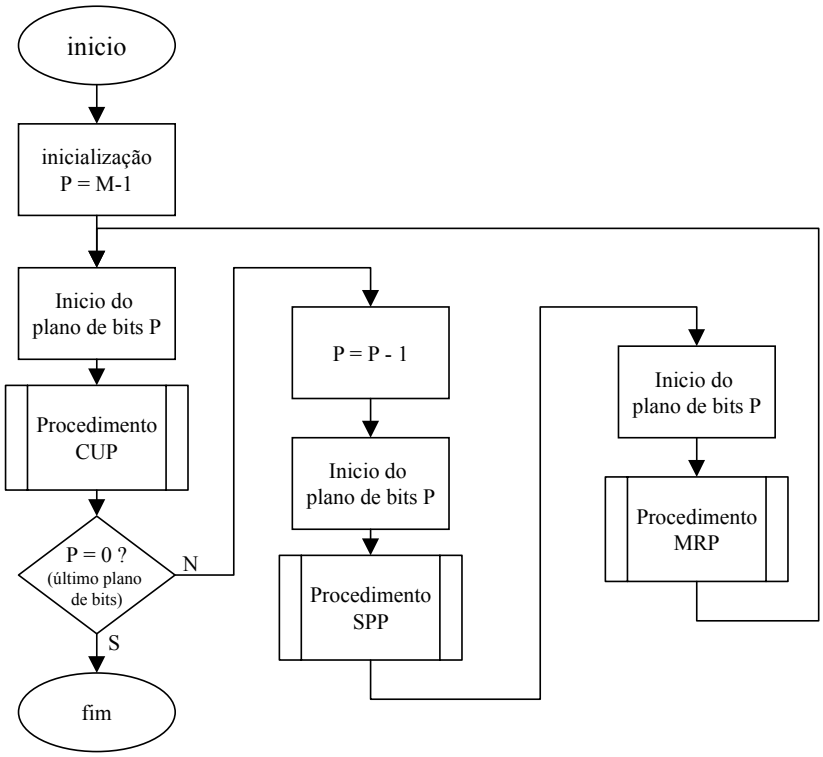

Figura 5.20: Fluxograma de operação do BPC.

O fluxograma mostrado na Figura 5.20 mostra a operação básica do BPC, onde o valor de P define o plano de bits corrente, M representa o número máximo de bits dos coeficientes (correspondendo ao número de planos de bits).

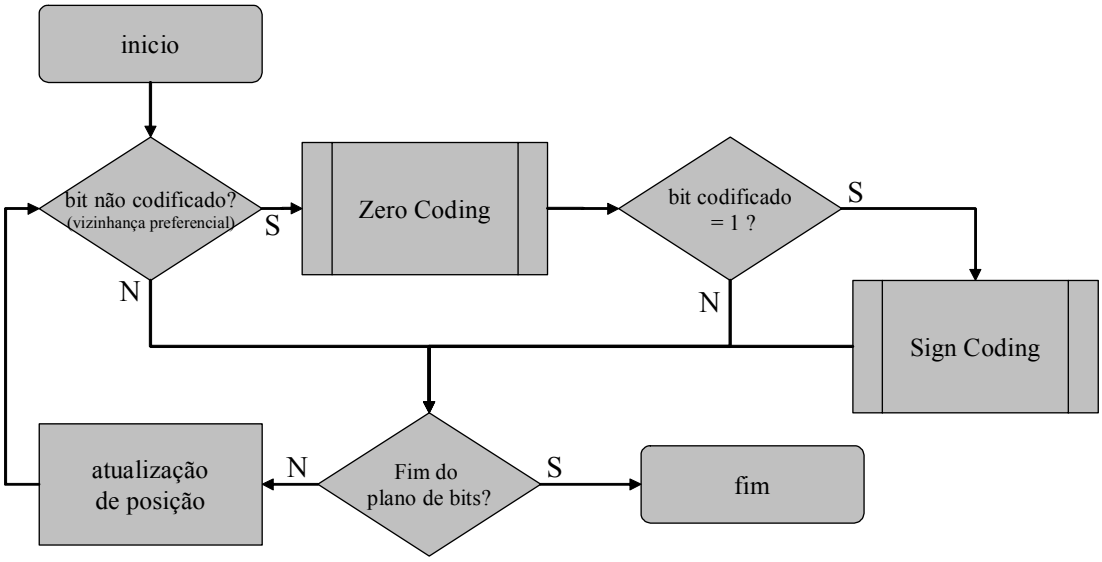

Figura 5.21: Fluxograma do algoritmo SPP.

Conforme algoritmo SPP mostrado Figura 5.21, todos os bits do plano de bits corrente que ainda não foram codificados e têm uma vizinhança preferencial são

codificados pelo algoritmo de *Zero Coding*. Este algoritmo gera um entre nove contextos conforme mostrado na Tabela 5.7, considerando que a tabela de contexto depende da sub-banda da qual o *code-block* foi originado.

Tabela 5.7: Determinação dos contextos do algoritmo *Zero Coding*.

Sub-bandas LL e LH			**Sub-banda HL**			**Sub-banda HH**		**Contexto**
ΣH	**ΣV**	**ΣD**	**ΣH**	**ΣV**	**ΣD**	**Σ(H+V)**	**ΣD**	**(CX)**
2	x	x	x	2	x	x	≥3	8
1	≥1	x	≥1	1	x	≥1	2	7
1	0	≥1	0	1	≥1	0	2	6
1	0	0	0	1	0	≥2	1	5
0	2	x	2	0	x	1	1	4
0	1	x	1	0	x	0	1	3
0	0	≥2	0	0	≥2	≥2	0	2
0	0	1	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0

O cálculo das vizinhanças vertical (ΣV), Horizontal (ΣH) e diagonal (ΣD) do bit X segue o diagrama mostrado na Figura 5.22.

O valor de ΣH é dado pela soma de H0 e H1;

O valor de ΣV é dado pela soma de V0 e V1;

O valor de ΣD é dado pela soma de D0, D1, D2 e D3.

D_0	V_0	D_1
H_0	X	H_1
D_3	V_1	D_2

Figura 5.22: Diagrama para cálculo das vizinhanças.

Conforme algoritmo da Figura 5.21, caso o bit codificado tenha valor Hi ocorre a codificação do bit de sinal, tendo para isso a geração do bit de sinal codificado e seu contexto.

Neste algoritmo os valores de H e V são calculados da seguinte maneira:

$H = \min[1,\max(-1,\sigma[m,n-1](-2\chi[m,n-1])+\sigma[m,n+1](-2\chi[m,n+1]))]$

$V = \min[1,\max(-1,\sigma[m-1,n](-2\chi[m-1,n])+\sigma[m+1,n](-2\chi[m+1,n]))]$

Onde:

- $\sigma[m,n-1 = 1$ indica que já ocorreu a codificação do primeiro bit não zero do coeficiente da esquerda (posição no *code-block*);

- $\sigma[m,n+1] = 1$ indica que já ocorreu a codificação do primeiro bit não zero do coeficiente da direita (posição no *code-block*);
- $\sigma[m-1,n] = 1$ indica que já ocorreu a codificação do primeiro bit não zero do coeficiente superior (posição no *code-block*);
- $\sigma[m+1,n] = 1$ indica que já ocorreu a codificação do primeiro bit não zero do coeficiente inferior (posição no *code-block*);
- $\chi[m,n-1]$ representa bit de sinal do coeficiente da esquerda (posição no *code-block*);
- $\chi[m,n+1]$ representa bit de sinal do coeficiente da direita (posição no *code-block*);
- $\chi[m-1,n]$ representa bit de sinal do coeficiente superior (posição no *code-block*);
- $\chi[m+1,n]$ representa bit de sinal do coeficiente inferior (posição no *code-block*);

A Tabela 5.8 é usada para determinar o contexto da codificação do sinal. O bit de dado é obtido pela função XOR entre χ da posição atual e o valor $\hat{\chi}$ dado na Tabela 5.8.

Tabela 5.8: Determinação dos contextos do algoritmo *Sign Coding*.

H	V	Contexto (CX)	$\hat{\chi}$
1	1	13	0
1	0	12	0
1	-1	11	0
0	1	10	0
0	0	9	0
0	-1	10	1
-1	1	11	1
-1	0	12	1
-1	-1	13	1

O segundo algoritmo utilizado na arquitetura interna do BPC é o MRP, ou *Magnitude Refinement Pass*. Este algoritmo realiza uma varredura completa em cada plano de bits (com exceção do primeiro) buscando todos os bits com magnitude 1 que não são o mais significativo bit em nível alto e não foram codificados pelo algoritmo SPP. A Figura 5.23 mostra o fluxograma de operação do algoritmo MRP.

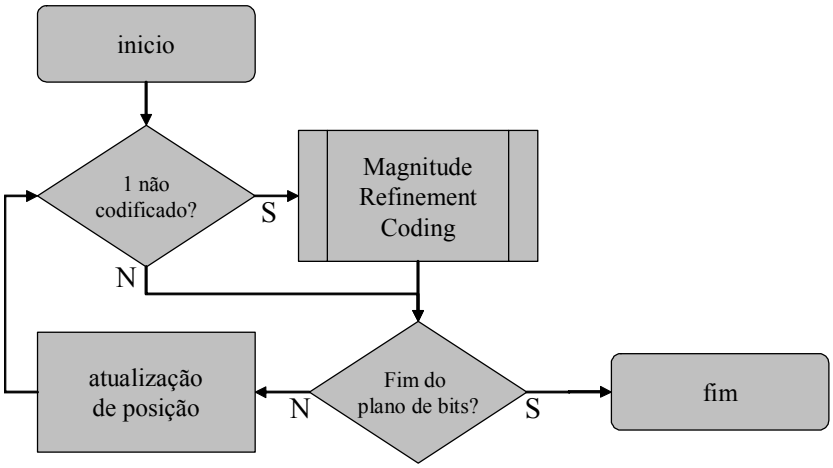

Figura 5.23: Fluxograma do algoritmo MRP.

De acordo com o fluxograma da Figura 5.23, a principal etapa do MRP é o *Magnitude Refinement Coding* (MRC). Nesta etapa, o valor do bit de dado gerado é igual ao valor do bit original e o valor do contexto obedece à Tabela 5.9.

Tabela 5.9: Determinação dos contextos do algoritmo *Magnitude Refinement Coding*.

Coeficiente já sofreu Refinamento de Magnitude	**# de vizinhos já codificados**	**Contexto (CX)**
1	x	16
0	≥1	15
0	0	14

A Tabela 5.9 determina que o contexto gerado terá:

- valor 16: quando o bit pertencer a um coeficiente ao qual já ocorreu refinamento de magnitude;
- valor 15: quando ainda não tiver ocorrido refinamento de magnitude no coeficiente ao qual pertence o bit a ser codificado, porém pelo menos um dos vizinhos horizontais ou diagonais deverá ter seu bit de magnitude 1 mais significativo codificado;
- valor 14: quando ainda não tiver ocorrido refinamento de magnitude no coeficiente ao qual pertence o bit a ser codificado e todos os vizinhos horizontais ou diagonais não tiveram a codificação do seu bit de magnitude 1 mais significativo;

O último algoritmo utilizado na arquitetura interna do BPC é o CUP, ou *Clean Up Pass*. Este algoritmo realiza uma varredura completa em cada plano de bits e codifica todos os bits que ainda não tiveram nenhum tipo de codificação. Basicamente este algoritmo realiza a codificação de bits de nível Lo, com exceção da primeira passagem que ocorre para o plano de bit dos MSBs. De acordo com o fluxograma da Figura 5.24, existem três algoritmos importante na operação do CUP. O algoritmo de

Zero Coding e algoritmo de *Sign Coding*, que foram apresentados anteriormente na descrição do algoritmo SPP, a principal diferença é a introdução do algoritmo *Run-Lengh Coding* (RLC).

O algoritmo RLC pode codificar de um a quatro bits consecutivos, que pertençam à mesma coluna no arranjo do *code-clock* que tenha bits em nível Lo em toda a sua vizinhança. A quantidade de bits codificados depende da posição do primeiro bit em nível Hi encontrado, neste caso o algoritmo RLC codifica até a posição do bit em nível Hi. Caso somente existam bits em nível Lo dentro do conjunto de quatro bits, todos os quatro bits são codificados.

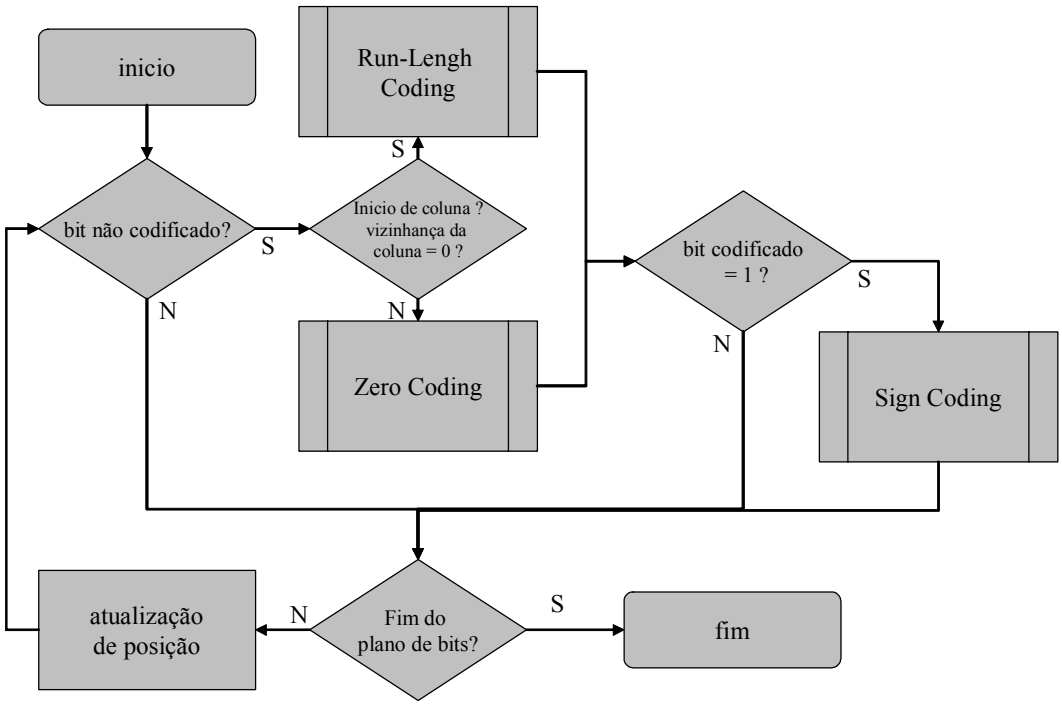

Figura 5.24: Fluxograma do algoritmo CUP.

Utilizando os algoritmos descritos nesta seção, foi implementada a arquitetura do BPC. Os sinais de interconexão, apresentados na Figura 5.25, são utilizados para as seguintes funções:

- CLK: Cadenciar as operações;
- RESET: Reinicializar o algoritmo, permitindo a codificação de outro *code-block*.
- Input Data: Entrada de dados a serem codificados;
- Sub-band: Define qual das quatro sub-bandas deve ser codificada (informação utilizada para a geração do contexto durante o algoritmo *Zero Coding*);
- DX: Saída do bit a ser codificado pelo próximo estágio (bloco BAC);

- CX: Saída com a informação de contexto;
- Coding OK: Sinaliza termino da codificação de todos os planos de bits.

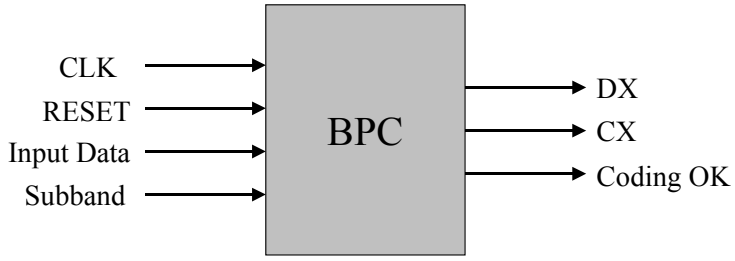

Figura 5.25: Sinais para interconexão do bloco BPC.

O bloco BCP foi implementado a partir do fluxograma apresentado na Figura 5.26. Como pode ser observado, este fluxograma possui 21 estados, logo é presumível que o bloco BPC apresente baixa taxa de saída. A operação do BPC de acordo com o fluxograma da Figura 5.26 apresenta os seguintes estados:

Após RESET, que define o inicio do processo de codificação, inicializando todas as variáveis que sinalizam a codificação de cada bit de entrada;

O estado *start* realiza a leitura de todo o *code-block* para a memória interna do BPC;

O estado *start CUP* marca o inicio do algoritmo CUP e posiciona as variáveis de endereçamento para apontar para a primeira posição do plano de bits P;

O estado *teste1 CUP* testa se o bit corrente já foi codificado. Caso sim redireciona o ponteiro para o próximo bit no estado *next CUP*;

O estado *teste2 CUP* testa se deverá ocorrer *Zero Coding* ou RLC no bit corrente;

Os estados *teste2 RLC CUP*, *teste2 1 CUP* e *teste2 RLC CUP* implementam o algoritmo RLC;

O estado *teste2 ZC CUP* implementa o algoritmo *Zero Coding*;

O estado *testebit CUP* testa a necessidade e implementa o algoritmo *Sign Coding*;

O estado *next CUP* posiciona o ponteiro de bit para o próximo bit;

O estado *end CUP* testa se todos os planos de bit foram codificados. Caso teste negativo reposiciona ponteiro de plano de bits;

O estado *start SPP* marca o inicio do algoritmo CUP e posiciona as variáveis de endereçamento para apontar para a primeira posição do plano de bits P;

O estado *teste SPP* testa necessidade de execução do algoritmo SPP para o bit corrente;

O estado *teste1 SPP* executa o algoritmo *Zero Coding*;

O estado *testebit SPP* testa a necessidade e implementa o algoritmo *Sign Coding*;

O estado *end SPP* posiciona o ponteiro de bit para o próximo bit;

O estado *start MRP* marca o inicio do algoritmo CUP e posiciona as variáveis de endereçamento para apontar para a primeira posição do plano de bits P;

O estado *teste MRP* testa necessidade de execução do algoritmo MRP para o bit corrente. Caso afirmativo executa o algoritmo MRP;

O estado *end SPP* posiciona o ponteiro de bit para o próximo bit;

O estado *end MQ* sinaliza fim do algoritmo BPC;

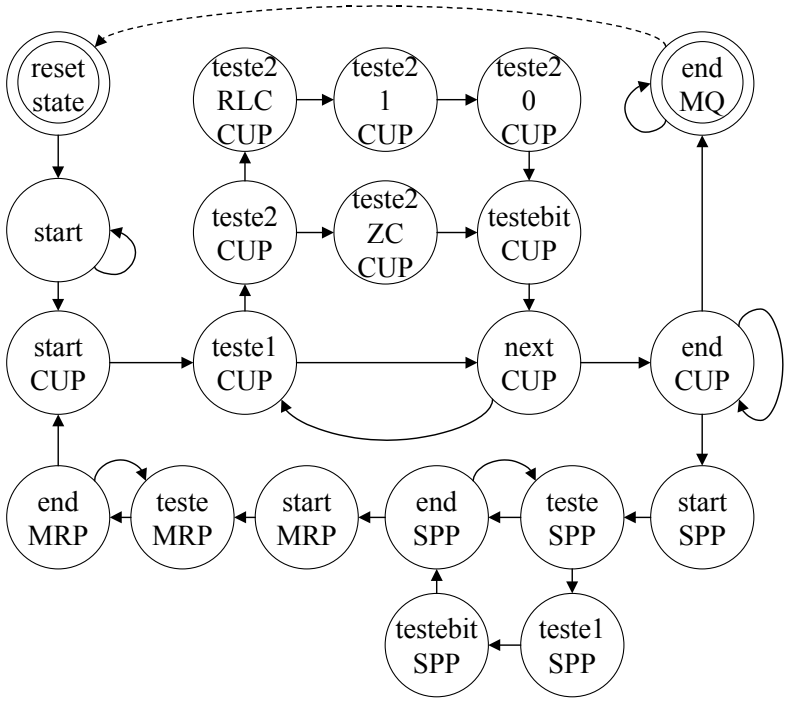

Figura 5.26: Fluxograma para operação do bloco BPC.

Considerando um arranjo de entrada de n x m coeficientes com P bits de magnitude e um bit de sinal em cada coeficiente:

- o estado *start CUP* será executado P vezes;
- a seqüência de estados *end CUP* → *start SPP* ocorre P-1 vezes;
- para as sub-bandas que possuam uma grande quantidade de zeros, o algoritmo CUP é muito requerido, ocasionando maior quantidade de estados.

O número de estados que o algoritmo executa pode ser calculado pela seguinte equação: **2 + P(2+[CUP]mn)+(P-1)(2+[SPP]mn+2mn)**

Onde SPP eqüivale a um valor médio entre 2 e 4 (cálculo dependente dos coeficientes de entrada) e CUP eqüivale a um valor médio entre 2 a 7. Deste modo o melhor caso será se for considerando CUP igual a 2 e SPP igual a 2, resultando em

4P+6mnP-4mn. O pior caso será se for considerando CUP igual a 7 e SPP igual a 4, resultando em 4P+13mnP-6mn.

Um *code-block* com 64x64 coeficientes de 12 bits (considerando o bit de sinal) apresenta 49.152 bits. O melhor caso de codificação apresenta **278.576 ciclos de *clock***, resultando em uma taxa de processamento de 5,67 ciclos de *clock* para cada bit de dados, ou 68 ciclos de *clock* para cada coeficiente de entrada. O pior caso de codificação apresenta **614.448 ciclos de *clock***, resultando em uma taxa de processamento de 12,5 ciclos de *clock* para cada bit de dados, ou 150 ciclos de *clock* para cada coeficiente de entrada.

A síntese do bloco BPC gerou uma arquitetura com **613 elementos lógicos** e pode operar até a freqüência de **57,27 MHz**.

Um *sub-frame* com 64 x 64 coeficientes de 12 bits resulta em um arranjo com **49.152 bits** e o BPC é capaz de processar todo o *frame* utilizando uma quantidade de ciclos de *clock* que varia entre 278.576 a 614.448 ciclos. Considerando a freqüência de 57,27 MHz, todo o *frame* pode ser computado entre **4,86ms** a **10,73ms**. E a uma taxa de processamento se situa entre **382 mil a 843 mil de coeficientes por segundo.**

5.3.2 BAC - Binary Aritmetic Coding

A codificação aritmética codifica incrementalmente, construindo uma palavra codificada diretamente a partir dos símbolos de entrada, não requerendo assim, memória para realizar o processo.

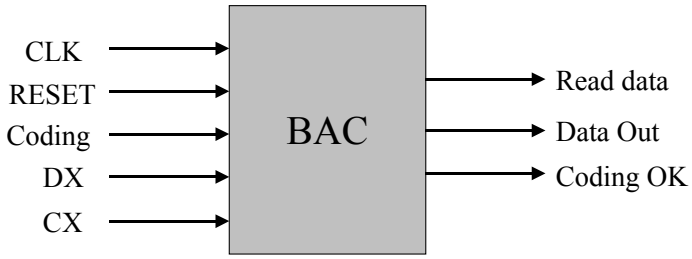

Figura 5.27: Sinais para interconexão do bloco BAC.

A arquitetura BAC implementada, conforme a Figura 5.27, possui 5 sinais de entrada e 3 sinais de saída. Cada sinal possui a seguinte função:

- CLK: Cadenciar as operações;
- RESET: Reinicializar o algoritmo, permitindo a codificação de outro *code-block*.
- DX: Entrada do bit a ser codificado;
- CX: Contexto do bit a ser codificado;
- Coding: Sinaliza a existência de bits não codificados;
- Read data: Sinaliza ciclo de leitura de DX e CX;
- Data Out: Saída de dados codificados;

- Coding OK: Sinaliza fim do processo de codificação do *code-block*.

A codificação aritmética binária é aplicada a um alfabeto com dois símbolos a partir de uma informação de probabilidade. O alfabeto de dois símbolos é obtido diretamente da entrada DX e a probabilidade é gerada a partir da tabela de codificação e da informação de contexto vinda da entrada CX.

A operação do bloco BAC se baseia em dois registradores principais, o registrador A onde são realizadas as comparações da análise da probabilidade acumulada, e o registrador C que acumula os sub-intervalos encontrados. A operação do BAC segue o fluxograma mostrado na Figura 5.28 e é dividido nos seguintes estados:

- estado *init*: ocorre a inicialização de todas as variáveis do sistema;
- estado *read input*: é onde efetivamente as informações de entrada são lidas, enquanto o valor de DX é armazenado para uso futuro, o valor de CX (contexto vindo do BPC) é utilizado como índice para uma tabela, para definir o valor da probabilidade do bit DX.
- estado *teste MPS LPS*: considerando que cada intervalo é dividido em dois subintervalos, neste estado é definido:
  - se o dado lido é o mais provável e caso ainda existam dados a serem codificados retorna para o estado *read input*, caso contrario passa para o estado *FLUSH*;
  - se o dado lido é menos provável ocorre renormalização do intervalo, passando para o estado *renorm*;
- estado *renorm*: realiza a renormalização do novo sub-intervalo. Após este estado pode ocorrer:
  - estado *renorm* novamente;
  - o estado *read input*, caso ainda existam dados a serem codificados;
  - o estado *byte out*, caso o tamanho da palavra codificada tenha sido alcançado;
  - o estado FLUSH, caso não existam mais dados a serem codificados;
- estado *byte out*: envia para a saída uma palavra codificada válida. Caso ainda existam dados a serem codificados o próximo estado será *read input*, caso contrário *FLUSH*;
- estado *FLUSH*: realiza o ajuste de *carry* do conjunto de bits que forma a palavra codificada (registrador C);
- estado *FLUSH byte out*: envia para a saída uma palavra codificada válida;
- estado *FLUSH_1*: realiza o ajuste de *carry* do conjunto de bits que forma a palavra codificada (registrador C);
- estado *FLUSH_1 byte out*: envia para a saída uma palavra codificada válida. Após este estado o BAC fica inoperante, retornando ao estado inicial, para o processamento de um novo *code-block* somente por RESET;

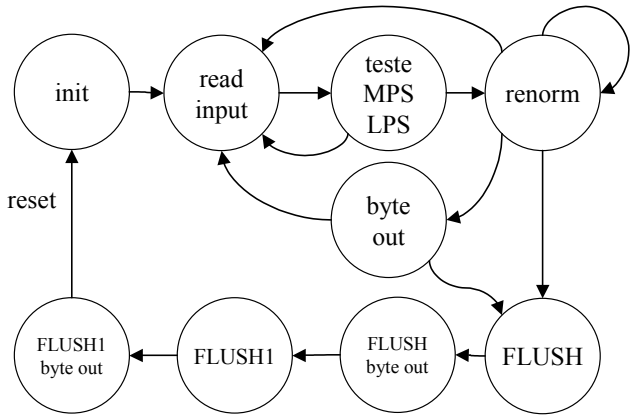

Figura 5.28: Fluxograma para operação do bloco BAC.

A síntese do bloco BAC gerou uma arquitetura com **1539 elementos lógicos** e pode operar até a freqüência de **53,1 MHz**. Considerando que o BAC é um codificador de comprimento de palavra variável não é possível determinar taxa de geração da palavra codificada, nem tampouco a taxa de leitura dos bits de entrada, porém neste caso é possível determinar os valores extremos.

No melhor caso ocorre: *read input* → *teste MPS LPS* → *read input*, ou seja, a cada dois ciclos ocorre uma nova leitura;

No pior caso ocorre: *read input* → *teste MPS LPS* → *renorm* → *byte out* →*read input*, assim a cada quatro ciclos ocorre uma nova leitura, porém o estado *byte out* ocorre no máximo uma vez a cada oito ciclos;

Um dos *sub-frames* de uma imagem 256 x 256 *pixels* que possui 64 x 64 coeficientes de 12 bits, resulta em um arranjo com **49.152 bits**. Considerando que o BAC é capaz de processar cada bit de entrada em uma média entre dois e quatro ciclos do sinal de *clock*, todos os 49.152 bits referentes ao *sub-frame* serão computados entre **98.304** e **196.608 ciclos** do sinal de *clock*, logo, na freqüência máxima de operação, todo o *sub-frame* será computado entre **1,85ms** e **3,7ms**, significando uma taxa de processamento entre **3,24 milhões e 6,5 milhões coeficientes por segundo.**

O codificador de entropia produz um resultado que não admite perda de dados, logo, independente da arquitetura utilizada o resultado deverá ser idêntico. Para validar esta arquitetura foi realizada uma simulação do processo de codificação utilizando como entrada um *code-block* de 4x4 coeficientes sugerido em ACHARYA (2005). Este *code-block* de 4x4 coeficientes, mostrado abaixo, representa 64 bits a serem codificados.

3	0	0	5
-3	7	2	1
-4	-1	-2	3
0	6	0	2

Os coeficientes do *code-block* mostrado acima foram aplicados ao BPC. A seqüência de saída do BPC possui duas informações, a primeira informação representa o bit codificado (para permitir uma maior compressão pelo BAC) e a segunda informação representa o contexto do bit. A Tabela 5.10 apresenta os resultados de saída do Codificador de planos de bits.

Tabela 5.10: Resultados de Saída do BPC.

Contexto	Bit
17	1
18	1
18	0
9	1
3	0
0	0
1	1
9	0
7	0
1	1
9	0
1	0
5	0
2	0
5	0
17	1

Contexto	Bit
18	0
18	0
9	0
3	0
0	0
0	0
1	1
9	0
7	1
12	1
7	0
7	0
7	0
6	0
6	1
12	0

Contexto	Bit
3	1
10	1
7	0
7	0
6	1
12	1
3	1
10	0
15	0
15	1
14	1
14	0
7	0
7	0
8	1
11	0

Contexto	Bit
7	0
8	0
7	1
13	0
15	1
15	1
16	0
16	1
16	0
15	0
15	0
16	1
15	1
15	0

As informações de saída do BPC são utilizadas para alimentar a entrada do BAC. A seqüência de saída do BAC é mostrada na seqüência abaixo:

0 0 0 0 1 0 0 1	09
0 1 0 0 0 1 1 0	46
0 0 0 0 1 0 0 0	08
0 0 1 1 0 1 1 1	37
1 1 1 1 0 1 1 0	F6
0 0 1 0 0 1 0 0	24
0 0 0 1 1 1 1 0	1E
1 0 0 0 1 0 0 1	89

Esta arquitetura foi validada, considerando que o conjunto de resultados da simulação é idêntico ao conjunto de valores sugeridos na bibliografia.

5.4 Implementação do Compressor de Imagens

A implementação do compressor de imagens no padrão JPEG 2000 requer o interfaceamento e síntese global das arquiteturas descritas nas seções 5.1, considerando a quantização inserida na etapa de escalonamento de coeficientes da 1D-DWT, 5.3.1 e 5.3.2 e uma etapa de controle para definir o momento de inicio da computação de cada bloco.

Tabela 5.11: Resultados da Implementação dos principais módulos do Compressor.

Arquitetura	Área utilizada (LEs)	Freqüência Máxima de Operação (MHz)	Throughput (coeficientes por segundo)
2D DWT	2.090	78,72	28,2M
BPC	613	57,27	382K a 843K
BAC	1.539	53,10	3,2M a 6,5M

A Tabela 5.11 apresenta os resultados da síntese de cada bloco individualmente de cada arquitetura conforme seções anteriores. Como pode ser observada, apesar da arquitetura do BPC apresentar uma freqüência máxima de operação razoável, sua taxa de computação (representado pelo valor do *throughput*) apresenta baixo valor. Com isso, uma implementação serializada composta de apenas uma unidade de cada uma das arquiteturas propostas, conforme sugerido na Figura 5.29, resultaria em uma arquitetura sub-utilizada, onde o *throughput* seria equivalente ao valor do pior caso, neste caso, equivalente ao resultado do BPC.

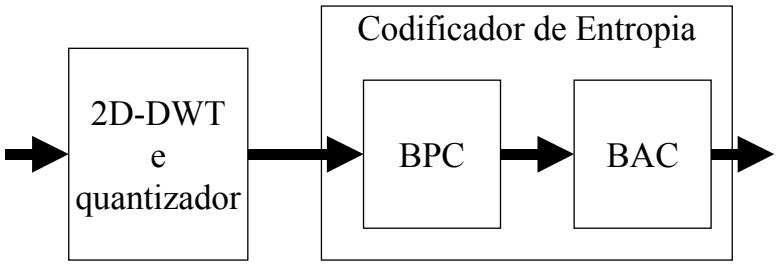

Figura 5.29: Compressor JPEG2000 em uma implementação serializada.

Esta implementação serializada utilizando todos os blocos do compressor operando na mesma freqüência de *clock* resulta que o *throughput* dos blocos 2D-DWT e BPC seria menor que o valor apresentado na Tabela 5.11. Uma alternativa de implementação mais complexa seria a utilização de blocos com diferentes domínios de *clock*, de modo que cada bloco opere em sua freqüência máxima de operação.

Durante o processamento da transformada *wavelet*, todos os coeficientes são armazenados em uma memória interna ao bloco 2D-DWT e quantizador, e ao final do processamento os dados são transferidos para fora do bloco formando um *stream* de dados. O bloco codificador de entropia requer *code-blocks* como entrada. Como os *code-blocks* são porções retangulares de um *frame* transformado, para realizar a interface entre a saída do bloco 2D-DWT e quantizador e o codificador de entropia é necessário realizar a separação dos *frames* transformados em *code-blocks*.

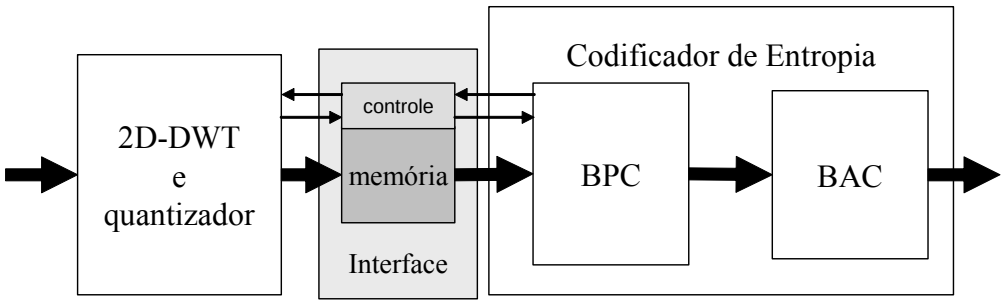

Figura 5.30: Interface entre os blocos do Compressor JPEG2000.

A implementação desta interface requer memória suficiente para armazenar todo o *frame* transformado e uma etapa de controle. Cada vez que o bloco 2D-DWT e quantizador termina a computação de um *frame*, pode transferi-lo para a memória do bloco de interface. A interface somente transfere o novo *code-block* para o codificador de entropia caso exista um *frame* armazenado na memória da interface e o BPC esteja aguardando um novo *code-block*. Porém, caso exista um *frame* armazenado na memória da interface e o bloco 2D-DWT/quantizador possua um *frame* completo transformado é necessário que a computação no bloco 2D-DWT seja inibida. A Figura 5.30 mostra a inserção da interface entre os blocos 2D-DWT/quantizador e codificador de entropia.

A interface entre os blocos BPC e BAC pode ser implementada de forma mais simples que a interface entre a saída da transformada e o codificador de entropia. Considerando que a taxa de saída de dados do bloco BPC, que é variável e dependente

dos dados de entrada, possui ainda taxa de processamento de dados menor que o bloco BAC. Assim a opção mais simples para esta *inteface*, é a alteração da etapa de controle do BAC, de modo a aguardar a existência de dado válido em sua entrada.

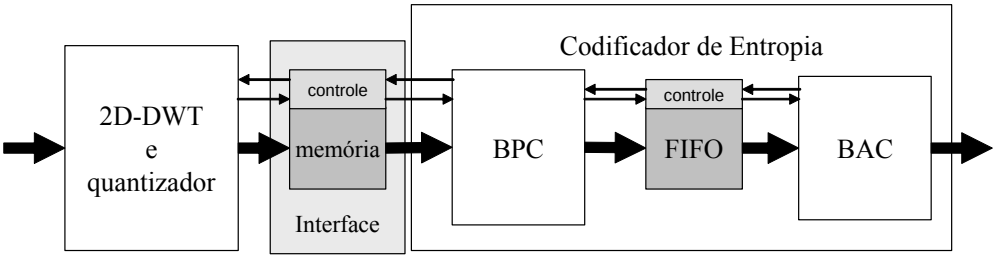

Figura 5.31: Interface entre os blocos BPC e BAC.

Em caso de melhoria da taxa de processamento do bloco BAC esta interface pode ser implementada por uma fila (FIFO) de registradores de profundidade *n* e um controle para detectar os casos condições extremas da fila (casos de fila cheia ou vazia), quando então o processamento de um dos dois blocos deve ser inibido. Para a definição da profundidade *n* da fila de registros deve ser observar a relação entre a taxa de processamento entre o BPC e o BAC e a quantidade máxima de bits no *code-block*. Esta opção de implementação é apresentada na Figura 5.31. O controle desta interface deve enviar um sinal para inibir o processamento do BPC sempre que a FIFO estiver cheia e deve enviar um sinal para inibir o processamento do BAC sempre que a FIFO estiver vazia, de modo que cada bloco processa sempre um coeficiente válido.

Considerando que o *throughput* do bloco da transformada é, pelo menos, dez vezes maior que o *throughput* do codificador de entropia, uma implementação mais eficiente propõe a paralelização dos blocos que compõem o codificador de entropia. A arquitetura para implementar esta paralelização é descrita na Figura 5.32 e apresenta três blocos idênticos do codificador de entropia, como sugerido em ANDRA (2003), de modo que cada bloco pode codificar cada um dos *sub-frames* LH, HL e HH gerados pelo bloco da transformada 2D. O *sub-frame* LL é codificado no codificador de entropia da sub-banda LH.

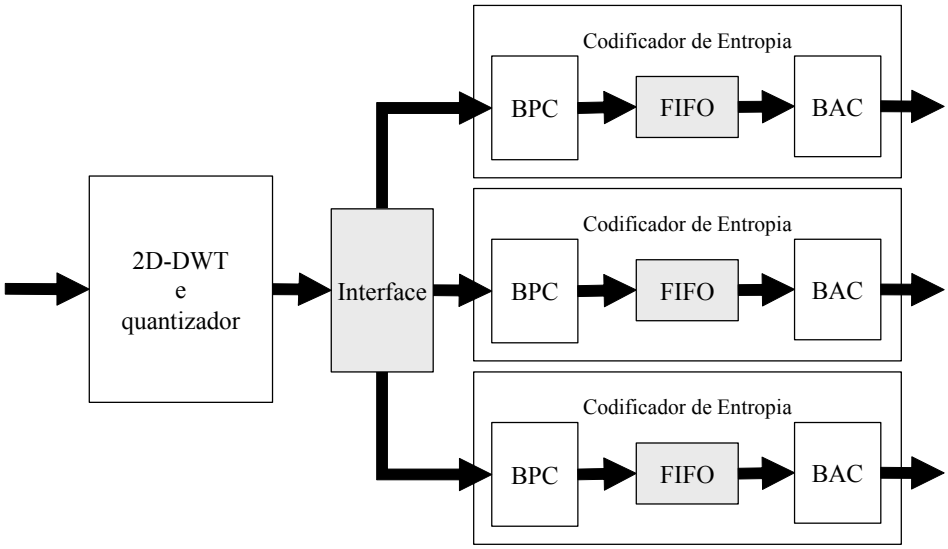

Figura 5.32: Compressor JPEG2000.

A interface entre os blocos 2D-DWT/quantizador e o codificador de entropia, opera de modo semelhante ao descrito para a arquitetura descrita para Figura 5.30, ou seja é necessário um bloco de controle para a interface e memória suficiente para armazenar o *frame*. A interface entre os blocos BPC e BAC pode ser implementada de modo idêntico ao descrito para a arquitetura da Figura 5.31, logo permite o reuso do mesmo bloco codificador de entropia. Considerando que o bloco codificador de entropia corresponde a cerca de 50% da área do codificador, esta abordagem proporciona acréscimo de área de aproximadamente 100%, entretanto proporciona uma redução de 65% no tempo de codificação de todo o *frame* da imagem. Esta redução do tempo de processamento pode ser inferida a partir da observação da Figura 5.33. Este paralelismo proporciona melhoria no desempenho do bloco codificador de entropia, fazendo com que sua taxa de saída varie de 1,1 milhões e 2,5 milhões de coeficientes por segundo.

Considerando que a transformada de cada um dos *sub-frames* relativos à sub-banda 0 requer n ciclos de processamento para ser codificada, a transformada de cada um dos *sub-frames* relativos à sub-banda 1 requer $4n$ ciclos de processamento e cada um dos *sub-frames* relativos à sub-banda 0 requer $16n$ ciclos de processamento, o compressor serializado mostrado na Figura 5.31 realiza a codificação das três oitavas em $64n$ ciclos de processamento. No compressor com codificador de entropia paralelizado mostrado na Figura 5.32, cada bloco codificador de entropia codifica apenas um *sub-frame* de cada oitava, e assim requer apenas $21n$ ciclos de processamento para realizar esta codificação. O bloco que também codifica o *sub-frame* LL_2 requer $22n$ ciclos de processamento. Assim neste caso todo o *frame* pode ser codificado em $22n$ ciclos de processamento, ou 35% do tempo de processamento da arquitetura serializado da Figura 5.31.

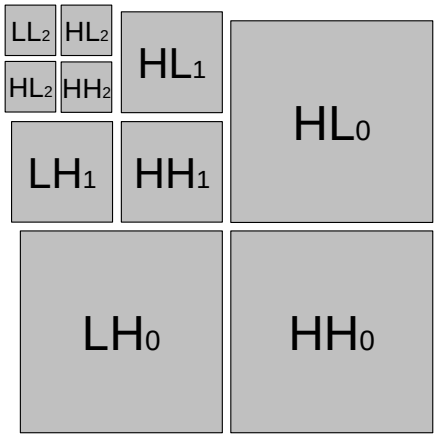

Figura 5.33: *Frames* resultantes da 2D-DWT de três oitavas.

Para qualquer variação no grau de paralelismo acima de três codificadores de entropia para este tipo de implementação resulta que cada codificador seria utilizado para codificar os coeficientes de cada sub-banda específica. Entretanto como cada codificador de entropia representa 50% da área da arquitetura serializada da Figura 5.31 e proporciona somente uma redução de 67% no tempo de processamento para graus de paralelismo maior que três, ou seja, apenas 2% melhor que a arquitetura paralelizada da Figura 5.32. Assim, conclui-se que para proporcionar um uma boa eficiência na compressão de imagens, o grau ideal para o paralelismo dos blocos codificadores de entropia é três, e qualquer grau acima implica em maior relação custo-benefício.

6 CONCLUSÕES E TRABALHOS FUTUROS

Esta dissertação apresentou a implementação de arquiteturas direcionadas à composição de um compressor de imagens seguindo o padrão JPEG 2000. As principais contribuições foram o desenvolvimento de diversas arquiteturas para a implementação da Transformada Wavelet Discreta em uma e em duas dimensões e a implementação de uma arquitetura para o compressor de entropia. A implementação deste compressor de imagens representa a aquisição dos conhecimentos necessários para a construção de uma arquitetura de alta eficiência dedicada à compressão de imagens.

As principais arquiteturas componentes do compressor foram completamente desenvolvidas. Para a arquitetura da 2D-DWT diferentes descrições foram sintetizadas e simuladas, desde uma arquitetura que consome somente 1.565 LCs de um FPGA da família STRATIX, que foi prevista sua operação com a freqüência máxima de 40MHz, até uma arquitetura que consome 2.090 LCs do mesmo tipo de FPGA e permitindo operação até 78MHz. Com o bloco 2D-DWT operando a 78MHz é possível obter uma taxa de processamento de 28 milhões de coeficientes por segundo, logo uma imagem com resolução de 256x256 pixels pode ser computada pela etapa da transformada com três oitavas em apenas 2,3ms.

Para a arquitetura do codificador de entropia, somente uma versão para validação foi desenvolvida. A realização da interconexão entre os blocos BPC e BAC prevê que a arquitetura resultante ocupa 2.152 LCs e pode apresentar freqüência máxima de operação de até 53MHz. Considerando que o codificador de entropia não apresenta taxa de saída constante, ou seja, o tempo para a geração de um byte codificado é dependente do conjunto de dados de entrada, taxa de saída pode variar entre 382 mil coeficientes por segundo até 843 mil coeficientes por segundo. Este rendimento pode ser melhorado mediante paralelização desta etapa, podendo resultar em uma taxa de saída entre 1,1 milhões e 2,5 milhões de coeficientes por segundo.

Os resultados obtidos demonstram que a etapa de transformada wavelet apresenta boa eficiência, quando comparada com os resultados de desempenho de arquiteturas comercias. O codificador de entropia apresenta um rendimento apenas satisfatório, que pode ser melhorado mediante paralelização.

O padrão JPEG2000 foi definido em 2000, entretanto ainda não é largamente disseminado como o seu padrão antecessor, o padrão JPEG e ainda não possui amplo suporte de *softwares* direcionado a ele. Porém é possível encontrar padrões semelhantes utilizados em aplicações importantes, como os padrões de transferência de imagens que a NASA tem utilizado ultimamente em suas sondas espaciais (KIELY, 2002). Este padrão da NASA é constituído por blocos internos semelhantes aos encontrados no padrão JPEG2000. A utilização destes padrões prova a importância da utilização de

técnicas de compressão de imagens que utilize técnicas como transformada wavelet e codificação aritmética.

No padrão JPEG2000 a transformada em duas dimensões é implementada pela computação da 1D-DWT aplicada a uma dimensão (por exemplo: as linhas da imagem), seguida da computação da 1D-DWT para a segunda dimensão (colunas da imagem). Logo uma arquitetura com processamento intercalado que visa à computação de todas as oitavas de uma dimensão e somente após a computação da DWT da outra dimensão, não é recomendada para o padrão JPEG2000, mas poderiam ser utilizadas em outros padrões de compressão proporcionando incremento da taxa de processamento dos coeficientes.

Algoritmos descritos em forma de fluxograma, para a descrição de uma determinada função, como as apresentadas na seção 5, podem ser traduzidos diretamente quando da implementação de uma descrição usando uma linguagem de programação, como a linguagem C, porém para uma descrição em VHDL o algoritmo em forma de seqüência de processos não produz resultados satisfatórios. As linguagens para descrição de *hardware* são utilizadas para descrever e simular a execução de circuitos reais, ou seja, uma das principais características desta linguagem é a execução simultânea das operações. Assim, a utilização de operações seqüenciais em uma descrição de hardware proporciona uma arquitetura com uma grande quantidade de estados na etapa de controle, resultando em um *throughput* reduzido.

Esta dissertação apresentou o desenvolvimento do compressor de imagens com perdas, utilizando a transformada irreversível, por permitir maior taxa de compressão, assim é proposto o desenvolvimento da arquitetura da transformada reversível e do compressor de imagens sem perdas como trabalho futuro. A implementação desta arquitetura representa a possibilidade de comparação entre o grau de inserção de erros nos coeficientes transformados e quantidade de coeficientes necessários para a implementação da arquitetura.

Um dos problemas apontados nesta dissertação diz respeito à inserção de erros na transformada *wavelet* ocasionada devido à truncagem dos coeficientes, então é sugerida a implementação de uma arquitetura para a transformada wavelet irreversível utilizando operadores de multiplicação inteira com uma quantidade maior de bits, para avaliar a relação entre o erro inserido nos coeficientes e o ponto de truncagem ideal dos coeficientes da transformada. Uma segunda variação deste tipo de implementação diz respeito a implementação de Operadores AMA com deslocamento diferente de 8 bits, ou seja a utilização de coeficientes em escala.

A dificuldade de avaliação da qualidade da imagem remete à necessidade de implementação da arquitetura DWT inversa. A arquitetura para implementar a DWT inversa segundo o esquema *lifting* utiliza o fluxo de operação exatamente inverso ao da DWT direta, deste modo, a descrição desta arquitetura pode prever a reutilização dos Operadores AMA utilizados na DWT direta.

Em relação ao nível de erro inserido pelo arredondamento dos coeficientes na DWT irreversível e na etapa de quantização, é recomendado um estudo matemático mais aprofundado.

A arquitetura da transformada *wavelet* descrita nesta dissertação pré-define a quantidade de oitavas da arquitetura, bem como o tamanho do *frame* de entrada. Estas variáveis são parametrizáveis, logo podem ser definidas no momento da síntese,

entretanto seria desejável uma arquitetura de compressor que permita a computação de qualquer quantidade de oitavas e qualquer quantidade de *frames*, possuindo pinos de entradas para estas definições.

Outro trabalho futuro importante é a descrição estrutural de todos módulos do compressor de imagens, de modo a permitir a geração de um circuito em *standard cells* reutilizável como um *core* IP.

REFERÊNCIAS

ACHARYA, T.; TSAI, P.-S. **JPEG2000 Standard for Image Compression:** Concepts, Algotithms and VLSI Architectures. New Jersey: Wiley, 2005.

ALTERA. **Apex20K Data Sheet**. v. 5.0, Jan. 2004. Disponível em: <http://www.altera.com/literature/ds/apex.pdf>. Acesso em: jul. 2005.

ALTERA. **FPGA, CPLD, & Structured ASIC Devices**: Altera, the Leader in Programmable Logic. Disponível em: <http://www.altera.com>. Acesso em: abr. 2005.

ALTERA. **Quartus II Development Software Handbook.** v. 4.2. (Complete Three-Volume Set), Jan. 2005. Disponível em: <http://www.altera.com/literature/hb/qts/quartusii_handbook.pdf>. Acesso em: jan. 2005.

ALTERA. **Stratix Device Handbook.** (Complete Two-Volume Set), Jan. 2005. Disponível em: <http://www.altera.com/literature/hb/stx/stratix_handbook.pdf>. Acesso em: jan. 2005.

ANALOG DEVICES. **ADV202 - JPEG 2000 Video CODEC**. Disponível em: <http://www.analog.com/UploadedFiles/Data_Sheets/85672960ADV202_b.pdf>. Acesso em: jul. 2005.

ANALOG DEVICES. **ADV601 - Low Cost Multiformat Video Codec**. Disponível em: <http://www.analog.com/UploadedFiles/Data_Sheets/646226113ADV601_0.pdf>. Acesso em: jul. 2005.

ANALOG DEVICES. **ADV611 - Closed Circuit TV Digital Video Codec**. Disponível em: <http://www.analog.com/UploadedFiles/Data_Sheets/461616590ADV611_2_0.pdf>. Acesso em: jul. 2005.

ANDRA, K.; CHAKRABARTI, C.; ACHARYA, T. A High-Performance JPEG2000 Architecture**. IEEE Transactions on Circuits and Systems for Video Technology**, New York, v.13, n.3, Mar. 2003.

BARCO SILEX. **BA112JPEG2000E - JPEG2000 Encoder**. Disponível em: <http://www.barco.com/subcontracting/Downloads/IPProducts/BA112JPEG2000EFact Sheet.pdf >. Acesso em: jul. 2005.

BARCO SILEX. **BA113FDWT - Discrete Wavelet Transform**. Disponível em <http://www.barco.com/subcontracting/Downloads/IPProducts/BA113FDWTFactSheet.pdf>. Acesso em: jul. 2005.

BARCO SILEX. **Barco - Visibly Yours**. Disponível em: <http://www.barco-silex.com>. Acesso em: jul. 2005.

BHASKARAN, V.; KONSTANTINIDES, K. **Image and Video Compression Standard Algorithms and Architectures**. 2nd ed. Norwell: Kluwer Academic Publishers, 1999.

BRIGHAM, E. O. **The Fast Fourier transform**. [S.l.]: Prentice-Hall, 1974.

BROWN, S. D.; FRANCIS, R. J.; ROSE, J. ; VRANESIC, Z. G. **Field-Programmable Gate Arrays**. Norwell: Kluwer Academic Publishers, 1992.

BROWN, S. D.; VRANESIC, Z. G. **Fundamentals of Digital Logic With VHDL Design**. New York: McGraw-Hill, 2000.

CADENCE DESIGN SYSTEMS. Disponível em: <http://www.cadence.com>. Acesso em: abr. 2005.

CAST, Inc. **CAST - IP cores for ASICs and FPGAs**. Disponível em: <http://www.cast-inc.com>. Acesso em: jul. 2005.

CAST. **JPEG2K_E - JPEG 2000 encoder**. Disponível em: <http://www.cast-inc.com/cores/jpeg2k_e/cast_jpeg2k_e-x.pdf>. Acesso em: jul. 2005.

CAST. **RC_2DDWT**. Disponível em: <http://www.cast-inc.com/cores/rc_2ddwt/rc_2ddwt-a.pdf>. Acesso em: jul. 2005.

CHAKRABARTI, C.; VISHWANATH, M. Efficient Realizations of the Discrete and Continuous Wavelet Transforms: From Single Chip Implementations to Mappings on SIMD Array Computers**. IEEE Transactions on Signal Processing**, New York, v.43, n.3, Mar. 1995

CHRISTOPOULOS, C.; SKODRAS, A.; EBRAHIMI, T. The JPEG2000 Still Image Coding System: An Overview. **IEEE Transactions on Consumer Electronics**, New York, v.46, n.4, p. 1103-1127, Nov. 2000.

DAUBECHIES, I.; SWELDENS, W. Factoring Wavelet Transform into Lifting steps**. J. Fourier Anal. Appl.**, [S.l.], v.4, n.3, p. 247-269, 1998.

DILLEN, G. et al. Combined Line-Based Architecture for the 5-3 and 9-7 Wavelet Transform of JPEG2000. **IEEE Transactions on Circuits and Systems for Video Technology**, New York, v.13, n. 9, Sept. 2003.

DIOU C.; TORRES, L.; ROBERT, M. An Embedded Core for the 2D Wavelet transform. In: IEEE CONFERENCE ON EMERGING TECHNOLOGIES AND FACTORY AUTOMATION, ETFA, 8., 2001, Antibes-Juan les Pins, France. **Proceedings…** [S.l.: s.n.], 2001. v.2, p.179-186.

DIOU C.; TORRES, L.; ROBERT, M. Implementation of a Wavelet Transform Architecture for Image Processing. In: INTERNATIONAL CONFERENCE ON VERY LARGE SCALE INTEGRATION, VLSI, 10., 1999. **Proceedings...** [S.l.: s.n.], 1999. p. 101-112.

FAIRCHILD. **CCD 595**. Fairchild Imaging Technology & Design. Disponível em: <http://www.fairchildimaging.com/main/ccd_area_595.htm>. Acesso em: maio 2005.

FRIDMAN, J.; MANOLAKOS, E. S. Distributed Memory and Control VLSI Architectures for the 1-D Discrete Wavelet Transform. In: IEEE VLSI SIGNAL PROCESSING, 7., 1994, La Jolla, California. **Proceedings…** [S.l.: s.n.], 1994. p. 388-397.

GANGADHAR, M.; BHATIA, D. FPGA based EBCOT architecture for JPEG 2000. In: INTERNATIONAL CONFERENCE ON FIELD-PROGRAMMABLE TECHNOLOGY, FPT, Tokyo. **Proceedings…** [S.l.:s.n.], 2003. p. 228-233.

GHANBARI, M. **Standard Codecs:** Image Compression to Advanced Video Coding. United Kingdom: [s.n.], 2003. (IEE Telecommunidations Series 49).

GONZALES, R. C.; WOODS, R. E. **Digital Image Processing**. [S.l.]: Addison-Wesley Publishing Company, 1992.

ITU-T. **T.800**. JPEG 2000 IMAGE CODING SYSTEM - Coding of Still Pictures. Mar. 2000.

ITU-T. **T.81**. Information Technology - Digital Compression and Coding of Continuous - Tone Still Images. Sept. 1992.

JAWERTH, B.; SWELDENS, W. An Overview of Wavelet Based Multiresolution Analysis**. SIAM Rev.**, Philadelphia, v.36, n.3, p.377-412, 1993.

KIELY, A.; KLIMESH M. **The ICER Progressive Wavelet Image Compressor**. [S.l.:s.n.], Nov. 2003. (NASA – JPL - IPN Progress Report 42-155).

KIELY, A.; KLIMESH M.; MAKI J. ICER on MARS: Wavelet-based Image Compression for the Mars Exploration Rovers. **IND Technology and Science News**, n.15, p.15-19, June 2002.

KNOWLES, G. VLSI Architectures for the Discrete Wavelet Transform. **Electronics Letters**, [S.l.], v.26, n.15, p. 1184-1185, July 1990.

LAKSHMINARAYANAN, G. et al. Design and FPGA Implementation of Image Block Encoders with 2D-DWT. In: CONFERENCE ON CONVERGENT TECHNOLOGIES FOR ASIA-PACIFIC REGION, TEKON, 2003. **Proceedings…** [S.l.: s.n.], 2003. v.3, p. 1015-1019.

MALLAT, S. G. **A Wavelet Tour of signal processing**. San Diego: Academic Press, 1998.

MALLAT, S. G. Multifrequency Channel Decompositions of Images and Wavelet Models. **IEEE Transactions on Acoustics, Speech and Signal Processing**, New York, v.37, n.12, Dec. 1989.

MASUD, S.; McCANNY, J. V. Reusable Silicon IP Cores for Discrete Wavelet Transform Applications. **IEEE Transactions on Circuits and Systems-I**: Regular Papers, New York, v.51, n.6, June 2004.

MOTRA, A. S.; BORA, P. K.; CHAKRABARTI, I. An Efficient Hardware Implementation of DWT and IDWT. In: CONFERENCE ON CONVERGENT TECHNOLOGIES FOR ASIA-PACIFIC REGION, TEKON, 2003. **Proceedings…** [S.l.: s.n.], 2003. v.3.

PARHI, K. K.; NISHITANI, T. Folded VLSI Architectures for the Discrete Wavelet Transform. **IEEE Transactions on VLSI Systems**, New York, 1993.

PENNEBAKER, W.; MITCHELL, J. **JPEG Still Image Data of Compression Standard**. New York: Van Nostrand Reinhold, 1992.

SALOMON, D. **Data Compression. The Complete Reference**. 2nd ed. New York: Springer-Verlag, 2000.

SILVA, S. V. **Algoritmos e Arquitetura Dedicada em FPGA para a Transformada Discreta do Cosseno (DCT)**. 2003. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

SILVA, S. V. **Arquiteturas para a Transformada Wavelet Discreta**. 2004. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

SILVA, S. V.; BAMPI, S. Area and Throughput Trade-offs in the Design of Pipelined Discreve Wavelet Transform Architectures. In: DESIGN, AUTOMATION AND TEST IN EUROPE, DATE, 2005, Munich, Germany. **Proceedings…** [S.l.: s.n.], 2005. p. 32-37.

SILVA, S. V.; BAMPI, S. Implementação de Arquiteturas para a transformada wavelet discreta Unidimensional em FPGAs. In: WORKSHOP IBERCHIP, 11., 2005, Salvador, Brasil. **Proceedings…** [S.l.: s.n.], 2005.

SWELDENS, W. The lifting scheme: A construction of second generation wavelets. **Siam J. Math. Anal. Journal**, [S.l.] v.29, n.2, p. 511-546, 1997.

SWELDENS, W. The lifting scheme: A custom-design construction of biorthogonal wavelets**. Appl. Comput. Harmon. Anal**, [S.l.], v.3, n.2, p. 186-200, 1996.

SWELDENS, W. The Lifting Scheme: A New Philosophy in Biorthogonal Wavelet Constructions. In: WAVELET APPLICATIONS IN SIGNAL AND IMAGE PROCESSING CONFERENCE, SPIE, 3., 1995. **Proccedings…** [S.l.: s.n.], 1995.

SYED, S.; BAYOUMI, M.; LIMQUECO, J. An Integrated Discrete Wavelet Transform Array Architecture. In: WORKSHOP ON COMPUTER ARCHITECTURE FOR MACHINE PERCEPTION, Como, Italy. **Proccedings…** [S.l.: s.n.], 1995. p. 32-36.

TAUBMAN, D.; MARCELLIN, M. W. **JPEG2000 Image Compression:** Fundamentals, Standards and Practice. Massachusetts: Kluwer Academic Publishers, 2002.

USEVITCH, B. E. **A** Tutorial on Modern Lossy Wavelet Image Compression: Foundations of JPEG 2000. **IEEE Signal Processing Magazine**, New York, Sept. 2001.

VISHWANATH, M. The Recursive Pyramid Algorithm for the Discrete Wavelet Transform. **IEEE Transactions on Signal Processing**, New York, v.42, n.3, Mar. 1994.

WEEKS, M.; BAYOUMI, M. Discrete Wavelet Transform: Architectures, Design and Performance Issues. **Journal of VLSI Processing**, [S.l.], v.35, 2003.

WIKIPEDIA. **Eye**. Disponível em: <http://en.wikipedia.org/wiki/Eye>. Acesso em: maio 2005.

WIKIPEDIA. **Graphics file format**. Disponível em: <http://en.wikipedia.org/wiki/Image_format>. Acesso em: maio 2005.

WIKIPEDIA. **Single-lens reflex camera**. Disponível em: <http://en.wikipedia.org/wiki/Single-lens_reflex_camera>. Acesso em: maio 2005.

ANEXO FORMATOS COMUNS PARA ARQUIVOS GRÁFICOS E IMAGENS

Este anexo apresenta um resumo dos formatos mais comuns para arquivos de gráficos e imagens de acordo com o apresentado em WIKIPEDIA (2005-b).

Extensão do arquivo	denominação	Descrição
.art	ART	ART é um formato de arquivo de imagens proprietário muito utilizado por software clientes da America Online®. O formato ART aplica uma alta taxa de compressão em uma única imagem em tons contínuos.
.bmp	Windows Bitmap	Normalmente utilizado por programas dos pacotes Microsoft Windows, e do sistema operacional Windows. Pode ser especificada a utilização de compressão com ou sem perdas, porém a maioria dos programas utilize somente no modo não comprimido.
.cin	Cineon	É considerado um subconjunto do formato de arquivos ANSI/SMPTE DPX com cabeçalho fixo.
.cpt	Corel Photo-Paint Image	Formato proprietário padrão para documento do Corel Photo-Paint. É suportado por poucos editores de imagem fora do pacote da Corel. Imagens deste padrão são usualmente menores que documento Photoshop.
.dpx	Digital Picture eXchange file format	O ANSI/SMPTE DPX é um padrão Kodak similar ao Cineon, porém com cabeçalhos de imagem variáveis e flexíveis.
.exr	Extended Dynamic Range Image File Format	OpenEXR é um formato de arquivo de código aberto de alta faixa dinâmica (*high dynamic-range* - HDR) desenvolvido pela *Industrial Light & Magic* para produção de vídeo. A principal vantagem deste formato é a utilização de pixels em ponto flutuante até 32-bits e algoritmos para a compressão de imagens sem perdas.
.fpx	Flashpix (1.0.2)	Formato que permite múltiplas resoluções para cada imagem. Desde sem compressão até compressão com perdas, para imagens desde tons de cinza de 8-bits até colorida de 24-bits.

.gif	Graphics Interchange Format	GIF é considerado um dos padrões mais utilizados na *web*. Suporta imagens animadas com até 255 tons por *frame*, utilizando quantização com perdas. Utiliza compressão LZW sem perdas.
.iff .ilbm	Interchange file format / Interleave bitmap	Formato de arquivo popular nos computadores Amiga. ILBM é um subtipo do formato IFF, podendo conter mais que somente imagens.
.jpeg .jpg	Joint Photographic Experts Group	JPEG é o padrão normalmente utilizado para fotos e outras imagens em tons contínuos na *web*. Utiliza compressão com perdas. A qualidade pode variar grandemente dependendo das configurações da compressão.
.jpg2 .jp2	Joint Photographic Experts Group	JPEG 2000 é o novo formato de arquivo baseado na utilização de *wavelets*. Possui opção para compressão com e sem perdas.
.mng	Multiple-image Network Graphics	Formato de animação utilizando *datastreams* similar a PNG e JPEG. Foi originalmente projetado para substituir o uso de animações GIF na *web*.
.pbm	Portable Bitmap Format	Formato simples para gráficos em preto e branco. Utiliza 1-bit por *pixel*.
.pcd	ImagePac Photo CD	Formato proprietário da Kodak, utiliza compressão com perdas em imagens coloridas de 24-bits.
.pgm	Portable Graymap Format	Formato simples para gráficos em escalas de cinza, utilizando 8-bits por *pixel*.
.png	Portable Network Graphics	È um formato de imagem que utiliza compressão com perdas, oferecendo profundidade de 1 até 32 bits. Foi projetado principalmente para substituir o padrão GIF na *web*.
.ppm	Portable Pixmap Format	Formato simples para gráficos coloridos. Utiliza 24-bits por *pixel*. (8-bits para vermelho, 8-bits para azul e 8-bits para verde).
.psd	application/ x-photoshop	Formato para documentos Adobe Photoshop.
.psp	Paint Shop Pro Document	Formato padrão para documentos Paint Shop Pro.
.tiff .tif	Tagged Image File Format	Formato utilizado extensivamente para gráficos para impressão. Suporta compressão com e sem perdas.
.wbmp	Wireless Application Protocol Bitmap Format	Formato utilizado principalmente com WML para dispositivos *wireless*.
.xbm	X BitMap	Formato originalmente em preto e branco, para o sistema X-Window. É suportado por muitos *web browsers*.
.xpm	X-Pixmap	Utilizado quase exclusivamente em plataformas UNIX, com o sistema X-Windows. Foi inspirado no formato XBM.