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

ARNALDO PEREIRA DE AZEVEDO FILHO

MoCHA: Arquitetura Dedicada para a Compensação de Movimento em Decodificadores de Vídeo de Alta Definição, Seguindo o Padrão H.264

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

Prof. Dr. Sergio Bampi
Orientador

Porto Alegre, maio de 2006

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Azevedo Filho, Arnaldo Pereira de

MoCHA: Arquitetura Dedicada para a Compensação de Movimento em Decodificadores de Vídeo de Alta Definição, Seguindo o Padrão H.264 / Arnaldo Pereira de Azevedo Filho. – Porto Alegre: PPGC da UFRGS, 2006.

120 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2006. Orientador: Sergio Bampi.

1. H.264. 2. Descompressão de vídeo. 3. Compensação de movimento. 4. Vídeo de alta definição. 5. Processamento de vídeo. I. Bampi, Sergio. II. Título.

"Passado.
Presente!
Futuro?"
— ARNALDO PEREIRA DE AZEVEDO FILHO

AGRADECIMENTOS

Quero agradecer primeiramente a Deus por me permitir e dar as condições de chegar a esse momento da minha vida. Aos meus pais, Arnaldo Pereira de Azevedo e Maria de Lourdes Silva Azevedo, que foram, são, e sempre serão meus mestres e que sempre me apoiaram nas minhas decisões. Aos meus irmãos Adelson e Aretuza, com os quais cresci e aprendi.

Agradeço também aos meus companheiro de vida estudantil, nos seus diversos estágios, que se tornaram grandes amigos e irmãos. Agradeço aos companheiros dos meus primeiros contatos com a computação, durante o segundo grau técnico na ETFRN, atual CEFET: Deyse Moura, Weldson Lima, Jorge Wanderson e Helder Lira. A todos os membros do CCCC (e grupos adjacentes) que estavam ao meu lado durante graduação na UFRN. Aos meus colegas (e também amigos) da pós graduação da UFRGS, pelo acolhimento e pelo compartilhamento de grandes momentos dessa jornada (os quais não cito nominalmente para não incorrer em injustiça por algum lapso). Aos participantes do projeto H.264 Brasil, os quais me auxiliaram para a conclusão deste trabalho.

Não poderia deixar de reconhecer o trabalho realizado pelos funcionários do Instituto de informática, desempenhando suas funções nos laboratórios, na biblioteca e na secretaria. Como também o suporte fornecido pelo CNPq ao conceder-me uma bolsa para dedicar-me a este trabalho.

Aos meus amigos Jadeilson Ferreira e Susanne Pereira. Aos demais membros do Tri-Ciclo: Nelson Silva e Kledmar Alves.

A todos os professores com os quais aprendi. Ao professor Ivan Saraiva Silva por me apresentar à pesquisa, ao professor Sergio Bampi, meu orientador. Ao Bruno Zatt, meu braço direito no desenvolvimento dessa dissertação.

Agradeço de forma especial a Cristina Meinhardt e a Luciano Agostini, por me cederem as condições de desenvolver esse texto no prazo e, junto com a Ana Pinto, por suas valiosas contribuições para o mesmo.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

1-D	Uma Dimensão
2-D	Duas Dimensões
1080HD	High Definition Television com 1080 linhas
720HD	High Definition Television com 720 linhas
AVC	Advanced Video Coding
ASO	Arbitrary Slice Order
CABAC	Context-Based Adaptive Binary Arithmetic Coding
CAVLC	Context-Based Adaptive Variable Length Coding
CIF	Common Intermediate Format
codec	codificador/decodificador
DDR	Double Data Rate
DIMM	Dual In-Line Memory Module
DRAM	Dynamic Random Access Memory
DVD	Digital Versatile Disk
FIFO	First In First Out
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
FRExt	Fidelity Range Extensions
GB	Giga Bytes
HDTV	High Definition Digital Television
H422P	High 4:2:2 Profile
H444P	High 4:4:4 Profile
Hi10P	High 10 Profile
HP	High Profile
HSI	Hue, Saturation, Intensity
IEEE	Institute of Electric and Electronics Engineers

INTRA	Intra Prediction
ISO	International Organization for Standardization
ITU-T	International Telecommunication Union - Telecommunication
JVT	Joint Video Team
KB	Kilo Byte
MB	Mega Bytes
MBAFF	Macroblock Adaptative Frame/Field
MC	Motion Compensation
ME	Motion Estimation
MoCHA	Motion Compensator Hardware Architecture
MPEG	Moving Picture Experts Group
NAL	Network Adaptation Layer
OPB	Open Protocol Bus
PAFF	Picture Adaptative Frame/Field
POC	Picture Order Count
PLB	Processor Local Bus
PMV	Predictive Motion Vector
PSNR	Peak Signal-to-Noise Ratio
Q	Quantization
Q^{-1}	Inverse Quantization
QCIF	Quarter Common Intermediate Format
QP	Quantization Parameter
RAM	Random Access Memory
RGB	Red, Green, Blue
SATA	Serial ATA
SBTVD	Sistema Brasileiro de Televisão Digital
SDTV	Standard Definition Television
SP	Switching P
SI	Switching I
T	Transform
T^{-1}	Inverse Transform
UFRGS	Universidade Federal do Rio Grande do Sul
USB	Universal Serial Bus
VCEG	Video Coding Experts Group

VCL	Video Coding Layer
VGA	Video Graphics Array
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
XUP	Xilinx University Program
YCbCr	Luminance, Chrominance Blue, Chrominance Red
WP	Weighted Prediction

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O padrão H.264 foi desenvolvido pelo JVT, que foi formado a partir de uma união entre os especialistas do VCEG da ITU-T e do MPEG da ISO/IEC. O padrão H.264 atingiu seu objetivo de alcançar as mais elevadas taxas de processamento dentre todos os padrões existentes, mas à custa de um grande aumento na complexidade computacional. Este aumento de complexidade impede, pelo menos na tecnologia atual, a utilização de codecs H.264 implementados em software, quando se deseja a decodificação de vídeos de alta definição em tempo real.

Essa dissertação propõe uma solução arquitetural de hardware, denominada MoCHA, para compensação de movimento do decodificador de vídeo de alta definição, segundo o padrão H.264/AVC. A MoCHA está dividida em três blocos principais, a predição dos vetores de movimento, o acesso à memória e o processamento de amostras.

A utilização de uma cache para explorar a redundância dos dados nos acessos à memória, em conjunto com melhorias propostas, alcançou economia de acessos à memória superior a 60%, para os casos testados. Quando uma penalidade de um ciclo por troca de linha de memória é imposta, a economia de ciclos de acesso supera os 75%.

No processamento de amostras, a arquitetura realiza o processamento dos dois blocos, que dão origem ao bloco bi-preditivo, de forma serial. Dessa forma, são economizados recursos de hardware, uma vez que a duplicação da estrutura de processamento não é requerida.

A arquitetura foi validada a partir de simulações, utilizando entradas extraídas de seqüências codificadas. Os dados extraídos, salvos em arquivos, serviam de entrada para a simulação. Os resultados da simulação foram salvos em arquivos e comparados com os resultados extraídos.

O processador de amostras do compensador de movimento foi prototipado na placa XUP Virtex-II Pro. A placa possui um FPGA VP30 da família Virtex-II PRO da Xilinx. O processador PowerPC 405, presente no dispositivo, foi usado para implementar um *test bench* para validar a operação do processador de amostras mapeado para o FPGA.

O compensador de movimento para o decodificador de vídeo H.264 foi descrito em VHDL, num total de 30 arquivos e cerca de 13.500 linhas de código. A descrição foi sintetizada pelo sintetizador Syplify Pro da Symplicity para o dispositivo XC2VP30-7 da Xilinx, consumindo 8.465 slices, 5.671 registradores, 10.835 LUTs, 21 blocos de memória interna e 12 multiplicadores. A latência mínima para processar um macrobloco é de 233 ciclos, enquanto a máxima é de 590, sem considerar misses na cache. A freqüência máxima de operação foi de 100,5 MHz. A arquitetura projetada é capaz de processar, no pior caso, 36,7 quadros HDTV de 1080 por 1920, inteiramente bi-preditivos, por segundo. Para quadros do tipo P, que não utilizam a bi-predição, a capacidade de processamento sobe para 64,3 quadros por segundo.

A arquitetura apresentada para o processamento de quadros bi-preditivos e a hierarquia de memória são, até o momento, inéditas na literatura. Os trabalhos relativos a decodificadores completos não apresentam a solução para esse processamento.

Os resultados apresentados tornam a MoCHA uma solução arquitetural capaz de fazer parte de um decodificador para vídeos de alta definição.

Palavras-chave: H.264, descompressão de vídeo, compensação de movimento, vídeo de alta definição, processamento de vídeo.

MoCHA: Dedicated Architecture for H.264 High Definition Video Decoder Motion Compensation

ABSTRACT

H.264 is a standard developed by JVT, a team of specialists from VCEG - ITU-T and MPEG-ISO/IEC. This standard reached his main objective: the highest processing rates among the others. This standard increases the computational complexity, however. This higher complexity taxes the use of H.264 codecs implemented in software, at least in the current technology, when the application expects high definition video decoding in real time.

This work presents a hardware architecture, called MoCHA, for high definition video decoder motion compensator for the H.264/AVC. MoCHA is divided in three main blocks: motion vectors prediction, memory access and sample processing.

A cache is used to explore the data redundancy in memory access. This technique, combined with others that we proposed, reduced the memory access up to 60% for all cases tested. When a penalty of one cicle is imposed for changing memory line, the saving of access cycles is higher than 75%.

In the sample processing, the architecture processes two blocks to generate the bi-predictive block, in serial mode. This method spends less hardware resources because it does not require the duplicity of the processing structure.

The architecture was validated from simulations with inputs extracted from software coded sequences. The extracted data, saved in files, have been used as inputs for simulations. The results of simulation have been saved in files and have been compared with the extracted data.

The sample processing has been prototyped in the XUP Virtex-II Pro card, which has an FPGA VP30 of Virtex-II PRO family from Xilinxs. The PowerPC 405 processor, inserted in this card, has been used to implement a test bench for the validation.

The H.264 video decoder motion compensator, developed in VHDL, has 30 files and about 13.500 code lines. The Synplify Pro from Synplicity synthesizes the description for the Xilinx XC2VP30-7 device. The synthesis consumes 8.465 slices, 5671 registers, 10835 LUTs, 21 internal memory blocks and 12 multipliers. The minimum latency is 233 cycles to process one macro block, and the maximum is 590, without cache misses. The maximum frequency is 100,5 MHz. The architecture proposed is able to process 36.7 frames per second of 1080 x 1920 HDTV, totally bi-predictive in the worst case. With P frames only, which do not use bi-prediction, the processing capability increases to 64.3 frames per second.

The architecture presented for bi-predictive frames processing and hierarchical memory is unpublished in the literature, at this moment. The related works dealing with complete decoders do not present a solution for this problem.

The results make the MoCHA an architecture that is able to integrate high definition video decoders.

Keywords: H.264, motion compensation, video decoding, high definition video, video processing.

1 INTRODUÇÃO

A codificação digital de vídeo é a solução para a transmissão e armazenamento de vídeo de alta definição. O padrão de codificação de vídeo MPEG-2 (ITU-T, 1994), que foi desenvolvido há mais de 10 anos, foi a tecnologia que tornou possível os sistemas de televisão digital ao redor do mundo (WIEGAND et al., 2003). Esse padrão é amplamente utilizado na transmissão de sinais de TV, sendo utilizado nos padrões de TV digital atualmente em uso, e no armazenamento de vídeo digital, como é o caso dos DVDs.

Assim como o MPEG-2 foi a tecnologia que possibilitou os sistemas de TV digital, o H.264 pode ser a tecnologia capaz de difundir a televisão em alta definição (HDTV - High Definition Television). O padrão H.264 foi desenvolvido pelo JVT (ITU-T, 2005), que foi formado a partir da união entre os especialistas do VCEG da ITU-T (THE ITU TELECOMMUNICATION STANDARDIZATION SECTOR , ITU-T) e do MPEG da ISO/IEC (ISO - INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2005). A primeira versão do H.264 foi aprovada em 2003. O padrão H.264 atingiu seu objetivo de alcançar as mais elevadas taxas de processamento dentre todos os padrões existentes, com uma redução de 50% na quantidade de bytes do resultado, em relação ao MPEG-2 e ao MPEG-4 Simple Profile. Para isso, foi necessário um grande aumento na complexidade computacional das operações dos codificadores e decodificadores (*codecs*) que seguem o padrão H.264, em relação aos demais padrões disponíveis na atualidade. Este aumento de complexidade impede, pelo menos na tecnologia atual, a utilização de codecs H.264 implementados em software, quando se deseja a decodificação de vídeos de alta definição em tempo real.

A codificação e decodificação do padrão H.264, devido ao seu grande interesse comercial, é alvo de vários estudos, tanto por instituições acadêmicas quanto por empresas privadas. Otimizações algorítmicas do padrão e seu mapeamento em hardware são alvos desses estudos. Essas otimizações são especialmente necessárias quando se trata de vídeo de alta definição, devido a grande massa de dados envolvida no processo.

Este trabalho é parte do esforço de pesquisa para auxiliar na definição do modelo do padrão brasileiro de televisão digital (SBDTV), com suporte a HDTV. Esse trabalho faz parte do estudo da viabilidade de se adotar o padrão de codificação de vídeo H.264 como o padrão de compressão de vídeo. Esse estudo de viabilidade passa por desenvolver um protótipo do codificador e do decodificador de vídeo em hardware que atenda os requisitos de tempo real, para vídeos de alta definição.

O objetivo dessa dissertação é apresentar uma solução arquitetural de hardware para a compensação de movimento do decodificador de vídeo segundo o padrão H.264/AVC. A arquitetura deve ser capaz de processar, em tempo real, vídeo de alta definição a 30 Hz. O formato de alta definição adotado nesse trabalho se refere ao 1080HD, cujo quadro possui 1080 linhas e 1920 colunas de pixels. Devido a natureza das aplicações alvo, a arquitetura

deve tratar seqüências no perfil (regras de conformidade) Main no nível 4.0.

A solução arquitetural tem como plataforma de desenvolvimento e prototipação os dispositivos FPGA. Esses dispositivos foram escolhidos devido ao seu baixo custo, quando comparados a soluções VLSI, permitindo, ao final do projeto, um protótipo funcional.

O capítulo dois desta dissertação apresenta alguns conceitos básicos de compressão de vídeo e uma introdução ao padrão H.264, que serão úteis no decorrer do texto e que servem para contextualizar o trabalho. O capítulo três apresenta o bloco da compensação de movimento, alvo deste trabalho, as ferramentas ligadas e as características desse bloco. Esse capítulo também apresenta o estado da arte em relação a soluções arquiteturais, presentes na literatura, para o compensador de movimento bem como para decodificadores completos. A solução arquitetural desenvolvida é apresentada no capítulo 4. Nesse capítulo são detalhados os requisitos, as restrições arquiteturais, a metodologia e a arquitetura da solução proposta. Também são apresentados os resultados alcançados com a solução e uma comparação com o estado da arte. O capítulo 5 apresenta a estratégia de validação e os procedimentos de prototipação de parte da arquitetura. Por fim, o capítulo 6 apresenta as conclusões dessa dissertação.

2 VÍDEO DIGITAL E O PADRÃO DE COMPRESSÃO DE VÍDEO H.264

Neste capítulo serão abordados os conceitos básicos sobre vídeo digital e, em especial, sobre o padrão de compactação de vídeo H.264/AVC.

Na próxima seção será dada uma visão geral sobre os processos de captura e formatação da representação digital de vídeo. A seção 2.2 aborda os tipos de redundância explorados para se obter a compressão de vídeo digital. Um modelo genérico do mecanismo da compressão de vídeo digital será apresentado na seção 2.3. A seção 2.4 define os conceitos básicos relacionados ao padrão de compressão de vídeo H.264/AVC e a 2.5 o seu processo de codificação e decodificação.

2.1 Conceitos sobre vídeo digital

Esta seção aborda os conceitos básicos sobre vídeo digital, tais como os processos de captura, amostragem espacial e temporal, espaço e sub-amostragem de cores e, por fim, a representação da cena em quadros e campos.

2.1.1 Captura

Uma cena visual natural é contínua no tempo e no espaço. A representação digital de uma cena visual envolve amostragem espacial e temporal. A amostragem espacial é realizada, normalmente, como uma matriz retangular na imagem do vídeo. A amostragem temporal é realizada como uma série de quadros, ou componentes de quadro, estáticos, amostrados a certos intervalos de tempo, conforme esquematizado na Figura 2.1. Vídeo digital é a representação de uma cena de vídeo amostrada de forma digital. Cada amostra temporal-espacial, chamada elemento de figura (picture element ou pixel), é representada como um número, ou um conjunto de números, que representa as componentes de cor da amostra, ou o brilho e cor, de acordo com o espaço de cor utilizado.

Para se obter uma imagem amostrada bidimensionalmente, a câmera foca uma projeção bidimensional da cena de vídeo num sensor. Para o caso de captura de imagens coloridas, cada componente de cor é separada e projetada num sensor.

2.1.2 Amostragem espacial

A saída do sensor é um sinal de vídeo analógico, uma variação do sinal elétrico que representa uma imagem do vídeo. Amostrando o sinal em intervalos de tempo se produz uma imagem amostrada, ou quadro, que tem valores definidos em um conjunto de pontos de amostragem. A forma mais comum da imagem amostrada é um retângulo com os pontos de amostragem posicionados numa malha quadrada ou retangular. A amos-

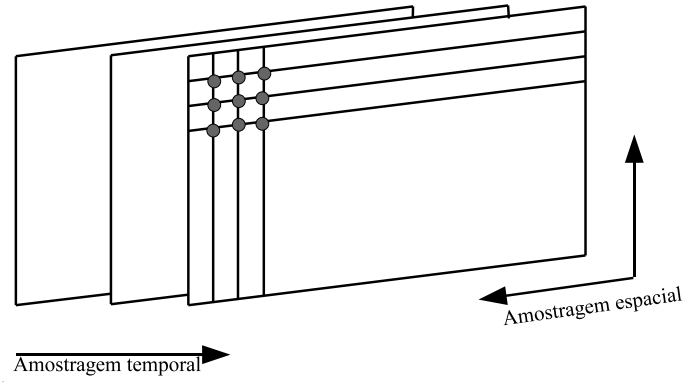

Figura 2.1: Amostragem espacial e temporal de uma cena

tragem ocorre em cada intersecção das linhas da malha e a imagem amostrada pode ser reconstruída representando cada amostra como um elemento de figura quadrado. A qualidade da imagem é influenciada pela quantidade de pontos de amostragem. Quanto mais espaçados são os pontos de amostragem menor a resolução da imagem amostrada, assim como quanto mais próximos são os pontos de amostragem maior é a resolução da imagem amostrada.

2.1.3 Amostragem temporal

Cada quadro de um vídeo é capturado por um sistema eletrônico complexo de sensores, amostradores e conversores, em intervalos regulares de tempo. Apresentando a série de quadros, produz-se a sensação de movimento. Quanto mais alta é a freqüência em que os quadros são "fotografados" (*frame rate*) a sensação de movimento torna-se mais suave, todavia requer uma maior quantidade de amostras a serem capturadas e armazenadas. O padrão de freqüência de amostragem das TVs fica entre 25 e 30 quadros por segundo. Taxas entre 10 e 20 quadros por segundo são utilizadas para comunicação de vídeos em canais de transmissão que suportam baixas taxas de transmissão, taxas de 50 ou 60 quadros por segundo são utilizadas para vídeos de alta qualidade.

2.1.4 Espaço de Cores e Sub-Amostragem de Cores

A representação digital de um vídeo colorido está associada à interpretação das cores pelo sistema visual humano. O sistema humano de visão possui elementos sensíveis à luz chamados bastonetes e cones. Os bastonetes são sensíveis à intensidade luminosa, enquanto os cones são sensíveis às cores primárias (GONZALEZ; WOODS, 2003). Em função desta estrutura do sistema visual humano, todas as cores são vistas como combinações variáveis das três cores primárias: vermelho (R - *red*), verde (G - *green*) e azul (B - *blue*). O sistema visual humano é capaz de discernir milhares de cores distintas a partir de combinações de intensidades distintas das cores primárias. Por outro lado, o sistema visual humano não consegue distinguir mais do que duas dúzias de tons de cinza, que, na verdade, indicam a intensidade luminosa da imagem (GONZALEZ; WOODS, 2003).

Existem muitas formas de se representar as cores de forma digital. Um sistema para representar cores é chamado de espaço de cores e a definição do espaço de cor a ser utilizado para representar um vídeo é essencial para a eficiência da codificação deste vídeo.

São vários os espaços de cores usados para representar imagens digitais, tais como: RGB, HSI e YCbCr (SHI; SUN, 1999). O espaço de cores RGB é um dos mais comuns,

tendo em vista que é este o espaço de cores utilizado na codificação da maior parte dos monitores coloridos. O RGB representa, em três matrizes distintas, as três cores primárias captadas pelo sistema visual humano: vermelho, verde e azul. Por isso o nome deste espaço de cores (do inglês *red*, *green*, *blue* - RGB). No espaço de cores YCbCr, as três componentes utilizadas são luminância (Y), que define a intensidade luminosa ou o brilho; crominância azul (Cb) e crominância vermelha (Cr) (MIANO, 1999).

Os componentes R, G e B possuem um elevado grau de correlação, tornando difícil o processamento de cada uma das informações de cor de forma independente, por isso, a compressão de vídeos é aplicada para espaços de cores do tipo luminância (luma) e crominância (croma), como o YCbCr (RICHARDSON, 2002).

Outra vantagem do espaço de cor YCbCr sobre o espaço RGB é que no espaço YCbCr a informação de cor está completamente separada da informação de brilho. Deste modo, estas informações podem ser tratadas de forma diferenciada pelos compressores de imagens estáticas e vídeos.

O sistema visual humano possui cerca de 240 milhões de bastonetes e 13 milhões de cones (GONZALEZ; WOODS, 2003). Deste modo, o sistema visual humano é muito mais sensível a informações de luminância do que a informações de crominância. Então, os padrões de compressão de imagens estáticas e vídeos podem explorar esta característica psicovisual humana para aumentar a eficiência de codificação, através da redução da taxa de amostragem dos componentes de crominância em relação aos componentes de luminância (RICHARDSON, 2002). Esta operação é chamada de sub-amostragem de cores e é realizada sob o espaço de cores YCbCr nos padrões de compressão de vídeos atuais.

Existem várias formas de relacionar os componentes de crominância com o componente de luminância para realizar a sub-amostragem. Os formatos mais comuns são o 4:4:4, o 4:2:2 e o 4:2:0, onde cada número representa a amostragem de um componente. No formato 4:4:4, para cada quatro amostras de luminância (Y), existem quatro amostras de crominância azul (Cb) e quatro amostras de crominância vermelha (Cr). Por isso, os três componentes de cor possuem a mesma resolução e existe uma amostra de cada elemento de cor para cada pixel da imagem e, assim, a sub-amostragem não foi aplicada. No formato 4:2:2, para cada quatro amostras de Y na direção horizontal, existem apenas duas amostras de Cb e duas amostras de Cr. Neste caso, as amostras de crominância possuem a mesma resolução vertical das amostras de luminância, mas possuem metade da resolução horizontal. No formato 4:2:0, para cada quatro amostras de Y, existe apenas uma amostra de Cb e uma amostra de Cr. Neste caso, as amostras de crominância possuem metade da resolução horizontal e metade da resolução vertical do que as amostras de luminância. A nomenclatura 4:2:0 é usada por motivos históricos, pois os números não representam a relação lógica entre os componentes de cor, que seria 4:1:1 (RICHARDSON, 2003).

A sub-amostragem de cor aumenta significativamente a eficiência da compressão, uma vez que parte da informação da imagem é simplesmente descartada, sem causar impacto visual perceptível. Considerando o formato 4:2:0 como exemplo, uma vez que cada componente de crominância possui exatamente um quarto das amostras presentes no componente de luminância, então um vídeo YCbCr no formato 4:2:0 irá utilizar exatamente a metade das amostras necessárias para um vídeo RGB ou YCbCr no formato 4:4:4. Isso implica em uma taxa de compressão de 50%, considerando apenas a contribuição da sub-amostragem.

O padrão H.264, foco deste trabalho, considera que os dados do vídeo de entrada estão codificados segundo a convenção de espaço de cores YCbCr.

2.1.5 Quadros e Campos

Um sinal de vídeo pode ser amostrado como uma série de quadros (*frames - progressive sampling*) ou como uma seqüência de campos entrelaçados (*interlaced sampling*). Numa seqüência de vídeo entrelaçado, metade dos dados de um quadro (um campo) é amostrada a cada intervalo temporal de amostragem, Figura 2.2. Um campo consiste ou de linhas pares ou de linhas ímpares de um quadro de uma seqüência inteira de quadros. Uma seqüência de vídeo entrelaçado contém uma série de campos, cada um representando metade da informação de um quadro de vídeo. A vantagem desse tipo de amostragem é que é possível enviar o dobro da quantidade de campos por segundo do que o número de quadros de uma seqüência progressiva com o mesmo *data rate*, dando a impressão de movimentos mais suaves.

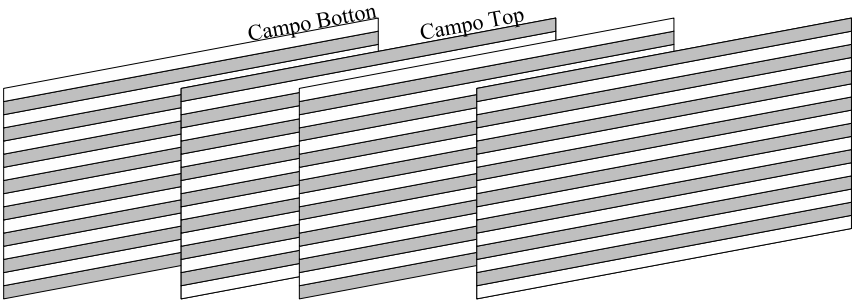

Figura 2.2: Campos do vídeo entrelaçado

2.2 Redundância de Dados na Representação de Vídeos

A compressão de vídeos busca diminuir a quantidade de dados considerados redundantes na representação computacional das informações existentes na imagem ou vídeo. Considera-se redundante aquele dado que não contribui com novas informações relevantes para a representação da imagem. Basicamente, existem quatro tipos diferentes de redundâncias exploradas na compressão de vídeos: redundância espacial, redundância temporal, redundância psicovisual e redundância entrópica. Existem controvérsias entre os autores a respeito da definição dos tipos de redundância. A classificação apresentada neste trabalho, com quatro diferentes tipos de redundância, é baseada em (SHI; SUN, 1999) e (GONZALEZ; WOODS, 2003). Cada uma destas redundâncias será resumidamente explicada nos próximos parágrafos e, na próxima seção do texto, será explicado como cada redundância é tratada em um codec de vídeo genérico.

- Redundância Espacial - A redundância espacial advém da correlação existente entre os pixels espacialmente distribuídos em um quadro. Pixels vizinhos tendem a possuir valores semelhantes e, por conseqüência, com elevado grau de redundância de informação. A redundância espacial é também chamada de "redundância intra-quadro (*intraframe*)" (GHANBARI, 2003) "redundância interpixel" ou "redundância geométrica" (GONZALEZ; WOODS, 2003). Os métodos de codificação que exploram a redundância espacial serão definidos como "codificação intra-quadro" neste trabalho.

- Redundância Temporal - A redundância temporal, também chamada de "redundância inter-quadro ((interframe)" (GHANBARI, 2003), é causada pela correlação existente entre quadros temporalmente próximos em um vídeo. Na verdade, a redundância temporal poderia ser classificada como apenas mais uma dimensão da redundância espacial, como faz (GONZALEZ; WOODS, 2003). Muitos blocos de pixels simplesmente não mudam de valor de um quadro para outro em um vídeo, como por exemplo, em um fundo que não foi alterado de um quadro para outro. Outros pixels apresentam uma pequena variação de valores causada, por exemplo, por uma variação de iluminação. Por fim, também é possível que o bloco de pixels simplesmente tenha se deslocado de um quadro para o outro, como por exemplo, em um movimento de um objeto em uma cena. Todos os padrões de codificação de vídeo atuais visam eliminar ou diminuir a redundância temporal. A exploração eficiente da redundância temporal conduz a elevadas taxas de compressão e é fundamental para o sucesso dos codificadores. Esse método é chamado de "codificação inter-quadro" neste trabalho.

- Redundância Psicovisual - As características do sistema visual humano fazem com que não consigamos captar alguns tipos de informações existentes na imagem. Além disso, algumas informações da imagem, como o brilho, por exemplo, são mais importantes para a percepção do sistema visual humano do que outras, como a crominância. Este tipo de redundância de informação é chamado de redundância psicovisual (GONZALEZ; WOODS, 2003). Para explorar este tipo de redundância, parte da informação original da imagem é eliminada de forma irreversível pelo codificador. Existem dois tipos principais de processos utilizados para este fim. O primeiro, chamado de "sub-amostragem" (que foi definida na seção anterior), é utilizado na entrada do vídeo e elimina parte das informações de cores. O segundo, conhecido por "quantização", normalmente é aplicado no domínio das freqüências e elimina ou atenua as freqüências de menor importância para o sistema visual humano. Por isso, este tipo de processo de codificação é chamado de "codificação com perdas". É importante destacar que a eliminação destas informações contribui para que o codificador atinja elevadas taxas de compressão, com pequeno impacto na qualidade visual da imagem que, eventualmente, pode mesmo ser nulo.

- Redundância Entrópica - A redundância entrópica está relacionada com a forma de representação computacional dos símbolos codificados e não se relaciona diretamente ao conteúdo da imagem. A entropia é uma medida da quantidade média de informação transmitida por símbolo do vídeo (SHI; SUN, 1999). A quantidade de informação nova transmitida por um símbolo diminui na medida em que a probabilidade de ocorrência deste símbolo aumenta. Então, os codificadores que exploram a redundância entrópica têm por objetivo transmitir o máximo de informação possível por símbolo codificado e, deste modo, representar mais informações com um número menor de símbolos. A "codificação de entropia", como é chamada, utiliza diferentes técnicas e algoritmos de compressão sem perdas para atingir este objetivo.

Alguns autores classificam as redundâncias apenas como temporal, espacial e entrópica (RICHARDSON, 2003) (BHASKARAN; KOSTANTINIDES, 1997) (GHANBARI, 2003) e não consideram a redundância psicovisual como uma categoria separada de redundância. Nestes casos, a quantização é classificada como uma operação para reduzir a redundância espacial.

2.3 Compressão/Descompressão de vídeo

Antes de entrar em maiores detalhes sobre o padrão H.264 propriamente dito, serão definidos um modelo simplificado de compressor e de descompressor de vídeo, com base na eliminação das redundâncias relacionadas na seção anterior. Este modelo tem como base os padrões atuais de compressão de vídeo e tem o intuito de localizar com mais clareza cada uma das principais operações realizadas em um codificador ou decodificador de vídeo.

Para os modelos apresentados nesta seção, são considerados apenas dois tipos de quadros. Os quadros do tipo I são quadros utilizados no início do vídeo e para sincronizar o sinal de vídeo. A codificação deste tipo de quadro não depende de quadros anteriores e é realizada considerando apenas as informações contidas no próprio quadro. Deste modo, nos quadros do tipo I a redundância temporal não é explorada. Os quadros tipo P são os quadros mais utilizados e são construídos com base em quadros I ou P previamente codificados. Os quadros tipo P, nos modelos propostos, não exploram a redundância espacial. Deste modo, as redundâncias espacial e temporal nunca são exploradas em um mesmo quadro nos modelos simplificados apresentados a seguir.

A Figura 2.3 apresenta um modelo simplificado de um compressor. Existem, neste modelo simplificado, dois quadros do vídeo sendo utilizados simultaneamente. Além do quadro atual, que está sendo comprimido, também é utilizado um quadro de referência anteriormente processado. Os quadros são divididos em diversas partes, normalmente chamadas de macroblocos, para serem processados. Os macroblocos do quadro atual são codificados através da codificação inter-quadro ou através da codificação intra-quadro. A chave seletora na Figura 2.3 representa a decisão de qual modo de codificação deve ser utilizado para cada quadro.

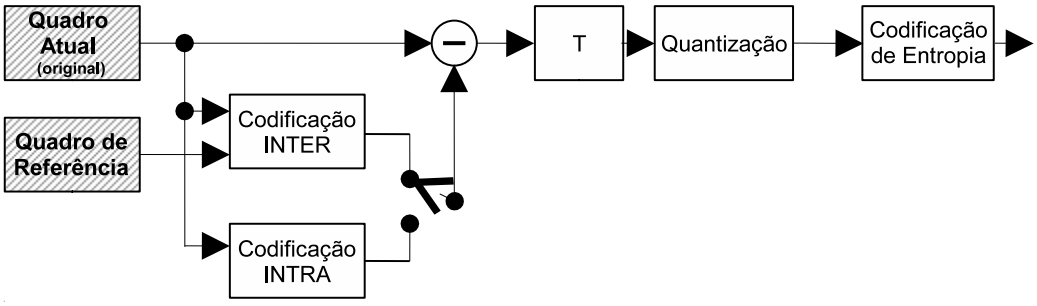

Figura 2.3: Modelo inicial de compressor de vídeo

O bloco de Codificação Inter-quadro é responsável por reduzir a redundância temporal, através da comparação dos macroblocos do quadro atual com os macroblocos do quadro de referência. A área do quadro de referência com maior semelhança ao macrobloco que está sendo processado do quadro atual é escolhida. Então um vetor de movimento é gerado, para identificar a posição desta área no quadro de referência. A codificação inter-quadro é utilizada apenas para quadros do tipo P.

O bloco de Codificação Intra-quadro é responsável por reduzir a redundância espacial, utilizando, para tanto, apenas a informação do quadro atual em processamento. Vários algoritmos podem ser utilizados para este fim, notadamente, os algoritmos utilizados em compressão de imagens estáticas podem ser utilizados na Codificação Intra-quadro. A codificação intra-quadro é utilizada apenas para quadros do tipo I.

A diferença residual após a codificação intra-quadro ou inter-quadro é obtida através

de uma subtração dos valores dos macroblocos do quadro atual e dos valores gerados por estas codificações. Esta diferença é chamada de resíduo.

O resíduo, então, é enviado para os blocos responsáveis por reduzir a redundância psicovisual. A primeira operação nesta direção é a transformada (bloco T na Figura 2.3). O bloco T transforma a informação do domínio espacial para o domínio das freqüências. Neste domínio, a Quantização pode ser aplicada de maneira mais eficaz, eliminando as freqüências menos relevantes ao olho humano, reduzindo, assim, a redundância psicovisual.

Por fim, a Codificação de Entropia reduz a redundância entrópica, que está relacionada à forma como os dados são codificados, em forma binária.

O modelo apresentado na Figura 2.3 atua na redução dos quatro tipos de redundância apresentados na seção anterior. Mas este modelo ainda não está completo. É possível observar que o codificador não armazenou o resultado codificado do quadro atual. Deste modo, quando um próximo quadro é processado, o quadro atual não codificado passa a ser o quadro de referência. Como apenas o quadro codificado é transmitido, o decodificador não receberá o quadro original, recebendo apenas o quadro codificado. Como a codificação gera perdas de informação, o quadro codificado será diferente do quadro original após a decodificação (RICHARDSON, 2003). Então, as referências utilizadas pelo codificador e pelo decodificador serão diferentes para o processamento de um mesmo quadro. Esta diferença na referência utilizada gera uma diferença entre o quadro que está sendo codificado e o que será reconstruído pela decodificação. Para resolver este problema, o codificador descarta o quadro original depois de ser processado, e armazena o quadro reconstruído. A informação reconstruída é relevante tanto para a codificação inter-quadro quanto para a codificação intra-quadro. Na codificação inter-quadro, o quadro codificado é usado como quadro de referência para a codificação do próximo quadro. A codificação intra-quadro, em alguns padrões, utiliza as amostras dos macroblocos vizinhos que acabaram de ser codificados como referência para a codificação dos próximos macroblocos. Em ambos os casos, a referência do codificador e do decodificador devem ser as mesmas. Para isso ser possível, parte da tarefa da decodificação é inserida também no codificador. A Figura 2.4 apresenta o novo diagrama de blocos do codificador, agora considerando a reconstrução da imagem.

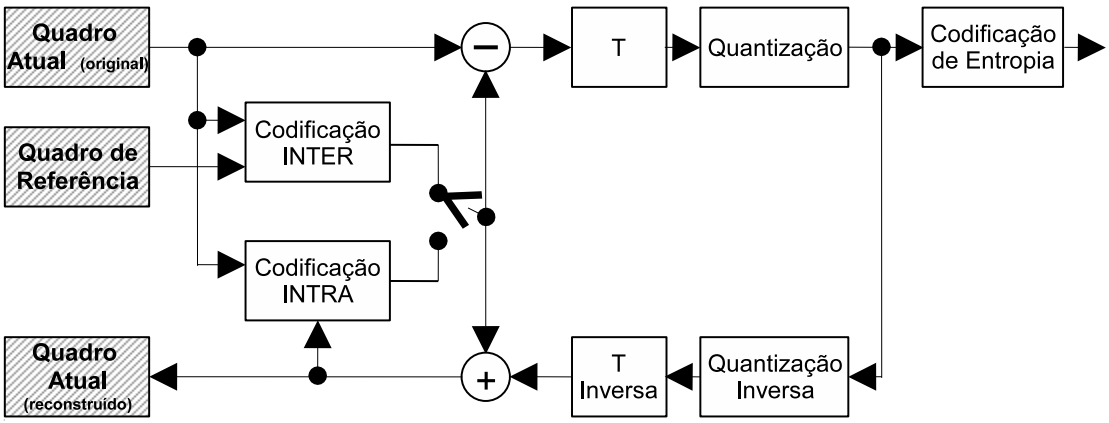

Figura 2.4: Modelo completo de compressor de vídeo

Como as perdas de informação, que geram a diferença entre o quadro atual original e o quadro atual reconstruído, acontecem no estágio de quantização, a imagem deve ser reconstruída a partir deste ponto. Deste modo, é aplicada a operação inversa da quanti-

zação. Como a quantização é uma operação irreversível, que gera perdas, alguns autores sustentam que é impossível realizar a quantização inversa, preferindo se referir a esta operação como *rescale* (RICHARDSON, 2003). Após a quantização inversa, é aplicada a transformada inversa, gerando os resíduos reconstruídos. Então, aos resíduos é somado o resultado da codificação intra-quadro ou inter-quadro do macrobloco reconstruído. Finalmente o macrobloco reconstruído está pronto e pode ser armazenado para ser utilizado pela codificação inter-quadro do próximo quadro ou pode ser utilizado diretamente pela codificação intra-quadro dos próximos macroblocos do quadro atual.

O modelo de descompressor está apresentado na Figura 2.5. Como pode ser observado a partir de uma comparação da Figura 2.5 com a Figura 2.4, a decodificação é muito parecida com a parte da reconstrução do quadro da codificação.

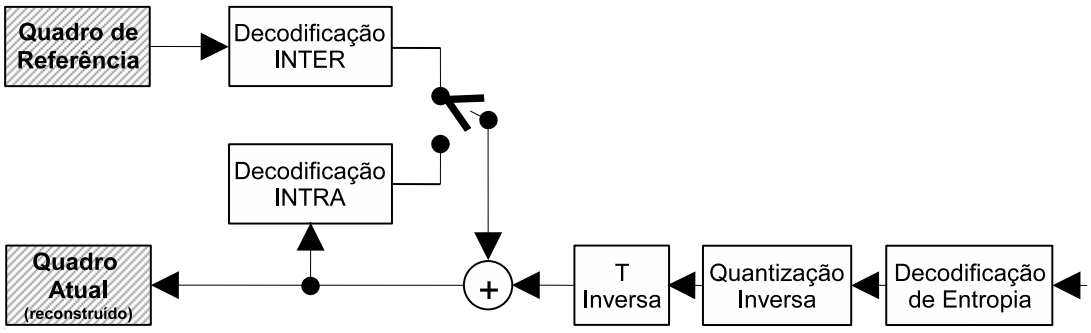

Figura 2.5: Diagrama de um descompressor de vídeo

A decodificação de entropia remonta as amostras codificadas. Como a codificação de entropia não gera perdas de informação, a operação de decodificação de entropia gera exatamente o mesmo resultado gerado pela saída da quantização no codificador.

Após a decodificação de entropia, as amostras remontadas são entregues para a quantização inversa, ou *rescale*, e para a transformada inversa, reconstruindo o resíduo. Estas operações são idênticas às apresentadas no codificador.

A decodificação inter-quadro utiliza o quadro de referência e as informações de controle (não apresentadas na Figura 2.5), para localizar os macroblocos dentro do quadro de referência. Estes macroblocos são, então, somados aos resíduos para gerar os macroblocos reconstruídos. Finalmente, os macroblocos reconstruídos são armazenados para serem usados como referência para a próxima decodificação e para a visualização do vídeo decodificado.

A decodificação intra-quadro utiliza os macroblocos reconstruídos do quadro atual e informações de controle (omitidas na Figura 2.5), para reconstruir os macroblocos codificados pelo codificador inter-quadro. De maneira similar a codificação inter-quadro, os macroblocos que passam pela decodificação intra-quadro são somados aos resíduos e armazenados para serem usados como referência e para visualização do vídeo.

É importante salientar que, nos modelos apresentados nas Figuras 2.4 e 2.5, não estão apresentados os fluxos de sinais de controle e de elementos sintáticos.

2.4 Introdução ao padrão de codificação de vídeo H.264/AVC

O padrão H.264, também conhecido como MPEG - 4 parte 10, foi desenvolvido, de forma colaborativa, pelo grupo de expertos em codificação de vídeo (CVEG - *Video Coding Experts Group*) da ITU-T e pelo grupo de expertos em imagem em movimento

(MPEG - *Moving Picture Experts Group*) da ISO/IEC. A essa força conjunta foi dado o nome de time unido de vídeo (JVT - *Joint Video Team*). O padrão H.264 descreve os formatos dos dados, suas sintaxe e semânticas, os processos para a decodificação de vídeo e o processo de parsing. A norma não trata do processo de codificação. A norma tem mais de 250 páginas e foi, nas palavras de Gary J. Sullivan, *chairman* do JVT, "escrita primeiramente para ser precisa, consistente, completa e correta e não para ser particularmente legível" (RICHARDSON, 2003, pg XVII).

Além do texto da norma, foi colocado no domínio público o software de referência, o qual tem a função de ser uma prova de conceito apenas. O software de referência não prima pelo desempenho, sendo desenvolvido tendo em vista apenas a corretude na implementação do padrão. O mesmo também foi desenvolvido de forma colaborativa.

Esta seção pretende apresentar uma introdução ao padrão H.264 de compressão de vídeo. Nesta seção são brevemente apresentados a estrutura básica, os diferentes perfis do padrão, o conceito de níveis e, por fim, o formato dos dados codificados.

2.4.1 Estrutura de codificação

Para o padrão H.264 a formação de uma imagem codificada pode acontecer a partir de um quadro, quando o vídeo é progressivo ou entrelaçado, ou de um campo, quando o vídeo é entrelaçado (RICHARDSON, 2003).

Um quadro codificado possui um número de quadro, que é sinalizado no fluxo de dados (*bitstream*). Este número de quadro não está, necessariamente, relacionado à ordem de decodificação deste quadro. Cada campo codificado de um quadro progressivo ou entrelaçado possui um contador que indica a sua ordem de decodificação. Os quadros codificados anteriormente podem ser utilizados como quadros de referência para a predição de outros quadros. Os quadros de referência são organizados em duas listas, chamadas de lista 0 e lista 1.

O padrão H.264 considera que o espaço de cores utilizado para representar o vídeo de entrada é do tipo luminância e crominância, como o YCbCr. A relação entre os elementos Y, Cb e Cr é dependente do perfil do padrão que está sendo considerado. Esta relação e os perfis do padrão serão apresentados ainda nesta seção.

Uma figura codificada consiste de um número de macroblocos, cada um contendo 16x16 amostras de luminância, associadas às amostras de crominância. Dependendo do perfil utilizado, o tamanho das amostras de crominância de um macrobloco pode variar, como ficará mais claro nos próximos parágrafos. Um macrobloco pode ainda ser dividido em blocos com 4x4 amostras de luma ou de croma.

Dentro de cada quadro, os macroblocos são organizados em *slices*, onde um *slice* é um grupo de macroblocos em uma ordem de varredura tipo *raster*. Um *slice* do tipo I pode conter somente macroblocos do tipo I. Um *slice* do tipo P pode conter macroblocos do tipo P e I e um *slice* do tipo B pode conter macroblocos do tipo B e I. Além dos *slices* tipo I, P e B, o padrão também permite a existência de outros dois tipos de *slices*: SI e SP. Um quadro do vídeo pode ser codificado em um ou mais *slices*, cada um contendo um número inteiro de macroblocos. O número de macroblocos num *slice* pode variar de um até o número total de macroblocos do quadro (RICHARDSON, 2003).

Os *slices* do tipo B são *slices* que permitem que a predição possa ocorrer a partir de um ou dois quadros de referência, que se encontram antes ou depois do quadro corrente, em ordem temporal. Essa possibilidade permite diversas opções para a escolha da predição mais adequada para cada macrobloco num *slice* do tipo B, permitindo maior eficiência na codificação.

Os *slices* SP (*Switching* P) e SI (*Switching* I) são *slices* que são transmitidos para permitir o chaveamento entre fluxos de bits diferentes, mas sem causar um grande prejuízo na eficiência de codificação (KARCZEWICZ; KURCEREN, 2003). É comum, em codificação de vídeo, o chaveamento entre fluxos de bits diferentes, mas este chaveamento não pode acontecer logo antes de um quadro P ou B, pois a codificação destes quadros usa como referência quadros do fluxo de bits atual. É possível fazer o chaveamento usando um quadro do tipo I, mas o problema é que estes quadros consomem muito mais bits do que os quadros P ou B. Por isso o padrão H.264 prevê a utilização de *slices* tipo SP e SI no seu perfil Extended, que será apresentado na próxima seção do texto.

Os macroblocos do tipo I são codificados usando a codificação intra-quadro a partir das amostras do *slice* atual. A codificação pode acontecer sobre macroblocos completos ou para cada bloco. Os macroblocos do tipo I também podem ser codificados através do modo I_PCM, onde o codificador transmite os valores das amostras diretamente, sem predição ou transformação. Em alguns casos anômalos, o modo I_PCM pode ser mais eficiente do que os demais modos de codificação (RICHARDSON, 2003).

Os macroblocos do tipo P são codificados usando a codificação inter-quadro, a partir de quadros de referência previamente codificados. Um macrobloco codificado no modo inter pode ser dividido em partições de macroblocos, isto é, em blocos de 16x16, 16x8, 8x16 ou 8x8. Se o tamanho de partição escolhido for o 8x8, então cada partição 8x8, chamada de sub-macrobloco, pode ser dividida novamente em partições de tamanho 8x8, 8x4, 4x8 ou 4x4. Cada partição de macrobloco é codificada utilizando como referência um quadro da lista 0. Se existir uma partição de sub-macrobloco, então esta partição é codificada utilizando o mesmo quadro da lista 0 utilizado para codificar a partição de macrobloco.

Os macroblocos do tipo B também são codificados usando a codificação inter-quadro. Cada partição de macrobloco pode ser codificada utilizando um ou dois quadros de referência, um na lista 0 e outro na lista 1. Se existir uma partição em sub-macrobloco, cada sub-macrobloco é codificado a partir dos mesmos um ou dois quadros de referência utilizados na codificação da partição de macrobloco.

Além dos macroblocos apresentados acima, o padrão H.264, tal qual os demais padrões de decodificação de vídeo, apresenta um tipo de macrobloco chamando *skip*. Macroblocos *skip* são macroblocos aos quais não é codificada nenhuma informação, nem mesmo o resíduo ou o vetor de movimento. Macroblocos do tipo *skip* são gerados quando o custo da taxa de distorção para codificar o macrobloco é mais elevado do que o custo de não enviar informação alguma sobre o macrobloco (KANNANGARA; RICHARDSON, 2005). Os macroblocos *skip* são gerados pela codificação inter-quadro.

2.4.2 Perfis e Níveis

O padrão H.264 é dividido em diferentes perfis. Cada perfil suporta um grupo particular de funções de codificação e especifica o que é necessário para cada codificador e decodificador que seguem este perfil. A primeira versão do padrão H.264, de maio de 2003 (ITU-T, 2003), define um grupo de três diferentes perfis: Baseline, Main e Extended. O perfil Baseline é direcionado a aplicações como videotelefonia, videoconferência e vídeo sem fio. O perfil Baseline suporta codificação intra e inter (usando somente *slices* I e P) e uma codificação de entropia com códigos de comprimento de palavra variável adaptativos ao contexto (CAVLC - *Context-Based Adaptive Variable Length Coding*). O perfil Main é focado na transmissão de televisão e no armazenamento de vídeo. O perfil Main inclui o suporte para vídeo entrelaçado, à codificação inter utilizando *slices* do

tipo B, predição ponderada e à codificação de entropia utilizando codificação aritmética adaptativa ao contexto (CABAC - *Context-Based Adaptive Binary Arithmetic Coding*). O perfil Extended é mais voltado para aplicações em *streaming* de vídeo e não suporta vídeo entrelaçado ou o CABAC, mas agrega modos para habilitar uma troca eficiente entre *bitstreams* codificados (através de *slices* do tipo SP e SI) e melhora a resiliência a erros (através do particionamento de dados).

Para os três perfis definidos na primeira versão do padrão, a relação entre os elementos de cores é fixa. Esta relação é de 4:2:0 para os elementos Y, Cb e Cr. Isso significa que os elementos de crominância Cb e Cr possuem metade da resolução horizontal e vertical dos elementos de luminância. Isso implica em uma sub-amostragem dos elementos de crominância já no vídeo original. Esta sub-amostragem contribui na redução da redundância psicovisual, como foi explicado na seção 2.2 do texto. Cada macrobloco, como já foi apresentado anteriormente, é formado por uma região de 16x16 pixels de um quadro do vídeo, incluindo as informações de luminância e crominância. Com a relação de 4:2:0, um macrobloco é formado por uma matriz de 16x16 amostras de luminância e por duas matrizes 8x8 amostras de crominância, uma para Cb e outra para Cr.

Os perfis Baseline, Main e Extended também possuem outra característica em comum, especificamente relacionada à quantidade de bits utilizados por amostra. Nos três padrões, são usados 8 bits por amostra.

Estes três perfis inicialmente propostos pelo padrão H.264 foram focados em vídeos de entretenimento ou de qualidade. Mas estes perfis não incluíram suporte para vídeos com resoluções mais elevadas, como as necessárias em ambientes profissionais. Para responder às exigências deste tipo de aplicação, uma continuação do projeto do JVT foi realizada para adicionar novas extensões para as capacidades do padrão original, para aplicações. Estas extensões foram chamadas de extensões para alcance de fidelidade (*fidelity range extensions* - FRExt). O FRExt produziu um grupo de quatro novos perfis chamados coletivamente de perfis High (SULLIVAN; TOPIWALA; LUTHRA, 2004).

Os perfis High possuem algumas características comuns que são inovadoras em relação aos perfis originais: suportam um tamanho de bloco adaptativo para a transformada (4x4 ou 8x8), suportam matrizes de quantização baseadas em percepção e suportam uma representação sem perdas de regiões específicas do conteúdo do vídeo (SULLIVAN; TOPIWALA; LUTHRA, 2004).

As matrizes de quantização baseadas em percepção foram usadas anteriormente no padrão MPEG-2 (ITU-T, 1994). Com esta ferramenta, o codificador pode especificar, para cada tamanho de bloco da transformada e separadamente para codificação intra e inter, um fator de escala customizado. Isso permite um ajuste da fidelidade da quantização de acordo com o modelo de sensibilidade do sistema visual humano para diferentes tipos de erros. Tipicamente, esta ferramenta não melhora a fidelidade objetiva (PSNR), mas melhora a fidelidade subjetiva, que é o critério realmente mais importante.

O perfil High (HP) inclui vídeo com 8 bits por amostra e com relação de cor 4:2:0. O perfil High 10 (Hi10P) suporta vídeo a 10 bits por amostra, também com uma relação de cor 4:2:0. O perfil High 4:2:2 (H422P) inclui suporte à sub-amostragem de cor 4:2:2 e vídeo a 10 bits por amostra. Finalmente, o perfil High 4:4:4 (H444P) dá suporte à amostragem de cor 4:4:4, vídeo com até 12 bits por amostra e, adicionalmente, suporta a codificação sem perdas de regiões do vídeo.

A Figura 2.6 apresenta resumidamente a relação entre os perfis Baseline, Main, Extended e High do padrão H.264.

Além da divisão em diversos perfis, o padrão H.264 também define 16 diferentes ní-

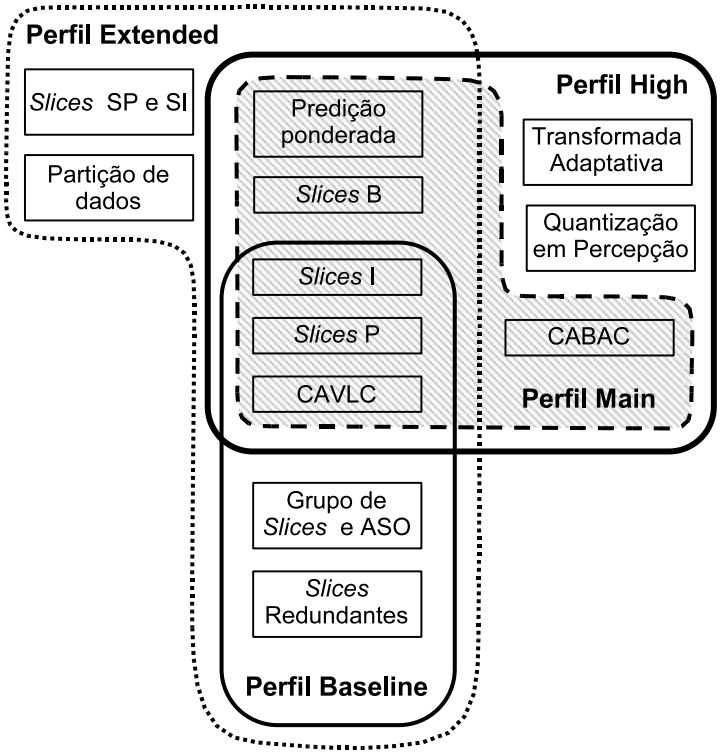

Figura 2.6: Perfis Baseline, Main, Extended e High do padrão H.264

veis em função da taxa de processamento e da quantidade de memória necessária para cada implementação. Com a definição do nível utilizado, é possível deduzir o número máximo de quadros de referência e a máxima taxa de bits que podem ser utilizados (SULLIVAN; TOPIWALA; LUTHRA, 2004).

2.4.3 Formato de Dados Codificados

Uma interessante inovação do padrão H.264 é que ele faz uma clara distinção entre a organização do vídeo codificado e da sua transmissão via rede. O padrão classifica a informação do vídeo em dois diferentes níveis de abstração, chamados de camada de vídeo codificado (VCL - *Vídeo Coding Layer*) e camada de abstração de rede (NAL - *Network Abstration Layer*) (RICHARDSON, 2003). Os dados de saída do processo de codificação estão na camada VCL, sendo formados por uma seqüência de bits que representam os dados do vídeo codificado. Estes dados são mapeados para unidades NAL antes da transmissão ou armazenamento. Uma seqüência de vídeo codificado é representada por uma seqüência de unidades NAL que podem ser transmitidas sobre uma rede baseada em pacotes, podem ser transmitidas via um link de transmissão de *bitstream* ou ainda, podem ser armazenados em um arquivo.

O objetivo de especificar separadamente a VCL e a NAL é diferenciar características específicas da codificação (na VCL) daquelas características específicas do transporte (na NAL). Neste trabalho não serão apresentados em maiores detalhes o mecanismo de transporte utilizado na NAL. Maiores detalhes podem ser encontrados na literatura (RICHARDSON, 2003; ITU-T, 2005).

2.5 Processo de codificação e decodificação no H.264/AVC

Esta seção do texto irá apresentar os principais blocos de um codificador e de um decodificador H.264, bem como suas funcionalidades.

O diagrama de blocos do codificador H.264 está apresentado na Figura 2.7. É possível perceber que a Figura 2.7 é muito parecida com aquela apresentada na seção 2.5 desta dissertação. Aquele modelo apresentado na Figura 2.4 foi apresentado para servir de base para discussão que será apresentada nesta seção do texto sobre o compressor H.264.

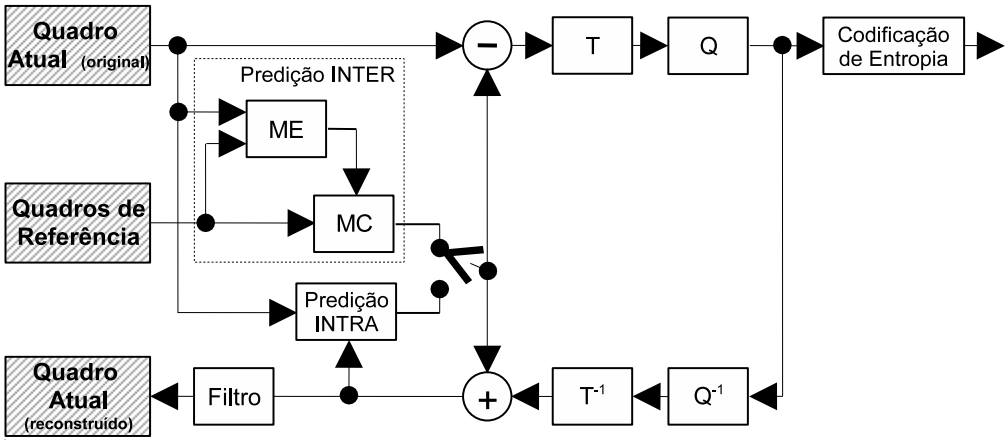

Figura 2.7: Diagrama em blocos de um codificador H.264

O bloco de Predição Inter-quadro na Figura 2.7 é formado pela estimação de movimento (ME) e pela compensação de movimento (MC). Este bloco desempenha função similar a do bloco de codificação inter-quadro da Figura 2.4, mas para desempenhar esta função o padrão H.264 agrega operações de grande complexidade computacional a este bloco. Esse bloco será detalhado no próximo capítulo dessa dissertação.

O bloco de Predição Intra-quadro, os blocos das transformadas diretas e inversas (T e T^{-1}), os blocos da quantização direta e inversa (Q e Q^{-1}) e o bloco da Codificação de Entropia na Figura 2.7 possuem a mesma função, respectivamente, que os blocos de codificação intra-quadro, das transformadas diretas e inversas (T e T^{-1}), os blocos da quantização direta e inversa e o bloco da codificação de entropia na Figura 2.4.

A única diferença mais significativa entre as Figuras 2.7 e 2.4 está no bloco chamado de Filtro na Figura 2.7, que não está presente na Figura 2.4. Este bloco foi inserido no padrão H.264 e é requisito obrigatório. Nos demais padrões de compressão de vídeo este filtro não é obrigatório, mas um filtro similar é usado na prática por várias implementações seguindo outros padrões. Este filtro, também chamado de filtro de deblocagem ou filtro de *loop*, é usado para minimizar o efeito de bloco.

A Figura 2.8 apresenta o diagrama em blocos de um decodificador H.264. Novamente esta figura é muito parecida com a Figura 2.5 apresentada na seção 2.5. Também no decodificador H.264 está presente o filtro, que não está originalmente apresentado na Figura 2.5.

Cada um dos blocos do codificador e do decodificador será brevemente descrito abaixo, de acordo com sua funcionalidade dentro do padrão H.264. Os blocos ME e MC serão detalhados no próximo capítulo dessa dissertação.

As etapas de estimação e compensação de movimento (KUHN, 1999) do padrão H.264 (bloco ME na Figura 2.7 e bloco MC nas Figuras 2.7 e 2.8) é onde se encontram a maior parte das inovações e dos ganhos obtidos pelo H.264 sobre os demais padrões.

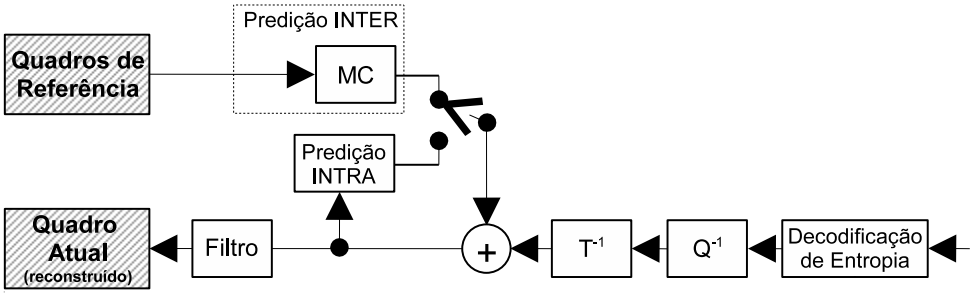

Figura 2.8: Diagrama em blocos de um decodificador H.264

Uma importante inovação é o uso de blocos de tamanho variável, onde se pode usar uma partição do macrobloco em blocos de tamanho 16x16, 16x8, 8x16, 8x8, 4x8, 8x4 ou 4x4 para realizar a ME e a MC. Outra inovação é a precisão de $\frac{1}{4}$ de pixel, para buscar o melhor casamento e para realizar a reconstrução do quadro. O H.264 também prevê o uso de múltiplos quadros de referência, que não precisam ser somente os quadros I ou P imediatamente anteriores ou posteriores. Também é uma inovação o uso de predição bi-preditiva, ponderada e direta para os *slices* do tipo B. Além disso, os vetores de movimento podem apontar para fora da borda do quadro. Por fim, a predição de vetores de movimento com base nos vetores vizinhos também é uma novidade (RICHARDSON, 2003; WIEGAND et al., 2003; ITU-T, 2003).

A etapa de predição INTRA (Figuras 2.7 e 2.8) é outra inovação introduzida pelo padrão H.264, pois mesmo nos macroblocos do tipo I é realizada uma predição antes da aplicação da transformada. Esta predição leva em conta as amostras já codificadas do *slice* atual.

No que diz respeito às transformadas direta e inversa (blocos T e T^{-1} nas Figuras 2.7 e 2.8), o H.264 introduziu algumas inovações. A primeira delas é que as dimensões da transformada foram definidas em blocos básicos de 4x4 ao invés do tradicional 8x8. Outra inovação é relativa ao uso de uma aproximação inteira da transformada DCT direta e inversa, de modo a facilitar a sua implementação em hardware de ponto fixo e evitar erros de casamento entre o codificador e o decodificador (MALVAR et al., 2003). Além disso, uma segunda transformada é aplicada sobre os elementos DC resultantes da DCT para todos os blocos de crominância (Hadamard 2x2) e para os macroblocos em que é feita a predição INTRA 16x16 (Hadamard 4x4) (RICHARDSON, 2003).

A quantização no padrão H.264 (blocos Q e Q^{-1} nas Figuras 2.7 e 2.8) é uma quantização escalar (RICHARDSON, 2002). O fator de quantização é função de um parâmetro QP de entrada, que é usado no codificador para controlar a qualidade da compressão e o *bit-rate* de saída.

O padrão H.264 normatiza a utilização de um filtro redutor do efeito de bloco (bloco Filtro nas Figuras 2.7 e 2.8). Este filtro era opcional na maioria dos demais padrões de compressão de vídeo, mas passou a ser obrigatório no padrão H.264. Uma inovação importante do filtro definido no padrão H.264 é que este filtro é adaptativo, conseguindo distinguir entre uma aresta real da imagem de um artefato gerado por um elevado passo de quantização.

Na codificação de entropia (Figuras 2.7 e 2.8) o H.264 também introduz ferramentas que aumentam bastante a eficiência de codificação deste bloco. Há, basicamente, dois tipos de codificação: a codificação de comprimento variável adaptativa ao contexto (CAVLC) (RICHARDSON, 2003) e a codificação aritmética adaptativa ao contexto (CABAC)

(RICHARDSON, 2003). A principal inovação é o uso de codificação adaptativa baseada em contextos. Nesta codificação, a maneira com que os diversos elementos sintáticos são codificados depende do elemento a ser codificado, da fase em que se encontra o algoritmo de codificação e dos elementos sintáticos que já foram codificados.

A CAVLC é utilizada para codificar os resíduos resultantes da quantização. A CAVLC produz códigos de comprimento variável que são dependentes do contexto da codificação, isto é, da fase em que o algoritmo de codificação se encontra e dos valores que já foram codificados.

A CABAC atinge elevadas taxas de compressão através da seleção de modelos de probabilidade para cada elemento sintático de acordo com o contexto deste elemento, então as estimativas de probabilidade são adaptadas com base nas estatísticas locais e, finalmente, a codificação aritmética é usada para codificar o elemento sintático.

Neste capítulo foram apresentados os conceitos básicos sobre vídeo digital e foi introduzido o padrão H.264. A estrutura do padrão foi brevemente apresentada, descrevendo a estrutura da codificação e decodificação, as características do padrão, como perfis e níveis, e uma visão geral de cada bloco do codificador e decodificador.

No próximo capítulo a compensação de movimento é detalhada, abordando-se com mais detalhes as ferramentas e características desse bloco.

3 O BLOCO DA COMPENSAÇÃO DE MOVIMENTO (MC)

Este capítulo apresenta a Compensação de Movimento (MC) no padrão H.264. Este capítulo também aborda, em menos detalhes, aspectos da Estimação de Movimento (ME), uma vez que a predição inter-quadro no codificador H.264 é formada pelos blocos da Estimação de Movimento (ME) e da Compensação de Movimento (MC), e ambos estão estreitamente relacionados.

O bloco da estimação de movimento está presente apenas nos codificadores H.264. Este bloco é o que apresenta a maior complexidade computacional dentre todos os blocos de um codificador H.264 (PURI; CHEN; LUTHRA, 2004). Este grande custo computacional é função das inovações inseridas na predição inter-quadro do padrão, que tiveram o objetivo de atingir elevadas taxas de compressão. Reside nos blocos da ME e MC as principais fontes de ganhos na taxa de compressão do padrão H.264 em relação aos demais padrões de compressão de vídeo (KWON; TAMHANKAR; RAO, 2003; RICHARDSON, 2003).

A estimação de movimento deve prover os mecanismos de codificação capazes de localizar, nos quadros de referência, qual área mais se assemelha ao macrobloco atual. Assim que é encontrada esta área, a ME deve gerar um vetor indicando a posição dela no quadro de referência. Este vetor é chamado de vetor de movimento e deve ser inserido junto com a codificação do macrobloco.

Para a realização da estimação de movimento é considerado apenas a componente de luminância do macrobloco. Cada componente de crominância possui a metade da resolução horizontal e vertical da componente de luminância, então os componentes horizontal e vertical de cada vetor de movimento são ajustados, de acordo com a sub-amostragem de cores, para serem aplicados aos blocos de croma.

A etapa de compensação de movimento (bloco Inter Predição nas Figuras 2.4 e 2.7), foco deste trabalho, deve adaptar-se às definições da estimação de movimento. A ME localiza o melhor casamento dentre os quadros de referência e gera um vetor de movimento. É função da compensação de movimento, a partir do vetor de movimento gerado na ME, localizar os blocos de melhor casamento na memória de quadros anteriormente codificados (quadros passados e/ou futuros) e montar o quadro predito. Na codificação, este quadro será subtraído do quadro atual para gerar o quadro de resíduos que passará pelas transformadas. Na decodificação, o resultado da compensação é somado aos resíduos gerados pela transformada inversa.

A compensação de movimento é realizada tanto no codificador quanto no decodificador. A MC no codificador pode ser simplificada, tendo em vista as possíveis simplificações que podem ser realizadas na ME no codificador. Por outro lado, no decodificador, a compensação de movimento deve ser completa, isto é, deve estar apta a operar sobre todas as funcionalidades do padrão.

A compensação de movimento deve atender as exigências da estimação de movimento. Deste modo, a MC deve:

- Tratar múltiplos tamanhos de partições de macroblocos;
- Utilizar múltiplos quadros de referência anteriores e posteriores;
- Tratar vetores que apontam para fora da borda do quadro;
- Interpretar corretamente os vetores de movimento e, em alguns casos, os índices dos quadros de referência, construídos com base na predição de vetores;
- Reconstruir os macroblocos considerando precisão de $\frac{1}{4}$ de pixel para os vetores de movimento;
- Reconstruir os macroblocos que utilizam as predições bi-preditiva, ponderada e direta para *slices* do tipo B;
- Inferir corretamente os pesos e ajustes da predição ponderada;
- Reconstruir corretamente os macroblocos do tipo *skip* para *slices* tipo P e B;

As partições de macroblocos em um *slice* tipo I não utilizam estimação ou compensação de movimento, uma vez que suas predições são do tipo intra-quadro, isto é, são realizadas apenas sobre amostras internas ao *slice* atual.

As partições de macroblocos em um *slice* tipo P podem ser codificados através de predições intra-quadro, de predições inter-quadro ou através de macroblocos do tipo *skip*. A predição inter-quadro utiliza a estimação de movimento e pode ser realizada sobre quadros passados ou futuros que tenham sido codificados anteriormente ao quadro atual, armazenados na lista 0.

As partições de macroblocos em um *slice* B são codificadas através de predições inter-quadro que podem ser geradas de diversas formas. É possível realizar a predição utilizando um quadro da lista 0, um quadro da lista 1 ou, ainda, utilizar um quadro de cada lista. Esta última opção é chamada de estimação bi-preditiva. Também é possível realizar a chamada estimação direta. Além destas opções, macroblocos do tipo *skip* também podem ser utilizados, como nos *slices* do tipo B.

A seção 3.1 desta dissertação apresentará cada uma das ferramentas associadas ao processamento das amostras da região do quadro de referência. A interpretação dos vetores de movimento e dos índices dos quadros de referência será abordada na seção 3.2. Por semelhança, também será abordada na seção 3.2 o processo de inferência dos pesos e ajustes da predição ponderada. O tratamento de vídeo entrelaçado será apresentado na seção 3.3. Fechando este capítulo, é apresentado o estado da arte nas implementações em *hardware* de compensadores de movimento e de decodificadores H.264.

3.1 Ferramentas

Os ganhos alcançados pela estimação e compensação de movimento do H.264 devem-se às ferramentas utilizadas no padrão para localizar o melhor casamento entre o macrobloco atual e a área do quadro de referência. Essas ferramentas são também utilizadas para reconstruir o quadro posteriormente. Essas ferramentas serão detalhadas nessa seção.

3.1.1 Tamanho de blocos variável

A principal inovação do H.264 no ponto de vista da estimação de movimento está na possibilidade de utilização de tamanhos de blocos variáveis para realizar a estimação e compensação de movimento. Ao invés de usar um macrobloco inteiro na estimação de movimento, o padrão H.264 permite o uso de partições de macrobloco e partições de sub-macroblocos. As partições de macroblocos possuem tamanhos de 16x16, 8x16, 16x8 e 8x8, como está apresentado na Figura 3.1 (KWON; TAMHANKAR; RAO, 2003; RICHARDSON, 2003).

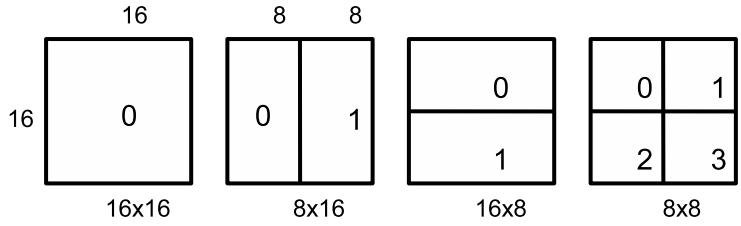

Figura 3.1: Divisão do macrobloco em partições de macroblocos

As partições de sub-macroblocos são permitidas somente se a partição de macrobloco selecionada foi a de 8x8. Neste caso, as quatro partições 8x8 do macrobloco podem ser divididas em mais quatro formas, chamadas sub-macroblocos. As partições de sub-macrobloco podem ter o tamanho 8x8, 4x8, 8x4 ou 4x4, como está apresentado na Figura 3.2.

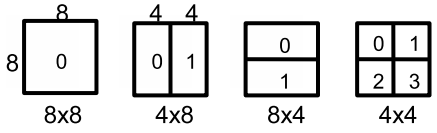

Figura 3.2: Divisão de uma partição de macrobloco em partições de sub-macroblocos

Este método de particionar macroblocos em sub-blocos de tamanho variável é conhecido como compensação de movimento estruturada em árvore.

Cada bloco de croma é dividido da mesma maneira que os blocos de luma, exceto pelo tamanho da partição, que terá exatamente a metade da resolução horizontal e vertical da partição de luma. Por exemplo, uma partição de 8x16 de luma corresponde a uma partição de 4x8 de croma.

Um vetor de movimento é necessário para cada partição ou sub-macrobloco. Cada vetor de movimento precisa ser codificado e transmitido e a escolha da partição precisa ser codificada no *bitstream* comprimido. A escolha de um tamanho de partição grande (16x16, 16x8, 8x16) implica na utilização de um número menor de bits para identificar o vetor de movimento escolhido e para indicar o tipo de partição utilizada. Por outro lado, o resíduo gerado pode conter uma quantidade significativa de energia em áreas com muitos detalhes. A escolha de um tamanho de partição pequeno (8x4, 4x4) pode gerar um resíduo com quantidade de energia mínima depois da compensação de movimento, mas esta escolha requer a utilização de um número de bits maior para representar o vetor de movimento e para identificar a partição escolhida. A escolha do tamanho da partição possui um impacto significativo no desempenho da compressão (RICHARDSON, 2003).

Em geral, uma partição grande é apropriada para áreas mais homogêneas do quadro e uma partição menor tende a ser mais apropriada para áreas com muitos detalhes.

A compensação de movimento localiza, no quadro de referência, cada uma das partições do macrobloco, ou do sub-macrobloco, e opera individualmente sobre ela.

3.1.2 Vetores apontando para fora do quadro

A possibilidade de utilizar vetores de movimento que apontam para fora dos limites dos quadros também faz parte do padrão H.264. Neste caso é realizada uma extrapolação dos quadros nas bordas. A Figura 3.3 apresenta um exemplo da extrapolação realizada. As amostras pertencentes às bordas são replicadas, nas direções vertical e horizontal. A amostra pertencente à borda do quadro é replicada para toda a área em que as duas componentes do vertor (vertical e horizontal) não pertencem às dimensões do quadro.

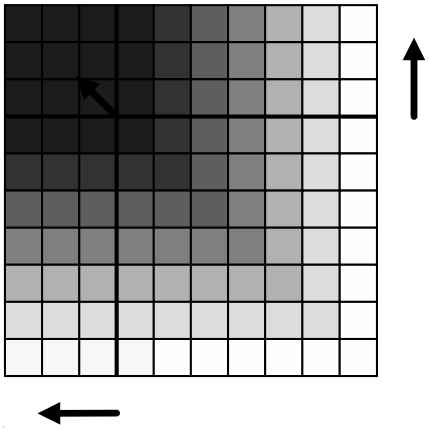

Figura 3.3: Extrapolação da borda superior esquerda

Esta ferramenta, quando implementada em software, tem complexidade baixa. Todavia, numa implementação em *hardware*, ela afeta diretamente os processos de localização e cópia da área do quadro de referência. Para cada acesso ao *buffer* de quadros de referência uma verificação é realizada para identificar se a área ultrapassa a borda do quadro e, se for o caso, os valores presentes na borda são extrapolados.

3.1.3 Múltiplos quadros de referência

Outra característica importante do padrão H.264 é o uso de múltiplos quadros de referência. No padrão H.264 os quadros de referência não precisam ser somente os quadros I ou P imediatamente anteriores ou posteriores ao quadro atual. Há a opção de se usar como referência múltiplos quadros temporalmente para frente ou para trás do quadro atual (RICHARDSON, 2003), como no exemplo apresentado na Figura 3.4. O padrão define, inclusive, que quadros do tipo B podem ser usados como referência na predição, diferentemente do que acontece no padrão MPEG-2 (ITU-T, 1994).

O padrão H.264 define duas listas contendo os quadros de referência. A lista chamada de **lista 0** contém como primeiro quadro o quadro passado mais próximo, seguido de quaisquer outros quadros passados, seguidos de quaisquer outros quadros futuros. A lista chamada **lista 1** contém como primeiro quadro o quadro futuro mais próximo, seguido de quaisquer quadros futuros, seguidos de quaisquer outros quadros passados.

É importante destacar que o padrão H.264 permite que o codificador escolha uma ordem dos quadros para codificação completamente diferente da ordem dos quadros para apresentação do vídeo (KWON; TAMHANKAR; RAO, 2005). Por isso, é possível armazenar nas listas 0 e 1, quadros futuros que, neste caso, são quadros futuros para a

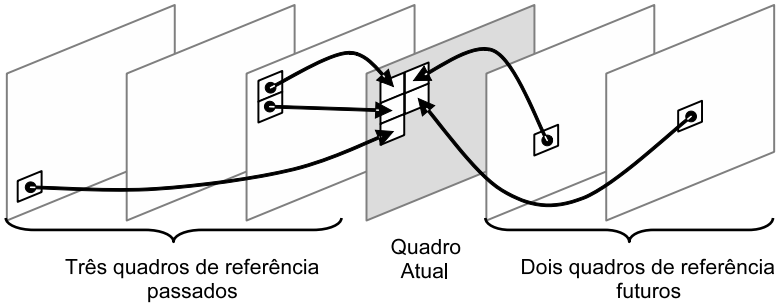

Figura 3.4: Uso de múltiplos quadros de referência

apresentação do vídeo, mas são quadros que já foram codificados.

O padrão H.264 define um tipo de armazenamento especial nas listas, o quadro long_term. Se um quadro se mostrar útil para ser referenciado por um longo período de tempo, o mesmo é assinalado como sendo um quadro de referência long_term, o que o mantém no *buffer* de quadros até que o gerenciador de listas o apague.

3.1.4 Predição de Quarto de Pixel

Uma outra característica importante da estimação de movimento do padrão H.264 é que ela prevê uma precisão de $\frac{1}{4}$ de pixel para os vetores de movimento. Normalmente, os movimentos que acontecem de um quadro para o outro não estão restritos a posições inteiras de pixel. Assim, se são utilizados apenas vetores de movimento com valores inteiros, normalmente não é possível encontrar casamentos bons. Por isso, o padrão H.264 prevê a utilização de vetores de movimento com valores fracionários de $\frac{1}{2}$ pixel e de $\frac{1}{4}$ de pixel. Na Figura 3.5 é apresentado um exemplo da precisão do vetor de movimento com valores fracionários.

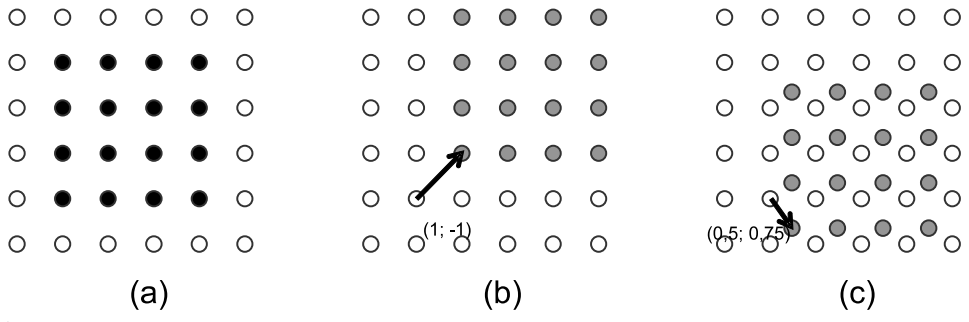

Figura 3.5: Estimação de movimento com precisão de frações de pixel

Na Figura 3.5 (a) está representado, com pontos pretos, um bloco 4x4 do quadro atual que será predito a partir de uma região do quadro de referência na vizinhança da posição do bloco atual. Se os componentes horizontal e vertical do vetor de movimento são inteiros, então as amostras necessárias para o casamento estão presentes no bloco de referência. Como exemplo, a Figura 3.5 (b) apresenta, com pontos em cinza, um possível casamento do bloco atual no quadro de referência, onde o vetor de movimento utiliza apenas valores inteiros, neste caso, o vetor de movimento é (-1, 1). Se um dos componentes do vetor de movimento apresenta valor fracionário, então as amostras preditas são geradas a partir da interpolação entre amostras adjacentes do quadro de referência. Como exemplo, a Figura 3.5 (c) apresenta um casamento com um vetor de movimento que uti-

liza dois valores fracionários, com as amostras sendo geradas por interpolação. No caso do exemplo, o vetor é (0,5; 0,75).

No padrão H.264, a estimação de movimento usando $\frac{1}{4}$ de pixel é obrigatória. Para os elementos de crominância a precisão é, na verdade, de 1/8 de pixel.

A interpolação para $\frac{1}{4}$ de pixel é realizada em dois passos. No primeiro passo, é realizada a interpolação dos quadros para $\frac{1}{2}$ pixel usando um filtro FIR de seis *taps*, com pesos (1/32, -5/32, 5/8, 5/8, -5/32, 1/32), da forma como está descrito na Figura 3.6. Na Figura 3.6, as posições inteiras são representadas por letras maiúsculas e as de $\frac{1}{2}$ pixel por letras minúsculas.

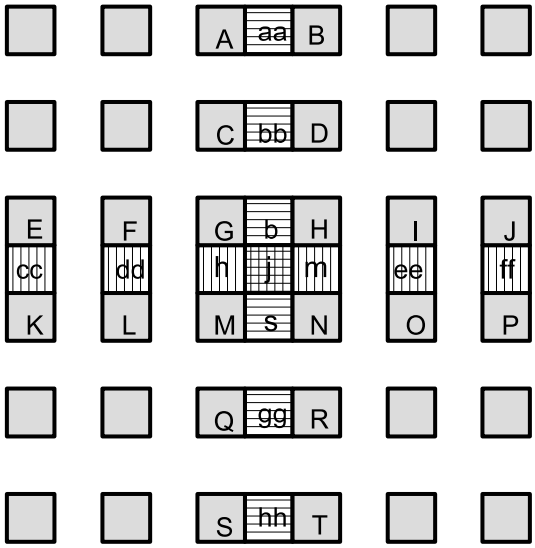

Figura 3.6: Interpolação para posições de $\frac{1}{2}$ pixel para a componente de luminância

Como exemplo, a posição **b** da Figura 3.6 é calculada como está indicado em 3.1.

$$b = (E - 5 * F + 20 * G + 20 * H - 5 * I + J)/32 \tag{3.1}$$

De forma similar, h é interpolado através da filtragem de **A**, **C**, **G**, **M**, **Q** e **S**. Por outro lado, **j** é gerado através da filtragem de **aa**, **bb**, **b**, **s**, **gg** e **hh**, que foram gerados a partir da primeira filtragem realizada apenas sobre elementos inteiros. É importante ressaltar que o filtro pode ser aplicado horizontalmente ou verticalmente.

No segundo passo, são interpoladas as posições de $\frac{1}{4}$ de pixel a partir das amostras construídas no primeiro passo e das amostras inteiras, usando a média simples entre dois pontos, na relação que está apresentada na Figura 3.7. Como exemplo, o cálculo da posição *a* na Figura 3.7 é realizado como está apresentado em 3.2.

$$a = (G + b)/2 \tag{3.2}$$

Para os vetores de crominância, que utilizam uma resolução de 1/8 de pixel, usa-se interpolação linear. As amostras interpoladas são geradas em intervalos de oito amostras entre amostras inteiras em cada componente de crominância. Considerando a Figura 3.8 como referência, cada posição interpolada *a* é uma combinação linear das amostras inteiras da vizinhança que, no caso da Figura 3.8, são as posições **A**, **B**, **C** e **D**.

A combinação linear das amostras inteiras é definida pela equação 3.3.

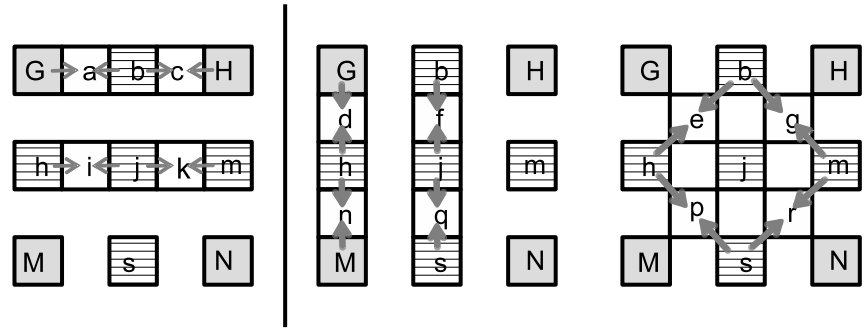

Figura 3.7: Interpolação para posições de $\frac{1}{4}$ de pixel

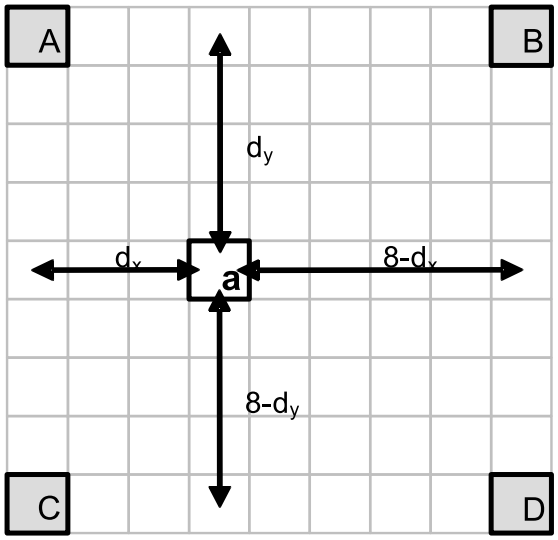

Figura 3.8: Interpolação para componentes de crominância

$$a = ((8-dx)*(8-dy)*A+dx*(8-dy)*B+(8-dx)*dy*C+dx*dy*D+32))/64 \quad (3.3)$$

No exemplo da Figura 3.8, **dx** é igual a três e **dy** é igual a quatro, então a é definido por 3.4 para este exemplo.

$$a = (20 * A + 12 * B + 20 * C + 12 * D + 32)/64 \quad (3.4)$$

3.1.5 Predição bi-preditiva

É possível realizar a predição utilizando um quadro da lista 0, um quadro da lista 1 ou, ainda, utilizar um quadro de cada lista. Esta última opção é chamada de predição bi-preditiva, ou bi-predição.

A predição bi-preditiva utiliza um bloco de referência que é construído a partir de dois blocos, um na lista 0 e outro na lista 1. Neste caso, são necessários dois vetores de movimento, um para o quadro de referência da lista 0 e outro para o quadro de referência da lista 1. Então, cada amostra do bloco de predição é calculada como uma média entre as amostras dos quadros das listas 0 e 1. Este cálculo é apresentado na equação 3.5, onde **pred0(i,j)** e **pred1(i,j)** são amostras derivadas dos quadros de referência das listas 0 e 1, respectivamente, e onde **pred(i,j)** é a amostra bi-preditiva.

$$pred(i,j) = (pred0(i,j) + pred1(i,j) + 1) >> 1 \tag{3.5}$$

A equação 3.5 é válida se a predição bi-preditiva não utilizar a predição ponderada, uma vez que a predição ponderada utiliza pesos diferentes sobre as amostras.

A predição bi-preditiva é aplicada após a predição em quarto de pixel da compensação de movimento. A Figura 3.9 exemplifica a aplicação da predição bi-preditiva. As Figuras 3.9 (a) e (b) representam áreas de referência das listas 0 e 1, respectivamente, já preditas em quarto de pixel. A Figura 3.9 (c) é o resultado da bi-predição aplicada sobre as áreas.

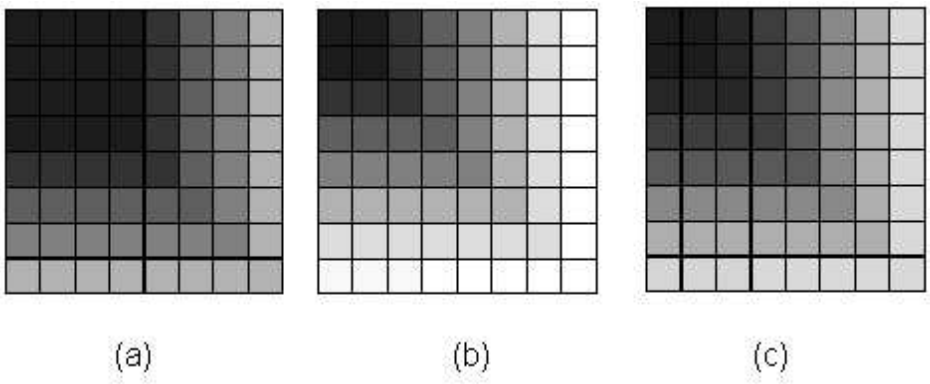

Figura 3.9: Exemplo da aplicação da bi-predição

3.1.6 Predição Ponderada

A predição ponderada é um método de escalonar os valores das amostras da compensação de movimento de um macrobloco num quadro do tipo P ou B. Ela faz parte dos perfis Main e Extended do padrão H.264. Existem dois modos de predição ponderada: modo explícito, que pode ser utilizada nos *slices* P, SP e B; e o modo implícito, que pode ser usado apenas em *slices* do tipo B.

Um único fator de escala está associado a cada quadro de referência, para cada componente de cor em cada *slice*. No modo explícito, os parâmetros que compõem o fator de escala estão codificados no cabeçalho do *slice*. No modo implícito, esses parâmetros são derivados baseados na distância relativa entre o quadro corrente e seus quadros de referências. Cada partição de macrobloco utiliza os parâmetros da predição ponderada associados ao seu índice de referência.

A predição ponderada no padrão H.264 não utiliza divisões, utilizando, para o caso da predição de um único quadro de referência, a fórmula 3.6 abaixo, onde **predX(i,j)** é a amostra derivada do quadro de referência, **pred(i,j)** a amostra ponderada, **logWD** o logaritmo de base 2 do denominador de todos os fatores de escala, **w** o fator de escala, **o** o ajuste aditivo e **X** representa a lista a qual pertence ao quadro de referência, 0 ou 1.

$$pred(i,j) = ((predX(i,j) * wX + 2^{logWD-1}) >> logWD) + oX \tag{3.6}$$

Quando uma compensação do tipo bi-preditiva é aplicada, a predição ponderada utiliza a fórmula 3.7 abaixo, onde **logWD** é o logaritmo de base 2 do denominador de todos os fatores de escala, **w0** o fator de escala da área da lista 0, **w1** o fator de escala da área da lista 1, **o0** e **o1** os ajustes aditivos das lista 0 e 1, respectivamente.

$$pred(i,j) = ((pred0(i,j) * w0 + pred1(i,j) * w1 + 2^{logWD}) >> (logWD + 1)) + ((o0 + o1 + 1) >> 1) \quad (3.7)$$

A predição ponderada é realizada após a predição em quarto de pixel da compensação de movimento.

Após a predição ponderada uma operação de saturação é aplicada para manter as amostras dentro da faixa de valores determinada pelo padrão. A saturação é realizada utilizando a fórmula 3.8, onde **pred(i, j)** é a amostra ponderada.

$$pred(i,j) = max(0, min(255, pred(i,j)) \quad (3.8)$$

A predição ponderada é especialmente útil em *fades* nas seqüências de vídeos. A Figura 3.10 mostra um exemplo de *fade to white*. A Figura 3.10 (a) representa a área de referência, a Figura 3.10 (b) o quadro temporalmente posterior, usando (a) como referência aplicando um fator de escala equivalente a 1.3. Na figura 3.10 (c) é o resultado da aplicação de um fator de escala equivalente a 1.6 sobre a área de referência, representada na Figura 3.10 (a).

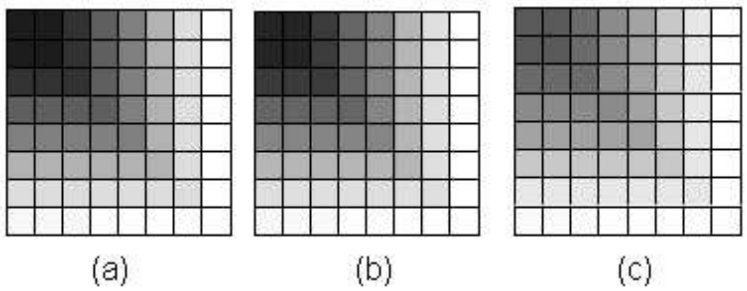

Figura 3.10: Exemplo de fade to white a partir de um mesmo quadro de referência

3.1.7 Predição skip e direta

O padrão H.264 também aprimora o conceito de macrobloco *skip* em relação aos padrões anteriores. Diferente dos outros padrões, um macrobloco *skip* no padrão H.264 usa uma predição de vetores de movimento, ao invés de simplesmente copiar o bloco co-localizado. O bloco co-localizado, no padrão H.264, é o bloco pertencente ao quadro de referência de índice 0 da lista 1, o qual se encontra na mesma posição que o bloco corrente. Essa predição calcula o vetor de movimento a partir dos vetores vizinhos para copiar uma área do quadro de referência. Além disso, se o macrobloco *skip* estiver em um *slice* B, o macrobloco é reconstruído no decodificador usando predição direta (SAHAFI, 2005). Além dos vetores de movimento, os índices de quadros de referência também são preditos baseados nos índices dos macroblocos vizinhos, utilizando a predição direta.

A predição direta pode ser aplicada tanto no macrobloco inteiro quanto, individualmente, em uma ou mais das 4 partições de macrobloco no formato 8x8.

A predição dos vetores de movimento será detalhada na seção 3.2, onde serão detalhados os procedimentos para se encontrar os vetores de movimento e índices dos quadros de referência, detalhando cada uma das possibilidades cobertas pelo padrão.

3.2 Predição dos vetores de movimento e dos índices dos quadros de referência das listas 0 e 1

Para minimizar o custo da codificação dos vetores de movimento, o padrão H.264 define que os vetores sejam preditos a partir dos vetores de movimento e índices de referência, previamente calculados, das partições de macroblocos vizinhas. Dessa forma, explora-se a correlação existente entre vetores de blocos vizinhos,

No *bitstream*, são codificados apenas os vetores de movimento diferenciais, que é a diferença entre os vetores de movimento do bloco e seus vetores de movimento preditos (*predictive motion vector* - PMV). O PMV é, normalmente, a mediana dos vetores de movimento dos blocos vizinhos ao bloco corrente. Os macroblocos do tipo *skip* têm uma forma diferenciada de predição dos vetores de movimento. Há, também, as partições que utilizam a predição direta. Essa seção detalha esses processos de predição dos vetores e índices de referência.

A predição dos vetores de movimento segue a estrutura de macrobloco, partição e sub-partição definidas na seção 3.1.1. A ordem de predição segue a ordem conhecida como duplo Z, ou *zig-zag scan*. A ordem de visitação dos blocos é representada na Figura 3.11. No caso de uma partição ocupar vários blocos, é considerado o bloco mais acima e mais a esquerda. Todos os blocos de uma partição compartilham os PMVs.

0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

Figura 3.11: Ordem de processamento dos blocos de um macrobloco

A vizinhança no H.264 corresponde às quatro partições ou sub-partições de macrobloco contíguos ao macrobloco, partição ou partição de sub-macrobloco corrente. Cada vizinho recebe uma letra, de ‘A’ a ‘D’ que o identifica.

Seja ‘E’ o macrobloco, partição de macrobloco ou de sub-macrobloco corrente: ‘A’ é a partição imediatamente à esquerda de ‘E’, ‘B’ é a partição imediatamente acima de ‘E’, ‘C’ é a partição acima e a direita de ‘E’ e ‘D’ é a partição acima e a esquerda de ‘E’, tal qual é vista na Figura 3.12 (a). Para o caso de existir mais de uma partição à esquerda, a superior é escolhida como ‘A’. Se existir mais de uma partição acima de ‘E’, a mais à esquerda é escolhida como ‘B’. Essa relação é exemplificada na Figura 3.12 (b).

O conceito de vizinho é válido para as partições internas ao mesmo macrobloco. Logo, o fluxo de predição é dependente dos PMVs do próprio macrobloco corrente, o que torna o processo essencialmente seqüencial.

Há casos em que os vetores de movimento diferenciais não são transmitidos no *bitstream*, mas inferidos a partir dos vetores de movimento das partições vizinhas. Num *slice* P, isso ocorre quando o macrobloco atual é do tipo *skip*. Num *slice* B, os vetores diferenciais não estão disponíveis em duas situações: quando o macrobloco é do tipo *skip* ou quando o macrobloco, ou partição de macrobloco, usa a predição direta. Os macroblocos *skip* num *slice* B utilizam a predição direta para encontrar os vetores de movimento.

A predição direta no H.264 pode ser realizada de dois modos, espacial ou temporal.

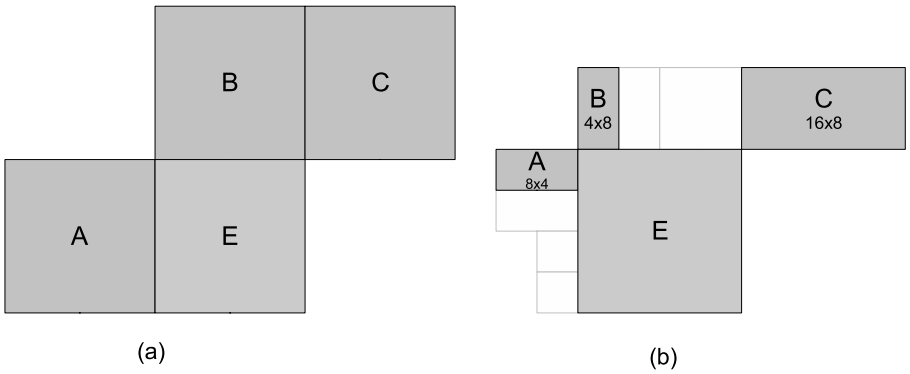

Figura 3.12: Exemplo de vizinhança entre partições. (a) partições de formatos idênticos. (b) partições de formatos diferentes

O modo espacial é uma inovação do padrão enquanto o modo temporal é uma versão simplificada da predição direta temporal encontrada no MPEG-4 original (PURI; CHEN; LUTHRA, 2004).

No modo espacial, os vetores de movimento são derivados se verificando os vetores de movimento do macrobloco co-localizado e, dependendo do resultado dessa verificação, usam-se os vetores de movimento dos blocos vizinhos para gerar os vetores de movimento.

No modo temporal o quadro corrente se encontra temporalmente entre duas outras. O vetor de movimento do macrobloco ou partição de macrobloco corrente, que utiliza a predição direta, será uma média ponderada dos vetores dos macroblocos ou partições de macrobloco co-localizados. Os pesos da média são dados pela distância relativa entre o quadro corrente e os quadros de referência.

Nos casos onde os vetores de movimento diferenciais não estão presentes no *bitstream* também não estão presentes os índices de referências. Assim, além de encontrar os vetores de movimento, é necessário encontrar, também, os índices de referências de cada partição do macrobloco que utiliza essa operação diferenciada.

Quando a predição direta é utilizada, não é conhecida a utilização das listas de referência para a predição. Assim, a utilização da lista 0, da lista 1, ou da predição bi-preditiva também é apontada pela predição. Para isso são utilizadas *flags* de utilização de cada lista.

Nas próximas 3 subseções serão abordados os mecanismos para se encontrar os vetores a partir dos PMVs e dos vetores de movimento diferenciais e, assim, realizar a predição dos vetores de movimento. Para o caso das predições diretas, também será abordado os mecanismos para se calcular os índices de referências e as *flags* de utilização das listas.

A predição ponderada nem sempre é utilizada para todos os quadros de referência referenciados pelo quadro atual. Também existem casos em que os parâmetros da predição ponderada não estão presentes no *bitstream*, a chamada predição ponderada implícita. Processos específicos são definidos para tratar desses casos. Estes processos serão apresentados na seção 3.2.5.

3.2.1 Predição padrão

A predição padrão dos vetores de movimento é dada pela soma dos PMVs, normalmente calculados pela mediana dos vetores de movimento dos blocos vizinhos, com os

vetores de movimento diferenciais da partição de macrobloco ou de sub-macrobloco corrente. A predição recebe como entrada, além dos vetores diferenciais, os vetores de movimento e os índices de referência das partições vizinhas.

A predição padrão é realizada nas seguintes etapas: verificação da disponibilidade dos blocos vizinhos; seleção dos vetores de movimento dos blocos vizinhos, baseada na disponibilidade dos mesmos; seleção do tipo de operação sobre os vetores vizinhos, baseada no compartilhamento do quadro de referência da partição corrente com as dos vizinhos e no formato da partição corrente; cálculo dos PMVs e sua adição aos vetores de movimento diferenciais presentes no *bitstream*; armazenamento dos vetores de movimento resultantes baseado no formato da partição.

A verificação da disponibilidade dos blocos vizinhos corresponde a verificação se os mesmos se encontram dentro das dimensões do quadro, se pertencem ao *slice* corrente e se já foram decodificados. A partição ou sub-partição vizinha que não satisfazer a alguma dessas condições é marcada como não disponível.

Para o cálculo da predição são utilizados até três vetores de movimento das partições ou sub-partições vizinhas. É atribuído o valor zero ao vetor do vizinho que tenha sido marcado como não disponível. No caso do vizinho 'C' ter sido marcado como não disponível é atribuído a ele o valor do vetor do vizinho 'D'.

A mediana é a operação padrão para se calcular o PMV. A mediana tem como parâmetro os vetores dos vizinhos 'A', 'B' e 'C', podendo esse último conter o valor do vetor do vizinho 'D', para o caso definido acima. A mediana é definida na equação 3.9.

$$Mediana(x, y, z) = x + y + z - Min(x, Min(y, z)) - Max(x, Max(y, z)) \quad (3.9)$$

Apesar de a mediana ser a operação padrão para a escolha do PMV, ela não é a única opção definida pelo padrão H.264. Se apenas um dos vizinhos tiver como quadro de referência o quadro de referência da partição corrente, o valor deste é escolhido como PMV.

Outra possibilidade, que se sobrepõe a anterior, se refere ao formato das partições do macrobloco corrente. Se o formato da posição for 16x8 ou 8x16 e caso compartilhe o mesmo quadro de referência do vizinho, o vetor de um dos vizinhos é selecionado para ser o PMV. Para partições no formato 8x16, a partição mais a esquerda tem o valor do vetor de movimento do vizinho 'A' selecionado e a partição inferior tem o vetor de movimento do vizinho 'C' selecionado, como exemplificado na Figura 3.13 (a). Para partições no formato 16x8, a partição superior do macrobloco tem o valor do vetor de movimento do vizinho 'B' selecionado, e a partição inferior tem o vetor de movimento do vizinho 'A' selecionado, representado na Figura 3.13 (b).

Se a operação da mediana for escolhida para se encontrar o PMV, mas o vizinho 'A' for o único disponível, o vetor do vizinho 'A' é atribuído a PMV. Esse caso evita que todos os vetores dos macroblocos, partições de macrobloco e partições de sub-macroblocos pertencentes à borda superior do quadro tenham o valor zero atribuído a seus PMVs.

Com o valor de PMV calculado, a ele é adicionado o valor do vetor de movimento diferencial codificado, presente do *bitstream*. O vetor de movimento resultante é então atribuído a todos os blocos pertencentes ao macrobloco, partição de macrobloco ou partição de sub-macrobloco corrente.

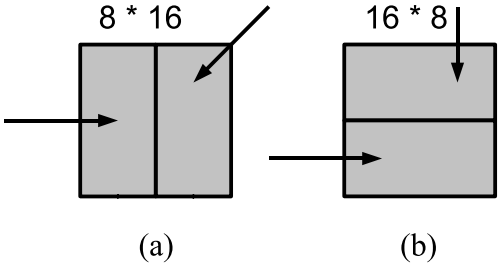

Figura 3.13: Casos especiais de escolha do PMV a partir do formato da partição. (a) partições 8x16. (b) partições 16x8

3.2.2 Predição skip em *slices* P

Um macrobloco *skip* num *slice* P não tem o vetor de movimento diferencial presente no *bitstream*, sendo a este atribuído o valor inicial zero. O quadro de referência desse tipo de macrobloco é o quadro de referência de índice 0 na lista 0.

Para se encontrar o vetor de movimento de um macrobloco *skip* em um *slice* do tipo P a verificação da disponibilidade dos blocos vizinhos é realizada. Ao vetor de movimento é atribuído zero caso alguma das seguintes condições seja verdadeira:

O vizinho 'A' ou 'B' está marcado como não disponível;

O vizinho 'A' ou 'B' tem como quadro de referência o índice 0 da lista 0 e seus vetores de movimento são iguais a zero.

Se nenhuma dessas condições for verdadeira a predição dos vetores de movimento padrão é realizada, como descrito na seção 3.2.1, e o valor do PMV calculado torna-se o valor do vetor de movimento do macrobloco.

3.2.3 Predição Direta Espacial

Num *slice* B, quando um macrobloco é *skip*, ou uma partição do macrobloco é do tipo direta, e a *flag* **direct_spatial_mv_pred_flag** é igual a 1, os vetores de movimento, as *flags* de utilização de listas e seus respectivos índices dos quadros de referência são determinados pelo processo de predição direta espacial.

O processo de predição direta espacial se inicia por encontrar os índices dos quadros de referência para as listas 0 e 1. Para encontrar os índices é verificada a disponibilidade das partições ou sub-partições vizinhas ao macrobloco corrente. Diferente do processo padrão, a vizinhança na predição direta leva sempre em conta o macrobloco corrente, mesmo quando se trata de uma partição direta 8x8. Se alguma partição ou sub-partição vizinha estiver marcada como não disponível, é considerado o valor -1 como seu índice de lista de quadro de referência. Semelhante ao que acontece com os vetores de movimento na predição padrão, quando o vizinho 'C' está marcado como não disponível, lhe é atribuído o índice do vizinho 'D'. Determinados os índices dos vizinhos, a fórmula 3.10 é aplicada, onde X é a lista a qual pertence o índice, **IdxLXA**, **IdxLXB** e **IdxLXC** são os índices de referência dos vizinhos A, B e C, respectivamente, e **IdxLX** é o índice de referência do macrobloco. A operação **MenorPositivo** é definida na fórmula 3.11.

$$IdxLX = MenorPositivo(IdxLXA, MenorPositivo(IdxLXB, IdxLXC)) \quad (3.10)$$

$$MenorPositivo(x, y) = \begin{cases} (min(x, y) \text{ se } x >= 0 \text{ e } y >= 0) \\ \text{senão } max(x, y) \end{cases} \quad (3.11)$$

Se ambos os índices de referência, **idxL0** e **idxL1**, forem menores que zero, é atribuído zero ao índice de referência de ambas as listas de referência da partição direta corrente. A *flag* **directZeroPredictionFlag** também é ligada, para esse caso.

Nessa fase também são selecionados os vetores de movimento dos blocos vizinhos, da forma já descrita. Neste processo são considerados os blocos vizinhos ao macrobloco corrente, tal qual a seleção dos índices dos quadros de referência.

Tendo sido calculados os índices, a *flag* **directZeroPredictionFlag** e a predição dos vetores de movimento, o macrobloco é dividido em 4 partições 8x8 (caso a predição não esteja sendo feita sobre uma partição 8x8 direta de um macrobloco não direto), que por sua vez são divididos em 4 partições de sub-macrobloco. Com isso, toda a partição do tipo direta tem o vetor de movimento predito para cada bloco 4x4 individualmente.

Para cada bloco 4x4 na região direta são executados, seguindo a ordem do duplo Z, os seguintes passos: são selecionados os vetores de movimento e índices de quadro de referência dos blocos co-localizados para ambas as listas; é definido o valor da *flag* **colZeroFlag** e, a partir do valor dessa *flag*, os valores dos vetores de movimento para o bloco são levados a zero ou então é realizada a predição padrão.

A *flag* **colZeroFlag** é ligada caso todas as condições abaixo forem satisfeitas:

- O quadro de referência de índice 0 da lista 1 não é do tipo long_term;
- O índice de referência da lista 0 do bloco co-localizado é igual a zero. Caso a lista 0 não estiver sendo utilizada é avaliado o índice de referência da lista 1;
- Os vetores de movimento, na resolução de quarto de pixel, se encontram na faixa de valores entre -1 e 1, inclusive.

Os vetores de movimento do bloco são levados a zero caso qualquer uma das seguintes condições seja satisfeita:

- A *flag* **directZeroPredictionFlag** está ligada;
- O índice da referência na lista avaliada, 0 ou 1, é igual a -1;
- O índice da referência na lista avaliada, 0 ou 1, é igual a 0 e a *flag* **colZeroFlag** é igual a 1.

A avaliação ocorre para cada lista individualmente, ou seja, os vetores de movimento da lista 0 podem ser zerados, enquanto os da lista 1 não.

Se nenhuma das condições for verdadeira a predição dos vetores de movimento padrão é realizada, como descrito na seção 3.2.1, e o valor de PMV torna-se o valor do vetor de movimento do bloco 4x4 corrente.

3.2.4 Predição Direta Temporal

Para as mesmas condições da predição direta espacial, citadas acima, mas com o *flag* **direct_spatial_mv_pred_flag** é igual a 0, os vetores de movimento, as *flags* de utilização de listas e seus respectivos índices dos quadros de referência são determinados pelo processo de predição direta temporal.

A predição direta temporal inicia por selecionar os índices de referência das listas 0 e 1. O índice de referência da lista 0, associado ao mesmo quadro de referência do bloco co-localizado, é atribuído ao índice de referência da lista 0. Como índice de referência da lista 1 é atribuído 0.

Encontrados os índices das listas de referência, inicia-se o processo da predição dos vetores de movimento. Similarmente à predição direta espacial, o processo ocorre sobre cada bloco 4x4, seguindo a seqüência do duplo Z.

Caso o quadro de referência tenha sido marcado como long_term, ao vetor de movimento da lista 0 de cada bloco é atribuído o valor do vetor de movimento do bloco co-localizado. Ao vetor de movimento da lista 1 do bloco, é atribuído 0 às suas duas componentes.

Caso contrário, os vetores de referência são derivados a partir das distâncias entre as imagens de referência e o quadro corrente e do vetor de movimento do bloco co-localizado.

O processo inicia por calcular o fator de escala, de acordo com as fórmulas 3.12 e 3.13, onde **TB** é a diferença em entre os valores do contador de ordem de imagem (*picture order counter* - POC) do quadro de referência da lista 1 e da lista 0, limitada ao intervalo de [-128, 127]. **TD** é a diferença entre os valores de POC da imagem corrente em relação à imagem de referência da lista 0, também limitada ao intervalo de [-128, 127].

$$tx = (16384 + Absoluto(TD/2))/TD \tag{3.12}$$

$$DistScaleFactor = Clip3(-1024, 1023, (TB * TX + 32) >> 6) \tag{3.13}$$

Calculado o fator de escala é aplicada a fórmula 3.14 para encontrar o valor do vetor de movimento **mvL0**, onde **mvCol** é o vetor de movimento do bloco co-localizado. Com **mvL0** calculado, o vetor de movimento da lista 1, **mvL1**, é derivado usando a fórmula 3.15.

$$mvL0 = (DistScaleFactor * mvCol + 128) >> 8 \tag{3.14}$$

$$mvL1 = mvL0 - mvCol \tag{3.15}$$

A Figura 3.14 exemplifica a derivação dos vetores usando a predição direta temporal.

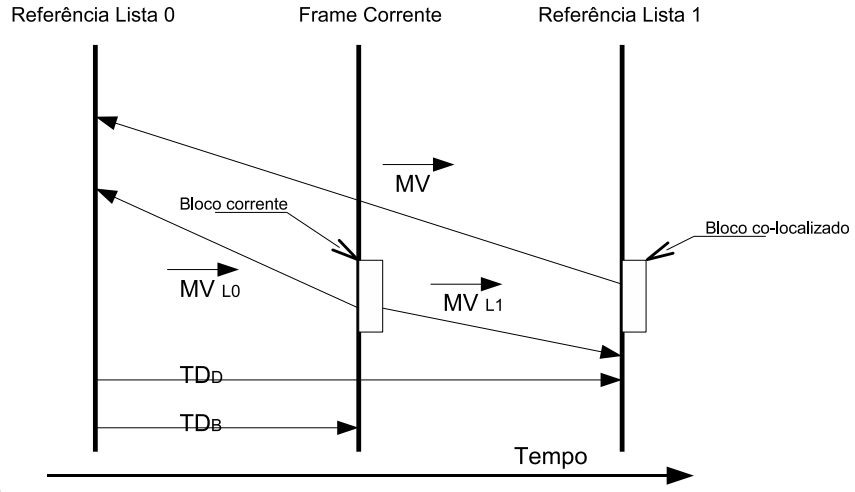

Figura 3.14: Derivação dos vetores de movimento da predição temporal

No modo de predição direta espacial o modo de predição bi-preditivo é sempre assinalado à partição corrente, sendo ligados as duas *flags* de utilização das listas.

3.2.5 Predição Ponderada Modo Explícito

No modo explícito os parâmetros da predição ponderada são determinados pelo codificador e enviados no cabeçalho do *slice*. O uso do modo explícito é indicado pelo campo **weighted_pred_flag** contendo o valor 1, para o caso de *slices* P, ou pelo campo **weighted_bipred_idc** contendo o valor 1, para o caso de *slices* B. Um fator de escala e um ajuste para cada componente de cor é codificado para cada índice de imagem de referência disponível na lista 0 para *slices* P e B, caso esses parâmetros sejam diferentes dos valores padrão.

Para cada índice de quadro de referência disponível na lista 0, e no caso de *slices* B também na lista 1, existem *flags* que indicam se os parâmetros da predição ponderada estão presentes no cabeçalho do *slice* para o quadro de referência apontado pelo índice. Cada componente de cor tem sua respectiva *flag* para indicar a existência ou não dos parâmetros no cabeçalho. Se os parâmetros não estiverem disponíveis no cabeçalho para um dado índice de quadro de referência, um valor padrão, equivalente ao fator de escala 1 e ajuste 0, será utilizado.

3.2.6 Predição Ponderada Modo implícito

No modo implícito os pesos **w0** e **w1** são calculados de acordo com a posição temporal relativa entre as imagens de referencia das listas 0 e 1. O uso deste modo é indicado pelo campo weighted_bipred_idc contendo o valor 2 nos *slices* do tipo B. Esses parâmetros da predição não estão presentes no cabeçalho, eles são calculados baseado na distância relativa, usando-se o POC, entre a imagem corrente e seus quadros de referência. O modo implícito é utilizado apenas para macroblocos e partições de macrobloco bi-preditivos, incluindo aqueles que usam predição direta. Os parâmetros da predição são calculados segundo as fórmulas 3.16 e 3.17 abaixo, onde DistScaleFactor é calculado segundo a fórmula 3.13. Os ajustes aditivos o0 e o1 são sempre iguais a zero, quando se utiliza a predição ponderada implícita.

$$w1 = DistScaleFactor >> 2 \tag{3.16}$$

$$w0 = 64 - w1 \tag{3.17}$$

Um fator de escala maior é aplicado caso a referência esteja temporalmente próxima da imagem corrente e um fator de escala menor é aplicado caso a referência esteja temporalmente próxima do quadro corrente.

Para macroblocos ou partições de macrobloco onde os preditores são de direções temporais diferentes a fórmula do modo implícito se transforma numa interpolação. Para preditores da mesma direção, o modo implícito se transforma numa extrapolação.

3.3 Vídeo Entrelaçado

Nas imagens entrelaçadas com regiões com movimento de objetos ou de câmera, duas linhas adjacentes tendem a apresentar um grau reduzido de dependência estatística quando comparado a imagens progressivas. Neste caso, pode ser mais eficiente codificar cada campo separadamente. Para aumentar a eficiência da codificação de vídeos entrelaçados o padrão H.264 possibilita ao codificador tomar uma das seguintes possibilidades enquanto codifica um quadro:

1. Combinar os dois campos e os codificar como uma única imagem (modo *frame*);

2. Não combinar os campos e os tratar cada campo como um quadro separado (modo *field*);

3. Combinar os dois campos e os comprimir como um único quadro, mas separar cada par de macroblocos verticalmente adjacentes em macroblocos *frame* ou *field* e então os codificar.

A escolha entre as três opções pode ser feita adaptativamente para cada quadro numa seqüência. A escolha entre as duas primeiras opções é denominada como codificação adaptativa quadro/campo (*picture adaptive frame/field* - PAFF). Quando uma imagem é codificada em dois campos, cada campo é particionado em macroblocos e são codificados de maneira muito similar a um quadro, com as seguintes exceções principais:

i. A compensação de movimento utiliza campos de referência no lugar de imagens de referência,

ii. O zig-zag scan dos coeficientes das transformadas é realizado de forma diferente, e

iii. A filtragem forte não é usada para a filtragem horizontal das bordas dos macroblocos em campos, pois as linhas dos campos são separadas com o dobro da distância, então o filtro deve cobrir uma área maior.

Se uma imagem é formada por regiões de movimento e outras estáticas, é tipicamente mais eficiente codificar as áreas sem movimento como quadros e as regiões com movimento como campos. Então, a decisão de codificar cada par verticalmente adjacente de macroblocos (uma região de 16x32 de luma), como campo ou quadro, pode ser feita de maneira adaptativa pelo codificador. Essa opção é denominada codificação quadro/campo adaptativa de macrobloco (*macroblock-adaptive frame/field coding* - MBAFF). Para cada par de macroblocos codificados no modo campo, o macrobloco superior contém as linhas do campo *top* enquanto o macrobloco inferior contém as linhas do campo *bottom*. A Figura 3.15 ilustra o conceito apresentado.

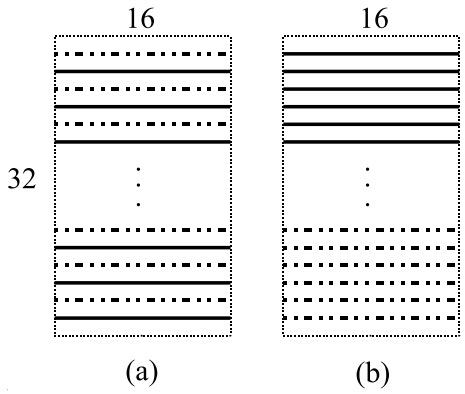

Figura 3.15: Codificação quadro/campo adaptativa de macrobloco (a) par de macroblocos tipo quadro, (b) par de macrobloco tipo campo

A decisão de se escolher entre os modos de quadro ou campo é tomada no nível de par de macrobloco, diferente do MPEG-2 que particiona o macrobloco em dois. Essa decisão foi tomada para se manter a estrutura básica da codificação e para permitir a compensação de movimento de áreas do tamanho do macrobloco.

Cada macrobloco de um par de macrobloco campo é codificado de maneira bastante similar a um macrobloco convencional dentro da codificação PAFF. Todavia, como a mistura de pares de macroblocos nos modos campo e quadro pode ocorrer dentro de uma imagem MBAFF, os métodos utilizados para

1. Scan zig-zag
2. Predição dos vetores de movimento
3. Predição dos modos intra-quadros
4. Predição das amostras intra-quadros
5. Filtragem da redução do efeito de bloco
6. Representação do contexto na codificação de entropia

são modificados para tratar essa mistura. A idéia principal é manter, tanto quanto possível, a consistência espacial.

Uma outra diferença importante entre MBAFF e PAFF é que no MBAFF um campo não pode usar os macroblocos do outro campo na predição de movimento, pois estes ainda não foram codificados.

3.4 Implementações em hardware do H.264

Essa seção revisa alguns trabalhos presentes na literatura sobre implementações em *hardware* do compensador de movimento para decodificadores de vídeo do padrão H.264. Implementações em *hardware* do decodificador completo também são apresentadas nessa seção.

Em (WANG et al., 2005) é apresentada uma arquitetura VLSI do compensador de movimento para o perfil Baseline, nível 4.0. A arquitetura é composta por um preditor de vetores de movimento, um interpolador, que implementa reuso de dados, e uma memória externa. O preditor de vetores de movimento é implementado em *hardware* e está de acordo com o padrão.

Os autores, nesse trabalho, propõem uma nova arquitetura para os interpoladores. O interpolador de luma, para blocos de 4x4, utiliza a propriedade da separabilidade do filtro 2-D para implementar o interpolador a partir de filtros 1-D. A solução apresentada separa o filtro 2-D em filtros interpoladores verticais e horizontais. Os registradores dos filtros são substituídos por *buffers* para armazenar as entradas. O armazenamento das entradas visa a reutilização dos dados, reduzindo o acesso a memória externa. Os autores afirmam que 30% dos acessos à memória externa são economizados, quando comparados com a interpolação sem reutilização de amostras.

Para os interpoladores de croma, os autores apresentam uma arquitetura que não utiliza multiplicadores para interpolar as amostras.

A arquitetura foi sintetizada para *standard cell*, empregando a tecnologia UMC 0.18 μm, utilizando 43 mil *gates*, sem considerar a memória para armazenar os quadros de referência. O interpolador, sozinho, requereu 20 mil *gates*. O trabalho não apresenta a latência completa do compensador de movimento, apenas a do interpolador, que, no pior caso, consome 560 ciclos para o processamento de um macrobloco.

A arquitetura apresentada decodifica, em tempo real, seqüências de vídeo compactadas no padrão H.264 no formato HDTV de 1080 por 1920, operando a 100 MHz.

Em (LIE et al., 2005) é proposta uma alternativa para a interpolação das amostras de luma, utilizando filtros de 4 *taps* no lugar dos filtros de 6 *taps* da norma. A alternativa também reduz a quantidade de etapas de filtragem do padrão, passando das 3 definidas, para

2. Uma outra contribuição do artigo é a eliminação das multiplicações na interpolação das amostras de croma.

Para a filtragem das amostras de luma, os autores pesquisaram coeficientes para um filtro de 4 *taps* que tivesse resultados semelhantes ao filtro adotado no padrão H.264. Através de experimentos, os coeficientes (-1/8, 5/8, 5/8, -1/8) foram escolhidos, uma vez que teve uma performance, em PSNR, similar ao filtro do padrão.

O trabalho, para reduzir o número de etapas de filtragem, apresenta uma alternativa para o cálculo da posição J, a amostra interpolada para a posição 0.5, 0.5. Como visto, essa posição é calculada a partir de amostras de metade de pixel. Para eliminar essa dependência, o trabalho propõe o uso das amostras inteiras diagonais para o cálculo de J.

Para as amostras de croma, o trabalho propõe uma estratégia recursiva em 3 passos para realizar a interpolação. Em cada passo o quadrado formado pelas amostras inteiras é dividido em quatro quadrantes, sendo os vértices dos quadrantes resultados de médias simples das posições dos vértices do quadro original. No terceiro passo do algoritmo, a amostra é computada como uma média simples dos vértices do quadrante.

A arquitetura proposta reduz a complexidade do compensador de movimento, utilizando menos camadas de cálculo e filtros com menos *taps*. A arquitetura também elimina os multiplicadores para o cálculo da interpolação das amostras de croma. Como a arquitetura não obedece ao padrão, um erro é adicionado ao processo, reduzindo a qualidade da imagem em 0,66 dBs PSNR em média. Esses resultados dizem respeito a vídeos codificados para transmissão com taxa de média a baixa e parâmetro de quantização, QP, maior que 25.

Não são apresentados resultados de síntese para a arquitetura, sendo o ganho em área para o *hardware* dado em utilização de somadores, registradores e deslocadores.

Em (WANG; LI; HUANG, 2005) são apresentadas quatro estratégias de melhoria da quantidade de acessos à memória externa para a interpolação para quarto de pixel na compensação de movimento do padrão H.264.

A primeira estratégia diz respeito à estruturação em árvore da compensação de movimento. Os autores sugerem à leitura da janela de filtragem correspondente a interpolação de uma partição 8x8, ao invés de acessar a memória externa para cada bloco 4x4. Com essa estratégia diminui-se a quantidade de leitura de amostras redundantes, devido ao tamanho das bordas para os blocos 4x4.

A segunda estratégia sugerida influi na metodologia de implementação do interpolador. A estratégia consiste na implementação de um filtro que calcule apenas as posições que serão utilizadas, chamado, no trabalho, de esquema direto de interpolação. Dessa forma, serão buscadas na memória apenas as amostras necessárias à interpolação da posição determinada. Dependendo da posição a ser interpolada, para um bloco 4x4 são necessárias janelas de 4x4, 4x9, 9x4 ou 9x9.

Outra estratégia apresentada é o armazenamento, na memória de quadros de referência, das componentes de croma de modo intercalado. As amostras de croma das componentes Cb e Cr compartilham a posição no quadro, dessa forma, os acessos para buscar as componentes de croma na memória seriam realizados para janelas de dados mais larga.

A última estratégia apresentada é a leitura das amostras de luma e de croma através de canais de memória separados.

O trabalho apresenta uma diminuição na utilização do barramento de memória média de 60%, para os casos de testes.

Em (WANG et al., 2005) é apresentada uma arquitetura paralela e *pipeline* para o filtro interpolador de quarto de pixel para luma, uma outra arquitetura para a interpolação

de croma sem multiplicação e a implementação de estratégias para redução do acesso à memória externa.

O filtro interpolador de luma utiliza uma abordagem diferente das anteriores. A interporlação é realizada com uma amostra compensada por ciclo. Para tal, fazem parte do interpolador de luma: um registrador de entradas com capacidade para armazenar nove amostras; uma matriz de 6x9 registradores, para armazenar as amostras de referência para um bloco 4x4; um controlador para o *pipeline* e uma estrutura de *pipeline* de 5 estágios. O *pipeline* para a interpolação das amostras é composto por oito filtros FIR de 6 *taps*, sendo sete presentes no segundo estágio de pipeline e o oitavo filtro no terceiro estágio. O quarto estágio é formado pelo clipping e pelo cálculo da média para a resolução de quarto de pixel. Os filtros são implementados a partir de somas e deslocamentos.

A arquitetura do compensador de croma utiliza de diversas transformações na estrutura da fórmula da interpolação. Por decomposição da fórmula, é encontrada uma estrutura única para o cálculo, que é realizado em uma árvore, com a estrutura replicada 3 vezes. A multiplicação presente na estrutura é substituída por um multiplexador que cobre os possíveis casos da multiplicação.

A arquitetura utiliza as três primeiras estratégias de diminuição do acesso à memória de referência citadas nem (WANG; LI; HUANG, 2005). A implementação das estratégias de melhoria na interpolação de luma e croma tiveram resultados semelhantes aos previamente apresentados.

A arquitetura apresentada interpola um macrobloco em 492 ciclos, sendo 356 para as amostras de luma e 136 para as amostras de croma.

A síntese da arquitetura do interpolador para *standard cell*, usando tecnologia 0.18 μm, consumiu 4.992 células. A freqüência alcançada foi de 142 MHz. Para a decodificação de vídeo no formato HDTV 1080 por 1920, a arquitetura precisa operar a 150 MHz.

Em (HU et al., 2004) é apresentado um ASIC para decodificação de vídeo H.264 para o perfil Main no nível 4.0. O artigo não apresenta detalhes arquiteturais, limitando-se a explicar as características de cada bloco. O artigo apresenta uma avaliação dos requisitos de banda de memória para a memória de quadros de referência para diferentes tipos de memória. Os requisitos de memória total para a implementação também é levantado. A solução em silício para o decodificador foi sintetizada e, utilizando *standard cell* na tecnologia TSMC 0.13 μm, consegue operar com uma freqüência de relógio de 200 MHz. A síntese utilizou 300 mil *gates*, além de 74kB de memória interna e 24MB de memória externa ao *chip*. O consumo de potência do *core* é de 160mW.

Já em (LIN et al., 2005) é apresentado um decodificador H.264 com *pipeline* no nível de blocos 4x4. O decodificador apresentado suporta a decodificação de vídeos HDTV no formato de 1080 por 1920 em tempo real, operando com uma freqüência de relógio de 100 MHz, para o perfil Baseline no nível 4.0.

O trabalho apresenta a arquitetura do *pipeline*, onde os blocos inter-quadro, intra-quadro trabalham em paralelo, e paralelo a estes, os blocos da decodificação de entropia, seguido pelo *rescalling* e pela transformada inversa. Cada um dos três caminhos de dados foi ajustado de modo a terem latências idênticas. A partir do somador, que soma os resíduos com as amostra preditas, a arquitetura assume um *pipeline* de macrobloco.

O trabalho faz uma análise da ordem de decodificação das amostras. Ao adotar colunas de 4 amostras como unidade básica de processamento, os autores afirmam uma economia de 17% no acesso à memória do bloco da predição intra-quadro e 28% para o bloco da predição inter-quadro. O artigo não apresenta dados seja de síntese RTL, seja

de síntese lógica. A latência e taxa de processamento também não são reportadas pelos autores.

A ST Microelectronics (STMICROELECTRONICS HOME, 2005), em (STMICROELECTRONICS, 2005), anunciou o primeiro chip decodificador de H.264 para HDTV. A arquitetura decodifica *bitstreams* H.264 no perfil High no nível 4.0. O chip trabalha a uma freqüência de relógio de 300 MHz. Não foram encontrados maiores detalhes arquiteturais do mesmo.

Este capítulo apresentou a compensação de movimento do padrão H.264. Foram apresentadas as ferramentas que o compõe e seu funcionamento. No capítulo 4 será apresentada a arquitetura desenvolvida para realizar a compensação de movimento para o perfil Main, no nível 4.0.

4 ARQUITETURA DO COMPENSADOR DE MOVIMENTO

Este capítulo apresenta a arquitetura do compensador de movimento para o decodificador de vídeo segundo o padrão ITU-T H.264. A arquitetura descrita nesse capítulo, denominada MoCHA (*Motion Compensator Hardware Architecture*), foi desenvolvida para o perfil Main do padrão, apresentado no capítulo 2 dessa dissertação.

A arquitetura apresentada é parte do estudo da viabilidade de se adotar o padrão de codificação de vídeo H.264 como o padrão de compressão de vídeo do padrão brasileiro de vídeo digital. Esse estudo de viabilidade passa por desenvolver um protótipo do codificador e do decodificador de vídeo em *hardware* que atenda os requisitos de tempo real. O protótipo deve estar apto a atender esses requisitos para vídeos de alta definição (HDTV), que podem chegar ao formato de 1080 por 1920 a 30 quadros por segundo.

Para o perfil Main a arquitetura deve ser capaz de processar um *bitstream* H.264 com as seguintes características: sub-amostragem de cor fixa no formato 4:2:0; tratamento de *slices* dos tipos P e B; predição direta; predição ponderada e vídeo entrelaçado.

A arquitetura foi projetada para alcançar taxa de processamento suficiente para decodificar vídeos de alta definição (HDTV), no formato de até 1080 por 1920, a 30 quadros por segundo. Foram definidas algumas diretivas arquiteturais a serem seguidas: taxa de saída de uma amostra compensada por ciclo de relógio; freqüência de operação de 100 MHz e sincronização com os demais componentes do decodificador no nível de macrobloco. Para atender os requisitos, adotou-se uma janela de 384 ciclos para a compensação de um macrobloco.

Os dados de entrada da arquitetura do compensador de movimento são originários de duas fontes distintas. A primeira fonte é o *parser*, que trabalha em conjunto com a decodificação de entropia. Esses blocos separam e descompactam os parâmetros a serem utilizados pelo compensador de movimento, para a decodificação. A segunda fonte é o *buffer* de quadros de referência, que armazena os quadros previamente decodificados. O *buffer* de quadros de referência fornece as amostras a serem compensadas, de acordo com os parâmetros fornecidos pelo *parser*.

A arquitetura do compensador de movimento, aqui apresentada, não cobre o gerenciamento das listas de referência e as operações de alimentação e retirada de quadros do *buffer* de quadros de referência. Essas operações são delegadas ao controle geral do decodificador ao qual o MoCHA estará inserido. Algumas funções, dada essa delimitação do escopo, foram repassadas ao controle geral, como a conversão do índice de referência para identificador de quadro.

A compensação de movimento pode ser separada em três blocos principais: predição; *buffer* de quadros de referência e processamento das amostras. Da predição fazem parte a predição dos vetores de movimento, a predição dos índices de referências e parâmetros da predição ponderada. O *buffer* de quadros, além de armazenar os quadros de referência,

deve tratar os casos em que os vetores de movimento apontam para áreas fora das bordas da imagem. A parte de processamento das amostras inclui a interpolação para quarto de pixel, a predição ponderada e a média dos blocos preditos, para o caso da bi-predição. A Figura 4.1 apresenta o modelo do compensador seguido nessa dissertação.

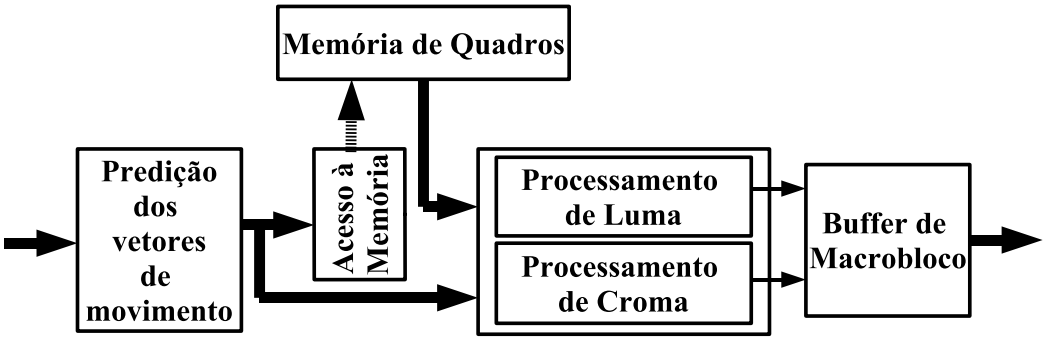

Figura 4.1: Estrutura do compensador de movimento

A arquitetura da MoCHA foi desenvolvida na forma de um *pipeline*. Cada um dos blocos principais perfaz um estágio de *pipeline*. Entre os blocos estão presentes *buffers* para armazenar os resultados, permitindo a independência entre os blocos.

As predições recebem como entrada parâmetros presentes no *bistream*, decodificados pela decodificação de entropia. Estes parâmetros podem corresponder a informações relativas ao *slice* e ao macrobloco corrente, além do formato do quadro. Para cada novo *slice* a predição deve receber informações referentes ao tipo de *slice*, ao tipo de predição direta e as informações da predição ponderada. Para o macrobloco corrente a predição recebe o tipo do macrobloco, o tipo de cada partição de sub-macrobloco, os índices de referência de cada lista de referência e os vetores de movimento diferenciais para cada lista de referência. As saídas das predições incluem os vetores de movimento, os índices de referência e as *flags* de utilização das listas de referência. A predição dos parâmetros da predição ponderada computa os pesos e ajustes aditivos para cada quadro de referência apontado pelo quadro atual.

O *buffer* de quadros de referência recebe os vetores de movimento, na resolução de pixel inteiro, a localização do bloco, dentro do quadro corrente, o índice da lista de referência e a *flag* de utilização de cada lista. Os vetores de movimento são somados à posição do bloco corrente e ao deslocamento referente às bordas utilizadas para a interpolação de quarto e oitavo de pixel. Os índices das listas são convertidos em contador de ordem de imagem (*picture order counter* - POC), através de consulta ao gerenciador de listas. Com o vetor de movimento ajustado e o POC do quadro de referência, são localizadas, na memória, as amostras pertencentes à janela de filtragem. As amostras são, então, enviadas ao processamento das amostras.

A parte fracionária dos vetores de movimento (vinda da predição) os pesos da predição ponderada, as *flags* de utilização das listas e as amostras enviadas pelo *buffer* de quadros de referência constituem as entradas do processamento das amostras. Sobre as amostras de luma é realizada a interpolação de quarto de pixel. Sobre as amostras de croma é realizada a interpolação de oitavo de pixel. O resultado das interpolações é ponderado e, caso a área utilize a bi-predição, é realizada uma média, uma a uma, das amostras processadas da lista 0 com as da lista 1, essas listas foram definidas na seção 3.1.3 destes

texto. Por fim, uma verificação é realizada para se averiguar se os valores das amostras se encontram no intervalo de valores permitidos pelo padrão. Os valores de saída do compensador de movimento são armazenados em um *buffer*, para sincronização com os demais componentes da arquitetura do decodificador H.264.

O controle geral da arquitetura é composto por um conjunto de máquinas de estados que sincronizam os blocos do compensador de movimento. Como cada módulo apresenta latências variáveis, são necessários mecanismos que controlem o fluxo de dados entre eles.

A arquitetura foi desenvolvida tendo como plataforma de prototipação alvo os dispositivos FPGA da Xillinx (XILINX: THE PROGRAMMABLE LOGIC COMPANY, 2005). A família de dispositivos escolhida foi a Virtex II Pro (VIRTEX SERIES, 2005), sendo o dispositivo XC2VP30-7 o disponível na plataforma de prototipação. A ferramenta utilizada para a síntese foi o ISE (PLATFORM STUDIO AND THE EDK, 2005), tendo como sintetizador o Symplify Pro (SYNPLICITY: PRODUCTS: SYNPLIFY PRO, 2005) da Synplicity (SYNPLICITY: HOME, 2005). O desenvolvimento da arquitetura e a sua descrição levaram em consideração as características do dispositivo e das ferramentas de síntese. Os resultados de síntese apresentados durante o capítulo foram extraídos dessas ferramentas. Para todas as sínteses, a restrição de freqüência de 100 MHz foi utilizada como parâmetro, com excessão da síntese da arquitetura completa.

O compensador de movimento para o decodificador de vídeo H.264 foi desenvolvido inteiramente em VHDL estrutural, com exceção do preditor dos vetores de movimento, que, por sua característica seqüencial, utilizou um misto de estrutural com comportamental e das máquinas de estados, que utilizaram descrição comportamental. Apesar da plataforma de prototipação definida, o VHDL foi escrito de forma a utilizar os mecanismos das ferramentas de CAD, que mapeam certas estruturas da linguagem em componentes presentes no dispositivo alvo. Desse modo obteve-se um VHDL portável, não se limitando a uma determinada ferramenta de síntese ou bibliotecas de componentes de empresas de FPGA. A descrição pode, inclusive, ser mapeada para *standard cell*. A descrição foi feita de forma hierárquica e visando o reaproveitamento dos componentes, sendo eles parametrizáveis.

A descrição do compensador de movimento está distribuída num total de 30 arquivos VHDL. Cerca de 13.500 linhas de código VHDL foram escritas para descrever o compensador de movimento.

A síntese do compensador de movimento, para o dispositivo XC2VP30-7 da Xilinx, consumiu 8.465 *slices*, 5.671 LUTs, 21 blocos de memória interna e 12 multiplicadores, sendo utilizado 61% dos *slices*, 39% das LUTs e 15% dos blocos de memória interna. Foram utilizados 1.197 pinos de entrada e saída, sem contar os pinos de alimentação. A freqüência máxima de operação, indicada pelo sintetizador Synplify, foi de 100 MHz. Considerando essa freqüência de operação, um quadro de 1080 por 1920, formado apenas por macroblocos P, é processado em 0,016 ms enquanto um quadro totalmente bi-preditivo leva 0,027 ms para ser processado, sem considerar o atraso do acesso à memória externa. Dessa forma, podem ser processados até 36,7 quadros totalmente bi-preditivos de 1920 por 1080 por segundo.

A presente versão da arquitetura possui algumas limitações. No que se refere à implementação completa das funcionalidades do padrão H.264 no perfil Main. A primeira limitação diz respeito à divisão da imagem corrente em *slices*. A arquitetura atual considera a existência de apenas um *slice* por imagem a ser codificada. As ferramentas de tratamento de seqüências de vídeo entrelaçadas não estão presentes na descrição atual.

Assim, apenas seqüências de vídeo entrelaçadas codificadas no modo quadro estão aptas a serem decodificadas pela arquitetura apresentada. A predição direta temporal também não foi implementa. Os detalhes das limitações, e os motivos que as impuseram, serão detalhados na parte do texto que descreve o componente da arquitetura onde a mesma se encontra.

Este capítulo está dividido em oito seções. A primeira seção apresenta a metodologia adotada para o desenvolvimento da arquitetura. A segunda apresenta a arquitetura do preditor de vetores de movimento, índices e *flags* de utilização das listas de referência. A terceira seção descreve o preditor de parâmetros da predição ponderada. A seção 4.4 apresenta a arquitetura do *buffer* de quadros de referência. A seção 4.5 apresenta, de forma global, a arquitetura do processamento de amostras da compensação de movimento. A seção 4.6 descreve o controle geral do compensador de movimento. A seção 4.7 apresenta os resultados da arquitetura e compara com implementações encontradas na literatura. A seção 4.8 apresenta as considerações finais e encerra o capítulo.

4.1 Metodologia de Desenvolvimento

Para o desenvolvimento da arquitetura do compensador de movimento, partiu-se da norma da ITU-T e do *software* de referência JM 9.5 (H.264/AVC SOFTWARE COORDINATION, 2005). Foram utilizadas diferentes metodologias de desenvolvimento, de acordo com as características do componente a ser desenvolvido.

O desenvolvimento do compensador de movimento se iniciou pelo particionamento do mesmo. O particionamento em três blocos é resultado da própria estrutura da norma. O padrão trata as predições e o processamento das amostras de forma separada. O processo de decodificação da predição inter-quadro é apresentado na seção 8.4 da norma, sendo as predições dos vetores de movimento e índices de referência descritas na subseção 8.4.1 e o processo de decodificação das amostras na subseção 8.4.2. Para a implementação em *hardware* foi adicionado um terceiro bloco, o *buffer* de quadros de referência, que não é tratado na norma. Definido o particionamento, partiu-se para a definição da metodologia de implementação dos blocos separadamente.

O bloco do processamento das amostras foi o primeiro a ser desenvolvido por ter sido considerado o bloco crítico no que diz respeito ao desempenho. A investigação de uma solução arquitetural passou pelo estudo do *software* de referência, da norma e da literatura sobre implementação em *hardware* deste. O estudo apresentou, para o processamento das amostras, uma estrutura de fluxo de dados, sendo possível o mapeamento direto das operações num fluxo de dados. Foi então desenvolvida diretamente uma solução arquitetural a partir dos casos apresentados na literatura, adaptando-os para o cumprimento dos requisitos especificados.

Para o bloco da predição dos vetores de movimento foi desenvolvida uma solução arquitetural partindo-se diretamente da norma. Essa abordagem se deve à característica intrinsecamente seqüencial das predições e pela falta de trabalhos que relatassem soluções arquiteturais de forma detalhada.

Após um estudo detalhado da norma, foi desenvolvido um programa em C para mapear o comportamento da predição. Como a predição dos vetores de movimento e índices de referência se encontra espalhada por diversas funções do código de referência, não foi possível a utilização do mesmo. Outro fator que impediu a utilização do código de referência foram as estruturas de dados utilizados, que inviabilizariam uma implementação em *hardware*. O programa em C foi desenvolvido de modo a ser um modelo para a im-

plementação em *hardware*. Dessa forma, as estruturas de dados utilizadas no programa foram estruturadas para serem passíveis de uma implementação em *hardware*. Com as estruturas de dados definidas, foi desenvolvido o núcleo do programa. Como resultado, obteve-se o comportamento das predições mapeado em, aproximadamente, 1.300 linhas de código C. Validado o programa, o mesmo serviu de guia para o desenvolvimento da descrição em HDL do preditor de vetores de movimento.

O bloco de acesso à memória foi desenvolvido tendo em vista a interconexão entre os blocos da predição e do processamento de amostras. O mesmo também foi o alvo das implementações para o implementação do *pipeline* no nível dos blocos principais. Para tal, *buffers* foram inseridos para armazenar as saídas da predição, assim como na entrada do processamento de amostras.

O controle geral da arquitetura foi dividido em diversas máquinas de estado para gerenciar cada estágio.

4.2 A Arquitetura do Preditor de Vetores de Movimento

Essa seção descreve as arquiteturas dos preditores dos vetores de movimento. A predição dos vetores de movimento restaura os valores dos vetores de movimento a partir dos vetores de movimento diferenciais, presentes no *bistream*, e dos preditores de vetor de movimento (PMV).

Foram desenvolvidas, de forma incremental, duas arquiteturas para a predição dos vetores de movimento onde, após a validação da primeira eram adicionados novos recursos, requeridos pela segunda. A primeira implementação trata apenas a predição padrão para *slices* tipo P. A segunda implementação adiciona à primeira os recursos para a predição direta espacial, bem como o tratamento dos vetores de movimento da lista 1, ambos utilizados pelos *slices* do tipo B. A implementação da versão para a predição direta temporal e a implementação das ferramentas de tratamento de vídeos entrelaçados não foram desenvolvidas no escopo desse trabalho e não estão disponíveis na versão atual da arquitetura MoCHA. Essas ferramentas não foram desenvolvidas em função da necessidade de finalização imediata deste mestrado, não restando tempo disponível para tais desenvolvimentos.

O tratamento de vídeo entrelaçado requer tratamento especial na predição dos vetores de movimento e dos índices de referência. A complexidade do processo tem um aumento significativo, uma vez que se deve prever todas as possibilidades com combinações entre as diversas formas de codificação dos macroblocos, suportadas pelas ferramentas para o vídeo entrelaçado. Cada macrobloco na memória de referências deve receber sinais que indicam qual opção de codificação foi utilizada. Dependendo do tipo de macrobloco atual e do tipo de macrobloco referenciado, transformações são realizadas nos vetores e nos índices. Dessa forma, não foi possível a implementação dessas ferramentas nos preditores descritos a seguir.

Cada arquitetura implementada será detalhada nas próximas subseções.

4.2.1 Arquitetura do preditor de vetores de movimento padrão

A arquitetura desenvolvida para a predição dos vetores de movimento de *slices* do tipo P suporta a predição padrão dos vetores de movimento e o tratamento dos macroblocos *skip*. Nesta versão do preditor de vetores de movimento, apenas os vetores de movimento e índices de referência pertencentes à lista 0 são tratados.

O processo de predição dos vetores de movimento, para cada partição de macrobloco

ou de sub-macrobloco, se inicia pela verificação da disponibilidade dos blocos vizinhos e pela seleção dos vetores de movimento destes blocos. Dependendo da disponibilidade e dos índices de referência dos vizinhos, seleciona-se se a mediana dos blocos vizinhos ou um dos vetores de movimento para ser somado ao vetor de movimento diferencial. O resultado é, então, armazenado nos blocos pertencentes à partição de macrobloco ou sub-macrobloco corrente.

A arquitetura foi descrita em VHDL utilizando um misto de descrição comportamental com descrição estrutural. Essa abordagem foi utilizada tendo em vista uma implementação eficaz das estruturas de dados definidas no modelo em C e para proporcionar ao sintetizador a liberdade de reuso dos componentes utilizados.

O preditor de vetores de movimento e índices de referência recebe em suas entradas, para cada novo macrobloco, o tipo do *slice*, o tipo do macrobloco e os quatro tipos dos sub-macrobloco. Dependendo do tipo de macrobloco e dos tipos de sub-macroblocos, são requisitados os vetores de movimento para cada partição de macrobloco ou de sub-macrobloco e os índices de referência para cada partição de macrobloco. Essa requisição é realizada apenas quando os vetores são necessários.

Os vetores de movimento e os índices de referência dos blocos 4x4 vizinhos da linha acima, da esquerda e do canto superior esquerdo do macrobloco corrente, tal qual apresentado na Figura 4.2, devem ser armazenados, bem como os vetores e índices dos blocos pertencentes ao próprio macrobloco. A arquitetura do preditor de vetores de movimento e índices de referência para *slices* do tipo P conta com dois *buffers* e pelos conjuntos de registradores, que armazenam os dados dos blocos vizinhos.

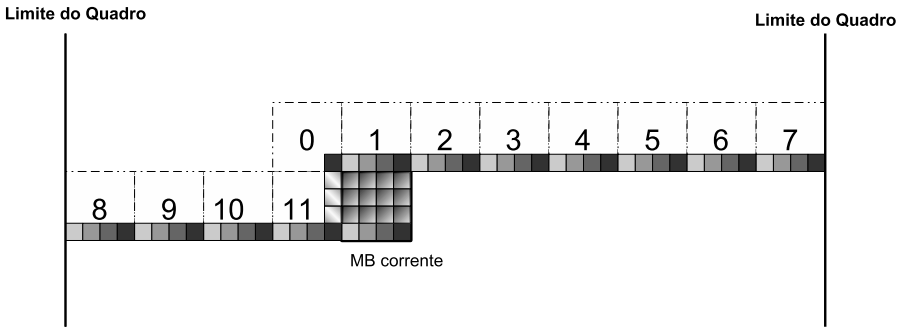

Figura 4.2: Armazenamento das informações dos blocos vizinhos

Para armazenar os vetores de movimento e índices de referência pertencentes à linha acima do macrobloco corrente a arquitetura conta com dois *buffers*, implementados como blocos de memória interna do FPGA. Um *buffer* de 512 palavras de 32 bits armazena os 480 vetores de movimento pertencentes a cada um dos blocos 4x4 de uma linha de blocos de um quadro HDTV de 1920 pixels de comprimento. O segundo *buffer* armazena os 240 índices de referência.

A arquitetura também conta com 10 conjuntos de registradores para o armazenamento dos vetores e índices vizinhos e internos do macrobloco corrente. A Figura 4.3 representa a disposição dos conjuntos de registradores que armazenam os vetores de movimento. As caixas com letras, na Figura 4.3, representam os vizinhos a qual armazenam os dados. As caixas com fundo em cinza representam os registradores que armazenam os vetores de movimento do próprio macrobloco. Dois conjuntos com 4 registradores armazenam os vetores de movimento imediatamente à esquerda e acima, vindos do *buffer*, do macrobloco

corrente. Esses vetores são referentes aos vizinhos A e B do macrobloco corrente, respectivamente. Dois registradores armazenam os vetores do bloco vizinho superior esquerdo e do superior direito, referentes aos vizinhos D e C, respectivamente. Um conjunto de 16 registradores armazena os vetores de movimento dos blocos do macrobloco corrente.

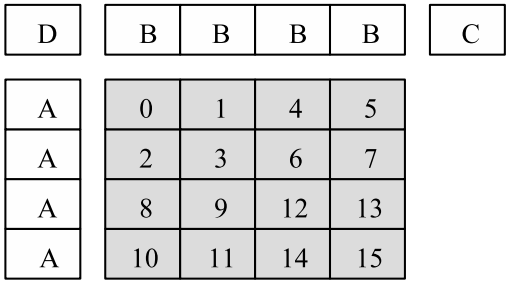

Figura 4.3: Registradores de armazenamento dos vetores de movimento

Os outros cinco conjuntos de registradores armazenam os índices de referência dos blocos vizinhos. Como cada partição de macrobloco (de até 8x8) compartilha as referências, são necessários menos registradores para armazenar as os índices de referência dos blocos vizinhos, tal qual representado na Figura 4.4.

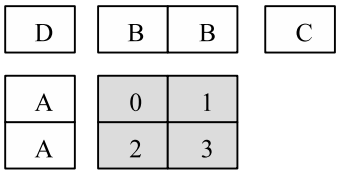

Figura 4.4: Registradores de armazenamento dos índices de referência

Uma máquina de estados implementa as operações sobre os dados. A máquina de Mealy implementada tem 33 estados. Um macrobloco pode ser completamente dividido em partições de 4x4. E, para cada uma dessas partições, é necessária à computação dos vetores de forma seqüencial. Assim, a máquina de estado deve processar até 16 partições, incluindo os ciclos de leitura e escrita, nos 384 ciclos requeridos para o processamento de um macrobloco. O particionamento dos estados foi então guiado por essa restrição na quantidade de ciclos e do período do relógio.

Ao receber o sinal de novo macrobloco, a máquina testa se o macrobloco é do tipo inter ou intra, consultando o tipo de *slice* e o tipo do macrobloco. Testado o tipo do macrobloco, a máquina entra no teste do loop das partições do macrobloco, onde é lido o índice de referência da partição, caso não seja um macrobloco *skip* ou intra. O estado seguinte implementa o teste do loop das partições de sub-macrobloco. Neste estado, são testadas as dimensões da partição corrente, que vão determinar os blocos vizinhos. As dimensões são consultadas em duas tabelas, uma para o caso do macrobloco não ter sub-macroblocos, e outra caso contrário. A posição do bloco também é atualizada neste estado.

Com as dimensões e a posição do canto superior esquerdo da partição do quadro, é avaliada a disponibilidade dos vizinhos. Se a posição horizontal for diferente de ‘0’ então o vizinho A está disponível. Se a posição vertical for diferente de 0 o vizinho B está disponível. Se a posição horizontal mais a largura da partição ou sub-partição

for menor que a largura do quadro, o vizinho C está disponível. Se A e B estiverem disponíveis, então D também está. Para se tratar múltiplos *slices* num mesmo quadro, cada macrobloco teria que ter uma identificação, informando a qual *slice* pertence. Os testes acima compariam o *slice* atual com o *slice* do vizinho, caso fossem diferentes, o vizinho não estaria disponível.

Como o vizinho C pode ser um dos blocos internos do macrobloco corrente e a ordem de decodificação obedece ao duplo Z, a disponibilidade de C depende de um segundo teste. Esse novo teste verifica, com base na posição e nas dimensões da partição, se C já foi ou não decodificado. Caso o mesmo já tenha sido decodificado é mantido o valor da disponibilidade previamente calculado, caso contrário ele é identificado como não disponível.

De acordo com a posição da partição corrente, e de suas dimensões, são selecionados os vetores de movimento e índices de referência. Num estado são calculados os seletores dos multiplexadores que selecionam de qual conjunto de registradores e de qual registrador especifico do conjunto, no estado seguinte, o dado é copiado para registradores específicos. Como os seletores dos multiplexadores são registrados, enquanto se copia os dados entre os registradores novos seletores são calculados para o vizinho seguinte. Essa operação está distribuída em 5 estados.

O estado seguinte zera os registradores dos vizinhos que estejam marcados como não disponíveis. Se C estiver marcado com não disponível o valor de D é copiado para ele. O mesmo acontece para os registradores de índice, mas em vez de serem zerados é atribuído -1 a eles.

São quatro os tipos de seleção possíveis para o PMV: a mediana ou a cópia do vetor de movimento de cada um dos 3 vizinhos. Caso uma partição possua apenas um dos vizinhos com o índice de referência igual ao seu, o vetor de movimento do vizinho é selecionado como PMV. Em um estado se compara o índice dos vizinhos com o da partição corrente e tipo da seleção é, se necessário, modificado.

Outro fator que influi na escolha do PMV é o formato da partição. É realizado um teste que verifica se a partição se encontra em um dos formatos que geram a exceção apresentada na subseção 3.2.1 e, se necessário, o tipo de seleção do PMV é modificado. Neste estado também é calculada a mediana dos vetores de movimento dos vizinhos. A mediana é calculada antes para diminuir o caminho crítico de dados. Neste estado também é testado se os vizinhos A e B têm vetores de movimento e índices de referência iguais a zero. Em caso afirmativo a *flag* **zero_motion** é ligada.

A atribuição do PMV é realizada baseada no tipo de seleção definido pelos testes anteriores. Caso a seleção não tenha sofrido alteração pelos testes realizados, a mediana dos vetores de movimento dos blocos vizinhos é selecionada como PMV, salvo o caso em que os vizinhos B, C e D, não estiverem disponíveis. Para este caso é selecionado o valor do vetor do vizinho A.

No mesmo estado da seleção do PMV é selecionada a posição da próxima partição a ter seus vetores de movimento preditos.

Se o macrobloco for do tipo intra ou se a *flag* zero_motion estiver ligada para um *slice* P, os vetores de movimento da partição são zerados. Caso contrário o PMV é somado ao vetor de movimento diferencial e salvo no conjunto de registradores referentes aos vetores de movimento dos blocos corrente. De acordo com a posição da partição e suas dimensões, dois multiplexadores selecionam quais blocos na horizontal e na vertical devem ter seus sinais de habilitação de escrita ligados. Cada multiplexador seleciona um conjunto de 4 bits. Cada bit na horizontal está associado a uma coluna enquanto cada bit

na vertical está associado a uma linha. Um E lógico entre cada par de sinais horizontais e verticais liga a habilitação de escrita num registrador. A Figura 4.5 ilustra o mecanismo de habilitação de escrita. Cada intercessão entre as linhas representa um E lógico que habilita o registrador.

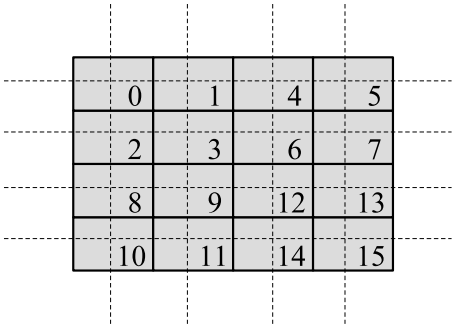

Figura 4.5: Estrutura de habilitação de escrita nos registradores

Depois de salvar os resultados nos registradores internos referentes a partição corrente, a máquina volta ao teste do *loop* de partições de sub-macroblocos que, por sua vez, passa para o estado do teste das partições de macrobloco, caso as partições de sub-macrobloco relativas à partição de macrobloco corrente já tenham sido percorridas. Se todas as partições de um macrobloco já tiverem sido percorridas, a máquina passa para os estados de atualização dos registradores dos vizinhos.

Nos estados de atualização, os conjuntos de registradores recebem os valores para o processamento do macrobloco seguinte. Os registradores referentes ao vizinho A recebem os valores dos registradores internos da borda direita do macrobloco. O registrador D recebe o valor do registrador B mais a direita. Os registradores B recebem os valores vindo do *buffer*, assim como o registrador C. Os vetores de movimento e índices de referência da última linha de blocos do macrobloco corrente são salvos nos *buffers*. Também nesses estados o conteúdo dos registradores internos, vetores e índices são enviados para a saída.

A Tabela 4.1 apresenta os resultados de síntese da predição para *slices* do tipo P. A freqüência máxima de operação foi de 111 MHz.

Tabela 4.1: Resultados de síntese da predição padrão

	Total	Percentual
Slices	1.929	14%
Flip Flops	2.216	8%
LUTs	2.028	7%
Blocos de RAM	2	1%
Multiplicadores	0	0%

O preditor de vetores de movimento e índices de referência padrão têm latência mínima de 13 ciclos para apresentar, na saída, o primeiro resultado válido do macrobloco predito. A latência máxima é de 219 ciclos de relógio. Após a saída do primeiro resultado válido, a arquitetura leva mais 16 ciclos para terminar de apresentar os resultados de um macrobloco.

4.2.2 Arquitetura do preditor de vetores de movimento da predição direta espacial

Desenvolvido o preditor para *slices* do tipo P, o mesmo foi incrementado para operar os *slices* do tipo B. Esse novo preditor opera com os vetores e índices pertencentes às duas listas, 0 e 1, com a seleção das predições de acordo com o tipo de *slice*, P ou B, e com a predição direta espacial, que também é usada para os blocos *skip* dos *slices* do tipo B.

Para o tratamento da lista 1 foram dobrados todos os conjuntos de registradores da arquitetura. Em cada estado da máquina de controle são, agora, tratados em paralelo os vetores de movimento e índices de referências de cada lista. Essa solução foi adotada uma vez que, com o particionamento dos estados, não é possível processar os 32 blocos de forma serial e satisfazer a restrição no número de ciclos para o pior caso.

Se um macrobloco, ou alguma de suas partições, utilizarem a predição direta, o fluxo da máquina de estados é desviado para tratar essa predição. Seis estados adicionais, inseridos antes dos loops, foram necessários para se calcular os índices de referência para a partição direta e a *flag* directPredictionZeroFlag. Essa *flag* indica se todos os vetores de movimento da partição devem ser zerados. O primeiro desses estados adicionais verifica a disponibilidade dos vizinhos e copia os índices de referências de cada um deles. No segundo estado, caso um vizinho não esteja disponível, ao seu índice de referência é atribuído -1. Nos dois estados seguintes é calculado, para cada lista, o menor índice não negativo dos vizinhos. Caso ambos os índices das listas 0 e 1 sejam negativos a *flag* directZeroPredictionFlag é ligada e os índices passam a conter o valor 0. Caso contrário, os índices da predição direta são os menores não negativos calculados.

Para os macroblocos pertencentes a um *slice* P existem 5 tipos de partições de macrobloco e 4 tipos de partições de sub-macrobloco. Nos *slices* do tipo B existem 23 tipos de partições de macrobloco e mais 13 tipos de partições de sub-macrobloco. Para cada tipo de partição de *slices* B está associada, além do formato da partição, a utilização dos vetores e índices para cada lista, ou seja, se a partição usa apenas a lista 0, apenas a lista 1 ou se usa ambas as listas (bi-predição). A *flag* de utilização também determina a leitura dos índices de referência de cada lista nas entradas do preditor. No teste do loop de partições é consultada a utilização das listas para a partição corrente e os índices são lidos, caso necessário. Se a partição for direta os índices de referência são os índices calculados no tratamento realizado para a predição direta.

As informações das dimensões das partições são carregadas no estado seguinte ao teste do loop das partições de sub-macrobloco. Se o macrobloco como um todo for do tipo direto, então o mesmo é dividido em 4 partições e cada uma delas em 4 sub-partições de 4x4. Se o macrobloco não for direto, mas alguma de suas partições o for, essa é dividida em sub-partições de 4x4. Cada um dos blocos 4x4 direto é predito segundo a ordem do duplo Z.

Caso o bloco 4x4 seja direto, o teste de disponibilidade dos vizinhos é realizado considerando-se o macrobloco inteiro como a partição corrente, não importando a posição do bloco 4x4 dentro do macrobloco.

Na predição direta espacial, além de se utilizar os vetores e índices dos blocos vizinhos do mesmo quadro, são utilizados os vetores e índices dos blocos co-localizados. O bloco co-localizado, no padrão H.264, é o bloco, pertencente ao quadro de referência de índice 0 da lista 1, o qual se encontra na mesma posição que o bloco corrente. Após a verificação da disponibilidade dos vizinhos, são solicitados, ao controle geral, os vetores de movimento e índices de referência dos blocos co-localizados. O quadro co-localizado depende da ordem da decodificação e as informações de vetores e índices de cada bloco, de um ou mais quadros, têm que estar disponíveis. Como o volume de dados é muito

grande para ser armazenado nos blocos de RAM interna do FPGA, esses são armazenados em memória externa sendo assim dependentes do mecanismo de acesso à memória. Com o recebimento dos vetores de movimento e índices de referência de ambas as listas, são realizados os testes das *flags* ColZeroFlag e VecZeroFlag. Essas *flags* determinam se os vetores do bloco corrente devem ser zerados ou devem ser preditos de acordo com a predição padrão.

Como visto na subseção 3.2.3 do capítulo 3, a ColZeroFlag é ligada caso o quadro de referência seja *long_term*, os vetores co-localizados entejam entre -1 e 1 e o índice de referência da lista 0 seja 0, ou o da lista 1, quando o da lista 0 for -1.

Existe um *flag* VecZeroFlag para cada lista. Se a *flag* directZeroPredictionFlag estiver ligada, se a referência co-localizada for igual a -1 ou se for igual a zero e ColZeroFlag estiver ligada a VecZeroFlag da lista avaliada é ligada. Neste estado são selecionados os vetores de movimento e índices de referência a serem copiados para os registradores que guardam os valores dos vizinhos (pertencentes ao quadro corrente), também realizada considerando-se o macrobloco inteiro como a partição corrente.

A execução então segue para o estado onde é testada a existência de apenas um vizinho com o mesmo índice que a partição corrente. A execução continua igual a predição padrão até o estado em que são salvos os vetores e índices de referência. Caso a partição seja uma partição direta e a *flag* VecZeroFlag de uma lista estiver ligada seus vetores de movimento a serem salvos são zerados, bem como o índice de referência.

A Tabela 4.2 apresenta os resultados de síntese da predição para *slices* do tipo B. A freqüência máxima de operação, para os dados da tabela, foi de 100 MHz.

Tabela 4.2: Resultados de síntese da predição

	Total	Percentual
Slices	4.552	33%
Flip Flops	4.649	16%
LUTs	4.947	18%
Blocos de RAM	3	2%
Multiplicadores	0	0%

A arquitetura do preditor, com suporte a predição direta espacial, utiliza mais que o dobro dos recursos utilizados pelo preditor padrão. Isso se deve a utilização das duas listas de referência em paralelo e aos mecanismos implementados para tratar a predição direta espacial.

O preditor de vetores de movimento e índices de referência com suporte a predição direta espacial tem latência mínima de 20 ciclos para apresentar, na saída, o primeiro resultado válido do macrobloco predito. A latência máxima é de 226 ciclos de relógio. Após a saída do primeiro resultado válido, a arquitetura leva mais 16 ciclos para terminar de gerar os resultados de um macrobloco. O tempo de processamento médio para processar as entradas referentes à 100 quadros da seqüência Foreman, no formato QCIF, foi de 123,5 ciclos por macrobloco.

4.3 A Arquitetura do Preditor dos parâmetros da predição ponderada

Cada índice de referência de cada lista tem seus próprios parâmetros da predição ponderada. Para cada índice de referência de cada lista, está presente, no *bistream*, a sinalização da utilização, ou não, da predição ponderada para o quadro de referência associado. Mesmo que a utilização da predição ponderada seja indicada no *bistream*, os parâmetros podem não estar presentes, caso da predição ponderada implícita. Para tratar esses casos foi desenvolvido um bloco para processar os parâmetros da predição ponderada.

O bloco de processamento dos parâmetros da predição ponderada recebe seus dados uma vez para cada novo *slice*. Para cada índice de referência, está presente no *bistream* uma *flag* que indica se o quadro de referência utiliza ou não a predição ponderada. O bloco deve, para cada índice de referência, tratar a utilização da predição e disponibilizar os parâmetros para o processamento das amostras.

O processamento dos parâmetros é tratado de duas formas: predição ponderada explicita (na qual a arquitetura trata do processamento dos parâmetros, que vêm explícitos no *bistream*) e predição ponderada implícita (utilizada apenas nos *slices* do tipo B, na qual o processamento define os parâmetros baseado na distância relativa dos quadros de referência).

Na predição ponderada explícita os parâmetros da predição ponderada estão presentes no *bistream*, cabendo a arquitetura tratar da existência ou não dos parâmetros e os enviar para a saída.

Na predição ponderada ímplicita os parâmetros devem ser calculados segundo as fórmulas 3.6 a 3.7, apresentadas na seção 3.6. Nesse tipo de predição, os parâmetros de deslocamento são fixados ao valor zero e o parâmetro LogWD é fixado ao valor 5. Portanto resta calcular os parâmetros multiplicativos W0 e W1.

Para obter os parâmetros multiplicativos na predição ponderada implícita, usa-se como base a distância temporal entre os quadros de referência e o quadro atual. Para calcular essas distâncias, são utilizados dois subtratores. As saídas dos subtratores passam por um operador de *clip*, são somadas à constante 16.384 e, em seguida, registradas. Uma divisão entre dois dos valores registrados é feita por um divisor multiciclo, de 15 bits, e seu resultado é novamente registrado. Este resultado é, então, multiplicado à outra das distâncias calculadas inicialmente, somado a uma constante e deslocado por um deslocador. Após o deslocamento, o dado é registrado novamente e passa por outro *clip* e outro deslocamento.

Um multiplexador seleciona o parâmetro W1 optando pela saída desse último deslocamento ou pela constante 32. O parâmetro W0 é selecionado por um segundo multiplexador. Este outro parâmetro será a constante 32 ou a constante 64 subtraída de W1. Estes dois multiplexadores fazem a seleção baseados na distância entre os quadros de referência e na saída do divisor, considerando, também, se as referências são do tipo *long_term*.

Para ambos os tipos de predição os parâmetros são armazenados em uma pequena *cache*. O nível 4.0 define a existência de até cinco imagens de referência armazenadas no *buffer* de quadros de referência. Assim, para um dado *slice*, não se utiliza mais que 10 índices de referência, sendo 5 para cada lista. Todavia, os índices podem assumir valores de -1 até 127. Para tratar esse comportamento, a solução encontrada foi a implementação de duas caches de 5 posições cada. A *cache* é alimentada pelo processamento dos parâmetros da predição e consultada pelo processamento das amostras. Um *miss* na *cache* indica o uso dos parâmetros padrão.

A Tabela 4.3 apresenta os resultados de síntese da predição para *slices* do tipo P.

Tabela 4.3: Resultados de síntese da predição implícita

	Total	Percentual
Slices	161	1%
Flip Flops	201	1%
LUTs	182	1%
Blocos de RAM	0	0%
Multiplicadores	1	0%

4.4 Buffer de Quadros de Referência

O *buffer* de quadros de referência armazena os quadros decodificados que serão utilizados pela compensação de movimento. A quantidade de quadros armazenados varia de acordo com o nível adotado. No caso do nível 4.0, adotado nessa dissertação, até 4 quadros HD de referência podem estar armazenados no *buffer* num dado momento, o que eqüivale a 12.228 KB.

O controle da escrita dos quadros no *buffer* de quadros de referência não faz parte do escopo dessa dissertação, sendo este delegado ao controle geral do decodificador. Foi também delegada ao controle geral, a gerência das listas de referência. Dessa forma, a conversão dos índices das listas para POC é realizada pelo gerenciador das listas de referência.

Como o dispositivo FPGA não comporta a quantidade de dados a serem armazenados pelo *buffer* de quadros de referência, foi definida, apenas, uma interface de comunicação com a memória, que implementará, de fato, o armazenamento dos quadros de referência. Essa estratégia foi necessária devido ao não tratamento do gerenciador de listas de referência por esse trabalho. Outro fator que contribuiu para essa decisão foi a reutilização da solução arquitetural. Cada placa de desenvolvimento pode ter diferentes tipos de memória externa ao dispositivo FPGA (SDRAM, DDR, etc) ou mesmo não possuir esse item, acarretando em mecanismos diferentes de acesso a memória. O mesmo aconteceria para um implementação VLSI da arquitetura.

Uma *cache* tridimensional foi definida para explorar a redundância espacial das áreas a serem processadas pelo processamento de amostras, tendo em vista a grande quantidade de amostras pertencentes à janela de filtragem.

O tratamento de áreas que ultrapassam a borda do quadro é tratado pelo *buffer* de quadros de predição. Os dados lidos da *cache*, caso seja necessário, passam por um replicador para atender a extrapolação definida no padrão.

Uma análise do impacto da utilização da *cache* é apresentado. Melhorias o acesso são sugeridas e avaliadas.

Faz parte do *buffer* dos quadros de referência: a interface de comunicação com a memória, a *cache* e o extrapolador das amostras. A *cache*, a ser apresentada na subseção a seguir, implementa, também, a interface de comunicação com a memória. Na subseção 4.3.2 é apresentado o processo de extrapolação das amostras. Os resultados da análise dos resultados da *cache* é apresentado na subseção 4.3.3.

4.4.1 Cache 3D de Amostras

Para cada bloco 4x4 de luma é processada uma área de 9x9. Para reduzir a comunicação com a memória, foi definida uma *cache* para aproveitar a redundância das áreas a

serem processadas.

Foi definida uma *cache* tridimensional para adequar a mesma a estrutura de dados e a redundância das áreas a serem processadas. Os dados a serem armazenados representam áreas de quadros de uma seqüência de vídeo, tendo-se então três dimensões: posições da amostra nos eixos 'x' e 'y' no quadro e a posição temporal do quadro na seqüência de vídeo, o POC. A *cache* também funciona como interface para a memória que armazena os quadros de referência.

A saída da *cache* é no formato de linha da área a ser processada, contendo as amostras das componentes de luma e das duas linhas componentes de croma. Isso se deve ao formato dos dados de entrada do processamento das amostras. Cada linha da área a ser processada é acessada a partir do POC do quadro, da linha da área a ser processada e da posição da coluna mais à esquerda da área. Para cada requisição, a estrutura da *cache* garante a presença de todas as amostras da linha, a partir da posição da coluna mais a esquerda da posição que for requisitada. O mesmo não ocorre para as linhas da área. Deste modo, para cada nova linha é necessária uma nova requisição à *cache*. As posições requeridas devem pertencer aos limites do quadro.

A *cache* foi dividida em 32 conjuntos com 40 colunas e 16 linhas cada, para componentes de luma. Para cada componente de croma, a *cache* está dividida em 32 conjuntos com 20 colunas e 8 linhas. Das 40 colunas da componente de luma apenas as 32 primeiras são registradas como presentes na *cache*, as últimas 8 posições são complementos que garantem que a linha solicitada esteja presente. O mesmo acontece para as duas componentes de croma, sendo as 16 primeiras posições registradas.

A configuração da *cache* foi baseada nos resultados de experimentos realizados. Foram decodificados 100 quadros da seqüência foreman.yuv no formato QCIF (144x174), codificados no perfil Main com intervalo de *slice* I-P-B. No processo de decodificação foram extraídos, para cada bloco e lista de referência, o POC, a posição da coluna e as posições inicial e final das linhas pertencentes a área de luma a ser interpolada. O mesmo foi realizado para o vídeo walkrun.yuv no formato SDTV (480x720).

Um simulador de *cache* foi então desenvolvido para explorar o espaço de projeto da *cache*. Para os dados extraídos da seqüência foreman foram executadas simulações para 32, 64 e 128 conjuntos. Para cada possibilidade de conjunto se testou todas as combinações da quantidade de linhas pertencentes ao conjunto [8, 16, 32] pelo conjunto das quantidades de colunas [24, 40, 72, 136]. A taxa de *misses* resultante dessas simulações para cada configuração se encontra nas Tabelas 4.4 a 4.6.

Tabela 4.4: Taxa de *misses* para 32 conjuntos

	24 Colunas	40 Colunas	72 Colunas	136 Colunas
8 Linhas	3,76%	2,31%	1,42%	0,97%
16 Linhas	1,85%	0,77%	0,47%	0,28%
32 Linhas	0,66%	0,34%	0,20%	0,10%

O número de conjuntos mostrou não ter um custo benefício satisfatório no resultado da taxa de *miss*. Com base nessa observação a opção por 32 conjuntos foi escolhida. Fixada a quantidade de conjuntos, foram realizadas simulações com os dados extraídos do vídeo no formato SDTV, para os mesmos parâmetros de coluna e linha. A taxa de *misses* resultante dessas novas simulações se encontra nas Tabelas 4.7.

Tabela 4.5: Taxa de *misses* para 64 conjuntos

	24 Colunas	40 Colunas	72 Colunas	136 Colunas
8 Linhas	2,94%	1,28%	0,77%	0,46%
16 Linhas	1,01%	0,55%	0,27%	0,12%
32 Linhas	0,60%	0,22%	0,09%	0,03%

Tabela 4.6: Taxa de *misses* para 128 conjuntos

	24 Colunas	40 Colunas	72 Colunas	136 Colunas
8 Linhas	1,83%	1,00%	0,44%	0,20%
16 Linhas	0,98%	0,31%	0,12%	0,06%
32 Linhas	0,36%	0,11%	0,05%	0,03%

A escolha da configuração do conjunto com 40 colunas e 16 linhas levou em consideração a taxa de *misses*, a quantidade de recursos utilizados e a latência para se copiar o conteúdo da memória para a *cache*. Devido a disposição dos dados na memória, colunas vizinhas têm tempo de acesso menor que linhas consecutivas. Assim uma área mais larga, mesmo que menos eficiente em termos de *misses*, é mais desejável. Para valores maiores que o escolhido o custo beneficio não é favorável, uma vez que se dobram os recursos de memória interna utilizados sem que se tenha uma redução de mesma ordem na taxa de *misses*. Também se aumenta o tempo de preenchimento do conjunto, para um caso de *miss*.

Devido a diferença na interpolação das componentes de luma com as de croma, as janelas de filtragem possuem tamanhos diferentes. Todavia, as requisições das linhas são realizadas sempre a partir das posições da componente de luma. As componentes de croma são posicionadas na saída, sendo a posição da coluna ajustada para o início das amostras de croma. As linhas da área de luma que não correspondem a área de croma são enviadas para a saída, mas o controle do compensador de movimento não as envia ao processamento de croma. Essa equivalência de áreas foi a alternativa encontrada para se evitar separar a *cache* em duas, uma para luma e outra para as componentes de croma, e duplicar recursos. Essa alternativa não causa aumento do uso dos recursos de memória, uma vez que o bloco de memória interna do FPGA tem tamanho fixo. O mesmo não é verdadeiro para implementações em *standard cell*.

A política de substituição de conjuntos adotada foi a FIFO. Essa política foi adotada devido a sua implementação em *hardware* demandar um número menor de recursos, quando comparadas a outras políticas. Considera-se desnecessária a utilização de outra política tendo em vista que não há convergência para uma mesma área de referência pelos blocos compensados.

A *tag* da *cache* tem 21 bits, 8 do POC, 6 da posição da coluna e 7 da linha. Para cada conjunto existe um registrador que armazena a *tag* a ele associada, um comparador que indica se a área requisitada se encontra na *cache* e um registrador que indica a posição na memória interna onde o conjunto está armazenado. O deslocamento dentro do conjunto é dado pelas demais 4 posições do endereço de linha requisitada. Dos demais bits do endereço de coluna, os dois bits mais significativos selecionam o conjunto de 16 amostras consecutivas para a saída de luma, e 4 para os de croma. A seleção das 9 amostras de luma

Tabela 4.7: Taxa de *misses* para 32 conjuntos com vídeo SDTV

	24 Colunas	40 Colunas	72 Colunas	136 Colunas
8 Linhas	4,21%	2,81%	2,12%	1,75%
16 Linhas	2,21%	1,48%	1,11%	0,98%
32 Linhas	1,18%	0,78%	0,56%	0,44%

e as 3 de cada componente de croma, correspondentes a linha da área a ser interpolada, é realizada pelo extrapolador. Um sinal de válido na saída do bloco é ligado quando há um *hit*.

São utilizados os blocos internos de memória do FPGA para armazenar os conjuntos. No dispositivo alvo da prototipação cada bloco de memória interna tem 16Kbits, com largura de palavra de até 32 bits. São utilizados 10 blocos para armazenar as amostras de luma, onde toda a capacidade de memória é utilizada. Não foi conseguido o mesmo para as amostras de croma na implementação, que também utilizam 10 blocos para armazenar os dois componentes, mas apenas 256 posições das 512 disponíveis no bloco de RAM.

Uma máquina de estados é responsável pela requisição dos dados no caso de *miss*. A interface faz a solicitação da área do conjunto, recebendo as 40 colunas de luma e as 32 colunas de croma, 16 de Cb e 16 de Cr, por vez. A solicitação é feita em termos de POC, posição de linha e coluna da área do quadro, um mecanismo externo deve converter esses parâmetros na posição de memória onde, de fato, se encontra os dados. Cada transação é controlada por um *handshake*. Ao final das 16 linhas os dados solicitados são posicionados na saída e o sinal de válido é ligado.

A Figura 4.6 apresenta a estrutura do acesso à memória através da *cache*.

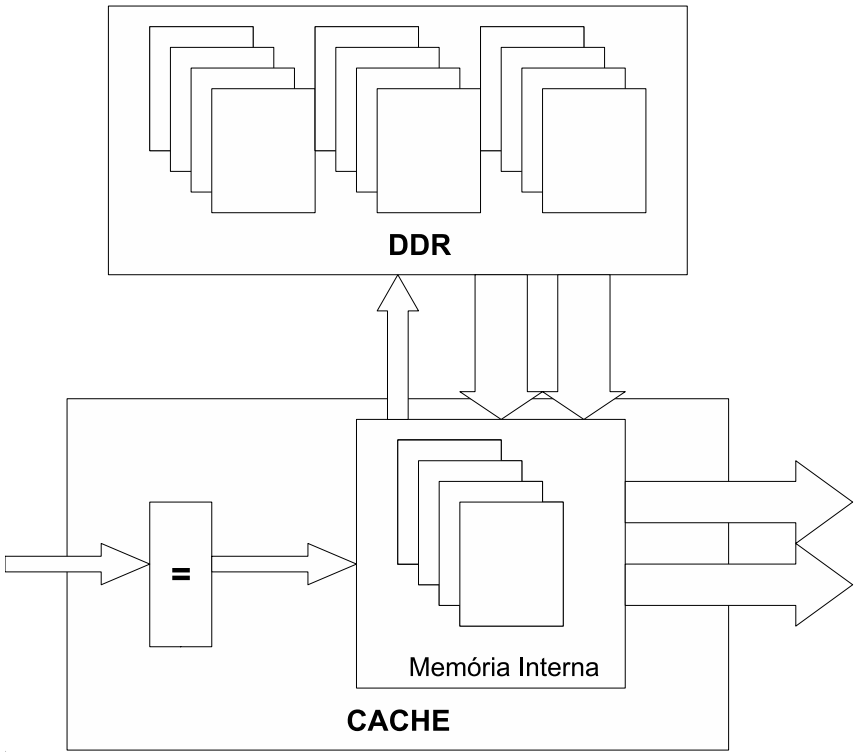

Figura 4.6: Estrutura do acesso à memória através da *cache*

A latência da *cache* é de 1 ciclo de relógio. A Tabela 4.8 apresenta os resultados de síntese da *cache*. Devido a quantidade de pinos na interface da *cache*, não foi possível a síntese pós *place and route*, sendo apresentados, os resultados referentes apenas a síntese, com estimativas de desempenho. A freqüência de operação da *cache* atingiu 132 MHz.

Tabela 4.8: Resultados de síntese da *cache*

	Total	Percentual
Slices	963	7%
Flip Flops	729	2%
LUTs	1.214	4%
Blocos de RAM	20	14%
Multiplicadores	0	0%

Pelos dados de síntese, apresentados na Tabela 4.8, nota-se que a *cache* é um bloco com um consumo considerável de recursos. A quantidade consumida de *slices* e blocos de RAM para esse bloco se justifica pela quantidade de acessos à memória externa economizados.

4.4.2 Extrapolação das amostras

Quando uma determinada área a ser processada ultrapassa as bordas do quadro de referência uma extrapolação dos valores presentes na borda é realizada.

Para as linhas, a extrapolação é realizada verificando se a posição da mesma se encontra dentro das bordas do quadro. Antes da requisição à *cache* é verificado se a posição da linha é negativa, se for, a requisição é feita para a posição zero. É testado, também, se a posição é maior que o número de linhas do quadro e é feita a requisição com a posição da última linha do quadro, para o caso do teste ser positivo. Como cada linha é acessada individualmente, esse mecanismo é suficiente para realizar a extrapolação.

Como as colunas são acessadas em blocos a simples verificação não é o suficiente para a extrapolação dos valores das colunas. Além da extrapolação, o extrator é responsável pela seleção das amostras a serem enviadas para o processamento de amostras.

Na extrapolação são passíveis de ocorrer 5 casos: área totalmente fora da borda esquerda; parte da área fora da borda direita, parte dentro do quadro; área totalmente dentro das dimensões do quadro; parte da área dentro do quadro, parte fora da borda direita; área totalmente fora da borda esquerda.

Caso a área a ser interpolada esteja dentro das bordas do quadro, as nove amostras a serem enviadas para o processamento de amostras de luma são selecionadas a partir dos três bits menos significativos do endereço de coluna, sem contar a parte fracionária, através de um multiplexador. O mesmo ocorre para as três amostras de cada componente de croma, mas utilizando o último bit, devido a sub-amostragem de cor.

Se a área estiver totalmente a esquerda da borda, o valor da primeira posição é replicado nove vezes e enviado ao processamento. O mesmo ocorre se a área estiver a direita, sendo que a posição replicada corresponde a oitava e a quarta posições da linha de saída da *cache* para as componentes de luma e croma, respectivamente.

Se a área estiver parte dentro do quadro parte fora do quadro, a primeira ou segunda metade das amostras é mantida e a outra é resultado de replicação. A primeira amostra é replicada e torna-se a primeira metade para a borda esquerda e a última para a borda

direita. Uma seleção, igual ao caso em que a área pertence ao quadro, é realizada sobre o resultado.

A latência do extrapolador é de 1 ciclo de relógio.

4.4.3 Resultados da hierarquia de acesso à memória

No intuito de avaliar a economia do número de acessos à memória, foram realizados experimentos, utilizando-se a configuração selecionada da *cache*. Para esses experimentos foram utilizadas 4 seqüências, sendo 3 em 720HD (1280x720) e uma SD no formato de 720 por 480. As seqüências no formato 720HD foram a *parkrun* (PRHD), *stockolm* (STOK) e *shields* (SHILD). A seqüência parkrun foi também utilizada no formato SD (PRSD).

Os experimentos consistem na contagem dos acessos à memória para a busca de amostras dos quadros de referência sem a *cache* e depois a contabilização desses acessos com a presença da *cache*. Cada acesso corresponde a leitura de uma palavra de memória. Considerou-se a disposição das componentes de luma e croma em áreas separadas da memória.

Primeiramente, o simulador de *cache* contabilizou o número de leituras efetuadas. Com esses números estimou-se a quantidade de acessos à memória externa. Os experimentos consideram uma memória com palavras de 64 bits. Dessa forma, para cada acesso a uma linha de amostras de luma, foram contabilizadas 2 leituras na memória externa. Esse fator de multiplicação deriva da quantidade de amostras presentes numa palavra na memória externa (oito) e na quantidade de amostras requerida por linha (nove). Assim sendo, são necessárias 2 leituras de memória externa para cada acesso à *cache*. De forma semelhante, para cada três acessos à *cache* foram contabilizados 2 acessos adicionais para as amostras de croma. Como a janela de amostras de croma é 1/3 da janela de luma, tanto em largura quanto em comprimento, e são duas as componentes a serem buscadas, chegou-se a proporção de 2/3 dos acessos de luma são requeridos para as amostras de croma. Não se levou em consideração a possibilidade das amostras de croma estarem em duas palavras distintas. A equação 4.1 define o cálculo do número de acessos à memória externa, onde Y representa a quantidade de acessos à *cache*.

$$N\_acessos\_direto = Y * 2 + Y * 2/3 \tag{4.1}$$

Para se encontrar a quantidade de acessos à memória externa realizada pela *cache*, utilizou-se a quantidade de *misses* pela quantidade de dados requerida para copiar os dados para preencher um conjunto. Para cada *miss* na *cache* são lidas 5 palavras de luma, uma vez que cada palavra tem oito amostras, para cada uma das 16 linha do conjunto. Semelhantemente para o croma, são lidas três palavras para cada uma das 8 linhas de croma para cada componente, Cb e Cr. Temos, então, a quantidade de acessos à memória realizada pela *cache* dada na equação 4.2 , onde M representa a quantidade de *misses*.

$$N\_acessos\_cache = M * 5 * 16 + M * 3 * 2 * 8 \tag{4.2}$$

A Fórmula 4.3 apresenta a porcentagem do número de acessos à memória externa economizado com a utilização da *cache*.

$$Porcent\_salvo = (N\_acessos\_direto - N\_acessos\_cache)/N\_acessos\_direto \tag{4.3}$$

Os resultados desses experimentos estão relacionados na Tabela 4.9. A Tabela 4.9 apresenta a quantidade de acessos à *cache*, a quantidade de *misses*, a quantidade estimada

de acessos à memória externa sem o uso da *cache*, a quantidade estimada de acessos à memória externa com o uso da *cache*, a quantidade de acessos salvos e a porcentagem de acessos economizados com a utilização da *cache*.

Tabela 4.9: Acessos à memória externa através da *cache*

Vídeo	Acessos à Cache ($*10^6$)	Misses ($*10^6$)	%Misses	Acessos à Memória S/ Cache ($*10^6$)	Acessos à Memória C/ Cache ($*10^6$)	Acessos Salvos ($*10^6$)	% Acessos Salvos
PRHD	19,47	0,30	1,54%	51,93	38,31	13,62	26,23%
SHILD	70,89	1,00	1,40%	189,03	127,46	61,57	32,57%
STOK	73,47	1,06	1,44%	195,93	135,28	60,65	30,96%
PRSD	29,11	0,42	1,45%	77,63	54,03	23,60	30,40%
Média	48,24	0,69	1,46%	128,63	88,77	39,86	30,04%

Como apresentado na Tabela 4.9, a economia média de acessos à memória externa foi de 30%. Esse resultado representa uma economia na quantidade de dados lidos da memória externa, com redução direta na banda de memória requerida para a aplicação.

Uma melhoria foi implementada no acesso à *cache*. O acesso à *cache* é realizada pelo controle geral do compensador de movimento, que será detalhado a seguir. A melhoria consiste em ler da *cache* apenas as linhas necessárias à interpolação para o dado vetor de movimento. A posição fracionária do vetor relativa às linhas é testada, caso a mesma seja igual a zero, são necessárias apenas as linhas pertencentes ao bloco 4x4.

Os experimentos foram novamente realizados para a melhoria implementada. A Tabela 4.10 apresenta os resultados de *misses* na *cache*, realizados de acordo com a melhoria, em relação a quantidade de acesso à memória do experimento original. Dessa forma, tem-se como resultado a melhoria alcançada sobre os acessos originais. São apresentados a quantidade original de acessos à *cache*, a quantidade de *misses* com a melhoria, a quantidade estimada de acessos à memória externa sem o uso da *cache*, a quantidade estimada de acessos à memória externa com o uso da *cache*, a quantidade de acessos salvos e a porcentagem de acessos economizados com a utilização da *cache* e a melhoria implementadas.

Tabela 4.10: Acessos à memória externa através da *cache* com a primeira melhoria

Vídeo	Acessos à Cache ($*10^6$)	Misses ($*10^6$)	%Misses	Acessos à Memória S/ Cache ($*10^6$)	Acessos à Memória C/ Cache ($*10^6$)	Acessos Salvos ($*10^6$)	% Acessos Salvos
PRHD	19,47	0,19	1,00%	51,93	24,87	27,06	52,11%
SHILD	70,89	0,54	0,76%	189,03	68,79	120,24	63,61%
STOK	73,47	0,50	0,68%	195,93	64,40	131,52	67,13%
PRSD	29,11	0,25	0,85%	77,63	31,75	45,88	59,10%
Média	48,24	0,37	0,82%	128,63	47,45	81,18	60,49%

Com a união das duas estratégias de acesso, o uso da *cache* e o acesso apenas às linhas da *cache* realmente necessárias à interpolação, a melhoria na quantidade de acesso

à memória externa é o dobro da alcançada quando considerada apenas a solução inicial, como está apresentado na Tabela 4.10, atingindo uma média de 60%.

Investigou-se, também, a taxa de melhoria alcançada pela união da *cache* com o acesso apenas às linhas necessárias em relação à implementação do acesso apenas as linhas necessárias, sem o uso da *cache*. Nesse experimento foram contabilizados apenas os acessos à memória externa referentes às linhas necessárias à interpolação. A Tabela 4.11 apresenta os novos dados, onde a segunda coluna representa os acessos à memória externa utilizando a melhoria descrita.

Tabela 4.11: Acessos à memória externa através da *cache*, em relação ao acesso direto com a primeira melhoria

Vídeo	Acessos à Cache ($*10^6$)	Misses ($*10^6$)	%Misses	Acessos à Memória S/ Cache ($*10^6$)	Acessos à Memória C/ Cache ($*10^6$)	Acessos Salvos ($*10^6$)	% Acessos Salvos
PRHD	10,75	0,19	1,81%	28,68	24,87	3,81	13,28%
SHILD	34,11	0,54	1,58%	90,96	68,79	22,17	24,37%
STOK	29,40	0,50	1,71%	78,40	64,40	13,99	17,85%
PRSD	14,36	0,25	1,73%	38,31	31,75	6,55	17,11%
Média	22,16	0,37	1,71%	59,08	47,45	11,63	18,15%

Tal qual esperado, o impacto do uso da *cache* é reduzido quando comparado com a implementação a melhoria. Tem-se uma redução de 40% em relação aos dados apresentados na Tabela 4.9, com ganho de 18% em média.

Uma segunda melhoria, apresentada em (WANG; LI; HUANG, 2005), é passível de ser implementada. A melhoria consiste na forma da disposição, na memória externa, das amostras de croma. O trabalho sugere o entrelaçamento das amostras, na forma CbCrCbCr...CbCr. Apesar da disposição das componentes de cor das amostras e a organização da memória não fazerem parte do escopo desse trabalho, é de relevância o estudo do impacto dessa melhoria.

A influência dessa segunda melhoria foi realizada alterando-se a forma da Fórmula 4.1. Com a nova disposição das amostras, o preenchimento do conjunto da *cache* é dado pela equação 4.4, onde no lugar de 3 palavras para cada linha de cada componente de cor, tem-se o acesso a duas palavras e meia. A outra metade da palavra é referente a segunda componente de cor e, por isso, não é desperdiçada, como na disposição original.

$$N\_acessos\_cache\_m2 = M * 5 * 16 + M * 2,5 * 2 * 8 \tag{4.4}$$

Com a implementação desse segunda melhoria, os dados das Tabelas 4.9, 4.10 e 4.11, teriam a forma apresentada nas Tabelas 4.12, 4.13 e 4.14, respectivamente.

As Tabelas 4.12, 4.13 e 4.14 mostram ganhos de 13, 5 e 27%, respectivamente, sobre os resultados originais, para as três abordagens. A *cache*, mais as duas melhorias apresentadas, como apresenta a Tabela 4.13, tem uma melhoria de 63%, em média, no número de acessos à palavras de memória sobre o acesso direto.

Apesar dos números satisfatórios, os experimentos realizados incidem apenas sobre a quantidade de palavras acessadas na memória externa, sem contemplar a quantidade de ciclos para se trazer os dados da memória. As memórias RAM costumam ter penalidades

Tabela 4.12: Acessos à memória externa através da *cache* com a segunda melhoria

Vídeo	Acessos à Cache ($*10^6$)	Misses ($*10^6$)	%Misses	Acessos à Memória S/ Cache ($*10^6$)	Acessos à Memória C/ Cache ($*10^6$)	Acessos Salvos ($*10^6$)	% Acessos Salvos
PRHD	19,47	0,30	1,54%	51,93	35,91	16,02	30,84%
SHILD	70,89	1,00	1,40%	189,03	119,49	69,54	36,79%
STOK	73,47	1,06	1,44%	195,93	126,82	69,11	35,27%
PRSD	29,11	0,42	1,45%	77,63	50,65	26,98	34,75%
Média	48,24	0,69	1,46%	128,63	83,22	45,41	34,41%

Tabela 4.13: Acessos à memória externa através da *cache* com a primeira e segunda melhorias

Vídeo	Acessos à Cache ($*10^6$)	Misses ($*10^6$)	%Misses	Acessos à Memória S/ Cache ($*10^6$)	Acessos à Memória C/ Cache ($*10^6$)	Acessos Salvos ($*10^6$)	% Acessos Salvos
PRHD	19,47	0,19	1,00%	51,93	23,31	28,62	55,11%
SHILD	70,89	0,54	0,76%	189,03	64,49	124,54	65,88%
STOK	73,47	0,50	0,68%	195,93	60,38	135,55	69,18%
PRSD	29,11	0,25	0,85%	77,63	29,77	47,86	61,66%
Média	48,24	0,37	0,82%	128,63	44,49	84,14	62,96%

no acesso à palavras que se encontram em linhas diferentes. No intuito de levantar dados sobre o impacto dessas penalidades, as equações de acesso a memória foram modificadas para tratar do tempo efetivo de acesso aos dados.

Para os fins do experimento, foi considerado uma penalidade de um ciclo de acesso para cada troca de linha de memória. Foi considerado, por simplicidade, que cada linha do quadro cabe numa linha de memória.

Dessa forma as equações 4.1 e 4.2 passam para a forma das equações 4.5 e 4.6, respectivamente, onde N_ciclos_direto_L é quantidade de ciclos para se acessar diretamente a memória e N_ciclos_cache_L é a quantidade de ciclos de acesso à memória realizados pela *cache*.

$$N\_ciclos\_direto\_L = Y * (2 + 1) + Y * (2 + 2/3) \tag{4.5}$$

$$N\_ciclos\_cache\_L = M * (5 + 1) * 16 + M * (3 + 1) * 2 * 8 \tag{4.6}$$

Aplicando-se essas novas fórmulas sobre os dados, os resultados apresentados nas Tabelas de 4.9 a 4.11 assumem a forma nas Tabelas 4.15 a 4.17, respectivamente.

Quando a penalidade de troca de linhas de memória é avaliada, a economia de ciclos de acesso à memória cresce de forma considerável. Os resultados apresentados na Tabela 4.15 mostram que a utilização da *cache* resulta em economias de ciclos de acesso em torno dos 55%. Esses resultados passam para 75% quando os experimentos são realizados com a primeira melhoria. Mesmo quando se compara a primeira melhoria sem

Tabela 4.14: Acessos à memória externa através da *cache* com primeira e segunda melhoria, em relação ao acesso direto com a primeira melhoria

Vídeo	Acessos à Cache ($*10^6$)	Misses ($*10^6$)	%Misses	Acessos à Memória S/ Cache ($*10^6$)	Acessos à Memória C/ Cache ($*10^6$)	Acessos Salvos ($*10^6$)	% Acessos Salvos
PRHD	10,75	0,19	1,81%	28,68	23,31	5,36	18,70%
SHILD	34,11	0,54	1,58%	90,96	64,49	26,47	29,10%
STOK	29,40	0,50	1,71%	78,40	60,38	18,02	22,98%
PRSD	14,36	0,25	1,73%	38,31	29,77	8,54	22,29%
Média	22,16	0,37	1,71%	59,08	44,49	14,60	23,27%

Tabela 4.15: Ciclos de acesso à memória externa através da *cache*

Vídeo	Acessos à Cache ($*10^6$)	Misses ($*10^6$)	%Misses	Acessos à Memória S/ Cache ($*10^6$)	Acessos à Memória C/ Cache ($*10^6$)	Acessos Salvos ($*10^6$)	% Acessos Salvos
PRHD	19,47	0,30	1,54%	110,36	47,89	62,47	56,61%
SHILD	70,89	1,00	1,40%	401,68	159,32	242,36	60,34%
STOK	73,47	1,06	1,44%	416,34	169,09	247,25	59,39%
PRSD	29,11	0,42	1,45%	164,97	67,54	97,43	59,06%
Média	48,24	0,69	1,46%	273,34	110,96	162,38	58,85%

cache com os resultados da junção dessas estratégias, os resultados, como apresentado na Tabela 4.17, ficam em torno dos 50%.

Na aplicação da segunda melhoria, o entrelaçamento das amostras de Cb e Cr, pode-se aplicar uma estratégia para melhora-la, quando a penalidade de troca de linhas é avaliada. Em vez de entrelaçar apenas as amostras de Cb e Cr, é possível entrelaçar todas as amostras. Dessa forma nas linhas pares propõe-se a organização por palavra de memória na forma YYYYCbCbCrCr. As linhas dos quadros ímpares mantém a estrutura original, contendo apenas amostras de luma. Dessa forma apenas a leitura das amostras de luma seriam penalizadas. A fórmula 4.6 assume a forma da fórmula 4.7.

$$N\_ciclos\_cache\_m2\_L = M * (5 + 1) * 16 + M * (2, 5 * 2) * 8 \tag{4.7}$$

Com essa nova melhoria, e considerando as latências, as os dados das Tabelas 4.12 a 4.14 passam para os apresentados nas Tabelas de 4.18 a 4.20.

Os resultados apresentados nas Tabelas de 4.18 a 4.20, quando comparados com os das Tabelas de 4.15 a 4.17, apresentam uma acréscimo em torno de 5% na economia de ciclos de leitura. A Tabela 4.19 apresenta uma economia média de 80% na quantidade de ciclos de acessos a dados na memória externa. A hierarquia de memória atinge quase 60% de economia de ciclos de acesso, como apresentado na Tabela 4.20, quando comparada com o acesso direto à memória externa, considerando-se as melhorias apresentadas.

As otimizações apresentadas têm baixo ou nenhum consumo adicional de recursos de *hardware*. Todavia, os ganhos, tanto em número de acesso e, ainda mais, na quantidade

Tabela 4.16: Ciclos de acesso à memória externa através da *cache* com a primeira melhoria

Vídeo	Acessos à Cache ($*10^6$)	Misses ($*10^6$)	%Misses	Acessos à Memória S/ Cache ($*10^6$)	Acessos à Memória C/ Cache ($*10^6$)	Acessos Salvos ($*10^6$)	% Acessos Salvos
PRHD	19,47	0,19	1,00%	110,36	31,09	79,27	71,83%
SHILD	70,89	0,54	0,76%	401,68	85,99	315,70	78,59%
STOK	73,47	0,50	0,68%	416,34	80,50	335,84	80,66%
PRSD	29,11	0,25	0,85%	164,97	39,69	125,28	75,94%
Média	48,24	0,37	0,82%	273,34	59,32	214,02	76,76%

Tabela 4.17: Ciclos de acesso à memória externa através da *cache*, em relação ao acesso direto com a primeira melhoria

Vídeo	Acessos à Cache ($*10^6$)	Misses ($*10^6$)	%Misses	Acessos à Memória S/ Cache ($*10^6$)	Acessos à Memória C/ Cache ($*10^6$)	Acessos Salvos ($*10^6$)	% Acessos Salvos
PRHD	10,75	0,19	1,81%	60,94	31,09	29,85	48,99%
SHILD	34,11	0,54	1,58%	193,29	85,99	107,30	55,51%
STOK	29,40	0,50	1,71%	166,59	80,50	86,09	51,68%
PRSD	14,36	0,25	1,73%	81,40	39,69	41,71	51,24%
Média	22,16	0,37	1,71%	125,56	59,32	66,24	51,86%

de ciclos de acesso, são expressivos.

Com as melhorias sugeridas, são necessários 120 acessos à palavras de 64 bits na memória externa, para preencher um conjunto da *cache*. Se se considerar uma memória DDR de 100 MHz com *pipeline* de comandos, em 64 ciclos o conjunto é preenchido. Aos 64 ciclos deve se adicionar ainda a latência inicial e 16 penalidades por troca de linha. Com uma latência inicial de 3 ciclos e uma penalidade de 1 ciclo por troca de linha tem-se 83 ciclos de relógio para o preenchimento de um conjunto da *cache*.

4.5 Arquitetura do Processamento das amostras

O processamento das amostras é o bloco responsável pelas transformações realizadas nas amostras das áreas de referência. Fazem parte do processamento das amostras: a interpolação para quarto de pixel; a predição ponderada e a bi-predição.

O processamento das amostras é o bloco crítico em termos de desempenho, tendo em vista a quantidade de dados a serem processados. Para vídeos de alta definição, no formato de 1080 por 1920 a uma taxa de 30 quadros por segundo, a taxa de entrada é de aproximadamente 629 milhões de amostras de luma e 69 milhões de amostras de croma por segundo, considerando um pior caso de 30 quadros totalmente formados por partições 4x4 bi-preditivas. A taxa de saída é de 93 milhões de amostras por segundo. Essa taxa de saída deve ser alcançada com um relógio a 100 MHz e saída de uma amostra compensada por ciclo.

Tabela 4.18: ciclos de acesso à memória externa através da *cache* com a terceira melhoria

Vídeo	Acessos à Cache ($*10^6$)	Misses ($*10^6$)	%Misses	Acessos à Memória S/ Cache ($*10^6$)	Acessos à Memória C/ Cache ($*10^6$)	Acessos Salvos ($*10^6$)	% Acessos Salvos
PRHD	19,47	0,30	1,54%	110,36	40,70	69,65	63,12%
SHILD	70,89	1,00	1,40%	401,68	135,42	266,26	66,29%
STOK	73,47	1,06	1,44%	416,34	143,73	272,61	65,48%
PRSD	29,11	0,42	1,45%	164,97	57,41	107,56	65,20%
Média	48,24	0,69	1,46%	273,34	94,32	179,02	65,02%

Tabela 4.19: Ciclos de acesso à memória externa através da *cache* com a primeira e terceira melhorias

Vídeo	Acessos à Cache ($*10^6$)	Misses ($*10^6$)	%Misses	Acessos à Memória S/ Cache ($*10^6$)	Acessos à Memória C/ Cache ($*10^6$)	Acessos Salvos ($*10^6$)	% Acessos Salvos
PRHD	19,47	0,19	1,00%	110,36	26,42	83,93	76,06%
SHILD	70,89	0,54	0,76%	401,68	73,09	328,59	81,80%
STOK	73,47	0,50	0,68%	416,34	68,43	347,92	83,56%
PRSD	29,11	0,25	0,85%	164,97	33,74	131,23	79,55%
Média	48,24	0,37	0,82%	273,34	50,42	222,92	80,24%

Para tratar os diversos formatos de macrobloco foi escolhido, como unidade de processamento, o bloco 4x4 de luma. O bloco 4x4 foi escolhido uma vez que o mesmo é a menor unidade tratada pelo padrão e porque todos os formatos do macrobloco podem ser formados a partir de conjuntos de blocos de 4x4 amostras. Essa também é a unidade de processamento adotada pelo *software* de referência e por diversos trabalhos na literatura pesquisada (H.264/AVC SOFTWARE COORDINATION, 2005; LIE et al., 2005; WANG et al., 2005). Para as amostras de croma, o bloco equivalente é o de 2x2 amostras.

O processador de amostras é formado por um caminho de dados para amostras de luma e outro para amostras de croma, que operam de forma independente.

Para evitar a duplicação dos recursos utilizados para o processamento de cada uma das regiões de um bloco bi-preditivo, os mesmos recursos computacionais são utilizados para processar as duas regiões de forma serial. A área referente a lista 0 é processada e tem seus resultados armazenados num *buffer*. Em seguida, a área referente a lista 1 é processada. A média dos dois resultados dá o resultado para o bloco bi-preditivo.

O caminho de dados para luminância conta com um *buffer* de entrada, o interpolador para $\frac{1}{4}$ de pixel, um ponderador responsável pelo processo de predição ponderada (WP), um *buffer* 4x4, uma média para bi-predição e, finalmente, um operador de clip. Essa arquitetura está apresentada pela Figura 4.7.

O caminho de dados para croma, cujo esquemático está representado na Figura 4.8, é semelhante ao de luma. O interpolador de croma é diferente, bem como o tamanho do *buffer*, que passa de 4x4 para 2x2 amostras. As outras operações utilizam instâncias diferentes dos mesmos componentes desenvolvidos para o caminho de dados de luma.

Tabela 4.20: Ciclos de acesso à memória externa através da *cache* com primeira e terceira melhoria, em relação ao acesso direto com a primeira melhoria

Vídeo	Acessos à Cache ($*10^6$)	Misses ($*10^6$)	%Misses	Acessos à Memória S/ Cache ($*10^6$)	Acessos à Memória C/ Cache ($*10^6$)	Acessos Salvos ($*10^6$)	% Acessos Salvos
PRHD	10,75	0,19	1,81%	60,94	26,42	34,52	56,64%
SHILD	34,11	0,54	1,58%	193,29	73,09	120,20	62,19%
STOK	29,40	0,50	1,71%	166,59	68,43	98,17	58,93%
PRSD	14,36	0,25	1,73%	81,40	33,74	47,66	58,56%
Média	22,16	0,37	1,71%	125,56	50,42	75,14	59,08%

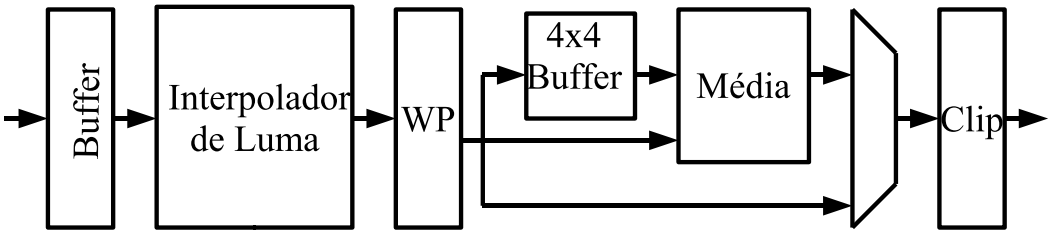

Figura 4.7: Caminho de dados do processador de amostras de luma

Para atender aos requisitos de desempenho, o processamento de amostras deve ser capaz de processar um macrobloco, com 256 amostras de luma e 128 de croma, a cada 384 ciclos. Uma vez que os caminhos de dados entre luma e croma são independentes os mesmos foram projetados para operarem em paralelo. Dessa forma, a quantidade de ciclos disponíveis, para o processamento de cada componente de cor de um macrobloco, passa a ser a quantidade de ciclos máxima disponível, ou seja, 384.

Para que a latência do processamento não se tornar crítica, o processamento das amostras foi desenvolvido na forma de *pipeline*. Assim, o processamento de amostras não precisa esperar o processamento completo de um bloco para dar início ao processamento do bloco seguinte. Dessa forma, a latência da arquitetura do processador de amostras influi apenas na quantidade de ciclos para se ter disponível o primeiro resultado válido.

Com o paralelismo do processamento das componentes e com o processamento realizado em *pipeline*, cada bloco tem disponível 24 ciclos para ser processado.

A filtragem para quarto de pixel de um bloco 4x4 de luma exige uma janela de filtragem de 9x9 amostras. Para as componentes de croma, os blocos 2x2 são filtrados a partir de uma matriz 3x3, cada. A quantidade de dados de entrada é dobrada para blocos bi-preditivos, ou seja, 18x9 amostras de luma e 6x3 amostras para cada componente de croma.

O atendimento aos requisitos de processamento exige que, para um quadro bi-preditivo, sejam processadas, no mínimo, 6,75 amostras de luma por ciclo de relógio. A solução encontrada foi a paralelização das amostras de um bloco. A cada ciclo de relógio, o processador de amostras lê uma linha da área a ser processada: 9 amostras de luma, 3 amostras de Cb e outras 3 de Cr. Essa taxa de leitura é requerida pela interpolação de

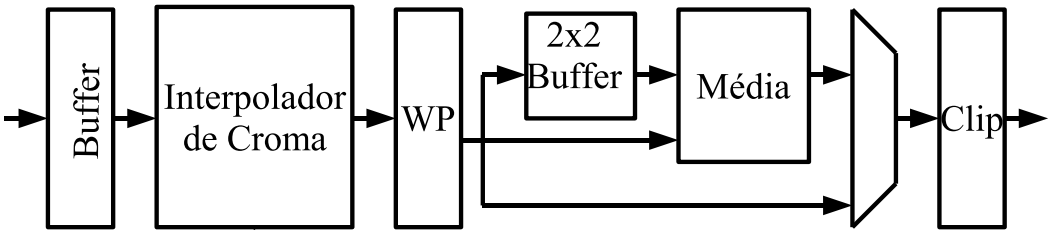

Figura 4.8: Caminho de dados do processador de amostras de croma

quarto e oitavo de pixel. Após as interpolações, a taxa de entrada dos demais blocos é igual a taxa de saída. Como o resultado da interporlação de luma resulta em blocos 4x4, e para manter a estrutura de processamento de linhas, os demais blocos do processador de amostras, processam 4 amostras por ciclo.

O processamento de luma tem uma taxa de entrada de dados maior que a taxa de entrada das componentes de croma. Para cada nove entradas de luma, o croma tem 3 entradas de Cb e 3 entradas de Cr. A latência do processamento de croma também é menor que a do processamento de luma. Foi possível, então, processar as amostras das componentes Cb e Cr de forma serial. A entrada no processamento de croma, para as duas componentes, ocorre em paralelo e são armazenadas em *buffers* distintos. Com as amostras de ambas as componentes armazenadas, o processamento lê as amostras referentes à componente Cb e as processa. Quando são lidas todas as amostras do bloco Cb, inicia-se a leitura e o processamento das amostras de Cr. Assim, evita-se a duplicação do caminho de dados do croma, sendo utilizado apenas um *buffer* para o armazenamento das amostras de Cb e outra para os de Cr, compartilhando os demais blocos do caminho de dados.

O processador de amostras processa as áreas a serem compensadas consumindo linhas das matrizes de entrada. Como cada palavra de memória armazena amostras consecutivas de uma mesma linha, o acesso orientado a linhas é mais eficiente, requerendo menos acessos. Para se evitar uma estrutura de transposição das amostras de linhas para colunas, o processamento das amostras foi transposto. A interpolação é a única operação, do caminho de dados do processamento de amostras, onde a ordem dos dados influi no resultado final. Para adequar a interpolação à transposição do processamento de amostras, bastou a troca dos valores dos vetores de movimento. A parte fracionária do vetor X é ligada à entrada para o vetor Y, e vice-versa.

Os interpoladores recebem as amostras, pertencentes às janelas de filtragem, e os parâmetros do processamento relativo ao bloco a ser compensado. O interpolador de luma recebe 9 ou 18 linhas de 9 amostras cada. O interpolador de croma recebe 3 ou 6 linhas de 6 amostras cada, sendo 3 amostra da componente Cb e mais 3 da componente Cr. Os parâmetros do processamento das amostras são: a parte fracionária dos vetores de movimento, os parâmetros da predição ponderada e uma *flag*, que indica se o bloco é bi-preditivo ou não. Quando as amostras estão carregadas nos *buffers* de entrada do processamento e os parâmetros posicionados nas entradas, o sinal para dar inicio ao processamento é ligado.

Para cada linha de 9 amostras de luma que entra no interpolador de luma, uma linha de 4 amostras é obtida na saída. A linha de saída de croma tem 2 amostras. Cada amostra de saída dos interpoladores passa pelo ponderador, o qual aplica a operação da predição ponderada. Caso o bloco não seja bi-preditivo, o mesmo é encaminhado para a operação de *clipping*. Caso contrário, a linha de resultados relativos a compensação do bloco da

lista 0, têm seus valores armazenados nos registradores de deslocamento do *buffer* 4x4, para luma, e 2x2, para croma. Os resultados do bloco da lista 1, após passarem pelo ponderador, são somados com os resultados armazenados no *buffer* e, então, divididos por 2. O resultado da divisão é submetido ao bloco de *clipping* e, em seguida, aos registradores de saída do processamento.

As arquiteturas dos componentes do processamento das amostras são apresentados em detalhes nas subseções a seguir. A subseção 4.4.1 apresenta a arquitetura do filtro interpolador de luma, enquanto a arquitetura do filtro interpolador de croma é apresentado na subseção 4.4.2. A arquitetura do ponderador é detalhada na subseção 4.4.3. A subseção 4.4.4 apresenta a arquitetura utilizada para processar os blocos bi-preditivos de forma seqüencial. A arquitetura do *clipping* é apresentada na subção 4.4.5. A subseção 4.4.6 apresenta as unidades de controle do processamento de amostras. Por fim, a subseção 4.4.7 apresenta os resultados do processamento das amostras.

4.5.1 Arquitetura do Filtro interpolador de Luma

A arquitetura do filtro interpolador de luma é um dos temas mais presentes na literatura, quando se trata da compensação de movimento para decodificadores segundo o padrão H.264, junto com os requisitos de memória para o mesmo. Esse interesse se deve à elevada taxa de entrada do interpolador e as diversas filtragens possíveis, de acordo com as diversas posições para o pixel interpolado.

A interpolação de quarto de pixel é realizada em duas etapas. Primeiro se interpola as amostras para a resolução de meio de pixel aplicando uma filtragem bidimensional de resposta finita ao impulso. Sobre o resultado da interpolação de meio pixel, é, então, realizada uma interpolação bilinear.

Dependendo da posição em resolução de metade de pixel a ser gerada pelo interpolador, é necessária uma filtragem horizontal, uma vertical ou uma vertical e outra horizontal. De modo a se manter uma estrutura fixa para a filtragem, a arquitetura do interpolador utiliza sempre uma filtragem horizontal e outra vertical.

O padrão H.264 define que a posição de meio pixel pode ser calculada a partir do resultado tanto da interpolação horizontal quanto da interpolação vertical. Desse modo, a ordem da filtragem não altera o resultado da interpolação.

O interpolador de luma, como mencionado, recebe como entrada uma linha da janela de filtragem por ciclo. A área de filtragem para um bloco 4x4 é uma matriz de 9x9 amostras. Para se alcançar o desempenho requerido, o filtro consome uma linha de entrada por ciclo.

O primeiro passo da filtragem para metade de pixel é a aplicação da filtragem horizontal em cada linha da entrada. Os resultados da filtragem horizontal preenchem as entradas da filtragem vertical. Uma vez que tenham sido filtradas linhas suficientes para preencher os seis parâmetros do filtro de 6 *taps* é iniciada a filtragem vertical. A Figura 4.9 exemplifica o funcionamento da filtragem. As amostras de c0 a c3 são as amostras a serem interpoladas. O quadrado com borda grossa é o bloco 4x4 a ser compensado e os retângulos com borda fina são as amostras sendo filtradas. As bolas brancas são as amostras da área de filtragem, as bolas cinzas claras são as amostras que estão sendo utilizadas para a filtragem, as bolas cinza escuras são os resultados do filtro horizontal.

O filtro interpolador de metade de pixel utilizado nesse trabalho é baseado na solução apresentada em (WANG et al., 2005). Essa solução usa, para calcular a amostras de meio pixel, a abordagem de separar o filtro 2D em filtros unidimensionais, verticais e horizontais, baseado na propriedade da separabilidade.

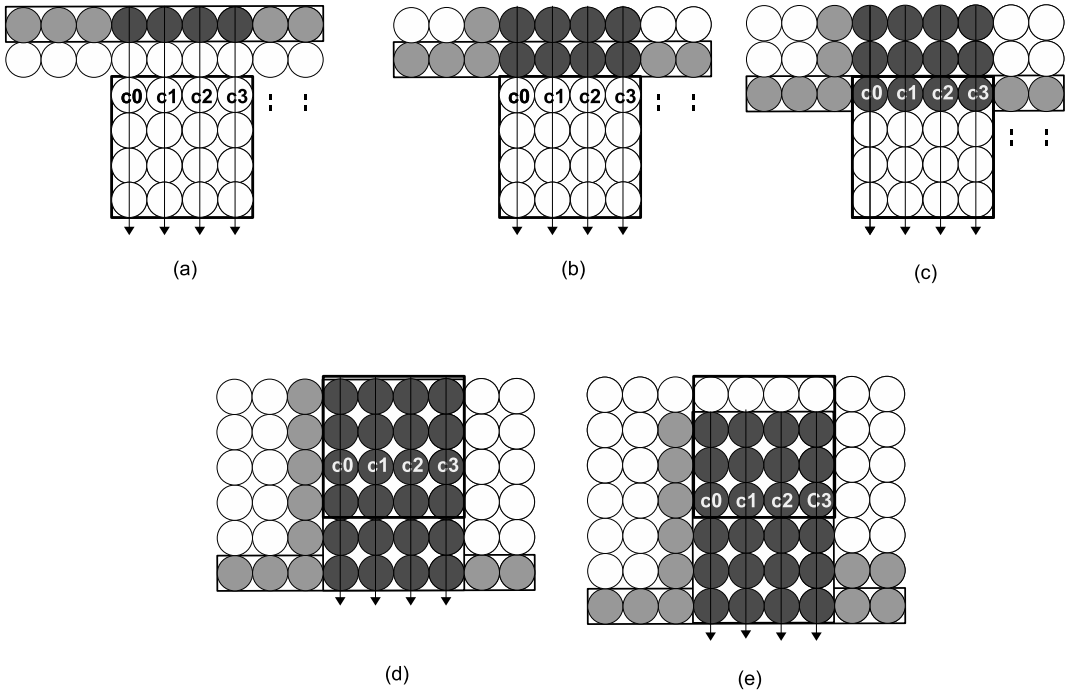

Figura 4.9: Filtragem de um bloco 4x4. (a) ciclo 1 (b) ciclo 2 (c) ciclo 3 (d) ciclo 8 (e) ciclo 9

Para uma linha de saída são utilizados um filtro horizontal e 3 verticais, essa disposição está representada na Figura 4.10(a). Compondo a filtragem para as quatro linhas do bloco, são utilizados quatro filtros horizontais e nove filtros verticais, representada na Figura 4.10(b). Essa arquitetura produz todas as posições na resolução de metade de pixel.

Os filtros foram implementados de forma completamente paralela. Os filtros verticais possuem *pipeline* de 2 estágios.

Quatro dos nove filtros verticais, representados em cinza na Figura 4.10, processam os resultados da filtragem horizontal. Estes quatro filtros têm largura de 14 bits e as entrada podem ter valores negativos. Nestes filtros foi utilizada a operação de multiplicação, utilizando-se os multiplicadores presentes no dispositivo FPGA. Nos demais filtros foram utilizados somas e deslocamentos para implementar a multiplicação.

Os valores de saída do filtro de meio pixel não podem estar fora da faixa de valores definida pela norma, que no caso do perfil Main, varia de 0 a 255. A interpolação de meio pixel pode gerar tanto valores negativos quanto valores que ultrapassam o valor máximo permitido. As amostras, depois da filtragem para meio pixel, têm 10 bits de largura. Se o bit mais significativo for igual a 1, então o valor é negativo e é enviado o valor zero para a saída. Se os dois bits mais significativos forem iguais a zero, os demais 8 bits são enviados para a saída. Se nenhuma das duas alternativas anteriores forem satisfeitas, é enviado o valor 255.

Uma vez disponíveis todas as amostras em metade de pixel, mais as amostras de posições inteiras, a segunda etapa da filtragem, a de quarto de pixel, pode ser realizada. A interpolação para quarto de pixel é realizada por uma média simples entre duas posições definidas, de acordo com a posição a ser gerada.

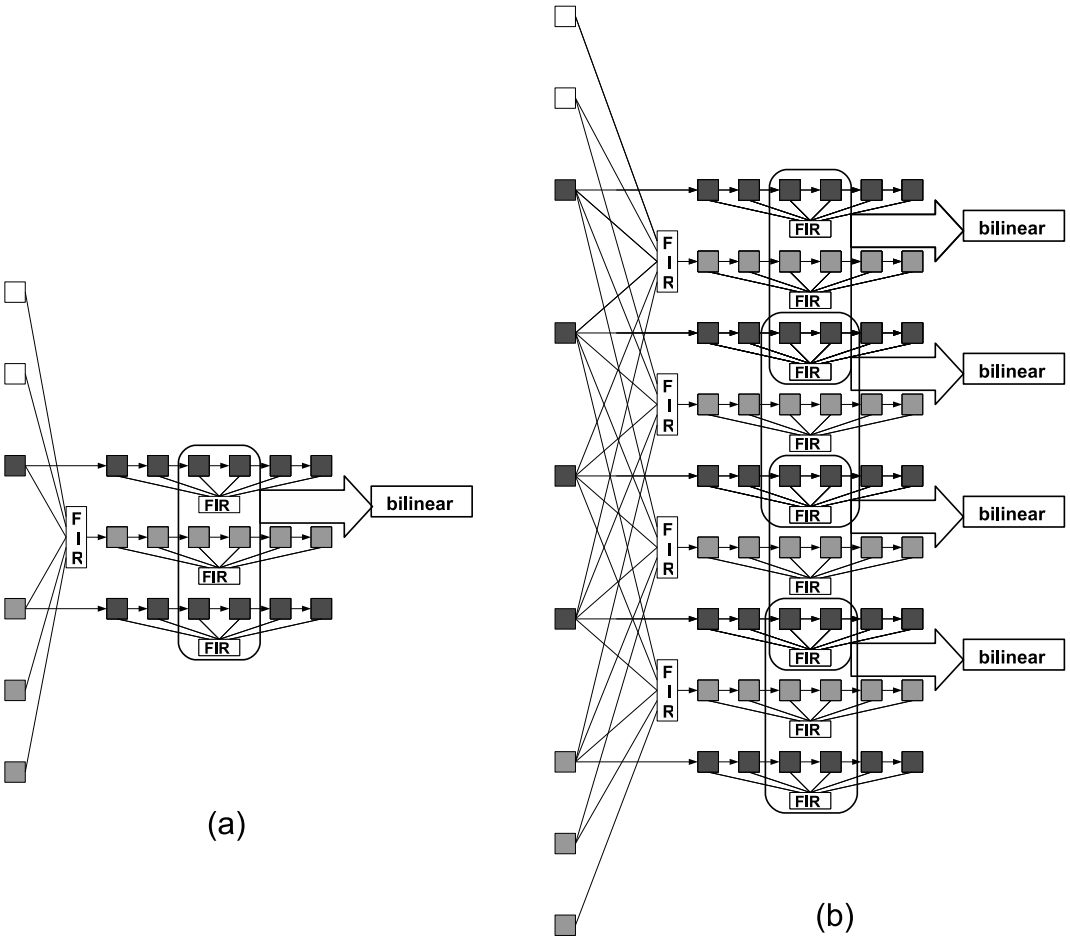

Figura 4.10: Filtro interpolador de luma

A partir dos valores da parte fracionária do vetor de movimento, são selecionadas as posições para cada termo da soma. Os termos são somados entre si e adicionados de 1, para o arredondamento. Em seguida, o bit menos significativo é descartado e os demais são enviados para a saída.

A latência da arquitetura é de nove ciclos de relógio. A Tabela 4.21 apresenta os resultados de síntese do filtro interpolador de luma.

4.5.2 Arquitetura do Filtro interpolador de croma

O filtro interpolador de croma, a exemplo do filtro de luma, recebe como entrada uma linha de bloco por ciclo de relógio. Cada linha tem 3 amostras, uma vez que os blocos de croma possuem o formato 2x2 e a interpolação utiliza-se de apenas mais uma amostra a direita e abaixo da área a ser interpolada, perfazendo uma área de 3x3.

Duas instâncias de interpoladores são utilizadas para processar o bloco. Cada interpolador recebe duas amostras por ciclo, sendo a amostra central da área de 3x3 compartilhada entre os dois.

Cada interpolador de croma conta com um conjunto de quatro registradores de deslocamento, no formato de 2x2, oito multiplicadores, dois subtratores e quatro somadores.

Cada termo da formula é multiplicado e tem seu resultado armazenado em um registrador. Em seguida, os 4 termos são somados e ao resultado é adicionado a 32. Os 6

Tabela 4.21: Resultados de síntese interpolador de luma

	Total	Percentual
Slices	4.552	33%
Flip Flops	4.649	16%
LUTs	4.947	18%
Blocos de RAM	3	2%
Multiplicadores	0	0%

bits menos significativos são descartados e os demais são armazenados no registrador de saída.

A latência do filtro de croma é de 2 ciclos de relógio. A Tabela 4.22 apresenta os resultados de síntese do filtro interpolador de croma, contendo as duas instâncias.

Tabela 4.22: Resultados de síntese interpolador de croma

	Total	Percentual
Slices	136	1%
Flip Flops	188	1%
LUTs	92	1%
Blocos de RAM	0	0%
Multiplicadores	12	8%

4.5.3 Arquitetura do Ponderador

O bloco ponderador, cuja arquitetura está representada na Figura 4.11, realiza a operação de multiplicação do valor da amostra, presente na entrada "Y", pelo fator multiplicativo, valor da entrada "W". Em paralelo com a multiplicação é selecionado o resultado de $2^{logWD-1}$, previamente calculado e armazenado numa tabela. O valor é selecionado pelo valor de logWD, que pode ir de 0 a 7. O resultado da soma da multiplicação com o valor selecionado da tabela é então deslocado em logWD posições. Para a compatibilidade da operação de deslocamento com valores tanto positivos quanto negativos, o bit mais significativo do resultado da soma é replicado nas posições mais significativas "vagas" pelo deslocamento. Ao final desse processo é adicionado o ajuste à saída do deslocamento e o resultado da soma é escrito no registrador de saída do bloco.

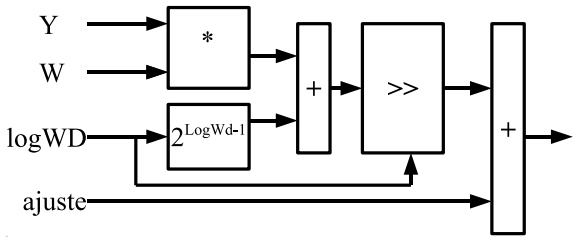

Figura 4.11: Ponderador

A predição ponderada gera amostras num intervalo de valores de 17 bits de largura. O bit mais significativo é necessário para a identificação de valores negativos, os outros

16 bits são necessários para alocar o valor máximo que pode ser gerado pelo processo de compensação de movimento.

No caminho de dados de luma, esse componente é replicado 4 vezes, enquanto no caminho de dados de croma estão presentes duas instâncias.

A latência do ponderador é de 1 ciclo de relógio. A Tabela 4.23 apresenta os resultados de síntese do ponderador.

Tabela 4.23: Resultados de síntese do ponderador

	Total	Percentual
Slices	748	4%
Flip Flops	856	2%
LUTs	806	2%
Blocos de RAM	0	0%
Multiplicadores	28	20%

4.5.4 Arquitetura da Predição Bidirecional

A predição bidirecional implica no processamento de duas regiões, de mesmo formato, de imagens referências distintas, sendo uma da lista 0 e outra da lista 1, e tendo como resultado a média entre as duas regiões compensadas. Para evitar a duplicação dos recursos utilizados no processamento de cada uma das regiões, os mesmos recursos computacionais são utilizados para processar as duas regiões de forma serial. A arquitetura para processamento da predição bidirecional é apresentada na Figura 4.12.

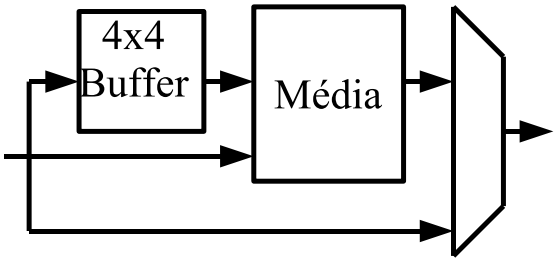

Figura 4.12: Arquitetura do tratamento da bi-predição

Os registradores de deslocamentos armazenam 16 amostras para o caminho de dados da luminância, dispostos em quatro linhas com quatro registradores cada. Para as 4 amostras de crominância, são dispostas duas linhas com dois registradores de deslocamento cada. A escrita nos registradores de deslocamento e, por conseguinte, os deslocamentos dos valores nos registradores de deslocamento são condicionados a presença do sinal de resultado válido na saída dos filtros interpoladores (atrasado em um ciclo devido ao processamento da predição ponderada) e ao sinal que indica o processamento da predição bidirecional do bloco corrente. À medida que os resultados válidos de processamento do bloco advindo da lista 1 são escritos, e processados pelo ponderador, são somados aos valores armazenados nos registradores de deslocamento, novamente ativados. O resultado da soma das amostras dos blocos resultantes da compensação das regiões das listas 0 e 1, é adicionado de 1 e o bit menos significativo é desprezado. Essa operação corresponde a média das regiões compensadas com arredondamento.

A latência da predição bidirecional é de 5 ciclos. Todavia, como há um intervalo de 5 ciclos entre os valores válidos de dois blocos, a latência efetiva é de 13 ciclos de relógio. A predição bidirecional não armazena seus resultados em um registrador, compartilhando o caminho combinacional com o *clipping*, a ser descrito na próxima subseção. A Tabela 4.24 apresenta os resultados de síntese da predição bidirecional.

Tabela 4.24: Resultados de síntese da bi-predição de luma

	Total	Percentual
Slices	180	0%
Flip Flops	272	0%
LUTs	72	0%
Blocos de RAM	0	0%
Multiplicadores	0	0%

4.5.5 *clipping*

O processo de *clipping* é responsável por manter os valores das amostras, ao final do processo de compensação de movimento, dentro do intervalo de valores válidos no processo de decodificação da seqüência de vídeo. O *clipping* é necessário uma vez que a predição ponderada gera amostras de 17 bits de largura.

O processo de *clipping* testa o valor do bit mais significativo, sendo este igual a '1', um valor negativo é caracterizado e na saída do componente é escrito o valor zero. Caso o bit mais significativo seja diferente de '1' são testados os outros 8 bits mais significativos e, em caso da constatação de que algum dos valores é diferente de '0', o valor 255 é escrito na saída. Não se constatando nenhuma das duas situações anteriores os 8 bits menos significativos do valor de entrada é escrito na saída do bloco de *clipping*. A Figura 4.13 apresenta a arquitetura do *clipping*.

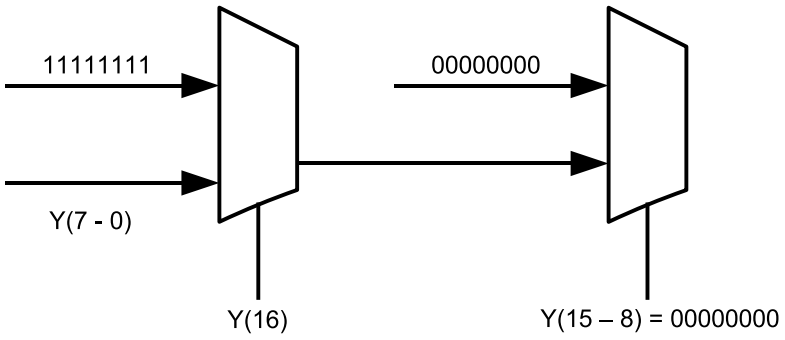

Figura 4.13: Arquitetura do clip

O *clipping*, como visto, compartilha o caminho de dados combinacional da saída da bi-predição. Ao final do processo o resultado é salvo num registrador de saída. A Tabela 4.25 apresenta os resultados de síntese do *clipping*.

Tabela 4.25: Resultados de síntese do clip

	Total	Percentual
Slices	6	1%
Flip Flops	0	0%
LUTs	10	1%
Blocos de RAM	0	0%
Multiplicadores	0	0%

4.5.6 Unidade de Controle do processamento das amostras

A unidade de controle do processamento das amostras foi implementada como máquinas de Mealy. Foram descritas duas máquinas, uma para o controle do processamento das amostras de luma e outra para o controle das amostras de croma. Cada máquina é responsável pela seleção dos parâmetros dos blocos, pela leitura dos *buffers* de entrada, pelo controle dos registradores de deslocamento, pela seleção dos resultados do bloco da bi-predição e por indicar a validade dos resultados. O controle de croma ainda é responsável pelo chaveamento do processamento entre as amostras de Cb e de Cr.

Os parâmetros dos blocos da lista 0 e da lista 1 a serem processados são associados de forma paralela na entrada do processador de amostras. Os valores dos vetores de movimento e dos parâmetros da predição ponderada de ambas as listas são armazenados e direcionados às entradas dos blocos no estado devido. A entrada em paralelo ocorre para os três componentes (Y, Cb e Cr) ao mesmo tempo.

Os controles implementam o processamento das amostras na forma de um *pipeline*. Assim, não é necessário esperar que um bloco esteja totalmente processado para se iniciar o processamento do seguinte. Os sinais de controle são escritos em registradores de deslocamento, com exceção dos sinais que são consumidos no primeiro estágio do *pipeline*. Os sinais são lidos de acordo com cada estágio ao qual são destinados.

A máquina de estados que controla o processamento das amostras de luma possui 19 estados. O primeiro estado é utilizado para sincronismo e início do processamento de um novo bloco, simplificada na Figura 4.14. Os nove estados seguintes controlam o processamento das amostras referentes à lista 0 (ou da lista 1, quando a lista 0 não é utilizada). Cada estado faz uma leitura do *buffer* de entrada. Os quatro últimos estados, desses 9, indicam se o resultado da interpolação é válido ou não, dependendo se o bloco é bi-preditivo ou não. Se o bloco for do tipo bi-preditivo nesses quatro estados é habilitada a escrita no *buffer* 4x4 e os resultados são marcados como inválidos. No último dos nove estados que controlam o processamento da lista 0, é selecionada a parte fracionária dos vetores de movimento da lista 0 para o interpolador de luma e os parâmetros da predição ponderada, sendo estes últimos atrasados em 1 ciclo antes de chegarem ao ponderador. Se o bloco for bi-preditivo, são executados os demais 9 estados que controlam o processamento das amostras relativas à lista 1. Os cinco primeiros destes apenas habilitam a leitura do *buffer* de entrada. Os quatro últimos, além de habilitar a leitura do *buffer*, habilitam a saída dos registradores de deslocamento do *buffer* 4x4 e selecionam a saída da média como entrada do processo de *clipping*. Estes também indicam como válidas as saídas do processamento de amostras. O último estado direciona os parâmetros da predição ponderada e a parte fracionária dos vetores de movimento da lista 1 para o ponderador e interpolador, respectivamente, e põe a máquina no estado de espera. A Figura 4.14 apresenta uma

simplificação da máquina de estados.

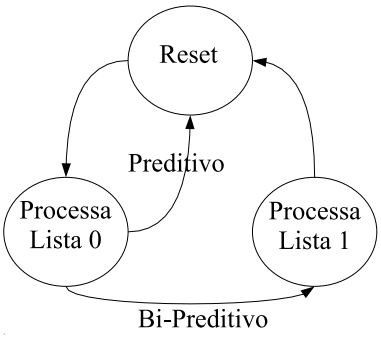

Figura 4.14: Controle do processador de luma

A máquina de controle do processamento de croma é semelhante à de luma. Como a interpolação de croma utiliza apenas uma posição extra e como os blocos são no formato 2x2, a máquina de estados possui 7 estados. O primeiro é responsável pelo sincronismo e pelo inicio do processamento. A lista 0 é tratada nos três estados seguintes e a lista 1 nos três últimos, tal qual acontece no controle do processamento de luma. Os processos realizados nos quatro últimos estados de responsáveis por cada lista no controle de luma, são realizados nos dois últimos estados de cada lista no controle de croma.

Como o processamento de croma acontece para amostras de Cb e Cr de forma serial são realizadas duas passagens pelos estados de processamento. Os sinais de controle são os mesmos da seleção do *buffer* de entrada e dos parâmetros do bloco. A primeira componente de croma processada é Cb e, no fim desta, a máquina retorna ao primeiro estado de processamento, dessa vez com o *buffer* de entrada e os parâmetros de Cr selecionados. Um sinal de saída indicando qual componente está sendo tratada foi adicionado a saída do processamento das amostras.

A Tabela 4.26 apresenta os resultados de síntese do controle do processamento de Luma. Os resultados de síntese do controle do processamento de croma estão descritos na Tabela 4.27.

Tabela 4.26: Resultados de síntese do controle de processamento de luma

	Total	Percentual
Slices	15	1%
Flip Flops	20	1%
LUTs	14	1%
Blocos de RAM	0	0%
Multiplicadores	0	20%

4.5.7 Resultados do Processador de amostras

O bloco processador de amostras apresenta uma arquitetura capaz de processar, de forma serial, partições bi-preditivas. Essa subseção detalha os resultados da arquitetura do processador de amostras.

A latência do processamento de amostras é de 13 ciclos de relógio, sem considerar o tempo para encher o *buffer* de entrada. Essa também é a latência para o processamento

Tabela 4.27: Resultados de síntese do controle de processamento de croma

	Total	Percentual
Slices	11	1%
Flip Flops	9	1%
LUTs	17	1%
Blocos de RAM	0	0%
Multiplicadores	0	0%

das amostras de luma. O processamento das amostras de croma, por sua vez, têm uma latência de 4 ciclos para apresentar a primeira amostra válida na saída.

A taxa de saída do processador de amostras é de um bloco 4x4 bi-preditivo a cada 19 ciclos de relógio. Caso o bloco não utilize a bi-predição, a taxa passa para um bloco a cada 10 ciclos de relógio. Esses valores consideram o *pipeline* cheio. Os valores acima são dominados pelo processamento das amostras de luma. O processamento das amostras de croma tem uma taxa de saída mais alta. a taxa de saída do processador de amostras de croma têm uma taxa de saída de um bloco de Cb e outro de Cr a cada 13 ciclos, para blocos bi-preditivos. Para blocos que não utilizam bi-predição, a taxa é de um bloco de Cb e outro de Cr a cada 7 ciclos.

As Tabelas 4.28, 4.29 e 4.30 apresentam os resultados de síntese para a arquitetura do processador de amostras de luma, de croma na forma serial e em paralelo, respectivamente. A freqüência de operação do processador de luma é de 100 MHz. Para as arquiteturas de processamento de croma os resultados foram de 100 e 104 MHz, para a arquitetura serial e paralela, respectivamente.

Tabela 4.28: Resultados de síntese do processador de luma

	Total	Percentual
Slices	1.768	13%
Flip Flops	1.569	5%
LUTs	2.579	5%
Blocos de RAM	0	0%
Multiplicadores	4	2%

Tabela 4.29: Resultados de síntese do processador de croma serial

	Total	Percentual
Slices	489	3%
Flip Flops	474	1%
LUTs	649	2%
Blocos de RAM	0	0%
Multiplicadores	14	10%

Os resultados de síntese mostram que a arquitetura cumpre os requisitos de desempenho esperados. O processamento das amostras, consumindo 19 ciclos para processar um

Tabela 4.30: Resultados de síntese do processador de croma paralelo

	Total	Percentual
Slices	748	4%
Flip Flops	856	2%
LUTs	806	2%
Blocos de RAM	0	0%
Multiplicadores	28	20%

bloco 4x4 bi-preditivo, tem capacidade, no pior caso, de processar 40,6 quadros 1080HD inteiramente bi-preditivos por segundo, trabalhando na freqüência de relógio apontada pela síntese. Para quadros do tipo P, que não utilizam bi-predição, a capacidade de processamento sobe para 77,1 quadros por segundo, também utilizando a freqüência apontada.

Comparando-se os resultados das Tabelas 4.30 e 4.29, nota-se uma redução significativa da quantidade de recursos.

O processamento das amostras de croma operando componentes Cb e Cr de forma paralela seria capaz de processar seqüências de vídeo que utiliza a sub-amostragem de cor 4:2:2, como é o caso dos perfis High.

O processador de amostras gerou um artigo aceito e apresentado na conferência Norchip de 2005, sob o título *Motion Compensation Sample Processing for HDTV H.264/AVC Decoder* (AZEVEDO et al., 2005).

4.6 Controle Geral do MC

O controle geral do compensador de movimento gerencia o fluxo de dados entre os preditores, a *cache*, o processamento das amostras e os *buffers* serializadores. Os preditores dos vetores de movimento entregam os vetores de movimento, índices de referência e *flags* de utilização das listas de um macrobloco. A saída dos preditores é armazenada em *buffers*. Um controle de acesso à *cache* lê esses dados armazenados e os consulta na *cache*, escrevendo a saída nos *buffers* de entrada do processamento de amostras. A saída do processamento das amostras é, então, armazenada num *buffer* para serializar a saída.

Como todos os componentes do compensador de movimento têm latência variável, o controle foi desenvolvido orientado a dados. Várias máquinas de estados controlam o consumo dos dados entre os blocos. Cada bloco inicia seu processamento quando os dados necessários ao seu processamento e o espaço disponível para a escrita dos seus resultados estejam disponíveis.

Entre os blocos e ao fim do compensador de movimento existem *buffers* para armazenar os resultados de cada bloco. Na saída do preditor três *buffers* armazenam os vetores de movimento preditos, os índices de referência e as *flags* de utilização de cada lista, para cada bloco do macrobloco corrente. Na saída da *cache* três *buffers* armazenam as amostras de cada uma das três componentes, Y, Cb e Cr, a serem processadas. Um *buffer* na saída do processamento armazena as amostras compensadas das três componentes.

O controle geral da compensação de movimento está dividido em três máquinas de estado. A primeira controla a leitura da *cache*, a segunda o início do processamento das amostras e a terceira a escrita dos dados no *buffer* de saída. Essas máquina são detalhadas nas próximas subseções.

4.6.1 Controle de leitura da Cache

A função do controle de leitura da *cache* é copiar os dados presentes na *cache* para o *buffer* de entrada do processamento. Uma vez que os *buffers* de vetores e índices possuem valores válidos e os *buffers* de entrada do processamento não estão cheios, o controle de leitura da *cache* é iniciado.

O controle lê as *flags* de utilização das listas para o bloco 4x4 corrente. Para cada *flag* de utilização de lista ligada, o controle lê os vetores de movimento e consulta o POC relativo ao índice de referência da lista. O vetor de movimento é somado à posição do bloco no quadro atual e ajustado à janela de filtragem. Com a posição absoluta da área a ser processada e o identificador de quadro, inicia-se a consulta a *cache*. São realizadas nove consultas, uma vez que a *cache* retorna linhas da área a ser filtrada onde, a cada consulta, a posição da linha é incrementada de um. Para cada consulta é verificada a presença do dado na *cache* e se o *buffer* de entrada do processamento está cheio. A operação descrita é realizada para cada lista, primeiro pra lista 0, se disponível, e depois para a lista 1.

A primeira melhoria citada na subseção 4.3.3, a qual se refere a leitura apenas das linhas necessárias a interpolação, dado a parte fracionária do vetor de movimento. Para cada leitura é testado o valor da parte fracionária do vetor relativa a posição vertical. Se esta for zero, ou seja, não havendo interpolação na vertical, apenas os dados relativos as quatro linhas pertencentes ao bloco é lida da *cache*.

4.6.2 Controle do Processamento de Amostras

Uma vez que a área a ser processada se encontra nos *buffers* de entrada do processamento, são selecionados os parâmetros do processamento. Uma segunda máquina de estados controla a seleção dos vetores de movimento, dos parâmetros da predição ponderada e se o bloco é bi-preditivo ou não. Caso apenas o quadro referente à lista 1 for processado, os parâmetros referentes a ele são escritos nas entradas do processamento referentes à lista 0. Se o *buffer* de saída não estiver cheio o sinal de inicio do processamento é ligado.

4.6.3 Controle do Buffer de Saída

Os requisitos da arquitetura pedem que a saída seja na taxa de uma amostra por ciclo. A ordem de saída também é definida: primeiro as 256 amostras de luma seguidas por 64 da componente Cb de croma e, por último, as 64 amostras da componente Cr. Todavia, para cumprir os requisitos de desempenho, a saída do processamento das amostras é tanto paralela em relação às componentes de cor quanto em relação às amostras de uma mesma componente. Para cumprir os requisitos do projeto, um *buffer* serializador é necessário.

Para o *buffer* foi utilizado um bloco de memória interna do FPGA, com uma porta de escrita e outra de leitura. Como o processamento de luma entrega quatro amostras por ciclo, uma largura de palavra de 32 bits foi utilizada. Como as amostras de croma são entregues em pares, o primeiro par de amostras válidas é armazenado e concatenado com o segundo. Dessa forma, temos sempre 4 amostras por posição no *buffer*. Nessa configuração, o bloco de memória comportaria até 5 macroblocos. Mas, para simplificar o controle, o mesmo armazena 4 macroblocos.

Para utilizar apenas um bloco de memória, as saídas válidas de luma e croma tiveram que ser organizadas de forma a não haver sobreposição de saídas válidas entre os processadores de amostra de luma e de croma.

O *buffer* guarda as amostras na ordem de saída. Sendo, para cada macrobloco, as primeiras 64 posições dedicadas a amostras de luma, as 16 seguintes de Cb e as 16 demais para Cr. Como as amostras válidas de luma e croma se intercalam, é necessário um mecanismo para os escrever de forma organizada no bloco de memória. São necessários três endereços para armazenar o próximo grupo de 4 amostras, onde cada endereço armazena a próxima posição de uma determinada componente.

Uma máquina de estados é responsável pelo controle de escrita no *buffer*, outra responde pela leitura do *buffer*. Cada componente de cor requer um registrador para manter o endereço o qual a próxima amostra deve ser escrita. A máquina de estados de escrita reveza a escrita entre a componentes, ordenando-as para a seqüência de saída.

Na saída do *buffer* existe um sinal que indica a presença de um macrobloco válido no *buffer*. Para cada um dos 4 macrobloco que podem ser armazenados no *buffer* existe um registrador que indica se o mesmo é válido ou não. O controle de escrita é responsável por indicar quando um macrobloco foi escrito e o controle de leitura por indicar quando o mesmo foi lido.

Um multiplexador na saída do bloco de memória seleciona qual das quatro amostras de cada posição do *buffer* deve ser enviada para a saída. O multiplexador é controlado pelo controle de leitura, que é acionado por uma requisição.

4.7 Resultados da arquitetura do compensador de movimento

Essa seção apresenta os resultados obtidos com o desenvolvimento da MoCHA. São abordados os resultados de desempenho da mesma e uma comparação com outras soluções presentes na literatura.

A latência mínima do compensador de movimento é de 233 ciclos de relógio, sem considerar a existência de *misses* na *cache*, para apresentar o primeiro resultado válido, já serializado, na saída. Essa também é a latência mínima de processamento de um macrobloco inteiro, uma vez que o primeiro dado serializado só está disponível quando todo o macrobloco já foi processado. Sem a serialização, a latência mínima é de 42 ciclos, sendo que a primeira amostra válida é uma amostra de croma. Para cada *miss* na *cache* são necessários, no mínimo, mais 16 ciclos de relógio. A latência mínima apresentada considera um macrobloco contendo uma partição única no formato 16x16 em um *slice* do tipo P, o qual não utiliza a bi-predição. A latência máxima da arquitetura do compensador de movimento, sem considerar a existência de *misses* na *cache*, é de 590 ciclos de relógio. Da mesma forma, essa também é a latência máxima de processamento de um macrobloco inteiro. Essa latência máxima corresponde a quantidade de ciclos necessária para se gerar o primeiro resultado válido, serializado, de um macrobloco direto, com a predição direta espacial, no qual todos os blocos 4x4 são bi-preditivos. Essa latência considera, também, que os dados do bloco co-localizados estejam disponíveis quando solicitados. Sem a serialização, a latência máxima é de 275 ciclos.

A utilização de uma *cache* para explorar a redundância dos dados, em conjunto com melhorias propostas na organização das amostras, alcança uma melhoria considerável nos acessos a memória externa. A economia na quantidade de acessos à memória externa é superior 60%, para os casos testados. Quando uma penalidade de um ciclo por troca de linha de memória é imposta, a economia de ciclos de acesso supera os 75%. Essas economia é dada comparando-se com as quantidades de acessos e ciclos originais, sem a *cache* nem melhorias.

A arquitetura realiza o processamento dos dois blocos, que dão origem ao bloco bi-

preditivo, de forma serial. Dessa forma, são economizados recursos de *hardware*, uma vez que a duplicação da estrutura de processamento não é requerida. Mesmo com a serialização, não é necessário um aumento na freqüência de operação, pois a taxa de processamento atingida é suficiente para atender os requisitos deste projeto. A freqüência de operação é igual ou inferior às apresentadas na literatura para o perfil Baseline, que não trata a bi-predição, indicando que a solução desenvolvida é adequada aos objetivos deste trabalho. Essa estratégia requer um bloco adicional, apresentado na subseção 4.4.4, que não apresenta um consumo de recursos considerável, e um *buffer* para serializar a saída.

Em decorrência das estratégias implementadas nas máquinas do controle geral, que alteraram a taxa de saída do processamento de amostras, a taxa de saída do compensador de movimento é de, no mínimo, um macrobloco a cada 192 ciclos de relógio e de, no máximo, um macrobloco a cada 336 ciclos, desconsiderando-se a leitura do serializador. Esses dados são referentes às mesmas situações apresentadas para o cálculo das latências mínima e máxima. Esses valores consideram o *pipeline* cheio.

A síntese da arquitetura utilizou, tal qual as demais sínteses apresentadas, o dispositivo XC2VP30-7 da família Virtex II Pro (VIRTEX SERIES, 2005) da Xilinx (XILINX: THE PROGRAMMABLE LOGIC COMPANY, 2005). A ferramenta utilizada para a síntese foi o ISE (PLATFORM STUDIO AND THE EDK, 2005), tendo como sintetizador o Symplify Pro 8.1 (SYNPLICITY: PRODUCTS: SYNPLIFY PRO, 2005) da Synplicity (SYNPLICITY: HOME, 2005). A Tabela 4.31 apresenta os resultados de síntese para a arquitetura do compensador de movimento. A freqüência máxima de operação reportada foi de 100.5 MHz.

Tabela 4.31: Resultados de síntese da MoCHA

	Total	Percentual
Slices	8.465	61%
Flip Flops	5.671	27%
LUTs	10.835	39%
Blocos de RAM	21	15%
Multiplicadores	12	8%

Nos resultados apresentados na Tabela 4.31, não estão incluídos os resultados relativos à predição ponderada implícita. Apesar de esta predição estar desenvolvida, até o momento da escrita dessa dissertação, este bloco não tinha sido integrado à arquitetura.

Os resultados de síntese mostram que a arquitetura cumpre os requisitos de desempenho esperados. O compensador de movimento desenvolvido é capaz de processar, no pior caso, até 36,7 quadros HDTV de 1080 por 1920, inteiramente bi-preditivos por segundo, trabalhando a freqüência de relógio de 100 MHz. Para quadros do tipo P (não bi-preditivo), a capacidade de processamento sobre para 64,3 quadros por segundo, para a mesma freqüência de operação.

Na literatura pesquisada, como foi apresentado na seção 3.4, não foi encontrado nenhum trabalho que descrevesse em detalhes uma solução arquitetural para a compensação de movimento para o perfil Main do padrão H.264. Os trabalhos descrevendo decodificadores completos não apresentam a arquitetura do compensador de movimento, nem detalhes sobre seu resultado de síntese.

Deve-se ressaltar que os trabalhos da literatura descrevem soluções para o perfil Baseline. Este perfil não utiliza a predição ponderada nem a bi-predição. Devido ao não

suporte da bi-predição, a taxa de processamento de dados de entrada, nesses trabalhos, é a metade da suportada pelo presente trabalho. A predição dos vetores de movimento é outro fator a ser considerado na comparação. A predição para o perfil Main é consideravelmente mais complexa do que a encontrada no perfil Baseline. Enquanto a predição dos vetores de movimento para o perfil Baseline suporta apenas a lista 0 e a predição dos macroblocos *skip* nos *slices* P, no perfil Main, além de suportar essas operações, suporta ainda a predição para a lista 1 e a predição direta.

A seguir serão apresentadas algumas comparações, que foram realizadas para cada módulo do compensador de movimento individualmente. A primeira arquitetura a ser comparada é a predição dos vetores de movimento e índices de referência.

(WANG et al., 2005) é o único trabalho, dos listados, que apresenta uma solução para a predição de vetores de movimento, porém apenas para o perfil Baseline. Todavia, nenhum resultado é apresentado sobre este bloco. Nos trabalhos sobre decodificadores completos, também não são apresentadas informações sobre a predição dos vetores. Dessa forma não foi possível uma comparação desse bloco com outros trabalhos da literatura.

Nos trabalhos que apresentam soluções para a compensação de movimento, são apresentadas soluções para a interpolação para oitavo de pixel das amostras de croma, sem uso de multiplicadores. Essas abordagens não foram consideradas para esse trabalho devido à disponibilidade, no dispositivo FPGA, de multiplicadores hardwired, não utilizado por outros blocos do decodificador. Dessa forma, a utilização dos multiplicadores para a interpolação foi a opção mais econômica em termos de células lógicas.

O interpolador de luma é um dos principais focos dos trabalhos na literatura, relativos à compensação de movimento. A seguir, são apresentadas as comparações arquiteturais entre o interpolador de luma proposto e os apresentados na revisão bibliográfica.

Esse trabalho utiliza a mesma estrutura de filtros de luma apresentada em (WANG et al., 2005). Todavia, a implementação foi alterada para atingir a taxa de processamento necessária para tratar quadros bi-preditivos. Deste modo, foi possível atingir o dobro da performance em relação ao trabalho original, considerando a mesma freqüência de operação. Em ambos os casos, o processamento se dá sobre blocos 4x4, gerando todas as posições de metade de pixel e processando as posições de quarto de pixel a partir dos vetores de movimento. Devido a comunicação direta com a memória, a interpolação descrita em (WANG et al., 2005), processa uma linha ou coluna a cada três ciclos de relógio, levando até 560 ciclos para processar as amostras de luma de um macrobloco. Para o formato 1080HD, a freqüência requerida para a arquitetura original, é de 100 MHz.

Em (WANG et al., 2005) é apresentada uma solução para o interpolador de luma a qual interpola apenas as amostras de metade de pixel que serão necessárias. A estrutura utiliza oito filtros e processa uma amostra por vez. Uma estrutura de *buffers* é utilizada para o reuso de dados na entrada. As amostras de luma de um macrobloco levam 356 ciclos para ser processada por essa arquitetura. Os resultados de síntese apresentados não contemplam o formato 1080D, atingindo uma freqüência de operação inferior a requerida para esse formato. Para o formato 1080HD a freqüência requerida é de 150 MHz. Esta arquitetura não foi utilizada neste trabalho, pois, avaliando seus resultados, concluiu-se que ela não seria capaz de atingir o desempenho necessário.

A alternativa arquitetural apresentada em (LIE et al., 2005), por não apresentar uma solução compatível com o padrão, não foi considerada.

O acesso às amostras dos quadros de referência é crítico no compensador de movimento. As arquiteturas desenvolvidas buscam a reutilização dos dados de modo a reduzir o número de acessos à memória.

Em (WANG et al., 2005) a arquitetura do interpolador foi desenvolvida para reduzir o acesso à memória externa. Esta solução utiliza *buffers* nos filtros para armazenar e reutilizar os dados de entrada. O trabalho apresenta economia de 30% na quantidade de acessos às amostras.

A arquitetura proposta nessa dissertação utiliza-se de uma hierarquia de memória para acessar as amostras dos quadros de referência. Essa abordagem é inédita na literatura e alcança uma economia de 60% no acesso à memória externa, para a configuração utilizada.

Em (WANG; LI; HUANG, 2005) são apresentadas estratégias para a redução do número de acessos à memória. Em (WANG et al., 2005) alguma dessas estratégias são utilizadas. A economia no acesso à memória é de 60%, em média, em relação a solução sem essa otimização. O presente trabalho utiliza duas dessas estratégias apresentadas, adaptadas às características particulares do presente trabalho.

A simples quantidade de acessos à memória não é uma abordagem adequada. As memórias requerem ciclos adicionais para trocar a leitura de dados entre linhas e bancos. Os resultados em (WANG; LI; HUANG, 2005) (WANG et al., 2005) não avaliam essa característica, nem as estratégias apresentadas a levam em consideração. Tal qual apresentado na subseção 4.3.3, quando essa penalidade é avaliada, a utilização da *cache* oferece ganhos significativos.

Da mesma forma, as estratégias apresentadas em (WANG; LI; HUANG, 2005) têm perdas em relação a utilização do barramento. Como os dados a requeridos não estão alinhados com a memória, existe um desperdício das dados que trafegam pelo barramento. A *cache*, por copiar blocos de amostras alinhados com a memória, tem utilização de 100%.

Uma comparação das áreas consumidas pelas diferentes implementações é difícil. Os trabalhos listados focam principalmente na interpolação e, além disso, a tecnologia de implementação é completamente diferente. Enquanto os trabalhos listados utilizam *standard cell* para a síntese do circuito, a arquitetura proposta utiliza os FPGAs e suas características particulares. A comparação da freqüência de operação enfrenta o mesmo problema, uma vez que a tecnologia de implementação é decisiva na definição da freqüência máxima de operação.

A arquitetura proposta para o compensador de movimento, com suporte a quadros bi-preditivos, é, até o momento, inédita na literatura, sendo a única que descreve uma arquitetura para a bi-predição. Os trabalhos relativos a decodificadores completos não apresentam a solução para esse processamento.

4.8 Considerações finais sobre a arquitetura do compensador de movimento

Este capítulo apresentou a arquitetura para compensação de movimento MoCHA. Foram detalhados os três principais blocos, que perfazem um *pipeline*, da arquitetura: o preditor de vetores de movimento, o acesso à memória e o processamento de amostras. Para o preditor de vetores de movimento e índices de referência foram apresentadas duas arquiteturas, uma para a predição dos *slices* P e outra para a predição dos *slices* P e B, com suporte a predição direta espacial. No bloco de acesso à memória, foram detalhados uma *cache*, para reduzir a quantidade de banda de memória requerida para a aplicação, e um extrapolador de amostras, que trata os casos em que os vetores apontam para fora do quadro. O processamento de amostras, e seus caminhos de dados para luma e croma em

paralelo, foram também detalhados neste capítulo. Por fim apresentou-se os resultados da arquitetura desenvolvida.

No próximo capítulo será apresentada a validação da arquitetura e a prototipação do processamento de amostras.

5 VALIDAÇÃO E PROTOTIPAÇÃO DA ARQUITETURA

Esse capítulo aborda na seção 5.1 os procedimentos para a validação da arquitetura do compensador de movimento e, na seção 5.2, os procedimento da prototipação do bloco de processamento de amostras do compensador de movimento. Devido às restrições temporais, não foi possível a prototipação dos demais blocos do compensador de movimento. As considerações finais sobre a validação e a prototipação são apresentadas na seção 5.3.

5.1 Validação

Para comprovar o funcionamento adequado da arquitetura descrita em VHDL, foi desenvolvida uma série de testes para a validação desta descrição. Como uma validação formal se faz inviável, dentro do cronograma disponível, partiu-se para a abordagem da validação através de simulações exaustivas.

Para esta abordagem de validação é necessária a existência de uma grande quantidade e variedade de casos de teste, os quais devem apresentar dados de entrada e resultados comprovadamente corretos. A busca desses casos foi feita através do *software* de referência do padrão H.264. Então foram localizadas, no código do *software* de referência, as variáveis necessárias para os testes e inseridas funções para capturá-las em arquivos de texto.

Obtendo os casos de teste teve início a fase de simulação, onde se utilizou o *software* ModelSim (MODELSIM, 2005) da Mentor Graphics para inserir os dados obtidos na arquitetura desenvolvida. Antes das simulações, preferiu-se converter os dados de teste para valores em binário, já que essa é a forma com que os dados são tratados pela arquitetura. Durante a simulação, os resultados calculados pelo compensador de movimento foram salvos em outros arquivos de texto.

De posse dos resultados gerados pelo *software* de referência e dos resultados extraídos através da simulação da arquitetura, foram escritos outros pequenos programas capazes de fazer a comparação entre esses resultados.

A validação da arquitetura do compensador de movimento se deu de forma incremental e à medida que os blocos principais eram descritos. O primeiro bloco a ser validado foi o processador de amostras. Quando este estava validado, partiu-se para a validação dos preditores dos vetores e índices de referência. Com os dois blocos principais validados, deu-se início à validação do compensador de movimento como um todo.

Para o processador de amostras e para o preditor de vetores de movimento e índices de referência, o procedimento de validação dependeu apenas de entradas capturadas. Assim todas as entradas de cada um dos blocos estavam presentes nos arquivos de entrada da simulação.

Para a validação da arquitetura do compensador de movimento como um todo, nem

todas as entradas são passíveis de serem armazenadas. Como o compensador de movimento depende de funções do controle geral e de dados armazenados em memória, a simulação teve que prever o comportamento dessas estruturas.

A subseção 5.1.1 apresentará o procedimento de validação do processamento de amostras e do preditor de vetores de movimento e índice de referência, uma vez que usam o mesmo procedimento de validação. A subseção 5.1.2 detalhará os procedimentos para a validação da arquitetura do compensador de movimento.

5.1.1 Validação do processador de amostras e do preditor de vetores de movimento e índices de referência

Essa subseção apresentará cada uma das etapas do processo de validação do processador de amostras e do preditor de vetores de movimento e índices de referência. Serão detalhados os processos de localização, captura e tratamento dos dados de entrada e saída, a estrutura do *test bench*, a simulação e a comparação dos resultados.

Para a validação dos blocos, foram utilizado 100 quadros da seqüência Foreman, no formato QCIF. Primeiramente, foram utilizados os 50 quadros do tipo P. Em seguida, foi utilizada a seqüência com o intervalo de quadro PB, sendo o primeiro quadro da seqüência, obrigatoriamente, do tipo I.

5.1.1.1 Localização dos casos de teste

Para obter os casos de teste necessários, primeiramente foi feito um estudo do *software* de referência do padrão H.264. O objetivo do estudo foi o de localizar as variáveis necessárias, e o ponto onde deveriam ser capturadas. Para esse trabalho de busca foi utilizada a versão JM 9.5 (H.264/AVC SOFTWARE COORDINATION, 2005) do *software* de referência.

Descobriu-se que as funções relativas à predição do tipo inter-quadro, no JM 9.5, estão bastante restritas ao arquivo *macroblock.c* do decodificador (ldecod). Foi possível, então, localizar todas as variáveis de controle, dados a serem processados e resultados obtidos, dentro do arquivo *macroblock.c*.

O processamento das amostras ocorre dentro da função *decode_one_macroblock*. As variáveis relativas a esse bloco se encontram dentro dessa função.

As variáveis de controle a serem capturadas são as mesmas tanto para luma quanto para croma. No entanto, o ponto no qual foram capturados é distinto. Dentro da função *decode_one_macroblock* existem dois laços de iteração principais, um para as amostras de luma e outro para as amostras das duas componentes de croma. Assim as variáveis a serem capturadas para cada componente de cor se encontram em locais diferentes da função.

Além das variáveis de controle, são necessárias as matrizes com os dados de entrada, bem como das matrizes com os resultados já processados. Quando se trata de capturar os dados de entrada para luma, o ponto utilizado é imediatamente após a chamada da função *get_block*, que é a função responsável pela interpolação de quarto de pixel. O resultado da compensação se encontra em atribuições realizadas dentro de estrutura de controle condicional *if* a qual testa o valor da *flag img->apply_weights*. Para obter os dados corretos para croma, dentro do laço de croma, devem-se capturar os valores das amostras após o cálculo das variáveis *if1*, *jf1*, *if0* e *jf0*, enquanto a saída deve ser salva junto aos testes com a *flag img->apply_weights*.

Para a predição dos vetores de movimento e índices de referência, os dados a serem capturados estão espalhados por todo o arquivo *macroblock.c*. Devido à estrutura de

processamento do *software* de referência, as variáveis com os dados para a predição dos vetores trocam de valores ao longo da execução. Após uma analise reversa das variáveis, se chegou ao ponto onde as mesmas são entregues pelo *parser*.

O *software* de referência tem uma opção a qual faz com que o mesmo salve os parâmetros do *bitstream* já decodificados. Essa função é implementada no *parser*, mas a *string* de saída é passada na chamada da função que pede o *token*. Com essa informação, localizou-se as chamadas do *parser* relativas às informações utilizadas pelo preditor de vetores de movimento. As variáveis devem ser capturadas logo após a chamada do *token*.

5.1.1.2 Obtenção dos casos de teste

Com a localização das entradas, o passo seguinte foi a captura dessas variáveis em arquivo texto. A partir da localização dos pontos em que as variáveis deveriam ser capturadas, foram inseridas funções para salvá-las em arquivo texto, num formato adequado para cada tipo de variável.

Para os dados relativos ao processamento das amostras, foram criadas matrizes bidimensionais para armazenar as amostras. Foram definidas duas matrizes de inteiros com 9x9 posições, *test_block* e *test_block_bw*, que armazenam dados de entrada, vindos da lista 0 e da lista 1, capturadas após a chamada da função *get_block*.

Em um arquivo, chamado de *INTER_IN.txt*, são salvas as variáveis de controle e contexto, bem como matrizes contendo as amostras a serem processadas. As informações são salvas a cada iteração de tratamento de bloco. O arquivo foi organizado de forma com que todas as variáveis e um contador de blocos ficassem na mesma linha, seguida de nove ou dezoito linhas de amostras a serem processadas. A necessidade de nove ou dezoito linhas de dados se explica pela existência de blocos do tipo P que utilizam uma matriz de 9x9 amostras, enquanto blocos do tipo B podem utilizar duas matrizes 9x9.

Da mesma forma, foi procedido com as variáveis de croma, que foram salvas em um arquivo chamado *INTER_IN_C.txt*. Esse arquivo contém variáveis de controle seguidas de uma ou duas matrizes de 3x3 amostras, tal qual as amostras de luma.

Dentre as variáveis de controle e contexto salvas, estão: tipo do slice, tipo do macrobloco e do sub-macrobloco, os vetores de movimento para lista 0 e lista 1, os valores de deslocamento e fator multiplicativo para predição ponderada, além do contador de bloco. Estas informações devem preceder os valores a serem interpolados, pois através delas pode-se reconhecer o tipo de bloco a ser processado e inserir, na simulação, a quantidade adequada de amostras.

Quanto aos dados de saída, estes são capturados em matrizes em memória, dentro dos testes da *flag img->apply_weight*. Os dados são salvos no arquivo *INTER_OUT_ref.txt*, para os resultados de luma, e *INTER_OUT_C_ref.txt*, para os de croma. Os resultados da interpolação de luma compõem uma matriz de 4x4 amostras precedidos pelo contador de bloco. Da mesma forma foram salvas as matrizes 2x2, com os resultados da interpolação dos valores de croma.

A captura dos dados da predição dos vetores de movimento se deu de forma ligeiramente diferente. Como algumas das entradas da predição nem sempre estão presentes e suas quantidades variam de acordo com o tipo de macrobloco, essas informações foram salvas em diversos arquivos separados. Essa estratégia também foi necessária devido à estrutura de *buffers*, dos quais são lidas essas informações e que tiveram que ser implementados no *test bench*.

As variáveis para a predição são armazenadas em arquivo tão logo são entregues pelo *parser*. Como as chamadas estão espalhadas por várias funções do código de referência,

cada arquivo constitui de uma lista de uma variável especifica. Assim tem-se um arquivo para armazenar os tipos de macrobloco e seus tipos de sub-macrobloco, um arquivo para armazenar os índices de referência da lista 0, outra para lista 1 e um arquivo para cada lista de vetor de movimento diferencial.

Os resultados da predição são capturados e escritos entre os laços de processamento de amostras de luma e croma da função *decode_one_macroblock*. Essa localização foi escolhida, pois até então, existe no código de referência operações relativas a predição de vetores. São salvos no arquivo *MV_out.txt* todas as referências de ambas as listas e os vetores de movimento, também de ambas as listas, para cada bloco do macrobloco.

5.1.1.3 Test benches

Para utilizar os dados já capturados e convertidos em uma simulação no ModelSim, foi escrito um test bench para o processador de amostras e outro para o preditor de vetores de movimento. O test bench é um código escrito em VHDL que descreve um módulo que envolve a arquitetura a ser testada e é responsável pela inserção de estímulos de entrada e pela leitura dos estímulos de saída desta arquitetura.

Na inserção dos estímulos, esse módulo se encarrega de ler os dados dos arquivos de texto. Para o processador de amostras, ele identifica, a partir dos parâmetros de cabeçalho do bloco, qual tipo de bloco será tratado, posicionando os parâmetros nas entradas. Em seguida são lidas, a cada ciclo de relógio, as linhas com as amostras, sendo estas posicionadas na entrada do *buffer* de entrada e acionando seu sinal de habilitação de escrita. Para o preditor, o *test bench* carrega os arquivos com as entradas, sendo cada linha de cada arquivo lida de acordo com a solicitação do preditor. A solicitação de leitura consome uma linha dos arquivos de entrada, assim, tem-se as entradas dos preditores funcionando como *buffers*.

Além de inserir os estímulos de entrada na arquitetura testada, o test bench foi escrito para observar os sinais de saída e identificar quando o sinal de validade estiver ativo. Este sinal de validade indica a existência de amostras já processadas e prontas para serem lidas. As amostras processadas são, então, salvas em arquivos de texto, acompanhadas de um contador de blocos.

5.1.1.4 Simulação

Apenas depois desses processos de busca dos dados e descrição dos test benches, pôde-se executar as simulações que validariam o processador de amostras e o preditor de vetores de movimento.

Para esta simulação, como já citado, foi escolhido o *software* ModelSim desenvolvido pela empresa Mentor Graphics em sua versão SE 6.0b (MODELSIM, 2005). Esta ferramenta oferece grande facilidade em detecção de problemas no módulo testado, uma vez que permite a visualização do comportamento de todos os sinais internos à arquitetura. Além disso, utilizando ModelSim, pode-se fazer simulações não apenas comportamentais, mas também simulações da arquitetura já mapeada em diversas famílias de FPGA.

Como referência para testes, utilizou-se o vídeo Foreman, de resolução 176x144 pixeis e duração de 100 quadros.

As simulações, para os dois blocos, foram primeiramente aplicadas para o vídeo citado, codificado apenas com quadros do tipo P, além de um quadros tipo I que deve ser o primeiro quadro obrigatoriamente. Uma vez terminada a simulação para os blocos P, passou-se para a simulação da arquitetura com o mesmo vídeo, mas desta vez codificado com quadros tipo P e quadros tipo B, intercalados entre si.

Os resultados processados pelas simulações do compensador de movimento foram armazenados em dois arquivos de texto. Com esses arquivos estão disponíveis dados suficientes para comparar com os resultados extraídos do JM 9.5.

5.1.1.5 Comparação dos resultados

De posse dos arquivos contendo os valores de saída do processador de amostras, do preditor de vetores de movimento e dos arquivos com os resultados gerados pelo *software* de referência, foram escritos programas em C/C++ capazes de comparar tais resultados.

A execução destes *softwares* de comparação mostrou que a arquitetura estava, de fato, de acordo com o perfil Main do padrão H.264.

A plataforma de validação desenvolvida permite que, facilmente se possam aplicar testes com diversos vídeos de diferentes definições, no entanto, os mais de 150 mil blocos processados de maneira correta, a partir do vídeo Foreman.yuv, dão uma grande confiabilidade no funcionamento do processador de amostras e do preditor de vetores de movimento e índices de referência.

5.1.2 Validação do compensador de movimento

Para a validação da arquitetura do compensador de movimento, a estrutura descrita na subseção 5.1.1 foi seguida. As entradas para o compensador de movimento são as mesmas utilizadas no preditor de vetores de movimento e as saídas a serem analisadas são as mesmas do processamento de amostras.

Todavia, o *test bench*, para o compensador de movimento, tem que lidar com algumas funcionalidades destinadas ao controle geral do decodificador. As principais funções que o *test bench* deve prover são: a conversão entre índice de referência e POC da imagem de referência; o armazenamento dos quadros decodificados; interface com a cache e o armazenamento dos vetores e índices dos quadros anteriores.

Para as seqüências contendo apenas quadros P, a conversão do índice de referência da lista 0 em POC consiste em subtrair do POC atual o valor do índice de referência, uma vez que, para os vídeos contendo apenas quadros P, a codificação se deu na ordem de exibição. Para as seqüências contendo quadros B, uma tabela, contendo o valor do POC para cada índice, foi descrita para cada quadro da seqüência.

O armazenamento dos quadros previamente decodificados foi implementado como uma ROM. O vídeo decodificado foi armazenado nessa memória, que faz interface com a cache. Cada componente de cor foi instanciada numa memória separada. A interface de comunicação com a cache foi implementada nessa ROM. Mesmo utilizando-se um vídeo na resolução de QCIF, não foi possível o armazenamento da seqüência de cem quadros utilizada nas demais validações. O Modelsim não conseguiu gerenciar uma quantidade de memória maior do que a correspondente para 16 quadros. Essa limitação também se deve a quantidade de memória física disponível no computador, 1 GB de RAM.

O armazenamento dos vetores de movimento e dos índices de referência dos quadros processados, a serem utilizados pela predição direta, também foram implementados do *test bench* como memórias. A predição direta utiliza os vetores e índices do quadro de índice 0 da lista 1. Apesar da ordem de codificação dos quadros ser arbitrária, o POC do quadro de referência da lista 1 é previamente conhecido para o vídeo utilizado. Assim, são armazenadas apenas as informações que sabidamente serão utilizadas. A ordem de codificação do vídeo permite o armazenamento das informações de apenas um quadro. O preditor acessa de essa memória de modo direto, nesta implementação.

Os procedimentos de simulação, previamente descritos, foram aplicados à arquitetura

completa do compensador de movimento. A o momento da entrega dessa dissertação, os trabalhos de validação da arquitetura MoCHA ainda não tinham sido concluídos, mas, para diversos casos, a arquitetura apresentou resultados corretos.

5.2 Prototipação do Processamento de Amostras do Compensador de Movimento

Essa seção abordará a metodologia e os procedimentos empregados para a prototipação do bloco de processamento de amostras da arquitetura do compensador de movimento do decodificador de vídeo segundo o padrão H.264.

Devido às restrições de tempo, apenas o processamento de amostras pôde ser prototipado. Como o processamento das amostras foi o primeiro bloco a ser implementado e validado, o mesmo esteve disponível para a prototipação em tempo hábil.

Antes de se partir para a prototipação, o bloco de processamento das amostras foi validado em uma simulação pós *place and route*. Nesse tipo de simulação a descrição é mapeada para as células lógicas do dispositivo FPGA, sendo estas já posicionadas e com as linhas de conexão definidas. Os atrasos das células e das conexões são anotados e utilizados na simulação. Tendo a descrição sido validada dessa forma, foi dado inicio a prototipação.

A subseção 5.2.1 apresentará a plataforma de prototipação utilizada. A metodologia de prototipação e os procedimentos serão apresentados na seção 5.2.2. A subseção 5.2.3 descreve os procedimentos para se validar o protótipo na freqüência requerida de operação.

5.2.1 Plataforma de prototipação

A plataforma de prototipação consiste do *software* EDK/ISE (PLATFORM STUDIO AND THE EDK, 2005) da Xilinx (XILINX: THE PROGRAMMABLE LOGIC COMPANY, 2005) e da placa de desenvolvimento XUP Virtex-II Pro (XUP V2P - VIRTEX-II PRO DEVELOPMENT SYSTEM, 2005) da produzida pela Digilent Inc. (DESIGN RESOURCES FOR DIGITAL ENGINEERS, 2005).

O *software* de desenvolvimento é responsável pelo fluxo de síntese e programação da placa de desenvolvimento. O mesmo conta com uma biblioteca de softcores que implementam o controle dos dispositivos presentes na placa e estruturas de comunicação dentro do FPGA, como barramentos OPB e PLB.

A placa de desenvolvimento XUP Virtex-II Pro conta com o dispositivo FPGA Virtex-II Pro XC2VP30-7 (VIRTEX SERIES, 2005). Esse dispositivo possui 30.816 células lógicas, 136 multiplicadores de 18 bits, 2.448Kb de memória interna espalhados em 153 blocos de RAM e dois processadores PowerPC´s 405. A placa possui um slot DIMM (168 pinos, para memórias com larguras de dados de 64 bits) e aceita memórias do tipo DDR DRAM de até 1GB de capacidade. Está instalado na placa um módulo de memória DDR DRAM de 512 MB PC2100 (relógio de 133MHz). Estão presentes na placa de desenvolvimento, entre outras, as interfaces de conexão: 10/100 Ethernet; USB2 para interface JTAG; vídeo XSGA com conversor D/A triplo de 8 bits por canal e capacidade para 180MSps; codificador de áudio compatível com AC97; interface serial ATA (SATA); duas interfaces PS/2 e uma interface RS-232. Além dessas interfaces, conectores de expansão de uso geral também se encontram disponíveis.

A Figura 5.1 apresenta uma foto da placa de prototipação utilizada. A placa é a da direta na foto. A esquerda da foto estão duas placas de extensão. A placa superior é uma

placa de captura de vídeo, enquanto a inferior é uma placa de extensão de entrada e saídas. As placas de extensão não foram utilizadas na prototipação.

Figura 5.1: Foto da placa

Para os procedimentos de prototipação foram de fato utilizadas, além do dispositivo FPGA, a memória RAM externa e as interfaces de vídeo e da RS-232. Foi utilizado também um dos dois processadores PowerPC disponíveis no dispositivo FPGA.

5.2.2 Procedimentos

Para a prototipação do processador de amostras do compensador de movimento, o mesmo foi mapeado para a área reconfigurável do FPGA. Suas entradas e saídas foram conectados a um processador PowerPC, presente do dispositivo, por meio de um barramento OPB. Um programa rodando no PowerPC fornecia as entradas e lia as saídas do bloco mapeado. O programa também gerenciava a comunicação com o ambiente, lendo e escrevendo dados através da porta serial e tratando os dados processados para serem exibidos no monitor conectado a placa de desenvolvimento.

Foi criado um projeto no ambiente de desenvolvimento EDK/ISE da Xilinx. O projeto, utilizando o arquivo de descrição da placa de desenvolvimento, utilizou os controladores da porta RS-232, da porta de vídeo, da memória RAM e de um controlador do barramento OPB. Dos recursos disponíveis no dispositivo FPGA, o projeto utilizou, ainda, 128KB de memória interna para armazenamento do código do programa e 64KB para armazenamento de seus dados. Por fim, o bloco do processador de amostras foi adicionado aos recursos utilizados e conectado, como escravo, ao barramento OPB implementado.

O bloco do processador de amostras foi instanciado numa descrição padrão VHDL do projeto, a qual define os componentes a serem mapeados na área reconfigurável do FPGA. Nessa descrição 9 registradores de 32 bits foram instanciados e mapeados como posição de memória de dados acessíveis pelo PowerPC. As portas do processador de amostras foram conectadas a esses registradores por meio de uma máquina de estados, que controla a escrita e leitura dos registradores através do barramento OPB.

As portas de relógio e de inicialização do sistema também foram conectadas aos registradores. Dessa forma, o relógio e a inicialização do sistema puderam ser controlados pelo programa desenvolvido, dando maior controlabilidade à plataforma de prototipação. Com essa estratégia, foi possível acompanhar ciclo a ciclo as saídas do processador de amostras, fundamental para a analise dos problemas apresentados durante o processo de prototipação. Essa abordagem, todavia, não permite uma validação da freqüência máxima de relógio efetivamente alcançada pelo bloco mapeado no dispositivo real.

Para o controle dos sinais a serem escritos no processador de amostras, bem como a leitura dos sinais de saída, foi desenvolvido um programa em C. O programa está configurado para processar um quadro QCIF a cada execução. Esse programa está dividido em três etapas. Primeiro o programa lê os dados da porta serial e os armazena, com os dados armazenados, inicia o envio dos mesmos para o bloco mapeado e verifica os sinais de saída. O resultado do processamento das amostras é formatado e exibido no monitor, podendo ser habilitado o envio dos mesmos para a RS-232. Essa estrutura foi adotada, pois o armazenamento dos dados enviados para a placa deve ser processada pelo PowerPC. De forma a não haver perda de dados, por estouro da capacidade do *buffer*, o PowerPC não pode executar mais que uma determinada quantidade de instruções entre duas leituras do *buffer*. Assim, a leitura dos dados teve que ser realizada de forma independente do resto do programa.

Os vetores de entrada usados na validação são enviados para a placa de desenvolvimento via *hyperterminal*, conectado por um cabo serial na COM1. São enviados de forma separada as amostras das componentes Y, Cb e Cr relativas a um quadro. Devido à quantidade de dados a serem tratados, os mesmos não eram passíveis de serem armazenados na memória interna do dispositivo, sendo armazenados na memória externa. O programa lê o fluxo de dados, os converte para inteiro e os armazena na estrutura de dados na memória externa.

Os dados de entrada são armazenados em seis estruturas na memória externa. Os parâmetros de cada bloco são armazenados em três matrizes de 1584 (quantidade de blocos) por 18 (quantidade de parâmetros), uma para cada componente de cor. Uma matriz tridimensional de 1584 por 18 por 9 armazena as entradas de luma. Como um bloco pode ser bi-preditivo, a matriz tem o dobro de linhas de entrada. Duas matrizes tridimensionais, de 1584 por 6 por 3, armazenam as entradas de croma. Para o armazenamento das saídas tem-se três estruturas, uma de 1584 por 4 por 4, para os blocos compensados de luma e outras 2, de 1584 por 2 por 2, para croma.

Após a leitura dos dados, escrita nas estruturas de dados, os mesmos são convertidos para valores de RGB, formatados e escritos na posição de memória relativa ao *buffer* de imagens para serem exibidas no monitor.

Para o processamento, o programa simula o comportamento de um *test bench* de hardware. Essa parte do programa foi descrita como um *loop*, onde cada iteração representava um ciclo de relógio. Em cada iteração do *loop* escrevem-se as entradas nos registradores, lêem-se os registradores para capturar as saídas daquele ciclo e se varia o valor do registrador ligado ao sinal de relógio do processador de amostras.

A primeira etapa da iteração é a verificação da *flag* de cabeçalho de bloco, que é iniciada ligada. Se ela estiver ligada então os parâmetros do bloco são escritos nos registradores. A segunda etapa consiste em enviar os dados para os *buffers* de entrada do processamento. Testa-se a *flag* de linha pronta, caso a mesma esteja ligada é escrita uma linha de luma, uma de croma Cb e outra de Cr nas entradas, caso o iterador de linhas de bloco não tenha ultrapassado a quantidade de linhas de um bloco de croma. O iterador de

linhas é testado novamente, caso ele seja igual a 3, então os sinais de inicio de processamento, tanto de luma quanto de croma, são ligados. Caso contrário, o registrador recebe o valor 0. Em seguida o iterador de linhas é incrementado. Se o bloco for bi-preditivo e o valor do iterador for 18 ou se o bloco não for bi-preditivo e o valor do iterador for igual a 9, então o sinal de linha pronta é desligado, o iterador de linha volta a zero e o iterador de bloco é incrementado, caso não tenha atingido o limite. Se o sinal de linha pronta estiver desligado, liga-se ele e o sinal de cabeçalho pronto, caso o iterador de bloco não tenha atingido o limite.

Depois dos testes de escritas, são testadas as saídas do processamento de amostras. Se os sinais de *buffer* cheio estiverem ligados, o programa gera um mensagem de erro e termina. Em seguida testem-se, individualmente, os sinais de saída válida do croma e do luma, caso estejam ligados, a saída é escrita num dos 3 vetores que armazenam cada uma das componentes. Se o sinal de válido de croma estiver ligado é então testado o valor do sinal que indica se as saídas são referentes à componente de cor Cb ou Cr. Para cada componente de cor existe um iterador de bloco, que é incrementado a cada par de saídas de linhas de Cb ou Cr, ou a cada 4 saídas de linhas de luma.

Por fim, a variação de valor no registrador que armazena o sinal de relógio é realizada, sendo escrito '1' e depois '0' bit no registrador referente ao relógio, e é verificado se o iterador de bloco de saída atingiu o limite. Caso tenha atingido o programa sai do *loop*.

A última etapa do programa é escrever os valores das amostras compensadas na saída. As amostras compensadas são, amostra a amostra, colocadas no formato de cor 4:4:4, convertidas em amostras no espaço de cor RGB e escritas no *buffer* de imagens, sendo então exibidas no monitor acoplado a placa de processamento. A Figura 5.2 apresenta a saída do protótipo. As duas primeiras colunas são as entradas, lista 0 e lista 1, respectivamente. A terceira coluna é o resultado da compensação. São apresentados as entradas de Y, Cb e Cr, sendo essas duas uma ao lado da outra, e a imagem colorida.

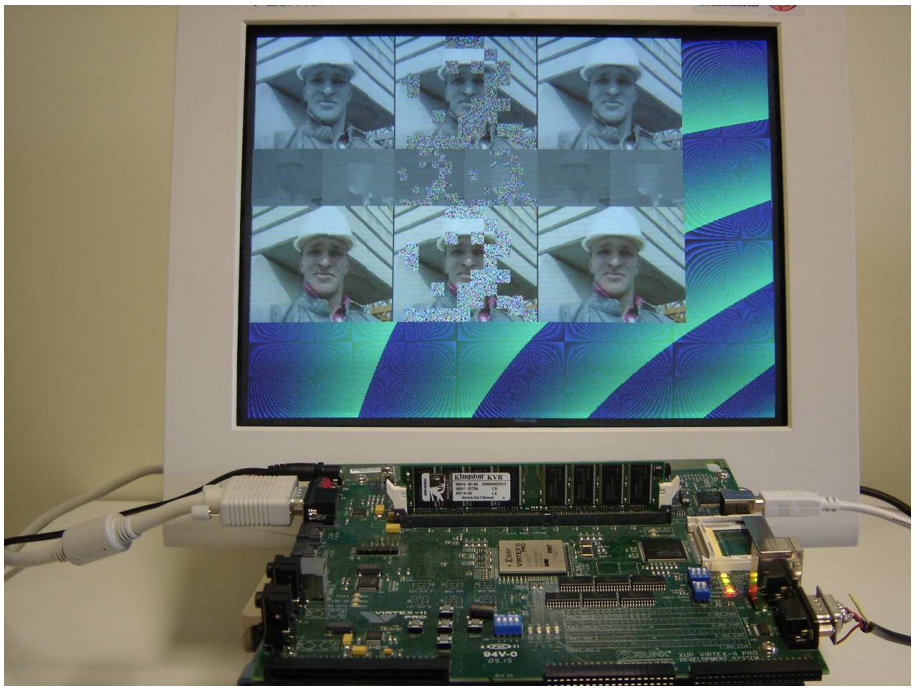

Figura 5.2: Foto do resultado do protótipo

A validação do protótipo do processamento das amostras foi realizada segundo a me-

todologia apresentada na sessão 5.1. Foram processados 25 quadros, um a um, com as amostras compensadas sendo enviadas para a porta RS-232. As amostras enviadas pela placa são salvas em arquivos no mesmo formato da saída do ModelSim e comparadas com as saídas capturadas no *software* de referência. A ausência de erros comprovou o bom funcionamento do protótipo. Foram induzidos alguns erros, como o estouro de *buffer*, para comprovar o funcionamento dos *buffers* de entrada do processador de amostras.

O presente trabalho de validação gerou um artigo, o qual foi aceito para publicação no *workshop* Iberchip de 2006, sob o título "Validação de uma arquitetura para compensação de movimento segundo o padrão H.264/AVC" (ZATT et al., 2006).

A prototipação do bloco de processamento de vetores, mais os *wrappers*, geraram os resultados presentes na Tabela 5.1.

Tabela 5.1: Resultados de síntese do protótipo

	Total	Percentual
Slices	6.530	47%
Flip Flops	6.596	24%
LUTs	7.440	27%
Blocos de RAM	65	47%
Multiplicadores	38	27%

5.2.3 Procedimentos para Prototipação com freqüência nominal de operação

O protótipo operando na freqüência nominal não foi conseguido devido às mesmas restrições temporais que impediram a prototipação da arquitetura como um todo. Todavia, essa subseção apresenta os procedimentos para a sua realização.

A prototipação visando a validação da freqüência máxima do relógio deverá ter mudanças na implementação. Para se alcançar a freqüência de 100 MHz, deve-se usar o relógio da placa, bem como o seu sinal de inicialização. Como o barramento de dados OPB não comporta a taxa de transferência de dados por ciclo de relógio necessária, *buffers* devem ser utilizados para armazenar os dados. Seis *buffers* devem ser adicionados ao bloco a ser mapeado no FPGA. Dois *buffers* são necessários para armazenar os parâmetros de entradas dos blocos, sendo um para luma e outro para croma; dois para armazenar as amostras de entrada e outros dois para armazenar as amostras compensadas, sempre um para luma e outro para croma. Tal qual na validação apresentada, o PowerPC seria o responsável pela comunicação entre o bloco e o meio externo por meio da porta RS-232 e por enviar e receber os dados dos *buffers*.

Como os sinais vindos do PowerPC são também mapeados em registrador, o valor do mesmo pode durar vários ciclos. Para que não se perca o sincronismo entre o PowerPC e o bloco, são necessários dois sinais de sincronismo. Assim, na descrição mapeada, para cada novo conjunto de dados uma porta de sinalização diferente deve ser lida de forma alternada. O PowerPC deve alocar os valores nos registradores de dados e ligar o primeiro bit de sinalização. Quando receber o sinal de lido do *buffer*, deve desligar o primeiro bit de sinalização, alocar os valores nos registradores de dados e só então ligar o segundo bit de sinalização. O mesmo deve ocorrer para os *buffers* de saída do processador de amostras, que deve alternar na leitura entre dois sinais de leitura.

Uma vez que os *buffers* de entrada estejam cheios, o sinal de inicio de processamento

deve ser ligado e assim permanecer até os *buffers* estejam novamente vazios. Uma vez que os *buffers* de saída tenham dados os mesmo podem começar a ser lidos pelo PowerPC.

A validação dos dados de saída deve seguir o mesmo procedimento apresentado para o protótipo desenvolvido.

5.3 Considerações finais sobre a validação e a prototipação

Neste capítulo foram apresentados a validação da arquitetura do compensador de movimento e os procedimentos de prototipação do bloco de processamento de amostras.

Apesar do conjunto de dados utilizado para a validação poder ser considerado insuficiente, as ferramentas para realizar novas simulações, com um conjunto de dados mais abrangente, se encontram disponíveis.

No próximo capítulo são apresentadas as conclusões dessa dissertação, bem como os trabalhos futuros decorrente dos resultados apresentados.

6 CONCLUSÕES

Esta dissertação apresentou o desenvolvimento da arquitetura do compensador de movimento para decodificadores de vídeo segundo o padrão H.264. Foram apresentados, de forma superficial, os conceitos básicos da codificação e decodificação de vídeo digital. Uma introdução ao padrão H.264 foi também desenvolvida. Essas apresentações foram relatadas para referenciar os conceitos básicos e ambientar o trabalho realizado.

A estrutura da compensação de movimento do padrão H.264, alvo deste trabalho, foi detalhada. Foram discutidas as ferramentas utilizadas pela estimação e compensação de movimento. A utilização de tamanho de bloco variável, a predição com precisão de quarto de pixel, a utilização de múltiplos quadros de referência, a predição ponderada e a bi-predição foram detalhadas. Os mecanismos da predição dos vetores de movimento e índice de referência foram apresentados em suas três formas: a predição padrão, predição direta espacial e a predição direta temporal. O cálculo dos parâmetros da predição ponderada foram também detalhados. O tratamento de vídeo entrelaçado, utilizado no perfil Main, foi apresentado de forma breve.

O estado da arte das implementações em hardware da compensação de movimento, bem como para o de decodificadores completos, foi apresentado.

Uma arquitetura para a compensação de movimento para decodificadores de vídeo foi desenvolvida. A arquitetura, denominada MoCHA (*Motion Compensator Hardware Architecture*), suporta a compensação de movimento para o perfil Main do padrão. A arquitetura está particionada em três blocos principais: predição, acesso à memória e processamento das amostras. Esses blocos funcionam na forma de um *pipeline*, existindo buffers entre os mesmos para armazenar os resultados intermediários.

No caminho de dados do bloco de acesso à memória foi inserida uma cache para reduzir a quantidade de acessos a memória externa e garantir a taxa de transmissão necessária a aplicação. A cache tridimensional implementada, baseada em experimentos que definiram sua configuração, atinge taxas de economia de acessos à memória externa em torno de 60%, segundo os experimentos realizados.

A arquitetura desenvolvida tem algumas limitações na implementação de todas as funcionalidades descritas no padrão H.264 para o perfil Main. A arquitetura do compensador considera que um quadro não contém mais que um *slice*. A arquitetura do preditor de vetores de movimento e índices de referência não implementa a predição direta temporal nem as ferramentas especificas para a decodificação de vídeo entrelaçado. A predição ponderada implícita, apesar de desenvolvida, não foi integrada à arquitetura do compensador de movimento.

A arquitetura MoCHA foi validada através de simulações. O processador de amostras e a predição dos vetores de movimento e índices de referência foram validados individualmente. Uma vez que os principais blocos foram validados, a validação da arquitetura

completa foi realizada. Os procedimentos de validação foram descritos, tal qual as ferramentas de suporte desenvolvidas.

O processador de amostras foi prototipado no dispositivo FPGA VP30 da família Virtex-II PRO da Xilinx. O processador PowerPC 405, presente no dispositivo, foi usado para implementar um *test bench* para validar a operação do processador de amostras mapeado para o FPGA.

O compensador de movimento para o decodificador de vídeo H.264 foi descrito em VHDL, num total de 30 arquivos VHDL e cerca de 13.500 linhas de código. A descrição foi sintetizada pelo sintetizador Syplify Pro da Symplicity para o dispositivo XC2VP30-7 da Xilinx, consumindo 8.465 *slices*, 5.671 registradores, 10.835 LUTs, 21 blocos de memória interna e 12 multiplicadores. A freqüência máxima de operação foi de 100,5 MHz. A arquitetura projetada é capaz de processar, no pior caso, 36,7 quadros HDTV de 1080 por 1920, inteiramente bi-preditivos por segundo, trabalhando a freqüência de relógio de 100 MHz. Para quadros do tipo P, que não utilizam a bi-predição, a capacidade de processamento sobe para 64,3 quadros por segundo.

A arquitetura apresentada para o processamento de quadros bi-preditivos é, até o momento, inédita na literatura. Os trabalhos relativos a decodificadores completos não apresentam a solução para esse processamento.

Foram aceitos dois artigos referentes ao tema da dissertação. O primeiro, sobre o processador de amostras, foi aceito na conferência Norchip de 2005, sob o título "*Motion Compensation Sample Processing for HDTV H.264/AVC Decoder*" (AZEVEDO et al., 2005). O segundo artigo foi aceito para publicação no *workshop* Iberchip de 2006, sob o título "Validação de uma arquitetura para compensação de movimento segundo o padrão H.264/AVC" (ZATT et al., 2006).

Os resultados apresentados tornam a MoCHA uma solução arquitetural capaz de fazer parte de um decodificador para vídeos de alta definição. Dessa forma considera-se que o trabalho atingiu os objetivos esperados para o mesmo.

Como trabalhos futuro, pretende-se adequar a arquitetura do compensador de movimento a todos os requisitos do padrão H.264 para o perfil Main. Essas atividades incluem: integrar, ao compensador de movimento, a arquitetura da predição ponderada implícita; adicionar o suporte a vários *slices* por quadro; desenvolver a predição direta temporal e incluir as ferramentas para a decodificação de vídeos entrelaçados a predição dos vetores de movimento e índice de referência. A prototipação da arquitetura completa também se encontra nos planos das próximas atividades a serem desenvolvidas.

Dada a complexidade da arquitetura e das operações realizadas pelo compensador de movimento, uma gama relativamente grande de atividades podem ser realizadas para explorar o seu espaço de projeto. Também a título de trabalhos futuros, são citadas algumas das possíveis explorações.

Uma versão do compensador de movimento com um comportamento mais síncrono de comunicação entre os blocos poderia diminuir os buffers existentes na arquitetura. O processamento de macroblocos com partições bi-preditivas poderia ser realizado para todos os blocos de cada lista por vez. Essa estratégia demandaria um aumento no buffer de bi-predição e tornaria o controle mais complexo, uma vez que cada bloco 4x4 de um macrobloco pode bi-preditivo ou não, a despeito dos demais blocos do macrobloco. Essa estratégia permitiria a filtragem de blocos adjacentes pertencentes a mesma partição, permitindo a diminuição do volume de dados a serem processados. Além disso, essa estratégia permitiria também a implementação da filtragem numa ordem *raster*, o que poderia diminuir a troca de contexto entre blocos.

REFERÊNCIAS

AZEVEDO, A.; ZATT, B.; AGOSTINI, L.; BAMPI, S. Motion Compensation Sample Processing for HDTV H.264/AVC Decoder. In: IEEE NORCHIP, 2005, Oulu, Finlândia. **Proceedings...** [S.l.: s.n.], 2005. p.110–113.

BHASKARAN, V.; KOSTANTINIDES, K. **Image and Video Compression Standards**: algorithms and architectures. 2nd.ed. Boston: Kluwer Academic, 1997.

DESIGN Resources for Digital Engineers. Disponível em: <http://www.digilentinc.com/>. Acesso em: fev. 2005.

GHANBARI, M. **Standard Codecs**: image compression to advanced video coding. United Kingdom: The Institution of Electrical Engineers, 2003.

GONZALEZ, R.; WOODS, R. **Processamento de Imagens Digitais**. São Paulo: E. Blücher, 2003.

H.264/AVC Software Coordination. Disponível em: <http://iphome.hhi.de/suehring/tml/>. Acesso em: fev. 2005.

HU, Y.; SIMPSON, A.; MCADOO, K.; CUSH, J. A High Definition H.264/AVC Hardware Video Decoder Core for Multimedia SoC's. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS CONSUMER ELECTRONICS, 2004. **Proceedings...** [S.l.: s.n.], 2004. p.385–389.

ISO - International Organization for Standardization. Disponível em: <http://www.iso.org>. Acesso em: fev. 2005.

ITU-T. **Recommendation H.262 (11/94)**: generic coding of moving pictures and associated audio information - part 2: video. [S.l.: s.n.], 1994.

ITU-T. **Recommendation H.264 (05/03)**: advanced video coding for generic audiovisual services. [S.l.: s.n.], 2003.

ITU-T. **Recommendation H.264 (03/05)**: advanced video coding for generic audiovisual services. [S.l.: s.n.], 2005.

KANNANGARA, C.; RICHARDSON, I. Computational Control of an H.264 Encoder through Lagrangian Cost Function Estimation. In: INTERNATIONAL WORKSHOP ON VERY LOW BITRATE VIDEO, 2005, Sardinia, Italy. **Proceedings...** [S.l.: s.n.], 2005.

KARCZEWICZ, M.; KURCEREN, R. The SP- and SI-frames design for H.264-AVC. IEEE Transactions on Circuits and Systems for Video Technology. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v.13, n.7, p.637–644, July 2003.

KUHN, P. **Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation**. Boston: Kluwer Academic, 1999.

KWON, S.-K.; TAMHANKAR, A.; RAO, K. R. Overview of H.264 / MPEG-4 Part 10. In: WSEAS INTERNATIONAL CONFERENCE ON MULTIMEDIA, INTERNET AND VIDEO TECHNOLOGIES, 5., 2003, Corfu Island, Grecia. **Proceedings...** [S.l.: s.n.], 2003.

KWON, S.-K.; TAMHANKAR, A.; RAO, K. R. Overview of H.264 / MPEG-4 Part 10. In: WSEAS INTERNATIONAL CONFERENCE ON MULTIMEDIA, INTERNET AND VIDEO TECHNOLOGIES, 5., 2005, Corfu Island, Grecia. **Proceedings...** [S.l.: s.n.], 2005.

LIE, W.-N.; YEH, H.-C.; LIN, T. C.-I.; CHEN, C.-F. Hardware-Efficient Computing Architecture for Motion Compensation Interpolation in H.264 Video Coding. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2005. **Proceedings...** [S.l.: s.n.], 2005. p.2136–2139.

LIN, T.-A.; WANG, S.-Z.; LIU, T.-M.; LEE, C.-Y. An H.264/AVC decoder with 4x4-block level pipeline. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2005. **Proceedings...** [S.l.: s.n.], 2005. v.2, p.1810–1813.

MALVAR, H.; HALLAPURO, A.; KARCZEWICZ, M.; KEROFSKY, L. Low-Complexity Transform and Quantization in H.264/AVC. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v.13, n.7, p.598–603, July 2003.

MIANO, J. **Compressed Image File Formats**: jpeg, png, gif, xbm, bpm. [S.l.]: Addison Wesley, 1999.

MODELSIM. Disponível em: <http://www.model.com/>. Acesso em: fev. 2005.

PLATFORM Studio and the EDK. Disponível em: <http://www.xilinx.com/ise/embedded_design_prod/platform_studio.htm>. Acesso em: fev. 2005.

PURI, A.; CHEN, X.; LUTHRA, A. Video Coding Using the H.264/MPEG-4 AVC Compression Standard. **Elsevier Signal Processing: Image Communication**, [S.l.], v.19, n.9, p.793–849, Oct. 2004.

RICHARDSON, I. **Video Codec Design - Developing Image and Video Compression Systems**. Chichester: John Wiley and Sons, 2002.

RICHARDSON, I. **H.264 and MPEG-4 Video Compression Video Coding for Next-Generation Multimedia**. Chichester: John Wiley and Sons, 2003.

SAHAFI, L. **Context-Based Complexity Reduction Applied to H.264 Video Compression**. 2005. 70p. Master in Applied Science — School of Engineering Science, Simon Frase University, Canada.

SHI, Y.; SUN, H. **Image and Video Compression for Multimedia Engineering**: fundamentals, algorithms and standards. Boca Raton: CRC Press, 1999.

STMICROELECTRONICS. **STMicroelectronics Launches the World's First Single-Chip Solution for Multi-Standard High Definition TV, DVD, Set-Top Boxes and Car Multimedia Center**. Disponível em: <http://www.st.com/stonline/press/news/year2005/p1567h.htm>. Acesso em: fev. 2005.

STMICROELECTRONICS Home. Disponível em: <http://www.st.com>. Acesso em: fev. 2005.

SULLIVAN, G.; TOPIWALA, P.; LUTHRA, A. The H.264/AVC Advanced Video Coding Standard: overview and introduction to the fidelity range extensions. In: CONFERENCE ON APPLICATIONS OF DIGITAL IMAGE PROCESSING, SPIE, 27., 2004, Denver. **Proceedings...** [S.l.: s.n.], 2004.

SYNPLICITY: home. Disponível em: <http://www.synplicity.com/>. Acesso em: fev. 2005.

SYNPLICITY: products: synplify pro. Disponível em: <http://www.synplicity.com/products/synplifypro/index.html>. Acesso em: fev. 2005.

THE ITU Telecommunication Standardization Sector (ITU-T). Disponível em: <www.itu.int/ITU-T/>. Acesso em: fev. 2005.

VIRTEX Series. Disponível em: <http://www.xilinx.com/products/silicon_solutions/fpgas/virtex/index.htm>. Acesso em: fev. 2005.

WANG, R.; LI, J.; HUANG, C. Motion Compensation Memory Access Optimization Strategies For H.264/AVC Decoder. In: IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, ICASSP, 2005. **Proceedings...** [S.l.: s.n.], 2005. v.5, p.97–100.

WANG, R.; LI, M.; LI, J.; ZHANG, Y. High Throughput and Low Memory Access Subpixel Interpolation Architecture for H.264/AVC HDTV Decoder. **IEEE Transactions on Consumer Electronics**, [S.l.], v.51, n.3, p.1006–1013, Aug. 2005.

WANG, S.-Z.; LIN, T.-A.; LIU, T.-M.; LEE, C.-Y. A New Motion Compensation Design for H.264/AVC Decoder. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2005. **Proceedings...** [S.l.: s.n.], 2005. p.4558–4561.

WIEGAND, T.; SCHWARZ, H.; JOCH, A.; KOSSENTINI, F.; ; SULLIVAN, G. J. Rate-Constrained Coder Control and Comparison of Video Coding Standards. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v.13, n.7, p.688–703, July 2003.

WIEGAND, T.; SULLIVAN, G. J.; BJONTEGAARD, G.; LUTHRA, A. Overview of the H.264/AVC Video Coding Standard. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v.13, n.7, p.560–576, July 2003.

XILINX: the programmable logic company. Disponível em: <http://www.xilinx.com/>. Acesso em: fev. 2005.

XUP V2P - Virtex-II Pro Development System. Disponível em: <http://www.digilentinc.com/Products/Detail.cfm?Nav1=Products&Nav2= Programmable&Prod=XUPV2P>. Acesso em: fev. 2005.

ZATT, B.; AZEVEDO, A.; AGOSTINI, L.; BAMPI, S. Validação de uma arquitetura para compensação de movimento segundo o padrão H.264/AVC. In: WORKSHOP IBERCHIP, 2006, San Jose, Costa Rica. **Proceedings...** [S.l.: s.n.], 2006.