UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

MARIANO MAJOLO

PROJETO DE DIPLOMAÇÃO

ARQUITETURA DOS MÓDULOS DE TRANSFORMADAS E DE QUANTIZAÇÃO DE UM CODIFICADOR DE VÍDEO H.264

Porto Alegre
2010

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

ARQUITETURA DOS MÓDULOS DE TRANSFORMADAS E DE QUANTIZAÇÃO DE UM CODIFICADOR DE VÍDEO H.264

Projeto de Diplomação apresentado ao Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul, como parte dos requisitos para Graduação em Engenharia Elétrica.

ORIENTADOR: Altamiro Amadeu Susin
CO-ORIENTADOR: Ronaldo Hüsemann

Porto Alegre
2010

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

MARIANO MAJOLO

ARQUITETURA DOS MÓDULOS DE TRANSFORMADAS E DE QUANTIZAÇÃO DE UM CODIFICADOR DE VÍDEO H.264

Este projeto foi julgado adequado para fazer jus aos créditos da Disciplina de "Projeto de Diplomação", do Departamento de Engenharia Elétrica e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____

Prof. Altamiro Amadeu Susin, UFRGS

Doutor pela Institut National Polytechnique de Grenoble

(Grenoble, França)

Banca Examinadora:

Prof. Dr. Altamiro Amadeu Susin, UFRGS

Doutor pela Institut National Polytechnique de Grenoble – Grenoble, França

Prof. Me. Ronaldo Hüsemann, UNIVATES

Mestre pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Prof. Dr. Walter Fetter Lages, UFRGS

Doutor pelo Instituto Tecnológico de Aeronáutica – São José dos Campos, Brasil

Porto Alegre, Dezembro de 2010.

DEDICATÓRIA

Dedico este trabalho aos meus pais, e família em especial pela dedicação e apoio em todos os momentos difíceis.

AGRADECIMENTOS

Aos colegas do PRAV que ajudaram muito na realização deste trabalho direta ou indiretamente.

Aos colegas de curso pelo auxílio e apoio na revisão deste trabalho e por estarem presentes nos momentos difíceis.

À Digitel pela flexibilidade de horário que permitiu o bom andamento das atividades acadêmicas.

Ao professor Altamiro Amadeu Susin pela orientação no trabalho.

À Universidade Federal do Rio Grande do Sul, professores e funcionários por oferecer um ensino público de qualidade.

À FINEP por ter financiado o projeto Rede H.264.

E um agradecimento especial ao professor Ronaldo Hüsemann que acompanhou o desenvolvimento deste trabalho do início ao fim.

RESUMO

Há poucos anos (2005) o Brasil entrou na era da Televisão Digital, adotando o padrão H.264 como mecanismo de codificação de vídeo para sua transmissão. Este padrão de codificador, bem mais eficiente que outros vigentes (H.262/MPEG-2 ou MPEG-4), trás como conseqüência um aumento da complexidade computacional. A fim de se garantir que a codificação de vídeo possa acontecer com características de tempo real, muitas soluções comerciais fazem uso de implementações em hardware ou em arquiteturas otimizadas (DSP ou MCP). A presente monografia apresenta soluções para os módulos de codificação direta de transformadas (DCT e Hadamard) e quantização, bem como de seus respectivos módulos inversos. O trabalho deve considerar abordagens com foco na redução da área ocupada na FPGA e/ou redução de atrasos e tempos de processamento, buscando soluções implementáveis na prática. Todas as arquiteturas desenvolvidas neste trabalho são descritas em VHDL e sintetizadas para FPGAs Virtex-II Pro da Xilinx.

Palavras-Chave: Codificação, Vídeo, H.264, DCT, Hadamard, Quantização, Engenharia Elétrica.

ABSTRACT

A few years ago (2005) Brazil has entered the digital television era, adopting the H.264 standard as a video compression mechanism for its transmission. This encoder standard, much more efficient than other existing (H.262/MPEG-2 or MPEG-4), has as consequence an increased computational complexity. In order to ensure a video encoding in real time, many commercial solutions make use of hardware or architecture optimized implementations (DSP or MCP). This monograph presents solutions to the direct transforms (DCT and Hadamard) and quantization encoding modules, as well as the inverses of their respective modules. The monograph should consider approaches focused on reducing the area occupied in the FPGA and / or reduction of delays and processing times, seeking for implementable solutions. All architectures developed in this monograph are described in VHDL and synthesized for Virtex-II Pro FPGAs from Xilinx.

Keywords: Encoding, Video, H.264, DCT, Hadamard, Quantization, Electrical Engineering.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

AVC:Advanced Video Coding

Cb: Chroma blue

CIF: Common Intermediate Format

CODEC: Encoder/Decoder

Cr: Chroma red

DCT: Discrete Cossine Transform

DELET: Departamento de Engenharia Elétrica

DPCM: Differential Pulse Code Modulation

DVD: Digital Video Disc

FQ: Forward Quantization

GPS: Global Position Satellite

HD: High Definition

HSI: Hue Saturation Intensity

IEC: International Electrotechnical Commission

ISO: International Organization for Standardization

ITU: International Telecommunication Union

JM: Joint Model

JPEG: Joint Photographic Experts Group

JVT: Joint Video Team

MPEG: Motion Picture Experts Group

MSE: Mean Square Error

NTSC: National Television System Committee

PAL TV: Phase Alternative Line Television

PF : Post-scaling Factor

PRAV: Projetos em Áudio e Vídeo

PSNR: Peak Signal to Noise Ratio

qbits: quantum bits

QP: Quantization Parameter

Qstep: Quantization Step

RGB: Red Green Blue

SBTVD: Sistema Brasileiro de Televisão Digital

SD: Standard Definition

SDTV : Standard Definition Television

SIF: Standard Image Format

SVC: Scalable Video Coding

TV: Television

UFRGS: Universidade Federal do Rio Grande do Sul

VCR: Video Cassete Recorder

VGA: Video Graphics Array

VHDL: VHSIC Hardware Description Language

INTRODUÇÃO

Um sinal de vídeo é responsável pela transferência da informação visual de um ponto a outro (JACK, 2001). Estes sinais podem ser transmitidos de uma emissora de televisão para uma residência via satélite ou mesmo provir de um aparelho de DVD (*Digital Video Disc*), filmadora ou de um celular com câmera. Portanto, o sinal de vídeo geralmente utiliza um meio de propagação, ocupando uma certa banda e/ou um local para seu armazenamento.

De forma geral, um sinal digital tem muita informação agregada, o que é um problema para a transmissão e armazenamento do mesmo. Considerando-se, por exemplo, uma imagem SD (*Standard Definition)* (720×480 pixels) no sistema de três cores primárias RGB (*Red Green Blue*), com 8 bits de representação para cada cor (3 cores = 24 bits por pixel). Essa imagem ocupa um espaço de memória de 1.036.800 Bytes (1012,5 KB) na sua forma original, ou seja, se não for comprimida.

$$720 \cdot 480 \cdot 3\ Bytes = 1036800\ Bytes \qquad (1)$$

Se imagens como essa fossem utilizadas para compor um vídeo SD com taxa de exibição de 30 quadros (frames) por segundo, o espaço necessário para armazenamento de um segundo do vídeo é de 29,66 MB.

$$(1036800 Bytes/frame) \cdot (30\ frames/s) \cdot (1s) = 29{,}66 MB \qquad (2)$$

Ou seja, um vídeo como esse demandaria uma taxa de transmissão de pelo menos 29,66 MB por segundo ou 237,3 Mbps. Esta taxa deve ser considerada alta mesmo para sistemas atuais de internet de banda larga, uma vez que se exige uma taxa mínima de 240 Mbps. Da mesma forma um disco rígido de 500 GB estaria cheio com 4,8 horas de gravação se o vídeo fosse armazenado desta forma, possivelmente com mais tempo sendo gasto gravando o vídeo no disco rígido que reproduzindo o mesmo.

Apesar das taxas de transmissão dos canais de comunicação continuarem crescendo ano após ano, juntamente com a capacidade de armazenamento dos discos rígidos, memórias *flash* e mídias ópticas, a transmissão e armazenamento de vídeo cru é impraticável na maioria dos casos. Considerando-se este contexto se torna importante o recurso de compressão de vídeo.

A presente monografia tem por objetivo apresentar o estudo sobre os módulos de Transformadas e Quantização presentes em um codificador H.264. Esse padrão de codificação foi adotado pelo Sistema Brasileiro de Televisão Digital (SBTVD) para ser utilizado na transmissão e recepção do sinal aberto de televisão terrestre (SBTVD, 2010).

O estudo dos módulos de Transformadas e Quantização para implementação em *hardware* é de extrema importância. Ao implementar esses módulos em *hardware* buscou-se atingir os requisitos para a codificação em tempo real.

Sob financiamento da FINEP, o trabalho aqui apresentado está inserido no grande projeto Rede H.264 SBTVD. O objetivo da Rede H.264 é desenvolver produtos de interesse nacional na área de codificação e decodificação de áudio e vídeo para o SBTVD.

O texto está organizado de modo a apresentar conceitos básicos de codificação de vídeo no capítulo 2. Estes conceitos são importantes por serem abordados no decorrer do trabalho. O capítulo 3 apresenta um codificador de vídeo com foco especial no H.264, padrão estudado. No capítulo 4 são apresentados os módulos de DCT (*Discrete Cosine Transform*), Hadamard e Quantização que têm sua arquitetura para hardware detalhada no capítulo 5. O capítulo 6 apresenta otimizações interessantes de se fazer ao se trabalhar com os módulos de Transformadas e Quantização propostos. O capítulo 7 apresenta resultados práticos e as conclusões tiradas desse trabalho.

2 CONCEITOS BÁSICOS

2.1 DIGITALIZAÇÃO DO VÍDEO

Questões como quantidade de espaço usado para armazenamento ou taxa de bits para transmissão só fazem sentido quando o sinal de vídeo em questão é digital. Nesta seção se abordam conceitos relevantes relacionados à digitalização de sinais para a formação de vídeos digitais.

2.1.1 AMOSTRAGEM

Para representar um vídeo na forma digital é preciso amostrá-lo tanto temporalmente quanto espacialmente. A amostragem temporal consiste em representar o vídeo na forma de uma sequência de imagens paradas separadas por um espaço de tempo conhecido em sua reprodução. A amostragem espacial geralmente é feita dividindo-se cada imagem em vários pedaços ou blocos menores (RICHARDSON, 2003).

O formato mais comum para uma amostragem espacial é o retangular, como na Figura 1. As amostras são agrupadas em macroblocos, correspondendo à região de 16×16 pixels em um quadro (RICHARDSON, 2003).

Quanto maior a definição da imagem (largura × altura em pixels) maior será a fidelidade da mesma em relação à original, porém maior será também o número de blocos usados para compor cada imagem, o que aumenta proporcionalmente o espaço de memória ocupada e a demanda por dados a serem processados.

Uma imagem de um vídeo é representada por um conjunto de pixels segundo a amostragem espacial, onde cada pixel representa um ponto na tela, que deve apresentar uma cor específica.

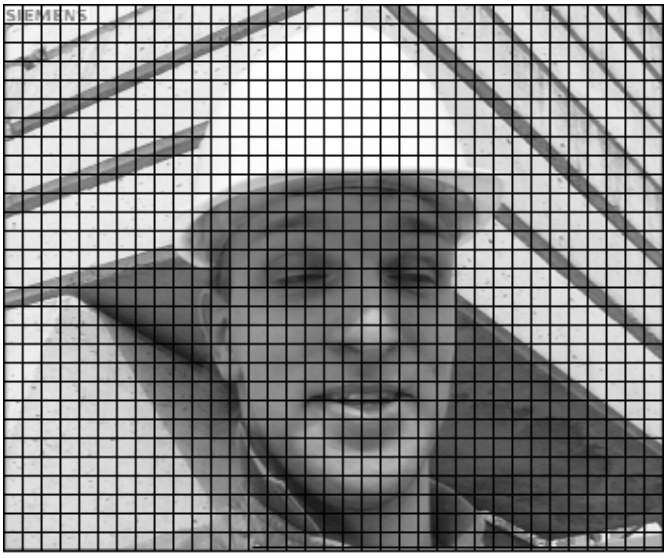


Figura 1 Imagem original com grade definindo o que serão os pixels.

2.1.2 ESPAÇO DE CORES

A cor de um dado pixel pode ser registrada como uma combinação de cores primárias, também chamadas espaço de cores, sendo que para as combinações existe mais de um padrão. Alguns espaços de cores usados na representação de imagens digitais são o RGB, HSI e YCbCr. (SHI & SUN, 1999).

Monitores de vídeo utilizam o espaço de cores RGB que são as três cores primárias captadas pelo sistema visual humano: vermelho, verde e azul (WOOTON,2005).

No espaço de cores YCbCr, também são utilizadas três componentes para representação de cores. Elas são a luminância (Y), que define a intensidade luminosa ou brilho; crominância azul (Cb – *Chroma blue*) e crominância vermelha (Cr – *Chroma red*). Este é o espaço de cores utilizado no sistema de TV (*Television*) e também nos codificadores de vídeo (BHASKARAN & KONSTANTINIDES, 1997).

O sistema visual humano é mais sensível ao brilho do que a cor e por esse motivo, este sistema permite certa economia de bits reduzindo o espaço para representação de Cb e Cr utilizados sem comprometer a qualidade visual do vídeo (RICHARDSON, 2003).

Em relação a este sistema foram definidos distintos formatos de representação YCbCr. O formato 4:4:4 é o padrão em que para cada 4 pixels de luminância, tem-se 4 pixels Cb e 4 Cr (neste caso, sem economia de bits). O formato 4:2:2 apresenta a cada 4 pixels de luminância apenas 2 pixels Cb e 2 Cr, ou seja, há uma prioridade para a representação da luminância. No formato 4:2:0, a cada 4 pixels de luminância, tem-se 1 pixel Cb e 1 pixel Cr. Assim, este formato usa a metade de bits em comparação com o 4:4:4. Estes são os padrões mais comuns e são ilustrados pela Figura 2. O padrão 4:2:0 é o padrão que será adotado neste trabalho.

As componentes Y, Cb e Cr são determinadas da seguinte maneira:

$$Y = 0{,}299R + 0{,}587G + 0{,}114B \quad (3)$$

$$Cb = 0{,}564(B - Y) \quad (4)$$

$$Cr = 0{,}713(R - Y) \quad (5)$$

Onde R, G e B são as cores vermelho, verde e azul do espaço de cores RGB (GHANBARI, 2003).

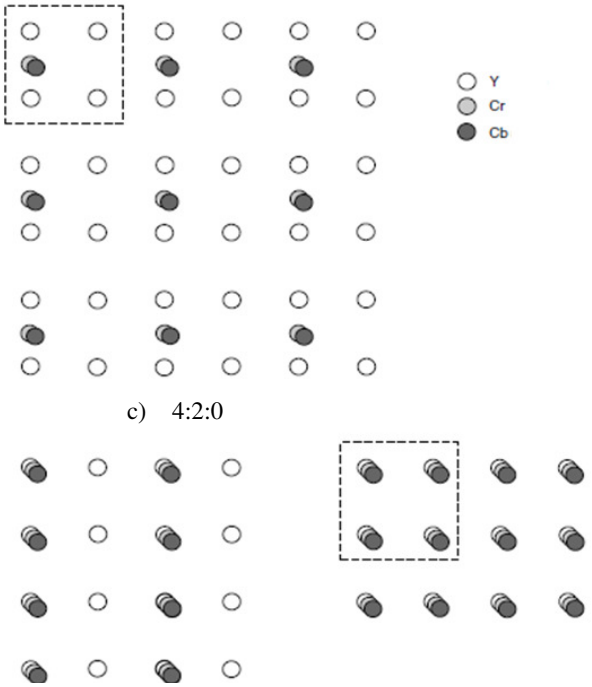


Figura 2 Padrões de amostragem de cores (RICHARDSON, 2003)

2.2 RESOLUÇÃO DE VÍDEO

Vídeos capturados apresentam formatos de resolução variados, dependendo da aplicação desejada. O formato CIF (*Common Intermediate Format*) é a base para uma série de outros formatos, que são listados na tabela abaixo. A informação do número de bits para representar uma imagem é calculado considerando-se o formato 4:2:0 e 8 bits para cada amostra de luminância e crominância:

Tabela 1 Formato CIF e derivados.

Formato	Resolução de Luminância (horiz. × vertic.)	Bits por frame (4:2:0, 8 bits por amostra)
SQCIF	128 × 96	147456
QCIF	176 × 144	304128
CIF	352 × 288	1216512
4CIF	704 × 576	4866048

Fonte: RICHARDSON, 2003

SDTV (*Standard Definition TV*) é definido diferente na Europa, nos Estados Unidos e outros lugares do mundo. Os padrões mais comuns de se encontrar são o NTSC e PAL TV. O padrão NTSC possui uma área visível de 640 × 480 pixels e uma taxa de atualização de 30 vezes por segundo. Já no padrão PAL TV a área visível é de 640 × 579 com 25 atualizações por segundo (WOOTTON, 2005).

Ainda existem os formatos de alta definição (HDTV-High Definition TV). Esses formatos também são definidos de forma diferente nos Estados Unidos e Europa. São algumas resoluções HD (WOOTTON, 2005):

- 1080 × 720 pixels;
- 1280 × 720 pixels;
- 1920 × 720 pixels;
- 1440 × 1080 pixels;
- 1920 × 1080 pixels;
- 1920 × 1200 pixels.

A escolha entre os diferentes formatos depende da aplicação e da capacidade de armazenamento ou transmissão. Por exemplo 4CIF é apropriado para representar vídeos de televisão enquanto que CIF e QCIF são comuns para videoconferências. As resoluções menores, QCIF e SQCIF, são utilizadas para dispositivos móveis como celular e GPS (*Global Position Satellite*) (RICHARDSON, 2003).

2.3 QUALIDADE

A qualidade de um vídeo não é uma medida de fácil obtenção e nem única, pois envolve o sistema visual humano. A impressão visual não pode ser fielmente representada por cálculos, portanto métodos subjetivos de medir qualidade são os mais adequados. Apesar dos

métodos subjetivos não poderem ser completamente substituídos, alguns métodos objetivos são utilizados em muitos casos por serem de mais fácil aplicação.

O método objetivo de medição qualidade mais comum é o PSNR (*Peak Signal to Noise Ratio*), que se baseia no erro quadrado médio (MSE – *Mean Square Error*) entre um vídeo original e um que passou por compressão e descompressão. A expressão que rege este método é a seguinte (RICHARDSON, 2003):

$$PSNR_{dB} = 10 \log_{10} \frac{(2^n - 1)^2}{MSE} \quad (6)$$

$$MSE = \frac{1}{\mathrm{n}^2} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \left(C_{ij} - R_{ij}\right)^2 \quad (7)$$

Onde

n é o número de bits por amostra de imagem.

C_{ij} é a amostra atual;

R_{ij} é a amostra de referência.

3 CODIFICADOR DE VÍDEO

Um codificador de vídeo explora as características de um sinal de vídeo para representá-lo da forma mais compacta possível e com menor perda de informação visual (RICHARDSON, 2003).

O codificador pode utilizar-se de diferentes algoritmos matemáticos ou entrópicos para converter a informação do vídeo para uma forma mais comprimida. Por sua vez, o decodificador converte esta representação reduzida dos dados de volta a uma forma visual, que não necessariamente é igual ao vídeo fonte, mas parecida (compressão com perda de informação). Para o conjunto codificador/decodificador dá-se o nome de CODEC (enCOder/DECoder).

A análise estatística de um vídeo indica a presença de correlação entre imagens consecutivas e até mesmo entre blocos da mesma imagem. O CODEC utiliza-se disso para diminuir a quantidade de bits necessária para a representação do vídeo (GHANBARI, 2003).

Em uma compressão sem perdas, a redundância estatística é removida para que o sinal original possa ser perfeitamente reconstruído no receptor. Infelizmente, os métodos sem perdas conseguem apenas uma modesta compressão de imagens e vídeos (algo entre 3 a 4 vezes). Já as técnicas de compressão com perdas conseguem chegar a uma redução bem maior do número de bits. Essa maior compressão, entretanto, pode comprometer a qualidade visual da imagem visto que remove redundâncias subjetivas do vídeo (RICHARDSON, 2003) .

Para ilustrar a importância de uma boa compressão, pode-se tomar como exemplo o padrão H.264/AVC de codificação de vídeo. Atingindo uma compressão de 50:1, a redução na qualidade do vídeo quase não pode ser percebida pelo sistema visual humano.

3.1 TIPOS DE REDUNDÂNCIA

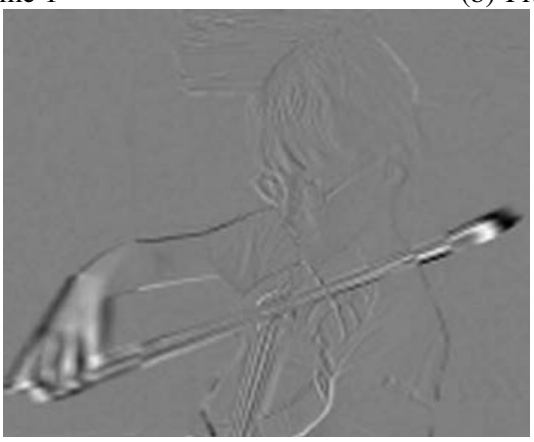
Os algoritmos de compressão de vídeo operam removendo redundância temporal e espacial (RICHARDSON, 2003).

3.1.1 REDUNDÂNCIA TEMPORAL

A redundância temporal, que também pode ser chamada de redundância entre-quadros, pode ser percebida ao analisar quadros consecutivos de um mesmo vídeo (GHANBARI, 2003). Tomando as imagens da Figura 3 como exemplo, (a) e (b) são duas imagens consecutivas em um vídeo. Mesmo que haja um pouco de movimentação da pessoa de um quadro a outro, todo o fundo que aparece na primeira imagem continua sendo o mesmo na segunda. Assim, toda a informação de fundo é considerada redundante. Para eliminar a redundância temporal, informações de um quadro são utilizadas na montagem de outro, sendo assim necessário acrescentar apenas a informação que difere entre uma imagem e outra. A diferença entre dois quadros é chamada de erro interquadro (GHANBARI, 2003).

A diferença entre as duas imagens da Figura 3, erro interquadro dessa sequência, é mostrado em (c). A representação do erro é uma imagem predominantemente cinza quando as diferenças entre quadros são pequenas porque é praticamente zero. Numa representação com valores positivos (claros) e negativos (escuros) o zero será representado por um tom de cinza. É fácil notar que houve uma grande redução na informação que deve ser codificada para gerar a figura (b) quando se tem (a) e (c). Houve uma troca da necessidade de codificar uma imagem com poucos zeros, ou seja, que exigem muitos bits de representação, pela possibilidade de codificar uma imagem representada por vários zeros.

(a) Frame 1 (b) Frame 2



(c) Diferença entre os frames

Figura 3 Remoção de redundância temporal(RICHARDSON, 2003).

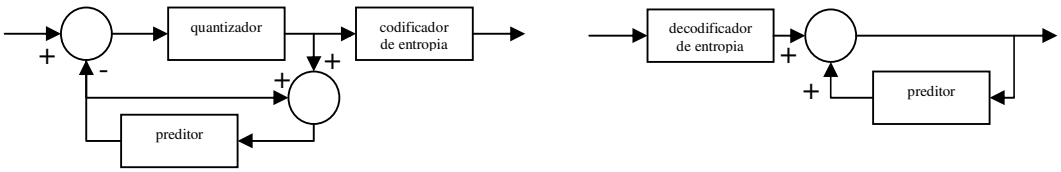
3.1.2 REDUNDÂNCIA ESPACIAL

A redundância espacial, também chamada de redundância intra-quadro, consiste em partes iguais ou parecidas em uma mesma imagem (GHANBARI, 2003). Pode-se observar na Figura 4 duas regiões destacadas, onde o conteúdo é bastante similar por toda a região, o que caracteriza a redundância espacial.

Baseado em valores já codificados, o método de redução da redundância espacial consiste em prever valores de pixels gerando, como no modelo temporal, o erro de predição. Este tipo de predição, chamado de DPCM (*Differential Pulse Code Modulation*) consegue prever melhor os pixels vizinhos ao pixel de referência (GHANBARI, 2003).

Figura 4 Imagem destacando redundâncias espaciais.

A Figura 5 mostra um diagrama de blocos de um codec DPCM. A diferença entre os pixels de entrada e da predição são quantizados (representados por valores aproximados e com menos bits) e codificados para transmissão. Na decodificação, o sinal de erro recebido é adicionado ao sinal de predição para a reconstrução do sinal (GHAMBARI, 2003).



a) Codificador b) Decodificador

Figura 5 Diagrama de Blocos de um codec DPCM (GHAMBARI, 2003)

Outro módulo utilizado para reduzir a redundância espacial é o módulo de transformadas. As transformadas convertem as amostras para outros domínios, nos quais a representação envolve menos informação útil. Um módulo de quantização utilizado após as transformadas elimina dos coeficientes transformados valores insignificantes, deixando a informação de saída mais compacta ainda.

3.2 PADRÕES DE CODIFICAÇÃO DE VÍDEO

Visando a padronização e a interoperabilidade entre diferentes equipamentos de diferentes fabricantes foram desenvolvidas normas internacionais de compressão de imagem e vídeo. Dentre elas as séries JPEG, MPEGx e H.26X (RICHARDSON, 2003).

Na década de 80, os estudos de codecs a base de blocos levou a escolha da DCT como unidade principal na compressão (GHANBARI, 2003).

Em 1989 a ITU-T (*International Telecommunication Union – Telecomunication Standardization Sector*) criou o padrão H.261. Com isso, lançou as bases do que é utilizado na maioria dos padrões de compressão de vídeo: estimação de movimento, transformada discreta do cosseno, quantização linear e codificação de entropia, tornando possível o uso de taxas de transmissão na ordem de 384 kbit/s (GHANBARI, 2003).

Após o H.261, o grupo MPEG (*Motion Picture Experts Group*) investigou técnicas para armazenamento de vídeos em CD-ROM, gerando o padrão MPEG-1, que atingia uma qualidade similar a dos VCRs (*Video Cassete Recorders*) a 1,5 Mbit/s (GHANBARI, 2003)

O padrão MPEG-2, que surgiu em seguida, também foi adotado pela ITU-T com o nome de H.262. Pode-se citar como exemplos de aplicações do padrão H.262/MPEG-2 o DVD, televisão digital via satélite ou cabo, entre outras. Esse padrão se tornou muito popular e foi adotado por muitos países como o padrão de televisão digital local.

Anos depois o H.263 foi criado e incorporou novos avanços que vinham sendo pesquisadas tanto pela academia como pela indústria (AGOSTINI, 2007).

MPEG-4 previu cenas sintéticas em conjunto com cenas naturais em um modelo de codificação baseada em objetos independentes. Outra inovação deste padrão é que as imagens são codificadas como objetos sem a restrição de blocos retangulares (GHANBARI, 2003).

Em 1997 os grupos de especialistas do MPEG e ITU-T formaram a equipe JVT (*Joint Video Team*) evitando a duplicação de esforços e acelerando na construção de um novo padrão (AGOSTINI, 2007).

A ITU-T decidiu adotar esse padrão com o nome de H.264, enquanto que a MPEG deu o nome de MPEG-4 parte 10 ou MPEG4-AVC (*Advanced Video Coding*). Muitos autores referenciam este padrão com o nome de H.264/AVC, para não confundir com a versão escalável SVC (*Scalable Video Coding*) do mesmo (AGOSTINI, 2007).

A Tabela 3 apresenta algumas aplicações para codecs de vídeo e o padrão de codificação recomendado. É interessante observar como o padrão H.264 é o mais abrangente.

Tabela 2 Formatos de codificação para diferentes aplicações.

Descrição	Formatos de Codificação Recomendados
DVD	MPEG-2, H.264
DVD-ROM	MPEG-1, MPEG-2, H.264
CD-ROM	MPEG-1, MPEG-2, MPEG-4 part 10, H.264
Internet discada	MPEG-1, H.264
Internet banda larga	MPEG-2, H.264
Vídeo conferência	H.261, H.263, H.264
Dispositivos móveis (3GPP&3GPP2)	H.264

Fonte: WOOTTON, 2005

3.3 CODEC H.264

No padrão H.264 cada quadro codificado pode ser de um dos três tipos:

- Quadro I: contém *slices* I;
- Quadro P: contém *slices* P;
- Quadro B: contém *slices* B.

Um *slice* é um conjunto de macroblocos. Seu tamanho pode variar de um macrobloco ao total de macroblocos de uma imagem.

Os quadros I são caracterizados por utilizar apenas o método de predição intra-quadro das amostras do mesmo slice. Quadros P e B utilizam a predição intra-quadro das imagens de referência para fazer a predição inter-quadros. Os *slices* que podem servir de referência para os quadros P são apenas os I enquanto que os quadros B podem se referenciar tanto a *slices* I quanto a *slices* P (RICHARDSON, 2003).

A unidade básica de um codificador H.264 é o bloco. Um bloco é formado por um grupo de 4 linhas com 4 amostras (4×4) na sua forma mais básica. Um macrobloco é formado por 4 blocos Y, 1 bloco Cb e 1 bloco Cr. (JACK, 2001).

A Figura 6 mostra um esquemático simplificado de um codificador H.264. Um quadro original é processado em unidades de blocos ou macroblocos. Cada bloco ou macrobloco é codificado no modo intra-quadro ou inter-quadro de predição. A predição inter-quadro é calculada com base no quadro atual e quadros de referência, enquanto que a predição intra-quadro é calculada com base em amostras reconstruídas da imagem.

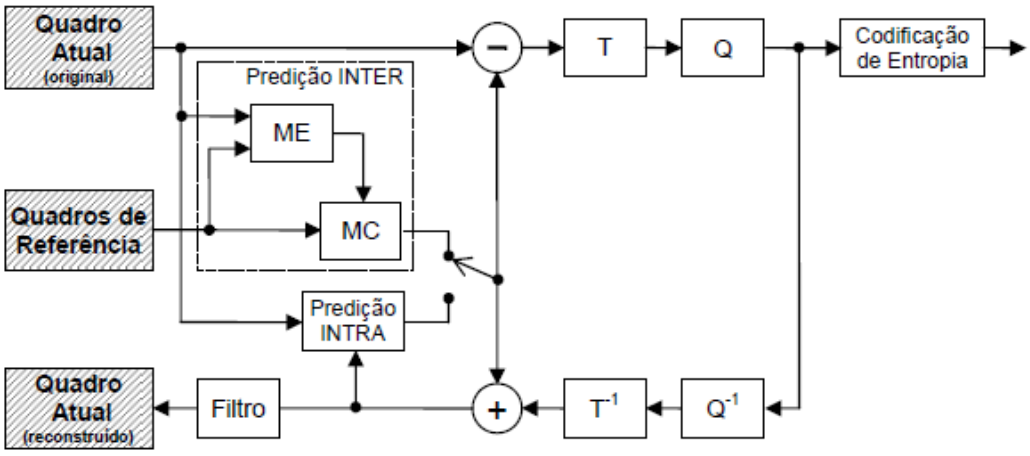


Figura 6 Codificador. (SILVA, 2006).

O resíduo das operações de predição é transformado e quantizado. O resultado é um conjunto de coeficientes que são reordenados e passam pelo codificador de entropia que gera os coeficientes codificados necessários para a decodificação de cada bloco.

O codificador além de codificar e transmitir cada bloco deve reconstruí-los para prover uma referência para as próximas predições. Isso é feito para que se tenha como referência para as predições os mesmos blocos que o decodificador terá ao decodificar a imagem. Para a reconstrução, os coeficientes quantizados passam por uma etapa de quantização inversa e transformada inversa. Logo após, um filtro é aplicado para reduzir os efeitos de distorção.

4 TRANSFORMADAS E QUANTIZAÇÃO

Os módulos de transformadas, quantização e suas operações inversas são parte importante de um codificador de vídeo. Apesar de serem menos complexos que outros módulos de um CODEC H.264, o estudo das transformadas e quantização é interessante por estarem no caminho crítico da predição intra-quadro. Ou seja, quanto maior for o desempenho deles, menor será a latência da predição intra-quadros (menos tempo se precisará esperar para trabalhar). Também pode ser visto que a reconstrução da imagem depende destes módulos, ou seja, estão no caminho crítico do codificador como um todo (AGOSTINI, 2007).

4.1 TRANSFORMADAS DIRETAS

O padrão H.264, define dois tipos distintos de transformada: Transformada discreta de cossenos (DCT) e Transformada de Hadamard.

4.1.1 DCT

A DCT é derivada da Transformada Discreta de Fourier (GHANBARI, 2003). Seu objetivo é converter um bloco de dados (imagem original ou resíduos calculados) para outro domínio. A Transformada do Cosseno é baseada em blocos de pixels, ou seja, opera sobre blocos de imagem de dimensão n×n e não sobre o quadro completo. A equação de transformação pode ser vista abaixo:

$$Y = AXA^T \qquad (8)$$

Onde

Y é a matriz de saída

X é a matriz de entrada, um bloco n×n de amostras de dados

A é a matriz de coeficientes da transformada (também n×n).

A^T é a matriz A transposta.

Os elementos da matriz A são:

$$A_{ij} = C_i \cos\frac{(2j+1)i\pi}{2n} \tag{9}$$

$$C_0 = \sqrt{\frac{1}{n}} \tag{10}$$

$$C_{i\neq 0} = \sqrt{\frac{2}{n}} \tag{11}$$

Onde *i* e *j* representam a posição do elemento da matriz, sendo número da linha e da coluna respectivamente. *n* é o número de elementos de uma linha ou coluna, quatro para a matriz 4×4.

Realizando a transformada, o resultado obtido é uma matriz com a mesma dimensão da entrada, n×n. O codificador H.264, caso trabalhado nesta monografia, normalmente opera sobre blocos de 4×4 amostras. Neste caso a matriz A fica assim:

$$A = \begin{bmatrix} \frac{1}{2}\cos(0) & \frac{1}{2}\cos(0) & \frac{1}{2}\cos(0) & \frac{1}{2}\cos(0) \\ \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{5\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{7\pi}{8}\right) \\ \sqrt{\frac{1}{2}}\cos\left(\frac{2\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{6\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{10\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{14\pi}{8}\right) \\ \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{9\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{15\pi}{8}\right) & \sqrt{\frac{1}{2}}\cos\left(\frac{21\pi}{8}\right) \end{bmatrix} \tag{12}$$

Uma matriz equivalente a essa e de mais fácil visualização e representação pode ser assim obtida:

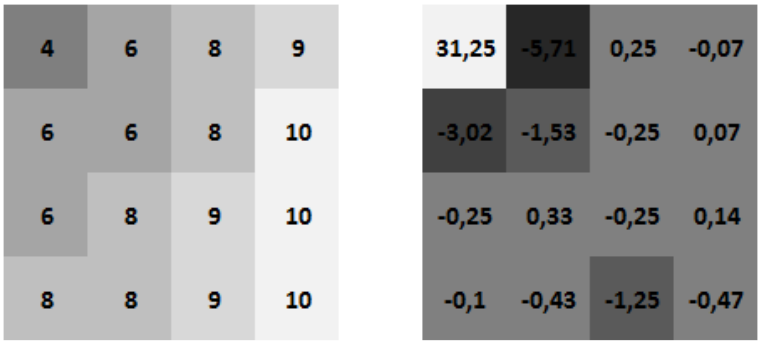
$$A = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \tag{13}$$

$$a = \frac{1}{2} \tag{14}$$

$$b = \sqrt{\frac{1}{2}}\cos\left(\frac{\pi}{8}\right) \tag{15}$$

$$c = \sqrt{\frac{1}{2}}\cos\left(\frac{3\pi}{8}\right) \tag{16}$$

Em um bloco 4×4 ocorre uma transformação de 16 valores de pixels em 16 coeficientes de DCT, ou seja, não há redução do número de dados a ser armazenado. Pode-se observar pela Figura 7 que a DCT tende a concentrar os valores mais significativos no canto superior esquerdo de sua representação de saída.



(a)Bloco Original (b)Bloco com coeficientes da DCT

Figura 7 Bloco de amostras antes e depois de passar pela DCT(RICHARDSON, 2003).

O padrão H.264 utiliza a transformada DCT sobre blocos 4×4 de amostras do vídeo original nos quadros I, e de informação residual resultante da compensação de movimento ou predição intra-quadro quando frames P ou B são processados (HUSEMANN, 2010b).

A transformada utilizada no padrão H.264 difere da DCT formal apresentada anteriormente em um aspecto: é uma transformada de coeficientes inteiros, criada para facilitar a sua implementação em hardware de ponto fixo e dispositivos de baixo custo, e evitar distorções entre o codificador e o decodificador (MALVAR, 2003).

Para chegar aos valores inteiros dessa DCT deve-se considerar (8), (13), (14), (15) e (16). Dessas tem-se:

$$Y = AXA^T = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} X \begin{bmatrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & -c \end{bmatrix} \quad (17)$$

Fatorando essa equação, é possível vê-la de uma nova forma.

$$Y = AXA^T = (CXC^T) \otimes E \quad (18)$$

$$Y = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} X \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix} \quad (19)$$

Onde ⊗ representa uma multiplicação ponto a ponto, ou seja, que cada elemento da matriz CXC^T é multiplicada pelo escalar de mesma posição na matriz E. Para simplificar a implementação da transformada, uma aproximação é feita para b:

$$a = \frac{1}{2} \tag{14}$$

$$b = \sqrt{\frac{2}{5}} \tag{20}$$

A multiplicação ponto a ponto é removida da DCT e compensada no módulo de quantização, chegando à equação final da DCT do padrão H.264:

$$\bar{Y} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \mathrm{X} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \tag{21}$$

Estas matrizes envolvem operações simples para hardware: somas, subtrações e multiplicação por 2, que nada mais é do que um deslocamento para a esquerda do valor (HUSEMANN, 2010a).

4.1.2 HADAMARD

O padrão H.264 propõe adicionalmente a transformada de Hadamard, que processa os valores DC após o cálculo da DCT. Esta segunda transformada foi proposta pelo padrão H.264/AVC para aumentar a compressão em áreas homogêneas (RICHARDSON, 2003).

Como a Transformada de Hadamard é aplicada apenas aos coeficientes DC (1 coeficiente por bloco) de um macrobloco que passou pela DCT, ao processar um macrobloco de luminância (composto por 16 blocos de 4×4 amostras), a dimensão da Hadamard é 4×4. Ao processar um macrobloco de crominância (composto por 4 blocos de 4×4 amostras), uma Hadamard 2×2 é necessária. A Figura 8 ilustra um macrobloco de luminância e um de crominância com respectivos valores DC destacados.

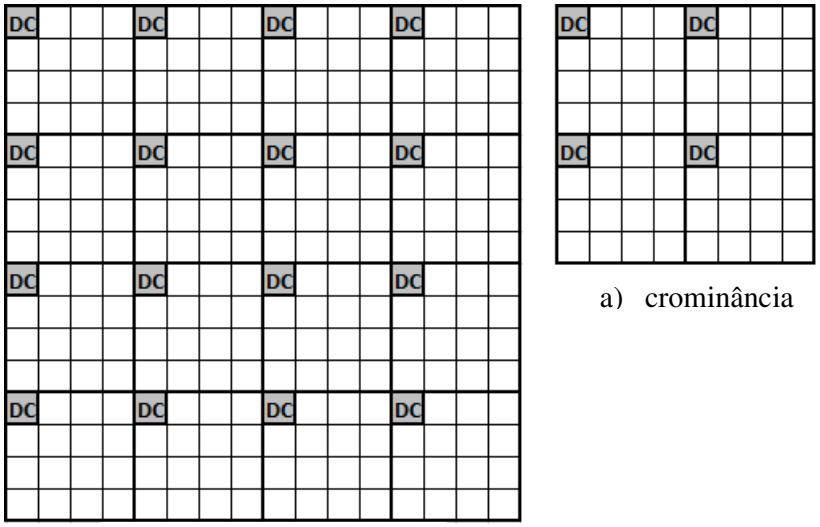


Figura 8 Macroblocos de luminância e crominância com elementos DC destacados

O cálculo executado pela Hadamard é o seguinte, para coeficientes de luminância:

$$Y_L = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} X_L \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \qquad (22)$$

Onde:

X_L é a matriz 4×4 dos valores DC resultantes do cálculo da DCT de luminância de 16×16 amostras;

Y_L é a matriz de valores resultantes da transformada Hadamard 4×4.

Para coeficientes de crominância, o cálculo do Hadamard é apresentado a seguir:

$$Y_C = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} X_C \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad (23)$$

Onde:

X_C é a matriz 2×2 dos valores DC resultantes do cálculo da DCT de crominância em função da relação 4:2:0;

Y_C é a matriz de valores resultantes da transformada Hadamard 2×2.

4.2 TRANSFORMADAS INVERSAS

A codificação de vídeo utiliza do recurso de transformadas para reduzir a quantidade de informação transmitida ou armazenada. Para a reprodução do vídeo, o decodificador precisa inverter essas transformações, colocando a informação obtida de volta ao domínio original. Os módulos que desempenham essa função são as transformadas DCT Inversa e Hadamard Inversa, e não aparecem somente nos decodificadores. No codificador há um laço de realimentação onde esses módulos desempenham a função de recuperar o quadro original.

4.2.1 DCT INVERSA

A DCT Inversa realiza a transformação dos coeficientes de DCT de volta a dados de pixels e que é regida por (24).

$$X = A^T Y A \tag{24}$$

Onde:

Y é a matriz transformada

X é a matriz recuperada de amostras

A é a matriz de coeficientes da transformada (também n×n).

A^T é a matriz A transposta.

Explorando essa equação, pode-se chegar a:

$$X = {C_i}^T (Y \otimes E) C_i \tag{25}$$

A mesma simplificação utilizada pela DCT direta no codificador H.264, eliminação da multiplicação escalar, será feita pela DCT inversa. Esse cálculo será compensado pela quantização inversa. Então o cálculo que realiza a DCT inversa é o seguinte:

$$X = \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} Y \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{bmatrix} \quad (26)$$

4.2.2 HADAMARD INVERSA

A equação que rege a transformada inversa de Hadamard pode ser obtida da seguinte maneira:

$$Y_L = \frac{1}{2}(HX_L H^T) \quad (27)$$

$$X_L = \frac{2}{2}(H^{-1}HX_L H^T H^{-T}) \quad (28)$$

$$X_L = 2(H^{-1}Y_L H^{-T}) \quad (29)$$

$$H^{-1} = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \quad (30)$$

$$X_L = \frac{1}{8}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} Y_L \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \quad (31)$$

A Hadamard inversa não faz a divisão por oito da equação acima, sendo essa divisão compensada na quantização inversa. Assim, a Hadamard Inversa executa o seguinte cálculo:

$$\overline{\mathrm{X}}_L = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \mathrm{Y_L} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \quad (32)$$

Para o processamento de blocos de crominância, a Hadamard Inversa é a seguinte:

$$\mathrm{X}_C = \begin{bmatrix} -1 & -1 \\ -1 & -1 \end{bmatrix} \mathrm{Y_C} \begin{bmatrix} -1 & -1 \\ -1 & -1 \end{bmatrix} \quad (33)$$

4.3 QUANTIZAÇÃO

4.3.1 QUANTIZAÇÃO DIRETA

A transformação de domínios dos pixels feita pela DCT e Hadamard tem por objetivo a concentração de parte significativa da energia da imagem em componentes de baixas frequências (GHANBARI, 2003). A quantização escalar dos coeficientes das transformadas leva a uma redução significativa da informação agregada. Seu cálculo é feito pela divisão dos valores dos coeficientes resultantes da transformada por um valor inteiro, diminuindo a gama de valores e levando muitos destes a zero. Essa forma de quantização, está representada pela equação a seguir (RICHARDSON, 2003):

$$FQ = round\left(\frac{X}{Qstep}\right) \quad (34)$$

Onde

FQ (*Forward Quantization*) é o resultado da Quantização;

X é o valor a ser quantizado;

Qstep é o tamanho do passo.

round é aproxima a expressão ao inteiro mais próximo.

Quanto maior o tamanho do passo (*Qstep*) maior a compressão e maior a perda de informação. A perda de informação é causada pela própria definição de *Qstep* e pode ser verificada ao se reverter essa operação, multiplicando o resultado da quantização pelo mesmo inteiro que o gerou. A Tabela 3 mostra a quantização dos valores originais com dois *Qsteps* distintos que comprovam as afirmações feitas quanto a perda de informação e compressão.

Tabela 3 Análise de compressão e perdas causadas pelo Qstep

Qstep = 3			Qstep = 5		
Valores Originais	Valores Quantizados	Valores Recuperados	Valores Originais	Valores Quantizados	Valores Recuperados
0	0	0	0	0	0
1	0	0	1	0	0
2	1	3	2	0	0
3	1	3	3	1	5
4	1	3	4	1	5
5	2	6	5	1	5
6	2	6	6	1	5
7	2	6	7	1	5
8	3	9	8	2	10
9	3	9	9	2	10
10	3	9	10	2	10

A operação que reverte o cálculo da Quantização é chamada de Quantização Inversa:

$$IQ = Qstep \cdot FQ \tag{35}$$

Se não houvesse a operação de arredondamento (*round*) na quantização, os valores recuperados seriam os próprios valores originais, mas também não haveria compressão. Havendo o arredondamento, o *Qstep* deve ser cuidadosamente escolhido para se conseguir uma boa compressão sem comprometer muito a imagem recuperada.

Explorando características do olho humano, percebeu-se que a sensibilidade a distorções em altas frequências é baixa. Portanto, utilizam-se diferentes *Qsteps* para altas e baixas frequências, sendo o *Qstep* para altas frequências maior (GHANBARI, 2003).

O padrão H.264 utiliza a Quantização escalar apresentada na equação (34), mas seu mecanismo é adaptado por levar em consideração as limitações de *hardwares*, ou seja, evita divisões e cálculos com ponto flutuante. A quantização do H.264 suporta um total de 52 valores de *Qstep*, indexados pelo QP (*Quantization Parameter*). Qstep dobra de tamanho a cada incremento de 6 do QP, como pode ser visto na Tabela 4 (RICHARDSON, 2003):

Tabela 4 Relações de QP e Qstep disponíveis no H.264.

QP	0	1	2	3	4	5	6	7	8	9	10	11	12	...
Qstep	0,625	0,6875	0,8125	0,875	1	1,125	1,25	1,375	1,625	1,75	2	2,25	2,5	...
QP	...	18	...	24	...	30	...	36	...	42	...	48	...	51
Qstep		5		10		20		40		80		160		224

Fonte: RICHARDSON, 2003

Além da divisão do coeficiente transformado por Qstep, a multiplicação escalar que a DCT deixou de realizar é calculada na quantização. Então o cálculo da Quantização pode ser melhor representado por:

$$FQ = round\left(X \cdot \frac{PF}{Qstep}\right) \tag{36}$$

Onde *FQ* significa *forward quantization* e o parâmetro *PF* (*Post-scaling Factor*) assume os valores de $a^2, b^2/4, ab/2$ de acordo com a Tabela 5:

Tabela 5 Valores de PF de acordo com a posição.

Posição	PF
(0,0), (2,0), (0,2) e (2,2)	a^2
(1,1), (1,3), (3,1) e (3,3)	$b^2/4$
Outras	$ab/2$

Fonte: RICHARDSON, 2003

Visando uma simplificação na complexidade do cálculo, o termo $\frac{PF}{Qstep}$ pode ser substituído por um termo equivalente.

$$\frac{PF}{Qstep} = \frac{MF}{2^{qbits}} \tag{37}$$

Onde

MF é o fator de multiplicação (*Multiplication Factor*);

qbits é definido por

$$qbits = 15 + floor(\frac{QP}{6}) \tag{38}$$

floor é uma função que realiza o arredondamento para o inteiro abaixo do resultado.

Assim, a equação (28) é calculada como segue:

$$|FQ| = (|X| \cdot MF + f) >> qbits \quad (39)$$

$$sign(FQ) = sign(X) \quad (40)$$

Onde

f é um valor de ajuste para o valor válido mais próximo;

>> indica deslocamento binário para a direita;

sign é uma função que retorna o sinal do número.

O valor de f para codificações em laços de intra-quadro e inter-quadro é dado por:

$$f_{intra} = \frac{2^{qbits}}{3} \quad (41)$$

$$f_{inter} = \frac{2^{qbits}}{6} \quad (42)$$

Para a equação (36) poder ser calculada a partir das equações (39) e (40), MF é definido da forma mostrada na Tabela 6:

Tabela 6 Fatores de Multiplicação (MF).

QP	Posições (0,0), (2,0), (0,2) e (2,2)	Posições (1,1), (1,3), (3,1) e (3,3)	Outras Posições
0	13107	5243	8066
1	11916	4660	7490
2	10082	4194	6554
3	9362	3647	5825
4	8192	3355	5243
5	7282	2893	4559
...	...	...	...

Fonte: RICHARDSON, 2003

A quantização, sendo feita dessa forma, apresenta uma multiplicação inevitável, porém com aritmética de inteiros: Valor absoluto de entrada pelo Fator de Multiplicação ($|X|\ MF$). Todas as outras operações são deslocamentos, somas, ou podem ser efetuadas por buscas em tabelas de constantes.

4.3.2 QUANTIZAÇÃO INVERSA

A quantização inversa realiza o cálculo de recuperação do valor anterior a quantização. Como a quantização apresenta perdas, o valor recuperado será uma aproximação do valor original. O cálculo executado por esse módulo é o seguinte:

$$|X| = (|FQ|\ IF) << qbits - 19 \quad (43)$$

$$sign(X) = sign(FQ) \quad (44)$$

O primeiro passo a ser realizado é a multiplicação do valor quantizado por *IF* (*Inverse Factor*). Essa matriz de constantes, assim como a quantização compensa a multiplicação escalar da DCT, compensa a multiplicação escalar da DCT inversa.

A matriz de multiplicação pronta pode ser vista na Tabela 7.

Tabela 7 Fatores de Multiplicação Inversa(IF).

QP	Posições (0,0), (2,0), (0,2) e (2,2)	Posições (1,1), (1,3), (3,1) e (3,3)	Outras Posições
0	160	256	208
1	176	288	224
2	208	320	256
3	224	368	288
4	256	400	320
5	288	464	368
...	...	...	...

Fonte: RICHARDSON, 2003

O deslocamento de *qbits* na quantização é um deslocamento para eliminar os bits menos significativos, ou seja, para a direita. Para inverter essa operação e voltar à escala anterior, deve-se realizar um deslocamento para a esquerda colocando uma série de zeros à direita do valor de quantização. A subtração presente no deslocamento, $(qbits - 19)$, pode ser explicada a partir da seguinte dedução:

$$|FQ| = (|X|\ MF + f) \gg qbits \quad (45)$$

$$|X| = \frac{(|FQ|)}{MF} \ll qbits \quad (46)$$

$$|X| = \frac{(|FQ|)IF}{MF \cdot IF} \ll qbits \quad (47)$$

$$|X| = (|FQ|\, IF) \ll qbits - 19 \quad (48)$$

A divisão presente na equação 47 é substituída pela subtração apresentada no deslocamento da equação 48.

Assim como na quantização, os cálculos são realizados sem sinal, mas este é recuperado ao final do cálculo.

5 ARQUITETURAS PROPOSTAS

Um codificador de vídeo pode operar em *software*, *hardware* ou em ambos, numa arquitetura mista. O módulo computacional tem uma tendência a ser trabalhado em *hardware* por apresentar um melhor desempenho dessa forma (AGOSTINI, 2006).

Arquiteturas em *hardware* normalmente permitem trabalhar com rotinas paralelas. Esse paralelismo permite que após um período de latência, a cada ciclo de relógio se tenha um novo dado de saída. A cada ciclo de *clock* é executada uma etapa de um cálculo e, passado esse *clock*, outro valor pode passar pela mesma operação. Assim vários cálculos são executados ao mesmo tempo, numa estrutura denominada *pipeline*.

Além disso, um mesmo dado de saída em *hardware* pode corresponder a várias saídas do software. Na Figura 9 pode-se ver uma comparação entre cálculos em *software* utilizando um processador apenas, e cálculos em *hardware*. Pode-se ver que o paralelismo do *pipeline* acelera o processo. As quatro entradas e variáveis intermediárias são apenas um exemplo, o limite do paralelismo para o hardware é o número de portas lógicas disponíveis. Outro limitante seria a aplicação.

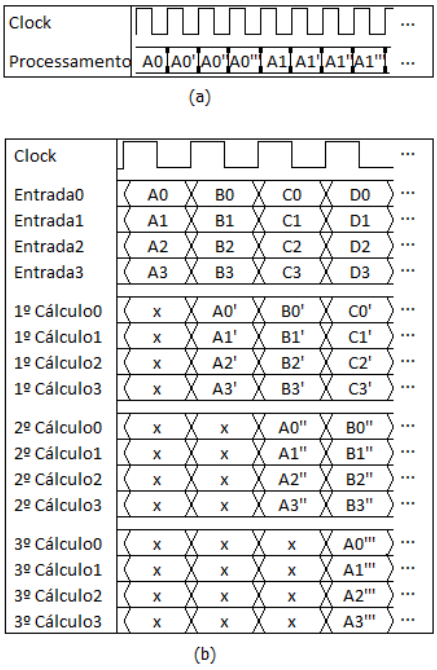


Figura 9 (a) Cálculos executados por um processador de uso geral. (b) Cálculos executados em hardware.

As arquiteturas desenvolvidas para esta monografia foram todas elaboradas no PRAV (Projetos em Áudio e Vídeo) do Instituto de Informática da UFRGS, dentro de um projeto que visa o desenvolvimento de um codificador híbrido (combinando *software* e *hardware*) que almeja realizar codificação escalável em tempo real. O desenvolvimento foi trabalhado em VHDL (*VHSIC Hardware Description Language*) e direcionado para a família Virtex-II Pro da Xilinx. A ferramenta de síntese utilizada foi o ISE e a ferramenta de análise foi o Chipscope, ambos da Xilinx.

Para fazer parte do codificador desse projeto, essas arquiteturas buscaram não apenas atender as exigências do padrão H.264/AVC, mas atingir um elevado desempenho, suficiente para processar vídeos de alta definição (HDTV) em tempo real até para a especificação escalável do codificador, o H.264 SVC (*Scalable Video Coding*). A especificação SVC agrega escalabilidade ao H.264, tornando a complexidade do codificador ainda maior, e os tempos ainda mais críticos. Para tanto, alguns ajustes nos algoritmos já existentes foram realizados, permitindo uma maior exploração do paralelismo (HUSEMANN, 2010a).

5.1 ESTUDO DE ARQUITETURAS ENCONTRADAS NA LITERATURA

Considerando a importância deste módulo para a codificação, muitas soluções para implementação da DCT em duas dimensões em *hardware* foram propostas. Prasoon (2009) apresenta em seu artigo uma arquitetura muito interessante de cálculo de DCT para o padrão H.264 com foco na redução de área ocupada na FPGA.

A arquitetura sugerida por Prasoon é serial, ou seja, uma amostra de cada vez é processada em uma arquitetura de *pipeline*. O *pipeline* utilizado possui cinco estágios para calcular as 16 amostras de um bloco 4×4. Representa uma solução bastante compacta, mas limita a performance do módulo.

Outras soluções relevantes são baseadas na propriedade da separabilidade. Nesta abordagem, a equação (49) é separada em duas equações que realizam o mesmo cálculo, (50) e (51), sendo necessário realizar uma transposição após o cálculo de Y_1.

$$Y = CXC^T \quad (49)$$

$$Y_1 = CX \quad (50)$$

$$Y = CY_1^T \quad (51)$$

A transposição atua em uma matriz de modo a transformar linhas em colunas. Assim, é realizado o mesmo cálculo em duas dimensões sendo o cálculo de uma dimensão chamado de DCT-1D e o cálculo completo DCT-2D. A Figura 10 ilustra o cálculo da DCT em três passos utilizando-se dois módulos de DCT-1D e uma transposição.

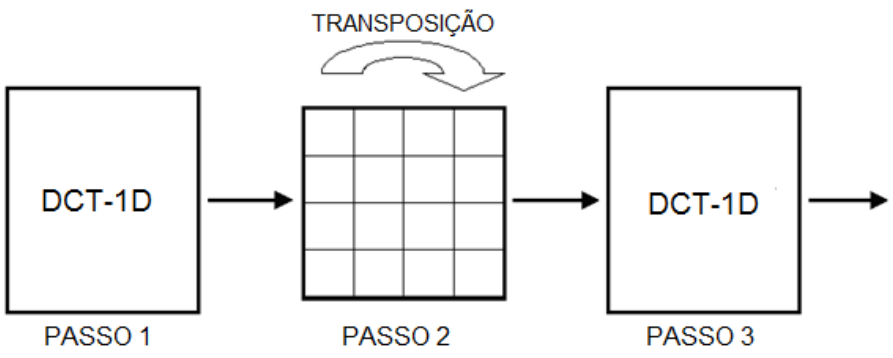


Figura 10 Divisão da DCT em dois módulos computacionais e uma transposição (HUSEMANN, 2010b).

Wang (2003) utiliza esta mesma abordagem (Figura 10) para implementar uma solução paralela. Nesta nova implementação o módulo DCT é baseado em duas transformadas unidimensionais (DCT-1D) conectadas por uma matriz de registradores de transposição. Cada DCT-1D realiza o cálculo sugerido pela Figura 11:

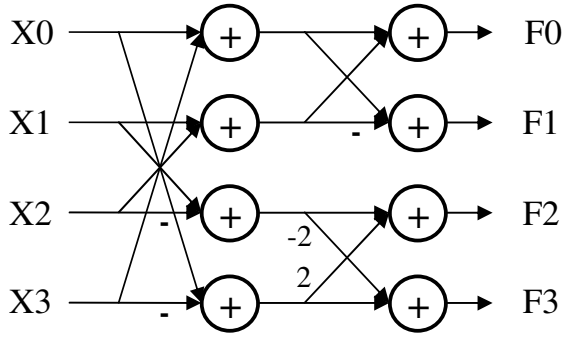


Figura 11 Cálculo simplificado da DCT-1D em duas etapas (WANG, 2003).

Isso significa que a saída F é calculada da seguinte forma:

$$F0 = X0 + X3 + X1 + X2 \quad (52)$$

$$F1 = 2(X0 - X3) + X1 - X2 \quad (53)$$

$$F2 = X0 + X3 - X1 - X2 \quad (54)$$

$$F3 = X0 - X3 - 2(X1 - X2) \quad (55)$$

Dividindo o cálculo matricial em cálculos de duas etapas, muitos cálculos podem ser aproveitados por serem repetidos. Desta forma, Wang (2003) propõe a utilização de quatro variáveis intermediárias para obter o mesmo resultado. Assim numa primeira etapa calcula-se:

$$I0 = X0 + X3 \quad (56)$$

$$I1 = X1 + X2 \quad (57)$$

$$I2 = X0 - X3 \quad (58)$$

$$I3 = X1 - X2 \quad (59)$$

Onde I0, I1, I2, I3 são as quatro variáveis intermediárias.

Em uma segunda etapa o cálculo da DCT-1D é finalizado:

$$F0 = I0 + I1 \tag{60}$$

$$F1 = 2 \cdot I2 + I3 \tag{61}$$

$$F2 = I0 - I1 \tag{62}$$

$$F3 = I2 - 2 \cdot I3 \tag{63}$$

A solução proposta por Wang (2003) pode processar quatro amostras por ciclo de relógio.

Cheng (2004) também apresenta uma arquitetura paralela como solução do cálculo da DCT-2D. Explorando as propriedades de simetria presentes na equação (49), Cheng propõe uma arquitetura que elimina o passo de transposição.

$$Y = CXC^T \tag{49}$$

Após a expansão da equação acima, encontra uma nova forma de apresentação para ela. Essa forma é:

$$\begin{bmatrix} y_0 \\ y_2 \end{bmatrix} = \begin{bmatrix} C_f & C_f \\ C_f & -C_f \end{bmatrix} \begin{bmatrix} x_0 + x_3 \\ x_1 + x_2 \end{bmatrix} \tag{64}$$

$$\begin{bmatrix} y_1 \\ y_3 \end{bmatrix} = \begin{bmatrix} 2C_f & C_f \\ C_f & -2C_f \end{bmatrix} \begin{bmatrix} x_0 - x_3 \\ x_1 - x_2 \end{bmatrix} \tag{65}$$

Onde

$$C_f = \begin{bmatrix} 1 & 1 & 1 & 0{,}5 \\ 1 & 0{,}5 & -1 & -1 \\ 1 & -0{,}5 & -1 & 1 \\ 1 & -1 & 1 & -0{,}5 \end{bmatrix} \tag{66}$$

Essa inovação permite o processamento de oito amostras por ciclo de relógio economizando portas lógicas. No entanto, esta solução, em comparação com soluções com transposição, apresenta atrasos maiores no caminho crítico pelo uso de estruturas computacionais sequenciais mais longas, o que acaba limitando a performance do conjunto (HUSEMANN, 2010a).

Agostini (2006) apresenta outra solução que explora o paralelismo em *hardware*. Utilizando uma estrutura complexa de somas e deslocamentos propõe realizar o processo de

transformada em duas dimensões de um bloco 4×4 completo em apenas um estágio. Através de uma estrutura em *pipeline*, 16 amostras podem ser processadas para cada ciclo de *clock*. Apesar da grande eficiência computacional em termos de número de amostras processadas por *clock* desta solução, altas frequências não podem ser alcançadas (AGOSTINI, 2006).

Como pode-se perceber a escolha da melhor solução dependerá de recursos de *hardware* e comunicação disponíveis para a aplicação alvo. A aplicação da arquitetura proposta por esta monografia é fazer parte de um projeto de implementação de um codificador híbrido rodando parte dos módulos em computador (*software*) e parte em FPGA (*hardware*) (HUSEMANN, 2010c). Sendo que apenas os módulos apresentados nessa monografia estarão em *hardware*, todo o ganho de velocidade adquirido é válido, não sendo limitado pelo desempenho de outros módulos. Assim, a frequência de operação é um dado interessante e deve ser levada em conta.

Um gargalo bastante conhecido para implementações em *hardware* é a largura de banda de memórias e o mecanismo de comunicação entre a unidade computacional e a memória (HUSEMANN, 2010b).

No estudo de caso montado para validação desta proposta, foi utilizada uma interface de comunicação PCI. A interface PCI utiliza um barramento de 32 ou 64 bits para a comunicação com velocidade de comunicação de 33MHz ou 66MHz.

Tendo em vista o gargalo de comunicação e a possibilidade de operação em altas frequências, a arquitetura proposta por este trabalho parte da abordagem de Wang (2003). Considerando amostras de 8 bits e comunicação de 32 bits, as quatro amostras por ciclo de relógio que Wang (2003) consegue atingir possibilita o desempenho de um acesso à memória por *clock* para adquirir os dados de entrada (32 bits). O uso da transposição em sua abordagem não limita a frequência de operação.

5.2 DCT PROPOSTA

A proposta de melhoria na DCT é fundir a transposição com o primeiro módulo de DCT-1D. Com a finalidade de reduzir a complexidade, aumentar a frequência de operação e diminuir a latência, essa inovação permite o aproveitamento de sinais e o reuso da mesma máquina de estados (HUSEMANN, 2010a). Essa alteração implica em módulos diferentes para o cálculo de cada DCT unidimensional. Dessa forma, os módulos foram renomeados para DCT-H (horizontal) e DCT-V (vertical). A figura abaixo mostra esses módulos.

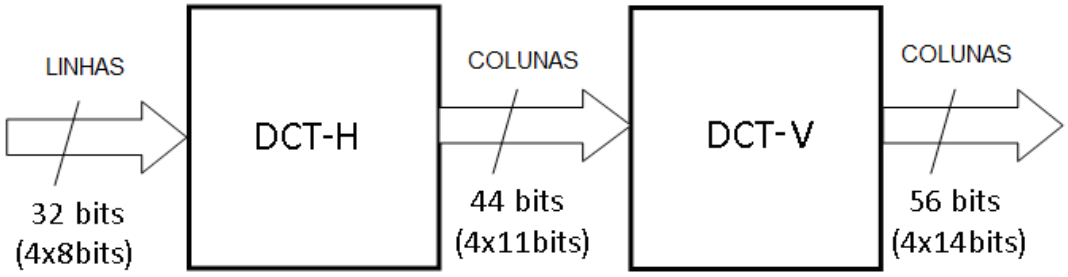


Figura 12 DCT-2D em dois estágios (HUSEMANN, 2010a).

O cálculo realizado internamente pela DCT-2D é o mesmo que Wang (2003) realiza, com duas etapas. No primeiro ciclo de *clock* (Cálculo Intermediário na Figura 13) são calculadas as equações (56), (57), (58) e (59). Apresenta somas e subtrações o que implica em um resultado com até 9 bits por amostra. No próximo ciclo de relógio, o cálculo apresenta multiplicações por dois além das somas, gerando um resultado com 2 bits a mais por amostra (equações (60), (61), (62) e (63)). A transposição, também incorporada na DCT-H, é realizada em 4 ciclos de relógio, pois precisa da matriz de 4×4 amostras inteira para começar a gerar a saída. Nessas condições, a saída da DCT-H é em coluna enquanto que a entrada era em linha, e a largura da saída é 11 bits por amostra devido às somas e multiplicações.

A DCT-V apresenta os mesmos cálculos da DCT-H, mas não apresenta transposição. Assim sendo suas saídas continuarão em colunas. Cada amostra de saída possui 14 bits devido aos cálculos internos, que demoram 2 ciclos de relógio para ser realizado.

A Figura 13 apresenta a análise da latência do cálculo da DCT-2D proposta. A latência total é de 8 ciclos de relógio e passado esse período tem-se 4 amostras a cada *clock* disponível na saída.

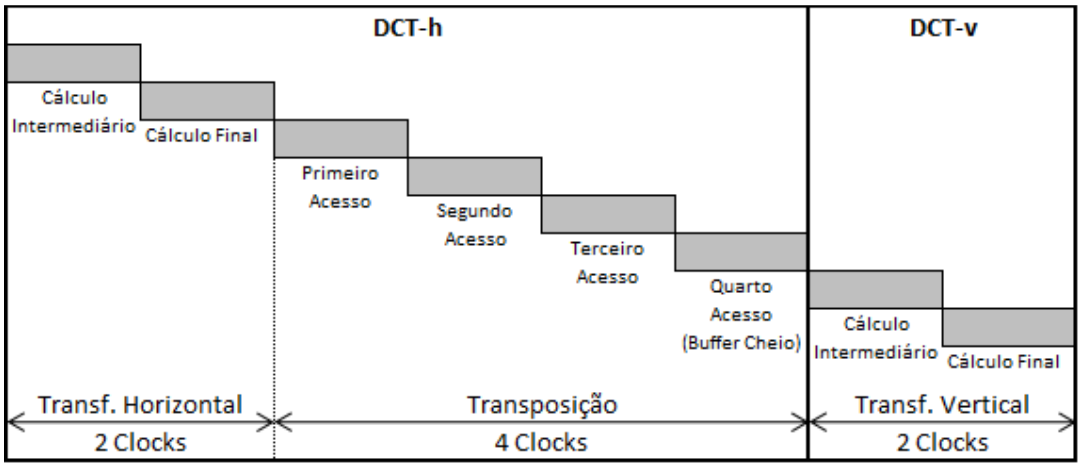


Figura 13 Diagrama dos estágios internos da DCT-2D proposta (HUSEMANN, 2010b).

A figura abaixo apresenta a DCT com seus sinais de entrada e saída.

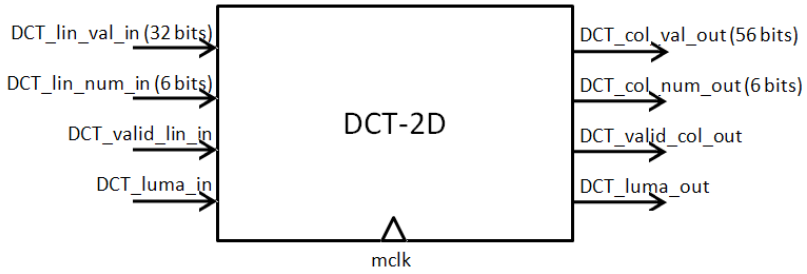


Figura 14 Módulo da DCT-2D.

Esse módulo foi sintetizado para o FPGA Xilinx V2P30FF896, apresentando o seguinte resultado:

Tabela 8 Resultado da implementação da DCT em FPGA.

Módulo	# LUTs	Período (ns)
DCT-2D	364	2,894

FPGA Xilinx V2P30FF896

5.3 HADAMARD PROPOSTA

Como já visto anteriormente, DCT e Hadamard possuem uma arquitetura muito parecida. Como foi verificado para a DCT uma abordagem que utiliza uma etapa de cálculos e transposição e outra etapa de cálculos, também foi implementada para a Hadamard.

As equações abaixo apresentam o cálculo executado por cada Hadamard de uma dimensão (HAD-1D) ao se processar um bloco de luminância.

$$Z_1 = HY \tag{67}$$

$$Y = \frac{1}{2}(HZ_1^T) \tag{68}$$

De forma resumida na arquitetura de Hadamard implementada há uma HAD-H (Hadamard horizontal), que envolve as funções de HAD-1D (equação (67)) e a transposição; e uma HAD-V (Hadamard vertical), que calcula a outra HAD-1D (equação (68)).

O cálculo de cada HAD-1D é dividido em duas etapas para simplificação e aproveitamento de cálculos intermediários (HUSEMANN, 2010b).

Abaixo pode ser visto o cálculo executado para blocos de luminância.

- Primeira etapa:

$$I0 = X0 + X3 \tag{69}$$

$$I1 = X1 + X2 \tag{70}$$

$$I2 = X0 - X3 \tag{71}$$

$$I3 = X1 - X2 \tag{72}$$

- Segunda etapa:

$$F0 = I0 + I1 \tag{73}$$

$$F1 = I2 + I3 \tag{74}$$

$$F2 = I0 - I1 \tag{75}$$

$$F3 = I2 - I3 \tag{76}$$

No final do cálculo das duas dimensões, ainda deve-se dividir os resultados por dois, ou seja, elimina-se o bit menos significativo do resultado obtido para cada amostra.

O módulo Hadamard projetado deve processar tanto blocos de luminância como de crominância. Em um macrobloco, os valores DC de crominância são apenas quatro, portanto a Hadamard de Crominância é calculada sobre uma matriz 2×2. Como o módulo proposto tem a capacidade de processar quatro amostras em paralelo (1 linha de amostras de luminância), o cálculo unidimensional de crominância processará as quatro amostras em um ciclo de relógio (HUSEMANN, 2010b).

O mesmo módulo será utilizado para executar os dois tipos de Hadamard reduzindo a lógica utilizada. Na primeira etapa, é executado todo o cálculo necessário para uma dimensão, e ao final da segunda etapa já se possui o cálculo completo da HAD-2D de crominância.

- Primeira etapa:

$$I0 = X0 + X1 \quad (77)$$
$$I1 = X2 + X3 \quad (78)$$
$$I2 = X0 - X1 \quad (79)$$
$$I3 = X2 - X3 \quad (80)$$

- Segunda etapa:

$$F0 = I0 + I1 \quad (81)$$
$$F1 = I2 + I3 \quad (82)$$
$$F2 = I0 - I1 \quad (83)$$
$$F3 = I2 - I3 \quad (84)$$

Pode-se notar que as equações (77), (78), (79) e (80) envolvem cálculos parecidos com os das equações (69), (70), (71) e (72). As equações (81), (82), (83) e (84) são iguais às equações (73), (74), (75), e (76). Portanto esses cálculos são aproveitados pelo *hardware* resultando em economia de lógica desejada. A Figura 15 mostra como o *hardware* executa essas operações na HAD-H. São necessários apenas quatro somadores, quatro subtratores, três multiplexadores e o buffer de transposição para o processamento da HAD-H. A transposição

não está representada na figura e os multiplexadores fazem a troca das entradas que são diferentes para luminância e crominância (luma e croma).

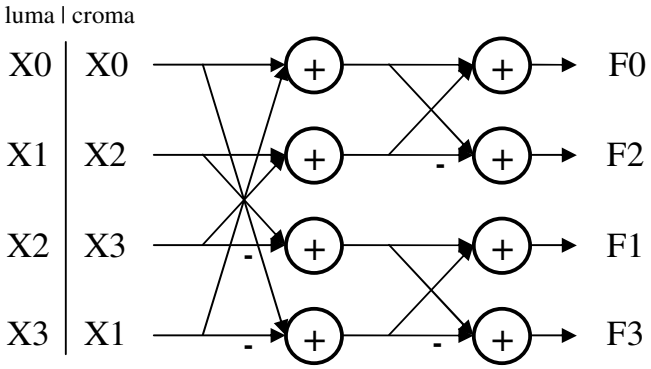


Figura 15 Aproveitamento da lógica para processamento de bloco de luminância e crominância na HAD-H.

No módulo HAD-V existem dois caminhos para as amostras, um para luminância e outro para crominância. Os blocos de luminância passama pelos mesmos cálculos ilustrados pela Figura 15 enquanto que os de crominância passam por um pipeline para o módulo não perder sincronização. A figura abaixo mostra o módulo HAD-H completo com parte computacional e transposição ligado ao módulo HAD-V. Neste módulo o pipeline deve ter dois estágios para que o tempo de propagação dos dados de crominância seja o mesmo apresentado pelos dados de luminância.

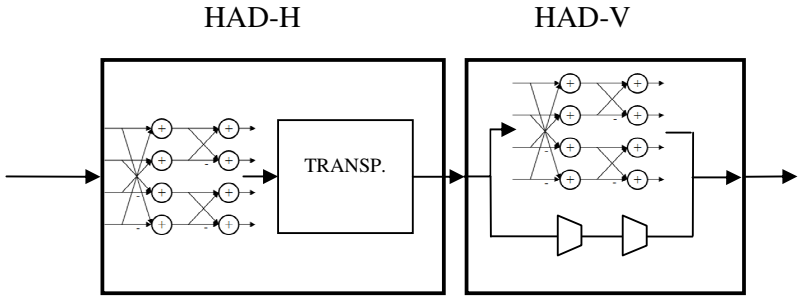


Figura 16 Fluxo de dados nos módulos HAD-H e HAD-V, este com dois caminhos diferentes para os dados.

Como a equação que rege a Hadamard não envolve multiplicações, o deslocamento para a direita, as somas e as subtrações causam apenas um aumento de representação de dois bits por amostra para o cálculo completo. Esse aumento foi confirmado pela análise da simulação do pior caso. Assim a entrada de 14 bits por amostra da Hadamard se transforma em uma saída de 16 bits por amostra (HUSEMANN, 2010b).

A HAD-H recebe como entrada os valores DC da saída da DCT-2D, ou seja, uma amostra entra na Hadamard a cada 16 amostras (4 colunas) saídas da DCT. Isso implica em duas considerações práticas:

- Um cálculo de Hadamard precisa de 16 blocos de luminância de DCT ou 8 blocos de crominância (4 blocos Cb e 4 blocos Cr) para todo processamento;
- As amostras que não passam pela transformada Hadamard precisam ser armazenadas durante o período do cálculo, e sincronizadas para a correta montagem da saída.

Devido à necessidade de um maior número de blocos serem processados de cada vez, a latência apresentada pela Hadamard é maior. O cálculo da HAD-2D projetado e implementado apresenta uma latência de 64 ciclos de relógio para blocos de luminância e 32 ciclos para calcular as duas crominâncias.

Resolvido o cálculo da Hadamard, é necessário resolver o armazenamento dos valores AC, que não entram na HAD-2D. Para isso, foi criado um módulo de gerência da Hadamard (Gerenciador Hadamard) com duas memórias. Uma dessas memórias armazena os valores AC recebidos do módulo de DCT. A outra trabalha em série com a HAD-2D com a finalidade de posicionar os coeficientes DC calculados para substituírem os valores DC originais correspondentes.

A Figura 17 apresenta o Gerenciador Hadamard com a memória para armazenamento dos valores AC (MB Buffer), a memória que ajusta o posicionamento dos coeficientes na

saída da Hadamard (HAD Buffer), o cálculo (HAD-H e HAD-V) e um módulo que faz a recomposição dos blocos (Output Composer). Seus sinais de entrada são compatíveis aos sinais de saída da DCT-2D.

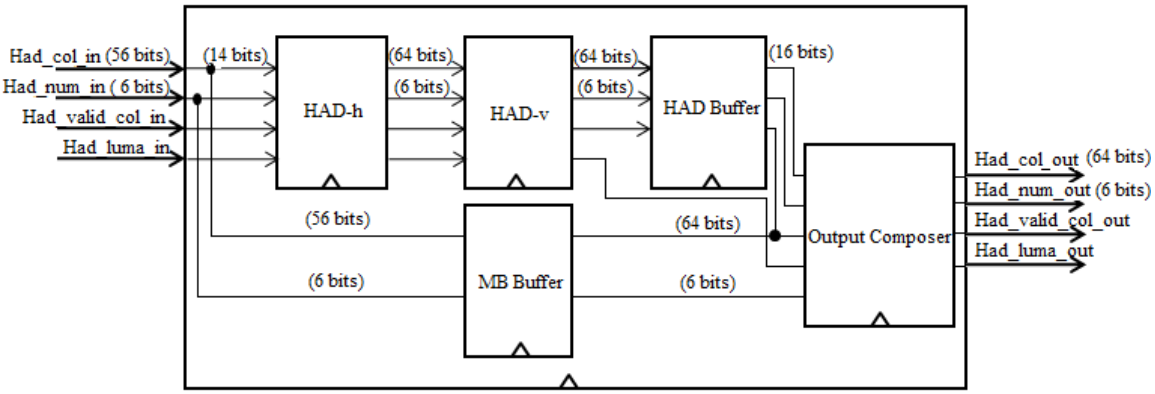


Figura 17 Gerenciador Hadamard adaptado de (HUSEMANN, 2010c).

MB Buffer é uma memória capaz de armazenar 384 amostras, 16 blocos de luminância e 8 de crominância (4 Cb e 4 Cr). HAD Buffer também deve suportar um bloco de luminância e dois de crominância, totalizando 24 amostras a serem armazenadas. Essas memórias foram projetadas como DP-RAMs (*DualPort RAM*) com a finalidade de diminuir a latência com a possibilidade de escrita e leitura de dados simultâneamente (HUSEMANN, 2010b).

Essa implementação de Hadamard, assim como a DCT, pode processar até 4 amostras por ciclo de relógio. A tabela abaixo apresenta os resultados obtidos para esse módulo.

Tabela 9 Resultado da implementação da Hadamard em FPGA.

Módulo	# LUTs	# RAMB 16	Período (ns)
HAD-2D	599	0	3,521
Gerenciador Hadamard	802	2	3,521

FPGA Xilinx V2P30FF896

Onde RAMB 16 é a unidade de memória utilizada pela XILINX e o Gerenciador Hadamard representa a HAD-2D, memórias e Output Composer. A HAD-2D representa os dois módulos HAD-H e HAD-V.

5.4 QUANTIZAÇÃO PROPOSTA

A quantização deve realizar os cálculos apresentados nas equações (42), (43) e (44). Para tanto, levou-se em consideração os seguintes pressupostos práticos (HUSEMANN, 2010c):

- Como entrada é esperado um vetor de amostras vindos do módulo de transformadas, o endereço dentro de um macrobloco, sinal indicando luminância/crominância, sinal indicando inter/intra-quadro e o valor de QP;
- MF e f são gravados em uma memória evitando ao máximo multiplicações e divisões que não sejam binárias (na base dois). Assim são utilizadas tabelas indexadas por QP, posição do coeficiente no bloco, sinal que indica se é intra-quadro e sinal que indica se é luminância;
- O sinal (positivo ou negativo) é retirado e recuperado com ajuda de um teste para ver se X é positivo ou negativo, um subtrator tira o sinal negativo quando necessário (|X| = 0-X para X negativo) e registradores guardando o sinal;
- A multiplicação entre |X| e MF que não pode ser evitada, adota blocos multiplicadores inteiros presentes no FPGA utilizado. Esses *Multiplier Blocks* (XILINX,2007) realizam multiplicações de dois valores com no máximo 18 bits cada e não dependem do ciclo de relógio, apresentado o resultado no mesmo ciclo em que foram fornecidas as entradas.
- qbits é calculado com ajuda de uma tabela indexada por QP;
- Outros cálculos são bastante simples, são deslocamentos ou somas.

O procedimento apresentado deve ser executado para cada amostra, sendo que para manter o fluxo de 4 amostras por ciclo de relógio, 4 destes procedimentos são realizados em paralelo, aproveitando as mesmas tabelas de memórias. As amostras de entrada possuem 16 bits cada e após quantizadas apresentam 8 bits. A estrutura interna é vista na Figura 18.

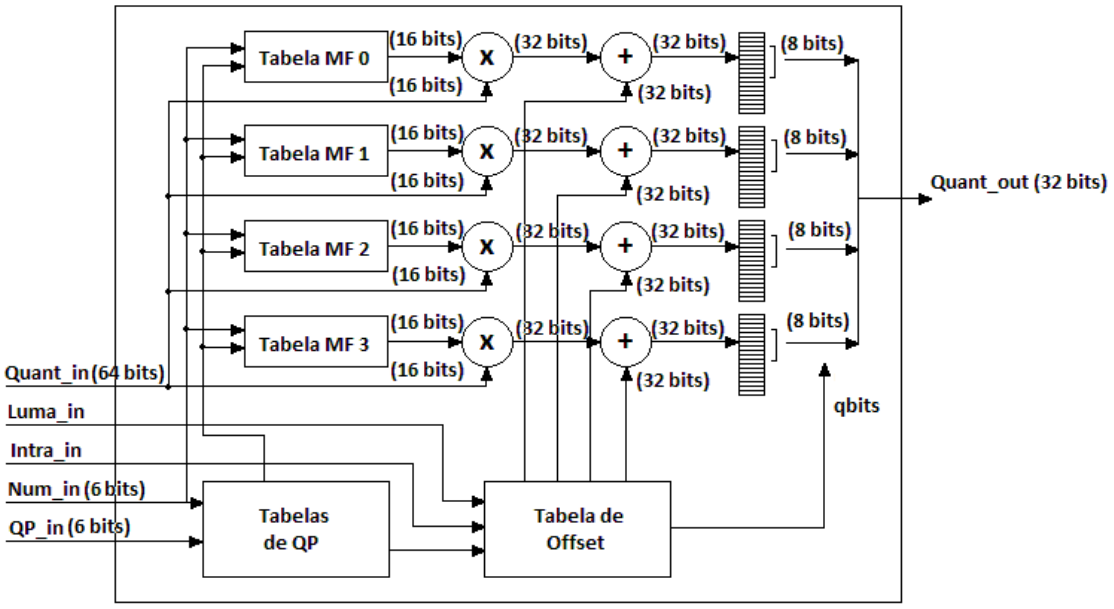


Figura 18 Estrutura interna da Quantização (HUSEMANN, 2010c).

A implementação desse módulo envolve operações mais complexas que as transformadas, como exemplo tem-se as multiplicações e deslocamentos variáveis. Essas operações implicam em um módulo que ocupa uma maior área e maior período de clock. O resultado dessa análise encontra-se na Tabela 10.

Tabela 10 Resultados da implementação da Quantização em FPGA.

Módulo	# LUTs	# RAMB 16	# MULT 18×18	Período (ns)
Quantização	1017	2	4	7,274

FPGA Xilinx V2P30FF896

Os valores quantizados devem ser enviados de volta para o computador na implementação do codificador híbrido. Com 8 bits por amostra, as 4 amostras somam 32 bits, portanto os valores a serem transmitidos utilizam de forma eficiente a banda de interface FPGA/Computador. A declaração da entidade deste componente encontra-se abaixo e apresenta alguns sinais não representados na Figura 18 que servem para controle do módulo.

5.5 HADAMARD INVERSA PROPOSTA

O primeiro módulo inverso implementado será o da Hadamard Inversa para ficar de acordo com o *software* de referência. O *software* utilizado como referência de implementação e comparação de resultados é o JM (Joint Model). No JM, o cálculo das inversas segue a ordem: Hadamard Inversa, Quantização Inversa e, por fim, DCT Inversa.

A Hadamard Inversa apresenta uma arquitetura bastante similar à Hadamard direta. A maior diferença é a largura de bits utilizada por amostra. Entram na Hadamard Inversa 8 bits por amostra e, devido aos cálculos internos, as amostras apresentam um aumento de 4 bits.

No cálculo, a única diferença em relação à Hadamard direta é que o deslocamento para a direita do resultado não precisa ser efetuado.

Nessa transformada, o módulo gerenciador também será utilizado. O gerenciador da Hadamard Inversa (Gerenciador IHadamard) encontra-se na Figura 19 com sinais internos representados.

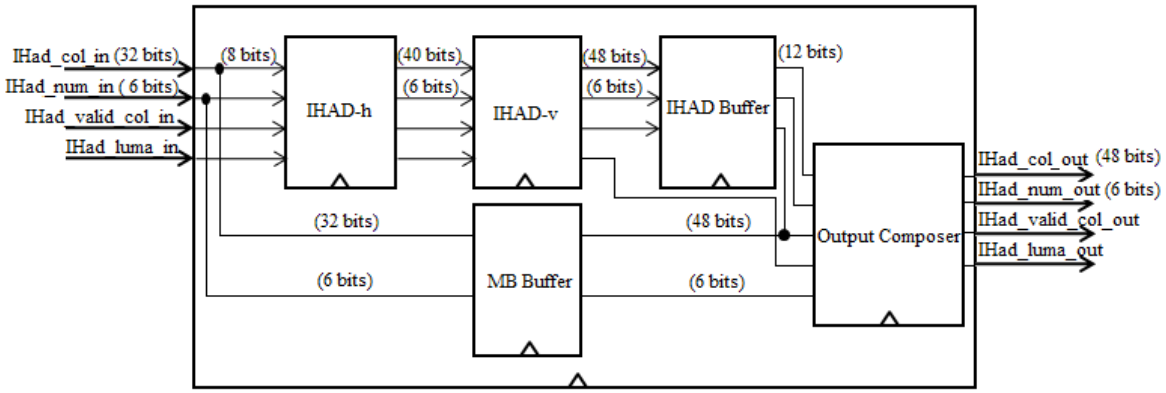


Figura 19 Estrutura interna do Gerenciador da Hadamard Inversa.
Adapatdo de (HUSEMANN, 2010c).

A implementação deste módulo apresenta como resultado a Tabela 11. A IHAD-2D é composta apenas pelos módulos IHAD-H e IHAD-V, enquanto que o Gerenciador IHadamard envolve além desses, o MB Buffer, IHAD Buffer e Output Composer (HUSEMANN, 2010b).

Tabela 11 Resultados da implementação da Hadamard Inversa em FPGA.

Módulo	# LUTs	# RAMB 16	Período (ns)
IHAD-2D	406	0	3,115
Gerenciador IHadamard	565	2	3,115

FPGA Xilinx V2P30FF896

5.6 QUANTIZAÇÃO INVERSA PROPOSTA

A quantização inversa deve realizar os cálculos apresentados nas equações (47) e (48). Esse módulo é mais simples que a quantização direta, mas apresenta uma linha de cálculo parecida (HUSEMANN, 2010c):

- MF' é gravado em uma memória evitando ao máximo multiplicações e divisões que não sejam binárias (na base dois).
- O sinal (positivo ou negativo) é retirado e recuperado com ajuda de um teste para ver se X é positivo ou negativo, um subtrator tira o sinal negativo quando necessário (|X| = 0-X para X negativo) e registradores guardam o sinal até o momento da recuperação;
- A multiplicação entre |X| e MF' que não pode ser evitada, adota blocos multiplicadores inteiros presentes no FPGA utilizado.
- qbits é aproveitado de cálculos executados na quantização direta;
- Outros cálculos não apresentam grande dificuldade, são deslocamentos ou somas.

Quatro desses procedimentos são executados em paralelo como mostra a Figura 20. Assim como qbits é aproveitado da quantização direta, um dos índice para as tabelas MF' é aproveitado. Esse índice está representado na figura por QP'_in, e foi obtido da tabela de QPs da quantização direta.

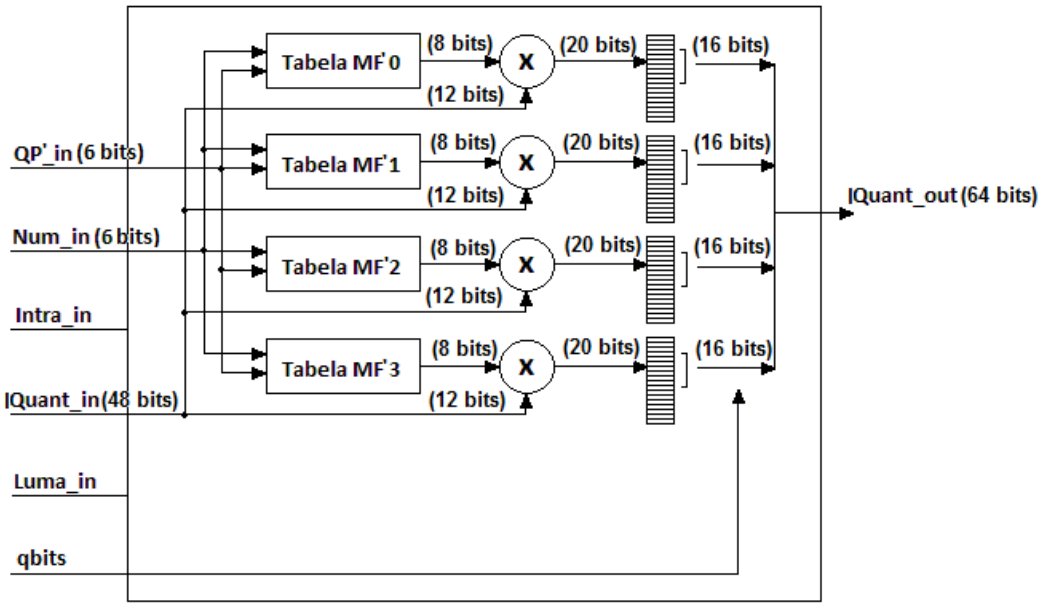


Figura 20 Estrutura interna da Quantização Inversa (HUSEMANN, 2010c).

A implementação desse módulo gera o resultado apresentado na Tabela 12.

Tabela 12 Resultados da implementação da Quantização Inversa em FPGA.

Módulo	# LUTs	# RAMB 16	# MULT 18×18	Período (ns)
Quantização Inversa	672	-	4	5,035

FPGA Xilinx V2P30FF896

5.7 DCT INVERSA PROPOSTA

A DCT Inversa proposta apresenta a mesma estrutura da DCT Direta apresentada. A diferença está nos cálculos de soma e deslocamento executados. Na primeira etapa do cálculo em uma dimensão calcula-se:

$$I0 = X0 + X2 \quad (85)$$

$$I1 = X0 - X2 \quad (86)$$

$$I2 = 2 \cdot X1 - X3 \quad (87)$$

$$I3 = X1 + 2 \cdot X3 \quad (88)$$

Na segunda etapa:

$$F0 = I0 + I3 \quad (89)$$

$$F1 = I1 + I2 \quad (90)$$

$$F2 = I1 - I2 \quad (91)$$

$$F3 = I0 - I3 \quad (92)$$

A transposição está junto ao cálculo da primeira dimensão da DCT. É importante notar que na DCT direta utiliza-se uma transposição fazendo as amostras que entram no módulo em linha se transformar em colunas. Agora, na DCT inversa, esse procedimento será revertido. As amostras que entram em coluna passam pela transposição e se transformam novamente em linhas. A Figura 21 apresenta essa troca de coluna por linhas.

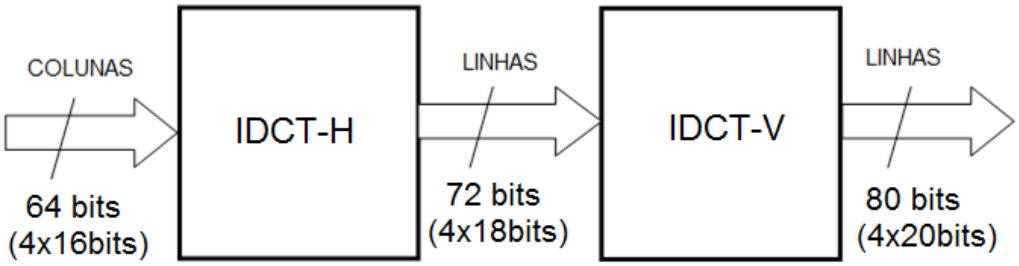


Figura 21 IDCT-2D em dois estágios

O resultado da síntese desse módulo está apresentado abaixo.

Tabela 13 Resultado da implementação da DCT Inversa em FPGA.

Módulo	# LUTs	Período (ns)
IDCT-2D	528	3,548

FPGA Xilinx V2P30FF896

6 OTIMIZAÇÃO DA OPERAÇÃO CONJUNTA DOS MÓDULOS

O foco principal deste trabalho está na análise prática do comportamento dos módulos apresentados e sua forma de conexão para fazer parte do módulo computacional de um codificador de vídeo.

A Figura 22 apresenta os módulos de transformadas (DCT e Hadamard) ligados à Quantização. Esse conjunto de módulos será discutido no texto que segue com a denominação de Módulos Diretos.

Figura 22 Ligação usual para os módulos diretos em um codificador.

O componente que faz essas ligações foi declarado em VHDL como uma entidade da seguinte forma:

```
entity Modulos_Diretos is
Port (
      MD_in                : in  STD_LOGIC_VECTOR (31 downto 0);
      MD_addr_in           : in  STD_LOGIC_VECTOR ( 5 downto 0);
      MD_valid_in          : in  STD_LOGIC;
      MD_luma_in           : in  STD_LOGIC;
      MD_qp_in_luma        : in  STD_LOGIC_VECTOR ( 7 downto 0);
      MD_qp_in_croma       : in  STD_LOGIC_VECTOR ( 7 downto 0);
      MD_rst               : in  STD_LOGIC;
      mclk                 : in  STD_LOGIC;
      MD_out               : out STD_LOGIC_VECTOR (31 downto 0);
      MD_addr_out          : out STD_LOGIC_VECTOR ( 5 downto 0);
      MD_valid_out         : out STD_LOGIC;
      MD_luma_out          : out STD_LOGIC;
      MD_qp_round_out_croma: out STD_LOGIC_VECTOR ( 3 downto 0);
      MD_qp_rem_out_croma  : out STD_LOGIC_VECTOR ( 3 downto 0);
      MD_qp_round_out_luma : out STD_LOGIC_VECTOR ( 3 downto 0);
      MD_qp_rem_out_luma   : out STD_LOGIC_VECTOR ( 3 downto 0));
end Modulos_Diretos;
```

A Figura 23 apresenta os módulos de transformadas Inversas (DCT e Hadamard Inversas) ligados à Quantização Inversa. Os módulos da Figura 23 serão abordados pelo nome de Módulos Inversos no texto que segue. Essas ligações colocam os módulos a operar de forma serial, sendo que um módulo depende da resposta dos módulos que o antecedem.

Figura 23 Ligação usual para os módulos inversos em um codificador.

O componente que envolve essas ligações foi assim declarado:

```
entity Modulos_Inversos is
Port (
      MI_in               : in  STD_LOGIC_VECTOR (31 downto 0);
      MI_addr_in          : in  STD_LOGIC_VECTOR ( 5 downto 0);
      MI_valid_in         : in  STD_LOGIC;
      MI_luma_in          : in  STD_LOGIC;
      MI_rst              : in  STD_LOGIC;
      MI_qp_round_in_croma: in  STD_LOGIC_VECTOR ( 3 downto 0);
      MI_qp_rem_in_croma  : in  STD_LOGIC_VECTOR ( 3 downto 0);
      MI_qp_round_in_luma : in  STD_LOGIC_VECTOR ( 3 downto 0);
      MI_qp_rem_in_luma   : in  STD_LOGIC_VECTOR ( 3 downto 0);
      mclk                : in  STD_LOGIC;
      MI_out              : out STD_LOGIC_VECTOR (31 downto 0);
      MI_addr_out         : out STD_LOGIC_VECTOR ( 5 downto 0);
      MI_valid_out        : out STD_LOGIC;
      MI_luma_out         : out STD_LOGIC);
end Modulos_Inversos;
```

A Figura 24 apresenta a ligação entre os módulos diretos e inversos que também ocorre de forma serial. Dessa forma, o desempenho do conjunto dos módulos será limitado ao desempenho do módulo mais lento (HUSEMANN, 2010b).

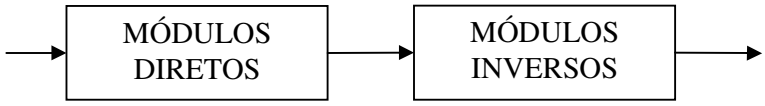


Figura 24 Ligação entre os módulos diretos e inversos no codificador.

Essas ligações foram realizadas e sintetizadas para a plataforma de desenvolvimento XUP Virtex 2P. Seu funcionamento foi comprovado comparando-se os resultados gerados pela arquitetura desenvolvida com o software de referência H.264, JM (*Joint Model*), para mesmos macroblocos de entrada.

Além disso, estes módulos foram ligados ao computador para substituir a parte do JSVM (*software* equivalente ao JM que realiza codificação escalável) que realiza os mesmos

cálculos. Para tanto as entradas e saídas dos módulos comunicantes com o computador deviam ter 32 bits. A entrada da DCT foi projetada para 32 bits e a saída da quantização também apresenta 32 bits. Só mais um módulo deve se comunicar com o computador, a DCT Inversa. Com essa comunicação funcionando, fez-se o codificador trabalhar de forma híbrida.

As Tabelas 14 e 15 resumem os resultados de ocupação e desempenho dos módulos diretos e inversos operando individualmente. A operação dos módulos em conjunto apresentam o desempenho da quantização, 7,274 ns de período mínimo para o ciclo de relógio. A Quantização, com o pior período, limita a velocidade de todo o conjunto por estarem ligados em série.

Tabela 14 Dados dos módulos individuais diretos.

Módulo	LUTS	RAMB 16	Período(ns)	MULT 18×18
DCT	364	-	2,894	-
HADAMARD	802	2	3,521	-
QUANTIZAÇÃO	1017	2	7,274	4
MÓDULOS DIRETOS	2192	4	7,274	4

FPGA Xilinx V2P30FF896

Tabela 15 Dados dos módulos individuais inversos.

Módulo	LUTS	RAMB 16	Período(ns)	MULT 18×18
HADAMARD INVERSA	565	2	3,115	-
QUANTIZAÇÃO INVERSA	672	-	5,035	4
DCT INVESA	528	-	3,548	-
MÓDULOS INVERSOS	1744	2	5,035	4

FPGA Xilinx V2P30FF896

Sendo que o conjunto apresenta um processamento de quatro amostras por ciclo de clock e período mínimo de 7,274 ns para o ciclo, a taxa de amostras processadas por segundo é de no máximo 551 milhões de amostras por segundo.

6.1 OTIMIZAÇÃO PARA ENTRADA DE 32 BITS

O objetivo da otimização do módulo computacional é, mantendo a largura de bits de entrada e saída que se trabalha, aumentar a velocidade de processamento das amostras. Utilizando os módulos já descritos, a quantização é o módulo que limita o desempenho do conjunto. Todos os módulos apresentam uma taxa de 4 amostras por *clock* e, sendo o período mínimo da Quantização maior, sua taxa de amostras por segundo fica menor.

Assim sendo, para conseguir um período mínimo de operação melhor, utilizou-se duas quantizações que operam em paralelo e uma barreira temporal (Figura 25).

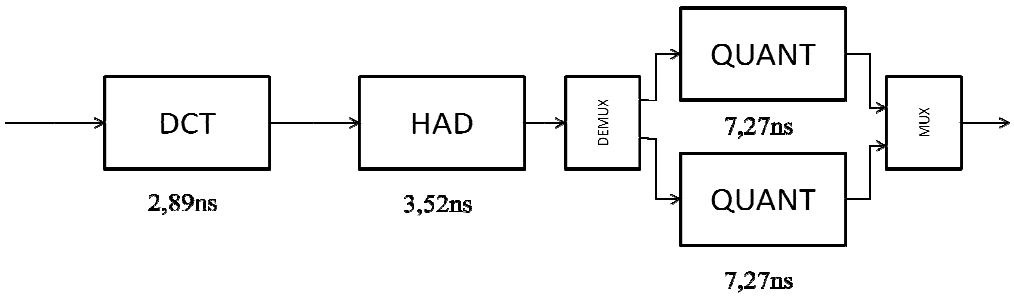


Figura 25 Módulos diretos com duas Quantizações.

Nessa abordagem, o *DEMUX* e o *MUX* se comportam como uma barreira temporal. Em um primeiro instante os dados passam pela DCT, Hadamard e uma das Quantizações. Em um segundo momento, os *DEMUX* e *MUX* são chaveados para que os dados passem pela DCT, Hadamard e a outra Quantização. Esse chaveamento ocorre a cada ciclo de *clock* e resulta em uma quantização equivalente com período de *clock* mínimo menor.

Nessa abordagem consome-se um pouco mais de área, mas utilizando a mesma largura de bits de entrada e saída se consegue um fluxo de amostras agora limitado pela Hadamard nos módulos diretos (HUSEMANN, 2010c). Isso equivale a um módulo 2,06 vezes mais rápido ocupando 1,48 vezes mais área (3238 LUTs para os Módulos Diretos).

Essa mesma barreira temporal deve ser inserida aos módulos inversos. Assim os ganhos em velocidade para a quantização não serão limitados pela quantização inversa.

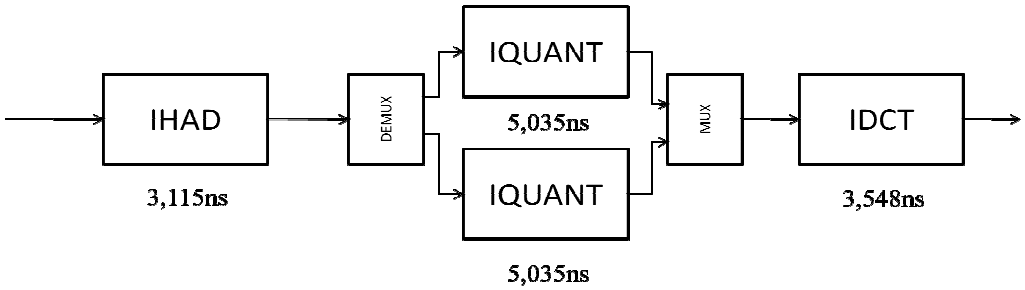


Figura 26 Módulos inversos com duas Quantizações Inversas.

Na Figura 26 pode ser vista a ligação das duas Quantizações Inversas aos demais módulos do sistema. Nessa abordagem a Quantização Inversa deixa de ser o limitador dos Módulos Inversos, a DCT Inversa passa a ser o novo limitador.

Sintetizando os módulos diretos e inversos em FPGA, o resultado obtido é:

Tabela 16 Resultados da otimização para 32 bits de entrada.

Módulo	LUTS	RAMB 16	Período(ns)	MULT 18×18
MÓDULOS DIRETOS	3238	6	3,521	8
MÓDULOS INVERSOS	2488	2	3,521	8

FPGA Xilinx V2P30FF896

Com essa abordagem, o processamento das amostras chega ao valor de 1136 milhões de amostras por segundo. O uso de maiores paralelizações não serão explorados para a largura de entrada de 32 bits pois todos os módulos apresentam tempos para o ciclo de relógio muito próximos (valores variam de 2,894 ns a 3,521 ns). Dessa forma, para diminuir tempos mantendo a estrutura, todos os módulos devem ser replicados chegando a uma relação de dobro de velocidade para dobro da área ocupada.

6.2 OTIMIZAÇÃO PARA ENTRADA DE 64 BITS

Visando uma comunicação que apresente 64 bits de entrada por ciclo de relógio, os módulos foram adaptados. A seguir são apresentadas as formas de operação dos módulos para processar 8 amostras (64 bits) por ciclo de relógio.

Duas DCTs operam em paralelo, processando dois blocos de cada vez (HUSEMANN, 2010c). Uma DCT processa o primeiro bloco enquanto que a segunda processa o segundo bloco. A Figura 27 apresenta dois blocos vizinhos em um macrobloco sendo processados simultaneamente. Entrada e saída de 8 amostras por ciclo de relógio.

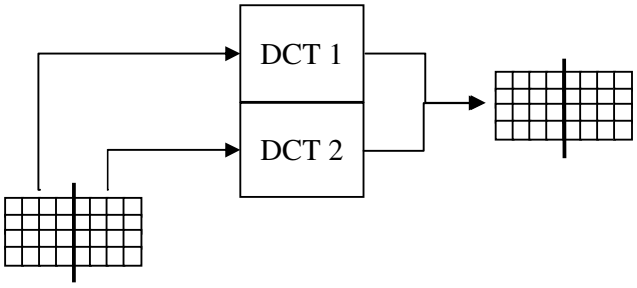


Figura 27 Processamento de duas DCTs em paralelo.

A Hadamard, por processar um macrobloco inteiro e não apenas um bloco teve de ser levemente modificada e não precisou ser duplicada. A alteração apenas está na entrada que o invés de adquirir um valor DC a cada 4 ciclos de relógio, deve adquirir 2 valores DC. E na saída que deve apresentar 8 valores transformados por ciclo de relógio o invés de 4. A Figura 28 apresenta a parte computacional da Hadamard adquirindo os dois valores DC que aparecem simultaneamente na entrada de dois blocos (HUSEMANN, 2010a).

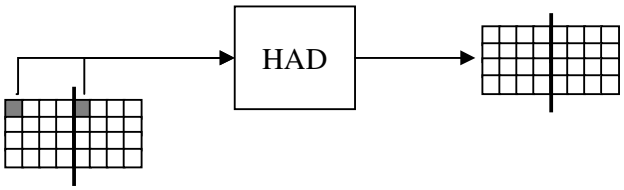


Figura 28 Hadamard processando 2 valores DC por vez.

O Gerenciador Hadamard nessa implementação apresenta o seguinte formato:

```
entity Gerenciador_Hadamard is
Port  (
      Had_col_in       : in  STD_LOGIC_VECTOR (111 downto 0);
      Had_num_in       : in  STD_LOGIC_VECTOR (  4 downto 0);
      Had_valid_col_in : in  STD_LOGIC;
      Had_luma_in      : in  STD_LOGIC;
      mclk             : in  STD_LOGIC;
      Had_col_out      : out STD_LOGIC_VECTOR (127 downto 0);
      Had_num_out      : out STD_LOGIC_VECTOR (  4 downto 0);
      Had_valid_col_out: out STD_LOGIC;
      Had_luma_out     : out STD_LOGIC);
end Gerenciador_Hadamard;
```

As larguras dos barramentos de dados foram todas dobradas de tamanho assim como a largura da memória que guarda os valores AC.

A Quantização, assim como a DCT, apenas teve de ser duplicada. Utilizando o recurso da barreira temporal, esse módulo fica com quatro quantizações trabalhando em paralelo (HUSEMANN, 2010c). O módulo direto para uma entrada de 64 bits é apresentado na Figura 29.

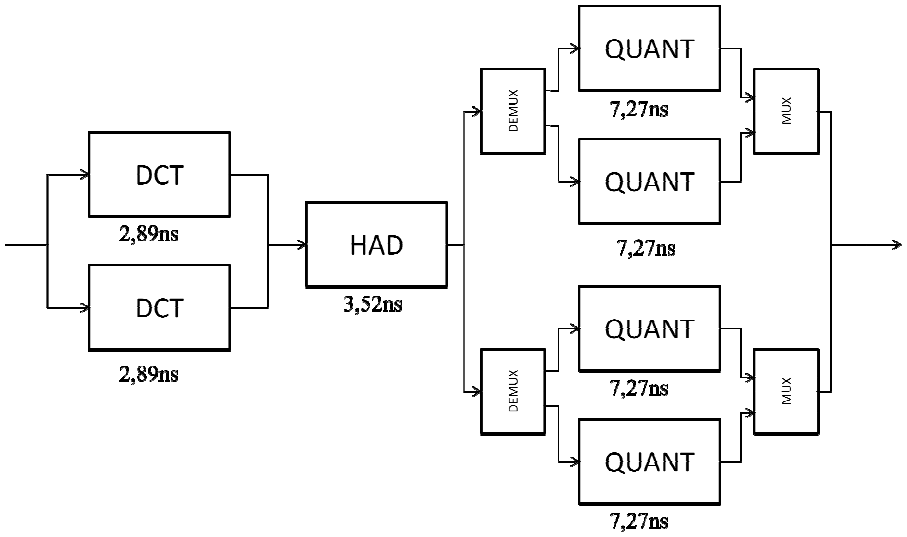


Figura 29 Módulos Diretos para uma entrada de 64 bits

O mesmo que foi feito para os módulos diretos é válido para os módulos inversos, chegando a seguinte arquitetura:

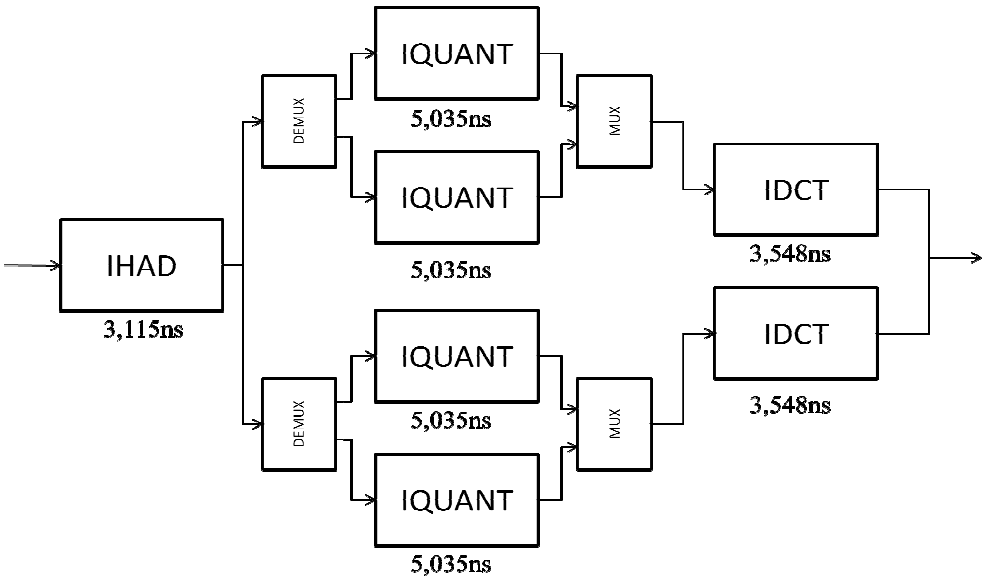


Figura 30 Módulos Inversos para uma entrada de 64 bits

Sintetizando os módulos diretos e inversos em FPGA, o resultado obtido é o seguinte:

Tabela 17 Resultados da otimização para 64 bits de entrada.

Módulo	LUTS	RAMB 16	Período(ns)	MULT 18X18
MÓDULOS DIRETOS	5469	16	3,532	16
MÓDULOS INVERSOS	4192	10	3,532	16

FPGA Xilinx V2P30FF896

Essa abordagem apresenta um período mínimo para o ciclo de relógio praticamente igual ao da abordagem para 32 bits de entrada, porém processa 8 amostras por ciclo. Dessa forma, processa 2265 milhões de amostras por segundo (HUSEMANN, 2010c).

O estudo desta arquitetura é interessante, pois para o processamento de 8 amostras por ciclo de clock não basta a simples duplicação da arquitetura que processa 4 amostras por clock, principalmente por a Hadamard proposta já trabalhar com macroblocos.

7 CONCLUSÃO

A elaboração das rotinas de Transformadas e Quantização em hardware se mostra válida por atingir os objetivos: codificação de HDTV em tempo real. Mesmo com a primeira abordagem estes requisitos são alcançados. O processamento de 551 milhões de amostras por segundo excede a taxa mínima de 93,3 milhões de amostras por segundo necessária para atingir tempo real quando processando vídeos HDTV com 1920×1080 pixels à taxa de 30 quadros por segundo(AGOSTINI,2007).

A otimização para comunicações de 32 bits abordada na seção 6.1 se mostra interessante ao se trabalhar com dispositivos de lógica programável de baixo desempenho. Pode ser utilizado como estratégia quando se tem lógica sobrando e os tempos não são atingidos.

A otimização para a comunicação de 64 bits vista na seção 6.2 é uma inovação que visa a crescente utilização de dispositivos que trabalham com essa largura de bits. Essa abordagem diminui o número de acessos à PCI necessários na transmissão e recepção de amostras.

Os módulos apresentados visam um melhor aproveitamento dos recursos de *hardware* no que diz respeito ao seu desempenho. Se a área lógica for a limitação do *hardware* a disposição, então sugere-se a utilização da simples implementação serial dos módulos como apresentado nas figuras 22, 23 e 24.

O teste executado com o FPGA V2P30FF896 foi realizado com um *clock* de 100MHz. Ao se colocar os módulos desenvolvidos a fazer parte do codificador híbrido rodando as outras operações em *software*, a comunicação com o computador através da PCI limita o codificador, sendo os módulos implementados mais rápidos que esta interface. Este teste foi feito com a arquitetura proposta para comunicação de 32 bits.

8 REFERÊNCIAS BIBLIOGRÁFICAS

Agostini, L.; Porto, R.; Guntzel, J.; Saraiva Silva, I.; Bampi, S. *"High throughput multitransform and multiparallelism IP for H.264/AVC video compression standard,"* Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on , vol., no., pp.4 pp.-5422, 0-0 0

Agostini, L. *"Desenvolvimento de Arquiteturas de Alto Desempenho Dedicadas à Compressão de Vídeo Segundo o Padrão H.264/AVC"*. 2007. Tese de Doutorado - UFRGS, Porto Alegre/RS.

Bhaskaran, V.; Konstmtinides, K. *Image and Video Compression Standards. Algorithms and Architectures*. Zd Edition. Kluwer Academic Publishers 1997.

Cheng, Z.; Chen, C.; Liu, B.; Yang, J. *"High throughput 2-D transform architectures for H.264 advanced video coders,"* IEEE Asia-Pacific Conference on Circuits and Systems, 2004, pp. 1141-1144.

Ghanbari, M. Standard Codecs: *Image Compression to Advanced Video Coding*. United Kingdom: The Institution of Electrical Engineers, 2003.

Husemann, R.; Majolo, M.; Susin, A.; Roesler, V.; Lima, J. V. (2010a) *"Highly Efficient Transforms Module Solution for a H.264/SVC Encoder,"* ISVLSI, pp.86-91, 2010 IEEE Annual Symposium on VLSI, 2010.

Husemann, R.; Majolo, M.; Roesler, V.; Lima, J. V.; Susin, A. A. (2010b) *"Highly Efficient Forward Two-Dimensional DCT Module Architecture for H.264/SVC"*. International Symposium on Rapid System Prototyping, 2010, Virginia. Proceedings of the 21st International Symposium on Rapid System Prototyping. Virginia, 2010. v. 1.

Husemann, R; Roesler, V.; Lima, J. V.; Susin, A. A. *"Hardware Integrated Quantization Solution for Improvement of Computational H.264 Encoder Module"*. 18th Very Large Scale Integration (VLSI), System-on-Chip, 2010, Madrid.

Jack, K. *Video Demystified*. Eagle Rock: LLH Technology Publishing, 2001.

Malvar, H.S.; Hallapuro, A.; Karczewicz, M.; Kerofsky, L.; , *"Low-complexity transform and quantization in H.264/AVC,"* Circuits and Systems for Video Technology, IEEE Transactions on , vol.13, no.7, pp. 598- 603, July 2003

Prasoon, A. K.; Rajan, K. *4 × 4 2-D DCT for H.264/AVC.* In Proceedings of the International Conference on Advances in Computing, Communication and Control (ICAC3 '09), 2009.

Richardson I. E. G.; *H.264 and MPEG-4 Video Compression*, England, Ed. Wiley & Sons, 2003.

Sistema Brasileiro de Televisão Digital. SBTVD Home. Disponível em: <http://forumsbtvd.org.br/>. Acesso em: Outubro de 2010.

Shi, Y.; Sun, M. *Image and Video Compression for Multimedia Engineering: Fundamentals, Algorithms and Standards*. Boca Raton: CRC Press, 1999.

Silva, A. M. C.; Silva, T. L.; Porto, M. S.; Porto, R. E. C.; Güntzel, J. L.; Silva, I. S.; Bampi, S.; Agostini, L. V. *"Exploração no Espaço de Projeto da Hadamard 4x4 Direta do Padrão de Compressão de Vídeo H.264/AVC"*. In: Workshop Iberchip, 12 San José, Costa Rica, 2006. p.99-102.

Wang, T. C.; Huang, Y. W.; Fang, H. C.; Chen, L. G. *"Parallel 4x4 2D Transform and Inverse Transform Architecture for MPEG-4 AVC / H.264"*, Proc. of IEEE ISCAS, 2003.

Wootton, Cliff . *A Practical Guide to Video and Audio Compression.* Oxford, UK: Elsevier's Science & Technology, 2005.

Xilinx University Program Virtex-II Pro Development System: Hardware Reference Manual Disponível em: <www.xilinx.com>. Acessado em:Outubro de 2010.