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

BRUNO ZATT

Modelagem de Hardware para Codificação de Vídeo e Arquitetura de Compensação de Movimento Segundo o Padrão H.264/AVC

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

Prof. Dr. Sergio Bampi
Orientador

Porto Alegre, outubro de 2008.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Zatt, Bruno

Modelagem de Hardware para Codificação de Vídeo e Arquitetura de Compensação de Movimento Segundo o Padrão H.264/AVC/ Bruno Zatt – Porto Alegre: Programa de Pós-Graduação em Computação, 2008.

121 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2008. Orientador: Sergio Bampi.

1.H.264/AVC. 2.Codificação de vídeo. 3.Arquiteturas VLSI. 4. Modelagem de Hardware I. Bampi, Sergio. II. Título.

AGRADECIMENTOS

Ao concluir mais uma importante etapa de meus estudos e, portanto, de minha vida é imprescindível parar, olhar para aquilo que se passou e agradecer a tudo e a todos que de alguma forma contribuíram para que esse obstáculo fosse vencido.

Gostaria de começar agradecendo à minha família, grande responsável pela pessoa que me tornei e pelos valores que comigo hei de carregar e compartilhar. Ao meu pai João Francisco Zatt e minha mãe Amantina Cardoso Ribeiro Zatt que sempre estiverem ao meu lado quando precisei de seu apoio e carinho e sempre vibraram com cada conquista ou alegria que alcancei. Meu irmão Gustavo Zatt, que com seu ânimo e bom humor me incentivou e me contagiou com sua força de vontade, além de ser um fiel companheiro para ir adentro das madrugadas estudando ou para virá-las tomando uma boa cerveja.

Aos meus avôs que sempre se mostraram exemplos como pessoas e uma base forte para minha família. Meus avôs paternos avô Rozimbo, que descansa em paz, e avó Maria. E meus avôs maternos avô Getúlio, que nos observa do céu, e avó Romilda. Aos meus tios e primos com os quais nunca perdi contato e que fazem de cada encontro familiar uma verdadeira festa. A Marli de Almeida, a Tita, que tomou conta de mim e de meu irmão desde que nascemos e que será sempre como uma segunda mãe para nós.

Agradeço, do fundo do coração, minha namorada, Arielli Sallet Nascimento, que teve importante contribuição quando da minha formação como engenheiro, mas que durante o mestrado se tornou imprescindível, sendo paciente e compreensiva quando meu humor, calma e atenção já não eram mais os mesmos, levados embora pela intensa rotina de trabalhos. Obrigado, meu amor, por ser essa pessoa tão especial e pelo amor e carinho que me dedica, sentimentos que só fazem crescer apesar da distância que insiste em nos afastar.

Aos meus mentores no meio acadêmico: M. Sc. Arnaldo Pereira de Azevedo Filho que me ensinou muito quando eu era apenas um estudante sem experiência; Dr. Luciano Volcan Agostini que sempre acompanhou e apoiou minha evolução e dedicou muito de seu tempo em meu auxílio; Dr. Altamiro Amadeu Susin, que mesmo não sendo meu orientador, esteve presente para estimular e discutir temas de interesse, apresentando sempre um ponto de vista novo com novas alternativas e questionamentos; Dr. Sergio Bampi que apostou em mim desde o início como bolsista de iniciação científica até o momento em que me incentivou a iniciar o doutorado, antes mesmo da conclusão do mestrado, confiando em minha capacidade de trabalho e de geração de resultados.

Aos meus colegas de trabalho do grupo de pesquisas Dieison Antonello Deprá, Marcelo Schiavon Porto e Roger Porto, meus antecessores no mestrado, que sempre contribuíram com sua experiência e parceria. Vagner Rosa, colega de doutorado que nunca permite que o desânimo se instale no laboratório. Thaísa Leal Silva, Cláudio

Machado Diniz e Guilherme Mauch de Freitas que foram colegas sempre presentes e dedicados durante esse ano e meio de estudos, em especial ao Cláudio que encontrou tempo, em meio a suas inúmeras tarefas, para revisar este texto. Agradeço muito a todos, pois sem esse grupo forte e dedicado, seria muito mais difícil chegar aos resultados alcançados.

Agradeço também a toda estrutura e aos funcionários do Instituto de Informática e do PPGC, que formam a base necessária para que possamos nos dedicar às pesquisas sem preocupações paralelas.

Aos amigos que fiz ao longo desse período em Porto Alegre e aos que trago de minha infância em Santo Ângelo. Enfim, agradeço a todos que de alguma forma contribuíram, apoiaram ou acompanharam minha caminhada.

A todos, meu muito obrigado!

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS 7

LISTA DE FIGURAS 11

LISTA DE TABELAS 13

RESUMO 15

Hardware Modeling for Video Coding and Motion Compensation Architecture for the H.264/AVC Standard 17

1 INTRODUÇÃO 19

2 O PADRÃO H.264/AVC 23

2.1 Introdução ao Padrão H.264/AVC 23
2.2 Perfis e Níveis 26
2.2.1 FRExt (Fidelity Range Extentions) 27
2.3 Módulos do Codec H.264/AVC 29
2.3.1 O Módulo da Estimação de Movimento (ME) 29
2.3.2 O Módulo da Compensação de Movimento (MC) 31
2.3.3 O Módulo de Predição Intra-quadro 32
2.3.4 O Módulo das Transformadas Diretas (T) 34
2.3.5 O Módulo da Quantização Direta (Q) 37
2.3.6 O Módulo das Transformadas Inversas (IT) 38
2.3.7 O Módulo da Quantização Inversa (IQ) 39
2.3.8 O Módulo do Filtro Redutor do Efeito de Bloco 40
2.3.9 O Módulo de Codificação de Entropia 42
2.3.10 Modo de Decisão 47

3 A ARQUITETURA DE COMPENSAÇÃO DE MOVIMENTO - *HP422-MOCHA* 49

3.1 Compensação de Movimento (MC) 49
3.1.1 Tamanho de blocos variável 50
3.1.2 Múltiplos quadros de referência 50
3.1.3 Vetores apontando para fora das bordas do quadro 51
3.1.4 Precisão de quarto de pixel 52
3.1.5 Bi-predição e Predição Ponderada 54
3.1.6 Predição Skip e Direta 55
3.1.7 Predição de Vetores de Movimento 56

3.2 Arquitetura HP422-MoCHA 58
3.2.1 Preditor de Vetores de Movimento (MVP) 60
3.2.2 Processador de Amostras 67
3.2.3 Acesso à Memória 72
3.3 Resultados de Síntese e Comparação 74
3.4 Verificação 75
3.5 Considerações Finais sobre o HP422-MoCHA 77

4 MODELAGEM SYSTEMC DO CODIFICADOR H.264/AVC 79

4.1 Modelagem 79
4.1.1 Níveis de Abstração TLM 80
4.1.2 Linguagem SystemC 81
4.1.3 Plataforma de Desenvolvimento 82
4.2 Arquitetura e Modelagem do Codificador H.264/AVC 83
4.2.1 Preditor Intra-Quadro 84
Figure 4. Diagrama do Codificador Intra-Quadro 86
4.2.2 T/Q 88
4.2.3 IT/IQ 90
4.2.4 ME 91
4.2.5 CAVLC 94
4.2.6 Controle 95
4.2.7 Modo de Decisão (MD) 96
4.3 Resultados 96
4.4 Considerações Finais sobre a Modelagem do Codificador H.264/AVC 102

5 CONCLUSÃO E TRABALHOS FUTUROS 103

REFERÊNCIAS 105

APÊNDICE A RELATÓRIO DE DISCIPLINA SOBRE UMA ARQUITETURA DE PREDIÇÃO INTRA-QUADRO 111

LISTA DE ABREVIATURAS E SIGLAS

AVC	*Advanced Video Coding*
CABAC	*Context-Based Adaptive Binary Arithmetic Coding*
CAVLC	*Context-Based Adaptive Variable Length Coding*
CIF	*Common Intermediate Format*
CODEC	codificador/decodificador
DC	*Direct Current*
DCT	*Discrete Cosine Transform*
DDR	*Double Data Rate*
DPCM	*Differential Pulse Code Modulation*
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*
I16MB	Macrobloco Tipo Intra 16x16
I4MB	Macrobloco Tipo Intra 4x4
IEC	*International Electrotechnical Commission*
IEEE	*Institute of Electric and Electronics Engineers*
INTRA	*Intra Prediction*
ISDTV	*International System for Digital Television*
ISO	*International Organization for Standardization*

ITU-T	*International Telecommunication Union – Telecommunication*
IT/IQ	Transformadas e Quantização Inversas
JVT	*Joint Video Team*
MB	*Mega Bytes*
MB	Macrobloco
MD	*Mode Decision*
MoCHA	*Motion Compensation Hardware Architecture*
MC	*Motion Compensation*
ME	*Motion Estimation*
MPEG	*Moving Picture Experts Group*
MVP	*Motion Vector Predictor*
OSCI	*Open SystemC Initiative*
PC	*Personal Computer*
PMV	*Predictive Motion Vector*
POC	*Picture Order Counter*
Q	*Quantization*
IQ	*Inverse Quantization*
QCIF	*Quarter Common Intermediate Format*
QP	*Quantization Parameter*
RAM	*Random Access Memory*
RDO	*Rate-Distortion Optimization*
RGB	*Red, Green, Blue*
SAD	*Sum of Absolute Distances*
SBCCI	*Symposium on Integrated Circuits and System Design*
SBTVD	Sistema Brasileiro de Televisão Digital
SDTV	*Standard Definition Television*
SP	*Switching P*
SI	*Switching I*
T	*Transform*
TLM	*Transaction Level Modelling*
T/Q	Transformadas e Quantização Diretas
IT	*Inverse Transform*
UFRGS	Universidade Federal do Rio Grande do Sul
UVLC	*Universal Variable Length Code*
VCD	*Value Change Dump*

VCEG	*Video Coding Experts Group*
VGA	*Video Graphics Array*
VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuit*
YCbCr	*Luminance, Chrominance Blue, Chrominance Red*
YCgCo	*Luminance, Chrominance Green, Chrominance Orange*
WP	*Weighted Prediction*

LISTA DE FIGURAS

Figura 2.1: Diagrama em blocos de um codificador H.264/AVC 24
Figura 2.2: Diagrama em blocos de um decodificador H.264/AVC 24
Figura 2.3: Perfis *Baseline*, *Main*, *Extended* e *High* do padrão H.264/AVC 27
Figura 2.4: Perfis High 28
Figura 2.5: Identificação das amostras para a predição intra-quadro 32
Figura 2.6: Nove modos da predição intra-quadro para blocos de luminância 4x4 33
Figura 2.7: Quatro modos da predição intra-quadro para blocos de luminância 16x16. 33
Figura 2.8: Ordem de processamento de amostras pelo módulo T 36
Figura 2.9: Ordem de filtragem de bordas em um macrobloco 41
Figura 2.10: Amostras adjacentes para bordas verticais e horizontais 41
Figura 2.11: Ordem ziguezague de leitura dos blocos 4x4 para a codificação de entropia 43
Figura 2.12: Exemplo de codificação aritmética 46
Figura 3.1: Divisão do macrobloco em partições de macroblocos 50
Figura 3.2: Divisão de uma partição de macrobloco em partições de sub-macroblocos 50
Figura 3.3: Uso de múltiplos quadros de referência. 51
Figura 3.4: Extrapolação da borda superior esquerda. 52
Figura 3.5: (a) quadro atual a ser predito; (b) predição com vetor de movimento inteiro com valores inteiros (1, -1); (c) vetor de movimento com valores fracionários (0,5 ; 0,75) 52
Figura 3.6: Interpolação para posições de ½ pixel para o componente de luminância. . 53
Figura 3.7: Interpolação para posições de ¼ de pixel 54
Figura 3.8: Interpolação para componentes de crominância em (a) sub-amostragem 4:2:0 e (b) sub-amostragem 4:2:2 54
Figura 3.9: Ordem de predição de vetores no MB 56
Figura 3.10: Relação entre blocos vizinho (a) em formatos idênticos e (b) em formatos diferentes 57
Figura 3.11: MoCHA Architecture 58
Figura 3.12: Blocos necessários ao cálculo do MVP 61
Figura 3.13: Estrutura de registradores para armazenamento de vetores internos ao macrobloco (0 a 15) e vetores de macroblocos vizinhos (A, B, C e D) 62
Figura 3.14: Estrutura de registradores para armazenamento de índices de referência internos ao macrobloco (0 a 3) e índices de referência de macroblocos vizinhos (A, B, C e D) 62
Figura 3.15: Diagrama de estados da FSM do MVP 63
Figura 3.16: Diagrama de estados do macroestado TEMPORAL 64
Figura 3.17: Caminho de dados para o cálculo do fator de escala 65
Figura 3.18: Interpolador de Luminância 68

Figura 3.19: Organização do Filtro FIR 6-taps 69
Figura 3.20: Unidade de Clipping 70
Figura 3.21: Filtro de Crominância 70
Figura 3.22: Esquemático da Memória Cache 72
Figura 4.1: Níveis de Modelagem (CAI, 2003) 81
Figura 4.2: Arquitetura do SystemC 81
Figura 4.3: Arquitetura e Estrutura do Modelo do Codificador 84
Figura 4.4: Diagrama do Codificador Intra-Quadro 85
Figura 4.5: Memória de Vizinhos 86
Figura 4.6: Preditor de Amostras 87
Figura 4.7: Módulo de Cálculo do SAD e Decisão I4MB 87
Figura 4.8: Estrutura de Modelagem do Bloco T/Q 89
Figura 4.9: Arquitetura das Transformadas e Quantização 90
Figura 4.10: Arquitetura das Transformadas e Quantização Inversas 91
Figura 4.11: Estrutura de Modelagem da ME 92
Figura 4.12: Área de Busca Lida da Memória Externa 93
Figura 4.13: Organização da Memória Externa 93
Figura 4.14: Estrutura de Modelagem do CAVLC 95
Figura 4.15: Diagrama de Tempo do Primeiro Macro-Estágio de *Pipeline* do Codificador quando Escolhido (a) Modo I16MB, (b) Modo I4MB, (c) Inter 99
Figura 4.16: Diagrama de Tempo do Primeiro Macro-Estágio de *Pipeline* do Codificador Explorando maior Área de Busca da ME 100
Figura 4.17: Diagrama de Tempo do Segundo Macro-Estágio de *Pipeline* do Codificador 100
Figura 4.18: Arquitetura do Codificador com as Conexões Analisadas 101

LISTA DE TABELAS

Tabela 2.1: Relação entre QP e Qstep 37
Tabela 2.2: Seis primeiros códigos de Exp-Golomb 44
Tabela 2.3: Tabela de probabilidades e sub-faixas para os símbolos do exemplo 46
Tabela 3.1: Síntese do MVP com Suporte a *Slices* B e Predição Direta Espacial e Temporal 65
Tabela 3.2: Taxa de processamento do MVP para HDTV 66
Tabela 3.3: Resultados de Síntese para Standard Cells TSMC 0.18 µm 71
Tabela 3.4: Resultados Comparativos com outras Arquiteturas de Interpolação 71
Tabela 3.5: Resultados de Síntese 75
Tabela 4.1: Resultados de Tráfego de Dados para Vídeos 1920x1080 @ 30fps 101

RESUMO

Esta dissertação é composta de duas partes principais em que apresenta, em sua primeira parte, o desenvolvimento de uma arquitetura de *hardware* para compensação de movimento para decodificadores de vídeo segundo o padrão H.264/AVC. A segunda parte apresenta a modelagem de uma arquitetura de *hardware* para codificação de vídeo segundo o mesmo padrão. Também são apresentados os conceitos básicos da codificação e decodificação de vídeo digital segundo o padrão H.264/AVC.

A arquitetura desenvolvida para compensação de movimento, denominada HP422-MoCHA (*High Profile 4:2:2 Motion Compensation Hardware Architecture*) (ZATT, 2008), baseada na arquitetura MoCHA (*Motion Compensator Hardware Architecture*) (AZEVEDO, 2007), suporta o conjunto de ferramentas da compensação de movimento para o perfil *High 4:2:2* do H.264/AVC. Esta arquitetura está particionada em três blocos principais: Preditor de Vetores de Movimento, Acesso à Memória e Processador de Amostras. Esses blocos funcionam na forma de um *pipeline*, existindo buffers entre os mesmos para armazenar os resultados intermediários. A descrição foi desenvolvida com a linguagem VHDL e alcança desempenho para decodificar, em tempo real, vídeos HDTV 1920x1080 a 30 quadros por segundo.

Na literatura atual não foi encontrada nenhuma solução detalhada para a compensação de movimento no perfil *High 4:2:2* do padrão H.264/AVC. Uma nova estrutura para interpolação de amostra na compensação de movimento foi proposta, sendo que sua versão para o Perfil *Main* se mostra 17% mais compacta, em termos de *gates*, que a solução mais compacta encontrada na literatura, sem degradação de performance.

A segunda parte do texto detalha a modelagem de uma arquitetura de codificação de vídeo segundo o H.264/AVC. A descrição utiliza a linguagem SystemC e consumiu aproximadamente 15.000 linhas de código. Seu projeto foi desenvolvido com o objetivo de codificar vídeo H.264/AVC segundo o perfil *Main* do padrão com desempenho para codificar vídeos 1920x1080 em tempo real, a 30 quadros por segundo.

A modelagem alcançou o objetivo principal de chegar a uma implementação funcional de um codificador, embora assumindo diversas restrições de codificação, permitindo a caracterização temporal e de comunicação do codificador. Dessa forma, o modelo se mostra uma poderosa ferramenta para o desenvolvimento do sistema de codificação em HW, desde a etapa de projeto até a verificação final.

Não foi encontrado na literatura, até o presente momento, nenhum trabalho que descreva uma modelagem em alto nível de um *hardware* para o codificador, ou mesmo para o decodificador, de vídeo H.264/AVC.

Palavras-Chave: H.264/AVC, Codificação de vídeo, Arquiteturas VLSI, Modelagem SystemC.

Hardware Modeling for Video Coding and Motion Compensation Architecture for the H.264/AVC Standard

This thesis is comprised by two main parts that present, in the first part, the development of a motion compensation hardware architecture for video decoders in compliance with the H.264/AVC standard. The second part presents a hardware architecture modeling for a video encoder compliant to the same video standard. The digital video coding basics in the H.264/AVC standard are also reviewed.

The developed motion compensation hardware architecture, named HP422-MoCHA (*High Profile 4:2:2 Motion Compensation Hardware Architecture*) (ZATT, 2008), is based on the MoCHA (*Motion Compensator Hardware Architecture*) (AZEVEDO, 2007) architecture. It supports the motion compensation toolset for the H.264/AVC High 4:2:2 profile. This architecture is divided in three main modules: Motion Vector Predictor, Memory Access and Sample Processor. These modules work in a pipeline and are interfaced by buffers to store the intermediate data. The architecture was described in the VHDL language and reaches the required throughput for real time decoding of HDTV 1920x1080 video sequences at 30 frames per second.

In the current literature another detailed motion compensation solution for the H.264/AVC High 4:2:2 could not be found. A new filtering organization for the motion compensation sample interpolator was proposed and its Main profile version reduces 17% the gate count in comparison to the smallest solution found in the literature, without any performance degradation.

The second part of the thesis details the modeling of a hardware architecture for a video encoder for the H.264/AVC standard. The model was described in SystemC language and used 15,000 source code lines. The project was designed for real time encoding of Main profile H.264/AVC for 1920x1080 video sequences at 30 frames per second.

The model supported the main objective which was to obtain a functional encoder implementation, despite of the several encoding restrictions, permitting the temporal and communications characterization of the encoder. The model is presented as a powerful tool for the hardware video encoder development, as it is useful from the initial design to the final verification.

No other hardware encoder or decoder modeling description was found in the current literature for the H.264/AVC video coding standard.

Keywords: H.264/AVC, Video Coding, VLSI Architectures, Modeling in SystemC.

1 INTRODUÇÃO

A compressão de vídeos digitais tem sido um assunto amplamente pesquisado e estudado, devido às suas diversas possibilidades de aplicação e utilização. O armazenamento e a transmissão de vídeos, principalmente em alta definição, exigem alta capacidade de compressão para se tornarem viáveis.

Durante aproximadamente 10 anos o padrão MPEG-2 foi o padrão de codificação de vídeo predominante no mercado, sendo utilizado pela grande maioria dos sistemas de televisão digital, bem como por mídias populares como o DVD e por uma vasta gama de dispositivos portáteis, como câmeras digitais. No entanto, a busca por maiores resoluções de imagem, trouxe a necessidade de armazenamento e transmissão de vídeos em alta definição (HDTV) e, conseqüentemente, a necessidade de um maior poder de compressão de vídeos.

Com o objetivo de aumentar a capacidade de compressão de vídeo em relação aos padrões já existentes, foi desenvolvido o padrão H.264/AVC. Esse padrão foi definido em 2003 pelo JVT (ITU-T, 2003), formado a partir da união de especialistas do VCEG da ITU-T (ITU-T, 2008) e especialistas do MPEG da ISO/IEC (ISO/IEC, 2008).

Após completar essa primeira versão do padrão, o JVT abriu uma nova chamada de propostas voltadas para o suporte de maior número de bits por amostra e diferentes formatos de subamostragens para crominância. Essa chamada, denominada como *Fidelity Range Extensions* (FREXt), foi feita devido à alta demanda de vídeos de alta fidelidade, especialmente para aplicações profissionais como produção de filmes. Os trabalhos foram encerrados em 2004 e a padronização feita em setembro de 2005 (ITU-T, 2005).

O padrão H.264/AVC atingiu seu objetivo de alcançar as mais elevadas taxas de compressão dentre todos os padrões existentes, mas para isso foi necessário um grande aumento na complexidade computacional das operações dos *codecs* que seguem este padrão, em relação aos demais padrões disponíveis na atualidade. Este aumento de complexidade dificulta, pelo menos na tecnologia atual, a utilização de codecs H.264/AVC completos implementados em software, quando as resoluções são elevadas e quando se deseja tempo real, com 30 quadros por segundo, por exemplo. A dificuldade de tratar o problema via software, somada ao enorme interesse comercial que reside neste padrão, têm impulsionado equipes de pesquisa e desenvolvimento ao redor do mundo a tratarem deste tema visando otimizações algorítmicas e/ou implementações em hardware para que os requisitos das aplicações sejam atendidos. Além disso, outro forte motivador para o desenvolvimento de codecs H.264/AVC em hardware está na crescente necessidade de sistemas embarcados capazes de manipular vídeos. São exemplos as câmaras e filmadoras digitais, os PDAs, os MP4 *players*, os Blu-ray *players,* os telefones celulares, os *set top boxes* de TV digital, etc. Neste caso,

questões como o consumo de energia e quantidade de recursos de hardware utilizados são críticas. Para estas aplicações, a única alternativa é o desenvolvimento destes sistemas em hardware dedicado, pois um software rodando em um processador de propósito geral não atende aos requisitos típicos de sistemas embarcados para aplicações computacionalmente muito complexas, como é o caso dos codecs de vídeo.

Esse texto é dividido em duas partes principais, cada uma responsável por apresentar uma arquitetura de hardware distinta para tratamento da codificação e decodificação de vídeo. Na primeira parte do texto é apresentado o HP422-MoCHA, uma arquitetura para compensação de movimento no H.264/AVC desenvolvida para atingir desempenho capaz de decodificar, em tempo real, vídeos de 1920x1080 pixels à 30 quadros por segundo. Essa arquitetura é uma extensão feita sobre a arquitetura MoCHA (AZEVEDO, 2007) original tornando-a capaz de suportar as ferramentas do perfil *High 4:2:2* do padrão. O suporte ao perfil *High 4:2:2* visa atender ao padrão ISDTV, adotado pelo Brasil como padrão de televisão digital.

Este trabalho está inserido no projeto de um codec H.264/AVC em hardware para TV de alta definição (HDTV – 1920x1080). O codec estudado foi utilizado nas investigações acadêmicas que serviram de subsídio para a construção do Sistema Brasileiro de Televisão Digital (SBTVD).

No momento do desenvolvimento do decodificador H.264/AVC, a urgência por parte do governo brasileiro em definir o SBTVD, impunha aos grupos de pesquisa um prazo extremamente curto para conclusão de suas tarefas. Tendo isso em vista, o grupo de desenvolvimento de hardware para o H.264/AVC coordenado pela UFRGS, viu-se forçado a pular etapas do fluxo de projeto, indo direto à implementação dos módulos de hardware em uma abordagem *bottom-up.*

Com a abordagem adotada, encontrou-se grande dificuldade para integrar os diferentes módulos de hardware desenvolvidos pelo grupo. O principal motivo é a dificuldade de sincronização e controle de cada um desses módulos em um sistema de decodificação completo.

Com base na experiência adquirida durante o desenvolvimento do decodificador e com mais tempo disponível, decidiu-se atacar o desenvolvimento do codificador com uma abordagem *meet-in-the-middle*. A estratégia é desenvolver um modelo, em alto nível de abstração, que descreva o comportamento do codificador como um todo, caracterizando cada um dos seus módulos componentes em termos de temporização, sincronização e comunicação com os demais módulos. Em paralelo são desenvolvidos os módulos de hardware propriamente ditos, em baixo nível de abstração, que irão compor o sistema modelado. É nessa abordagem que se encaixa a segunda parte desse texto.

A segunda parte do texto propõe uma arquitetura completa para codificação de vídeo segundo o padrão H.264/AVC. Essa arquitetura foi projetada e modelada para codificar vídeos de alta resolução, 1920x1080 pixels, em uma taxa de 30 quadros por segundo. Nessa arquitetura são implementadas ferramentas definidas pelo perfil *Main* do padrão, embora uma série de limitações tenha sido imposta para simplificar o codificador. Essas simplificações não fazem com que o codificador deixe de estar de acordo com o padrão H.264/AVC, pois o único requisito para o codificador é gerar um *bitstream* válido, capaz de ser decodificado por um decodificador conforme o padrão. A arquitetura foi modelada utilizando a linguagem SystemC e a plataforma de desenvolvimento foi o simulador ModelSim.

Este texto está organizado da seguinte forma. No capítulo 2 é apresentado o padrão de compressão de vídeo H.264/AVC, com suas principais características, perfis, níveis e definições úteis para a compreensão do restante do texto. Ainda no capítulo 2, são detalhados o codificador e o decodificador H.264/AVC, bem como seus principais módulos componentes. O capítulo 3 apresenta as extensões feitas ao padrão em 2006, o algoritmo da compensação de movimento e a arquitetura HP422-MoCHA. A arquitetura do codificador e sua modelagem SystemC são descritas no capítulo 4. Nesse capítulo também são abordados alguns tópicos sobre a linguagem SystemC, a plataforma de desenvolvimento e os níveis de abstração utilizados. O capítulo 5 apresenta as conclusões deste trabalho.

2 O PADRÃO H.264/AVC

Este capítulo apresentará, de forma breve, as principais características do padrão de compressão de vídeo H.264/AVC, seus perfis e níveis, bem como seus principais módulos componentes.

2.1 Introdução ao Padrão H.264/AVC

O padrão de compressão de vídeo H.264/AVC (*Advanced Video Coding)* (ITU-T, 2003), também conhecido como MPEG-4 Parte 10, foi desenvolvido conjuntamente pela ITU-T, através de seu grupo de especialistas em codificação de vídeo (VCEG – *Video Coding Experts Group*) (ITU-T, 2008), e pela ISO/IEC, representada pelo seu grupo de especialistas em imagem em movimento (MPEG – *Motion Picture Experts Group*) (ISO/IEC, 2008). Esta união deu origem ao JVT (*Joint Video Team*) (ITU-T, 2008) que concluiu a definição do padrão em 2003.

O padrão H.264/AVC opera sobre imagens no espaço de cores YCbCr (MIANO, 1999), este espaço de cores é composto por um canal de luminância e dois de crominância, sendo que a relação entre Y, Cb e Cr depende do perfil do padrão considerado. Os perfis serão apresentados ainda nesta seção.

As imagens são formadas por macroblocos que contém 16x16 amostras de luminância e as amostras de crominância associadas. Grupos de macroblocos são associados em *slices.* Cada *slice* pode ser classificado em cinco diferentes tipos (I, P, B, SI e SP). *Slices* do tipo I contém apenas macroblocos do tipo I. *Slices* do tipo P podem conter macroblocos tipo P e tipo I. Os *slices* B podem conter macroblocos I e B. Os *slices* SI e SP são definidos apenas no perfil *Extended* e visam permitir o chaveamento entre fluxos de bits diferentes, sem comprometer a eficiência da codificação. O número de macroblocos em um *slice* pode variar de um até o número total de macroblocos do quadro (RICHARDSON,2003).

Macroblocos do tipo I são codificados através da codificação intraframe ou intra-quadro, que utiliza amostras contidas no *slice* atual, explorando a redundância espacial dentro do *slice*. Estes macroblocos também podem ser codificados através do modo I_PCM onde são transmitidos os valores das amostras diretamente.

Os macroblocos tipo P são codificados utilizando a codificação interframe ou inter-quadros, que utiliza amostras contidas em um dos quadros de referência previamente codificados. Este tipo de codificação visa explorar a redundância temporal que existe entre os quadros de uma seqüência de vídeo. Os quadros de referência são organizados em uma lista, chamada pelo padrão de **lista 0** (ITU-T, 2003).

Macroblocos B também são codificados pela codificação interframe, no entanto, cada partição pode utilizar informações de até dois quadros de referência diferentes. As referências são organizadas em duas listas, chamadas de **lista 0** e **lista 1**. A forma como os macroblocos são particionados e como as listas de referências são organizadas serão melhor descritas na próxima seção.

O diagrama de blocos do codificador é apresentado na Figura 2.1, enquanto a Figura 2.2 apresenta o diagrama de blocos do decodificador. Estas figuras exibem as principais operações e a forma com que se relacionam.

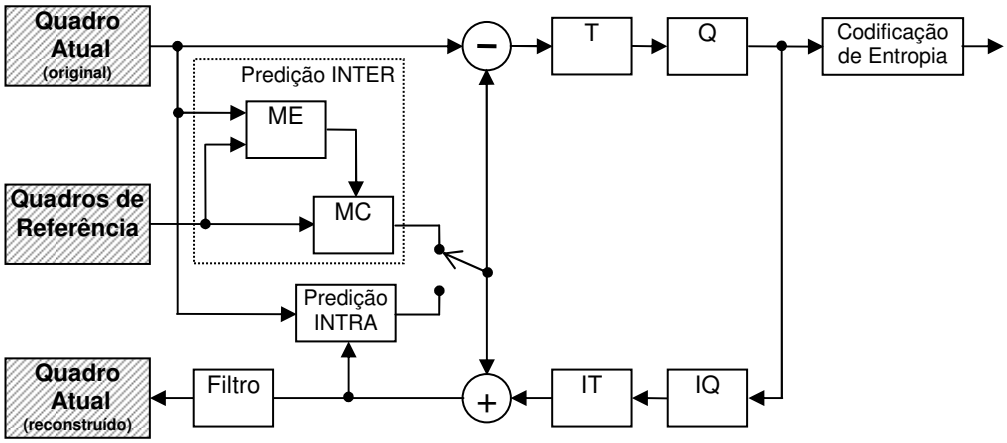


Figura 2.1: Diagrama em blocos de um codificador H.264/AVC

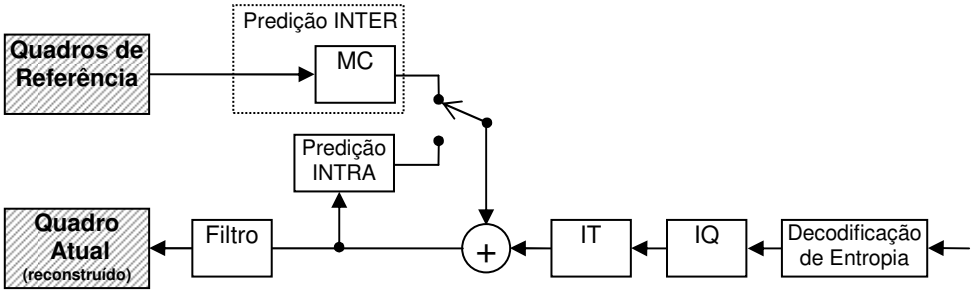


Figura 2.2: Diagrama em blocos de um decodificador H.264/AVC

As etapas de estimação e compensação de movimento (KUHN, 1999) do padrão H.264/AVC (módulo ME na Figura 2.1 e módulo MC nas Figuras 2.1 e 2.2) são onde se encontram a maior parte das inovações e dos ganhos obtidos pelo H.264/AVC sobre os demais padrões. 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 ¼ de

pixel, para buscar o melhor casamento e para realizar a reconstrução do quadro. O H.264/AVC 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 bipreditiva, 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 (ITU-T, 2003; RICHARDSON, 2003; WIEGAND, 2003a).

O compensador de movimento (MC), presente tanto no codificador quanto no decodificador, é responsável por reconstruir o quadro atual a partir dos vetores de movimento e quadros de referência. Seu funcionamento será detalhado na próxima seção.

A etapa de predição INTRA (Figuras 2.1 e 2.2) é outra inovação introduzida pelo padrão H.264/AVC, pois mesmo nos macroblocos do tipo I é realizada uma predição antes da aplicação da transformada, ainda no domínio espacial. Esta predição leva em conta as amostras já codificadas do *slice* atual.

No que diz respeito às transformadas direta e inversa (módulos T e IT nas Figuras 2.1 e 2.2), o H.264/AVC introduziu algumas inovações. A primeira delas é que as dimensões da transformada foram definidas em 4x4 ao invés do tradicional 8x8. Outra inovação é relativa ao uso de uma aproximação inteira das transformadas de modo a facilitar a sua implementação em hardware de ponto fixo e evitar erros de casamento entre o codificador e o decodificador (MALVAR, 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/AVC (módulos Q e IQ nas Figuras 2.1 e 2.2) 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/AVC normatiza a utilização de um filtro redutor do efeito de bloco (módulo Filtro nas Figuras 2.1 e 2.2). Este filtro era opcional na maioria dos demais padrões de compressão de vídeo, mas passou a ser obrigatório no H.264/AVC. Uma inovação importante do filtro definido no padrão H.264/AVC é que este filtro é adaptativo, conseguindo distinguir entre uma aresta real da imagem e um artefato gerado por um elevado passo de quantização.

Na codificação de entropia (Figuras 2.1 e 2.2) o H.264/AVC 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) e a codificação aritmética adaptativa baseada em contexto (CABAC). 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.

2.2 Perfis e Níveis

O padrão H.264/AVC 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/AVC, 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 vídeo-telefonia, videoconferência e vídeo sem fio. O perfil *Baseline* suporta codificação intra-quadro e inter-quadros (usando somente *slices* I e P) e uma codificação de entropia com códigos de comprimento de palavra variável adaptativos ao contexto (CAVLC). O perfil *Main* é focado na transmissão de televisão e no armazenamento de vídeo. O perfil *Main* inclui o suporte para vídeo entrelaçado, o suporte à codificação inter-quadros utilizando *slices* do tipo B e utilizando predição ponderada e o suporte à codificação de entropia utilizando codificação aritmética adaptativa ao contexto (CABAC). 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 subamostragem dos elementos de crominância já no vídeo original. 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: nos três perfis são usados 8 bits por amostra.

Estes três perfis inicialmente propostos pelo padrão H.264/AVC não incluíram suporte para vídeos com qualidade mais elevada, como as necessárias em ambientes profissionais. Para responder às exigências deste tipo de aplicação, uma continuação do projeto JVT foi realizada para adicionar novas extensões para as capacidades do padrão original. 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, 2004).

A Figura 3.1 apresenta, resumidamente, a relação entre os perfis *Baseline*, *Main*, *Extended* e *High* do padrão H.264/AVC.

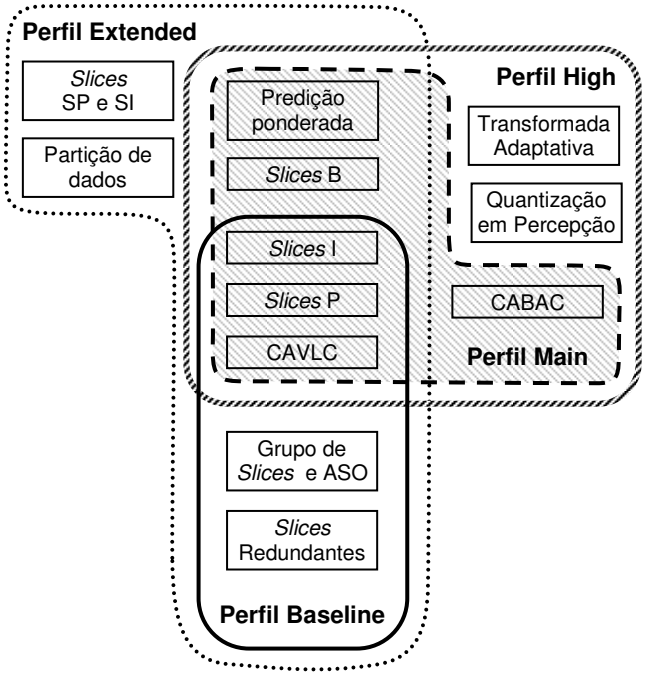


Figura 2.3: Perfis *Baseline*, *Main*, *Extended* e *High* do padrão H.264/AVC

Além da divisão em diversos perfis, o padrão H.264/AVC também define 16 diferentes níveis 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, 2004).

2.2.1 FRExt (Fidelity Range Extentions)

Conforme mencionado, quatro perfis *High* foram inseridos por meio da *Fidelity Range Extension* (FRExt). Como a arquitetura HP422-MoCHA foi desenvolvida para o perfil *High,* as diferenças dos perfis *High* com relação ao H.264/AVC original serão detalhadas nesta sub-seção. A Figura 2.4 resume as diferenças principais entre os perfis *High* e o perfil *Main.*

2.2.1.1 Perfil High

Neste perfil são definidas as principais mudanças inseridas pela FRExt, que são a predição intraframes 8x8 e as transformadas 8x8. É importante lembrar que além das inovações, os perfis *High* têm suporte a todas as ferramentas presentes no perfil *Main.*

Na primeira versão do padrão, a predição intraframes estava definida para dois diferentes tamanhos de bloco quando considerando luminância, blocos 16x16 e blocos 4x4. Para blocos 16x16 existem 4 modos de predição (vertical, horizontal, DC e planar) enquanto que para blocos 4x4 existem 9 modos (8 modos direcionais e modo DC). No perfil *High*, foi inserido o tipo de predição intraframe que trabalha sobre blocos 8x8 utilizando os mesmos 9 modos definidos previamente para blocos 4x4. A diferença da nova predição intra 8x8 é uma pré-filtragem das amostras de referência (vizinhos) antes da predição. Essa filtragem é definida como um filtro polinomial de segunda ordem.

As transformadas definidas para a primeira versão do padrão foram a aproximação inteira da DCT 4x4 e a transformada Hadamard 4x4 para os coeficientes DC da predição intraframes 16x16. Nesta extensão, foi inserida a transformada DCT inteira

8x8 no intuito de preservar texturas e detalhes finos da imagem. Essa transformada é utilizada apenas na predição intraframes quando o modo 8x8 é utilizado e na predição interframes quando não há nenhuma partição menor do que 8x8 no MB processado.

A FRExt trouxe o suporte para imagens monocromáticas, isto é, sub-amostragem de cores 4:0:0 e o suporte a matrizes de escalamento perceptuais especificados durante a codificação.

As matrizes de quantização baseadas em percepção foram usadas anteriormente no padrão MPEG-2. Com esta ferramenta, o codificador pode especificar, para cada tamanho de bloco da transformada e separadamente para codificação intra-quadro e inter-quadros, um fator de escala específico. 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.

2.2.1.2 Perfil High 10

Engloba todas as funcionalidades do perfil *High* e permite utilizar maiores larguras de amostra. Além da previamente suportada largura de 8 bits, suporta amostras de 9 e 10 bits, tanto para luminância quanto para crominância.

2.2.1.3 Perfil High 4:2:2

Suporta todas as ferramentas do perfil *High 10* e insere a capacidade de trabalhar sobre vídeos com sub-amostragem de cores 4:2:2. Nessa subamostragem cada canal de crominância apresenta metade da resolução horizontal e resolução vertical completa.

2.2.1.4 Perfil High 4:4:4

Permite codificação de vídeos com largura de amostra entre 8 e 12 bits, suporta vídeos com sub-amostragem de cores 4:0:0, 4:2:0 e 4:2:2 além de vídeos sem sub-amostragem (4:4:4). Define a transformada residual de cores que consiste em uma conversão do espaço de cores de RGB para YCgCo e insere a possibilidade de codificação sem perdas (*lossless coding*).

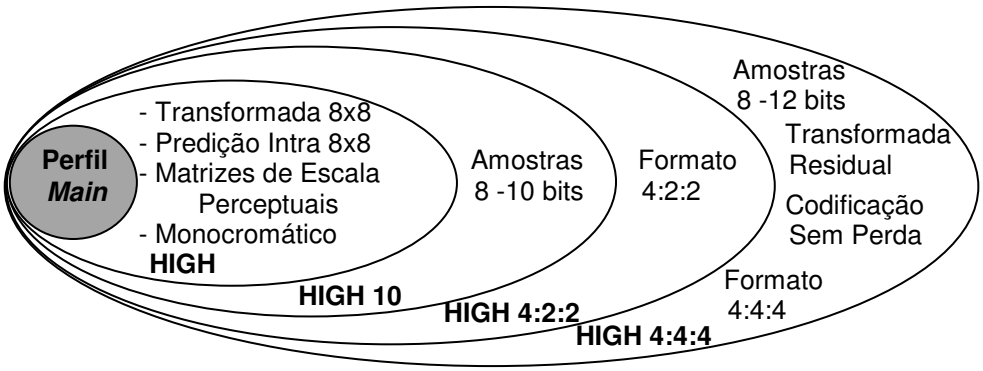


Figura 2.4: Perfis High

2.3 Módulos do Codec H.264/AVC

Esta seção apresentará os principais módulos do codec H.264/AVC. Esses módulos estão representados nas Figuras 2.1 e Figura 2.2.

2.3.1 O Módulo da Estimação de Movimento (ME)

A predição inter-quadros no codificador H.264/AVC, é formada pelos módulos da Estimação de Movimento (ME) e da Compensação de Movimento (MC). Esta seção focará o módulo da Estimação de Movimento, mas ambos os módulos estão estreitamente relacionados.

O módulo da estimação de movimento está presente apenas nos codificadores. Este módulo é o que apresenta a maior complexidade computacional dentre todos os módulos de um codificador H.264/AVC (PURI, 2004). Este grande custo computacional é função das inovações inseridas neste módulo do padrão, que tiveram o objetivo de atingir elevadas taxas de compressão. Residem nos módulos da ME e MC as principais fontes de ganhos do H.264/AVC em relação aos demais padrões de compressão de vídeo (WIEGAND, 2003, RICHARDSON, 2003).

A estimação de movimento deve prover as ferramentas de codificação capazes de localizar, nos quadros de referência, qual macrobloco mais se assemelha ao macrobloco atual. Assim que o macrobloco é encontrado, a ME deve gerar um vetor indicando a posição deste macrobloco no quadro de referência. Este vetor é chamado de vetor de movimento e deve ser inserido junto com a codificação do macrobloco.

A estimação de movimento no padrão H.264/AVC apresenta diversas inovações com relação aos padrões anteriores. Tais como o uso de tamanhos variáveis de bloco. O macrobloco pode ser particionado em blocos de 16x16, 8x16, 16x8 e 8x8 amostras. Os blocos 8x8 podem ainda ser subparticionados em blocos 8x4,4x8 e 4x4 amostras.

O tamanho de partição é escolhido no algoritmo do codificador a partir de alguma métrica que conduza a uma codificação mais eficiente, isto é, que analise o compromisso entre a geração de um resíduo mínimo e a geração de um número mínimo de vetores de movimento. A melhor escolha é realizada tomando por base os resultados da análise para todos os tamanhos de partição de cada macrobloco. Então, é escolhido aquele tamanho de partição que apresentar a maior eficiência de codificação. Pode-se concluir que a complexidade computacional desta operação é bastante grande, uma vez que os cálculos da estimação de movimento são realizados para vários tamanhos de partição diferentes e apenas uma destas partições é escolhida. Por isso, a estimação de movimento é o módulo mais crítico na implementação de codificadores H.264/AVC.

Outra característica importante da estimação de movimento do padrão H.264/AVC é que ela prevê uma precisão de ¼ 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/AVC prevê a utilização de vetores de movimento com valores fracionários de ½ pixel e de ¼ de pixel.

Outra característica importante do padrão H.264/AVC é o uso de múltiplos quadros de referência. No H.264/AVC 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 para frente ou para trás do quadro atual (RICHARDSON, 2003).

As partições de macroblocos em um *slice* tipo I não utilizam estimação de movimento, uma vez que suas predições são sempre 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-quadros ou através de macroblocos do tipo *skipped*. A predição inter-quadros 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. A predição Intra-quadro é a mesma utilizada para *slices* tipo I. Um macrobloco do tipo *skip* pode ser utilizado em *slices* tipo P ou B. Neste caso, nenhuma informação é transmitida para este macrobloco, 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, 2005). Quando o decodificador encontra um macrobloco tipo *skip* em um *slice* P, o decodificador reconstrói o macrobloco a partir do primeiro quadro armazenado na lista 0, calculando seu vetor de movimento a partir dos vetores vizinhos. Se o macrobloco *skip* estiver em um tipo *slice* B, então o macrobloco é reconstruído no decodificador usando predição direta (SAHAFI, 2005).

As partições de macroblocos em um *slice* B são codificadas através de predições inter-quadros 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 P.

A estimação bipreditiva 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.

Quando a predição direta é usada em um quadro B, nenhum vetor de movimento é transmitido. Ao invés disso, o decodificador calcula os vetores da lista 0 e da lista 1 baseado nos vetores previamente codificados.

A estimação de movimento do padrão H.264/AVC possui ainda características adicionais que contribuem para o aumento da eficiência de codificação. Uma destas características é a possibilidade de utilizar vetores de movimento que apontam para fora dos limites dos quadros. Neste caso é realizada uma extrapolação dos quadros nas bordas. Outra característica interessante é que quadros do tipo B podem ser usados como referência na predição, diferente do que acontece no padrão MPEG-2 (ITU-T, 1994). Por fim, para minimizar o custo da codificação dos vetores de movimento o padrão H.264/AVC prevê que os vetores sejam preditos a partir dos vetores calculados para as partições de macroblocos vizinhas.

Do que foi exposto até agora nesta seção é possível perceber que a estimação de movimento no padrão H.264/AVC é realmente muito complexa, por permitir o uso de técnicas variadas com o objetivo de atingir elevadas taxas de compressão. Uma questão importante e que deve ser considerada no projeto de codificadores H.264/AVC em hardware é que o padrão H.264/AVC normatiza apenas o decodificador. Por isso, a implementação do codificador pode ser realizada com vários graus de liberdade, como já foi citado. Então, várias das características da estimação de movimento previstas pelo padrão podem ser simplesmente ignoradas na construção do codificador para que a complexidade da compressão seja reduzida. Medidas de simplificação deste tipo podem gerar um impacto negativo na eficiência da codificação.

Existem diversos algoritmos utilizados para realizar a busca nos quadros de referência. Estes algoritmos variam desde a busca exaustiva (BHASKARAN, 1999) até algoritmos sub-ótimos que reduzem muito a complexidade da busca, mas que produzem impactos negativos na eficiência da codificação. Dentre estes algoritmos rápidos é possível citar a busca circular (TOURAPIS, 1999), a busca logarítmica (JAIN, 1981), a busca em três passos (LI, 1994), a busca espiral (KHUN, 1999; KROUPIS, 2005), a busca em diamante (YI, 2005) a busca hierárquica (CHU, 1997), entre outros.

Além dos algoritmos de busca é importante destacar a importância dos critérios de similaridade utilizados para decidir qual é o melhor casamento na procura sobre o quadro de referência. Existem vários critérios reportados na literatura dentre os quais cabe citar o erro quadrático médio (MSE), o erro absoluto médio (MAE), a soma de diferenças absolutas (SAD), também chamada de soma dos erros absolutos (SAE) (KUHN, 1999), entre outras.

As ferramentas apresentadas nessa subseção se aplicam também a compensação de movimento e serão detalhadas no capítulo 3 dessa dissertação.

2.3.2 O Módulo da Compensação de Movimento (MC)

A etapa de compensação de movimento (bloco MC nas Figuras 2.1 e 2.2) opera de forma complementar à 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 pela ME, copiar os blocos de melhor casamento na memória de quadros anteriormente codificados (quadros passados e/ou futuros) e montar o quadro predito. Este quadro será subtraído do quadro atual para gerar o quadro de resíduos que passará pela transformada.

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, conforme foi apresentado na seção anterior. 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. Assim, a MC no decodificador deve:

- Tratar múltiplos tamanhos de partições de macroblocos;
- Utilizar múltiplos quadros de referência anteriores e posteriores;
- Interpretar corretamente os vetores construídos com base na predição de vetores;

- Tratar vetores que apontam para fora da borda do quadro;
- Reconstruir os macroblocos considerando uma precisão de ¼ de pixel para os vetores de movimento;
- Reconstruir os macroblocos que utilizam as predições bipreditiva, ponderada e direta para *slices* do tipo B;
- Reconstruir corretamente os macroblocos do tipo *skip* para *slices* tipo P e B.

As ferramentas acima citadas serão mais bem apresentadas no capítulo 3 que trata exclusivamente da compensação de movimento.

2.3.3 O Módulo de Predição Intra-quadro

O módulo de predição intra-quadro do padrão H.264/AVC é responsável por realizar a predição nos macroblocos do tipo I. Esta predição é baseada nos valores previamente codificados do *slice* atual dos pixels acima e à esquerda do bloco a ser codificado. A predição intra-quadro para amostras de luminância pode ser utilizada sobre blocos 4x4 (macroblocos I4MB) ou 16x16 (macroblocos I16MB). Existem nove diferentes modos de predição intra-quadro I4MB e quatro modos para a predição I16MB (RICHARDSON, 2003). O módulo de predição intra-quadro está presente nos codificadores e nos decodificadores H.264/AVC (Figuras 2.1 e 2.2).

A utilização de um módulo de predição intra-quadro no domínio espacial é uma inovação no padrão H.264/AVC. Em função da predição intra-quadro e considerando também a predição inter-quadros, a transformada é sempre aplicada num sinal erro de predição. Existe um modo adicional de codificação para macroblocos do tipo I, chamado I_PCM. Neste caso, as amostras da imagem são transmitidas diretamente, sem predição, transformada e quantização (RICHARDSON, 2003).

A parte destacada em cinza na Figura 2.5 mostra um bloco 4x4 de luminância a ser codificado pela predição intra-quadro. As amostras acima e à esquerda (letras A a M na Figura 2.5) foram codificadas e reconstruídas antes deste bloco ser processado. Estas amostras serão utilizadas como referências para a predição intra-quadro.

M	A	B	C	D	E	F	G	H
I	a	b	c	d				
J	e	f	g	h				
K	i	j	k	l				
L	m	n	o	p				

Figura 2.5: Identificação das amostras para a predição intra-quadro

A Figura 2.6 apresenta os nove tipos diferentes de codificação no modo intra-quadro para blocos 4x4 de luminância. Os modos 0 e 1 fazem uma simples extrapolação (uma cópia) dos pixels das bordas verticais ou horizontais para todas as posições do bloco. O modo DC (2) faz uma média entre as amostras das bordas e copia o resultado para todas as posições do bloco. Os demais modos (3 a 8) fazem uma média ponderada das amostras das bordas, de acordo com a direção da seta na Figura 2.6.

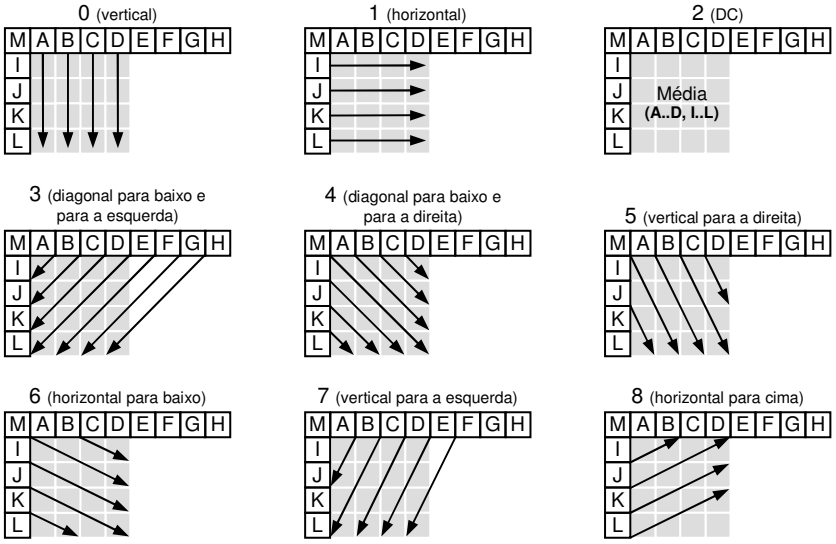


Figura 2.6: Nove modos da predição intra-quadro para blocos de luminância 4x4

Como já foi mencionado, a predição intra-quadro também pode acontecer sobre o componente de luminância completo de um macrobloco, ou seja, com blocos do tamanho 16x16. Neste caso, são quatro os modos possíveis de predição intra-quadro, conforme está apresentado na Figura 2.7. Os modos 0 e 1 são simples cópias na vertical ou na horizontal das amostras reconstruídas das bordas. O modo 2 é o modo DC e é formado por uma média simples dos elementos das bordas, cujo resultado é copiado para todas as posições do bloco. O modo 3, modo planar, aplica uma função linear que considera as amostras horizontais e verticais (H e V na Figura 2.7) das bordas.

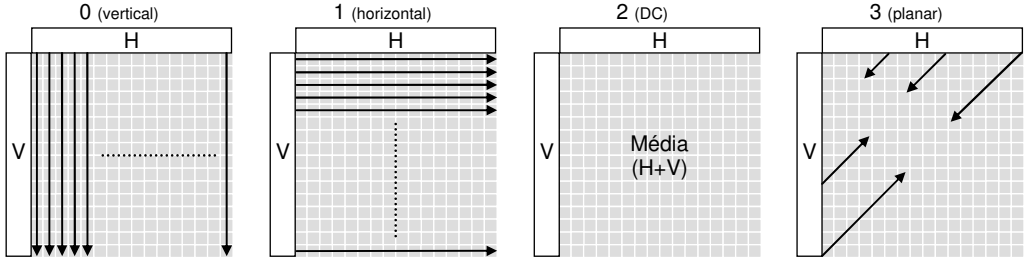


Figura 2.7: Quatro modos da predição intra-quadro para blocos de luminância 16x16

O modo planar é o mais complexo de ser calculado, por necessitar do cálculo de três parâmetros (a,b e c) como mostrado em (2.1), (2.2) e (2.3) sendo que o cálculo de (2.2) e (2.3) dependem dos parâmetros H e V definidos em (2.4) e (2.5).

$$a = 16 \cdot (p[-1,15] + p[15,-1]) \quad (2.1)$$

$$b = (5 \cdot H + 32) \gg 6 \quad (2.2)$$

$$c = (5 \cdot V + 32) \gg 6 \quad (2.3)$$

$$H = \sum_{x'=0}^{7} (x'+1) \cdot (p[-1,8+x'] - p[-1,6-x']) \quad (2.4)$$

$$V = \sum_{y'=0}^{7} (y'+1) \cdot (p[8+y',-1] - p[6-y',-1]) \quad (2.5)$$

A posição [y,x] representa a amostra contida na linha y e coluna x do macrobloco, sendo que [0,0] representa o canto superior esquerdo do macrobloco. A amostra p[y,x] representa a amostra vizinha, sendo que os índices [y,x] estão numerados relativos à amostra do canto superior esquerdo do macrobloco. A predição do modo planar é calculada, a partir dos parâmetros previamente calculados, de acordo com (2.6), onde P[y,x] é a predição da amostra [y,x] do macrobloco e Clip é uma função de saturação para adequar os valores na faixa de representação da imagem (0 a 255, para amostras de 8 bits).

$$P[y,x] = Clip((a + b \cdot (x-7) + c \cdot (y-7) + 16) \gg 5 \quad (2.6)$$

Todos os demais modos utilizam apenas as equações (2.7), (2.8) e (2.9) para gerar as predições.

$$p = (a + 2 \cdot b + c + 2) \gg 2 \quad (2.7)$$

$$p = (g + 3 \cdot h + 2) \gg 2 \quad (2.8)$$

$$p = (l + k + 1) \gg 1 \quad (2.9)$$

A predição da crominância é realizada diretamente sobre blocos 8x8 e utiliza 4 modos distintos de predição, mas os dois componentes de crominância utilizam sempre o mesmo modo. Os modos de predição para crominância são muito similares aos modos de predição de luminância para blocos 16x16 que foram apresentados na Figura 2.7, exceto pela numeração dos modos. Para a predição de crominância, o modo DC é o zero, o modo horizontal é o um, o modo vertical é o dois e o modo plano é o três (RICHARDSON, 2003).

Os diferentes modos de predição intra-quadro para luminância e crominância possibilitam a geração de uma predição para macroblocos do tipo I que conduz a uma codificação eficiente para este tipo de macrobloco. A escolha de qual modo de predição será utilizado é realizada pelo controle do codificador, que deve sinalizar o modo escolhido no cabeçalho do macrobloco. Para escolher o melhor modo, o codificador deve gerar a predição sobre todos os modos e escolher qual é o melhor do ponto de vista da eficiência de codificação. Esta tarefa possui uma complexidade computacional elevada para o codificador, porém muito inferior do que a complexidade da estimação de movimento.

2.3.4 O Módulo das Transformadas Diretas (T)

O módulo T, apresentado na Figura 2.1, é responsável pelas transformadas diretas e está presente apenas nos codificadores H.264/AVC. As entradas para o módulo T são blocos 4x4 de resíduos gerados pela etapa de predição. A DCT 2-D direta (FDCT 2-D) é aplicada a todas as amostras de entrada, tanto de luminância quanto de crominância. A FDCT 2-D do padrão H.264/AVC é uma aproximação inteira da DCT 2-D real. Esta aproximação, como já mencionado, foi realizada para reduzir a complexidade do cálculo e evitar erros de casamento entre o codificador e o decodificador. O cálculo da FDCT 2-D inteira é definido no padrão H.264/AVC como está apresentado em (2.10).

$$Y = C_f X C_f^T \otimes E_f = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} & X & \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 2 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix} \quad (2.10)$$

Em (2.10), **X** é a matriz 4x4 de entrada, $\mathbf{C_f}$ é a matriz da FDCT inteira em uma dimensão, $\mathbf{C_f^T}$ é a transposta de da matriz da DCT e $\mathbf{E_f}$ é a matriz de fatores de escala. O símbolo ⊗ na equação indica uma multiplicação escalar. As letras **a** e **b** na matriz $\mathbf{E_f}$ são as constantes definidas em (2.11).

$$a = \frac{1}{2}, \qquad b = \sqrt{\frac{2}{5}} \qquad (2.11)$$

O cálculo da FDCT 2-D transfere para o módulo de quantização (Q), que é o passo que segue a aplicação das transformadas diretas na compressão H.264/AVC, a tarefa de realizar a multiplicação escalar por $\mathbf{E_f}$. Esta tarefa adicional não implica em aumento na complexidade do módulo Q.

Como pode ser observado em (2.10), as matrizes $\mathbf{C_f}$ e $\mathbf{C_f^T}$ possuem apenas valores -2, -1, 1 e 2. Isso implica na realização de operações bastante simples, consistindo de somas e subtrações diretas e multiplicações por dois, que são equivalentes a um deslocamento de uma casa binária para a esquerda.

O cálculo da FDCT 2-D é aplicado sobre todos os dados de entrada, mas, no caso de amostras com informação de crominância ou com informação de luminância cuja predição tenha sido do tipo intra-quadro 16x16, um cálculo adicional é realizado. Em ambos os casos é aplicada a transformada Hadamard 2-D direta sobre os coeficientes DC resultantes da FDCT 2-D. Esta segunda transformada é uma inovação do padrão H.264/AVC e foi inserida com o objetivo de atingir ainda mais compressão em áreas homogêneas do vídeo. Os coeficientes DC dos blocos de luminância cuja predição tenha sido intra-quadro 16x16 são submetidos a uma transformada Hadamard 4x4 direta. Por outro lado, uma transformada Hadamard 2x2 direta é aplicada sob os coeficientes DC dos blocos de crominância.

A transformada Hadamard 4x4 direta definida pelo padrão H.264/AVC está apresentada em (2.12). Esta transformada é aplicada apenas sobre os elementos DC dos blocos 4x4 (resultantes da aplicação da FDCT 2-D) de um macrobloco 16x16 de luminância que tenha utilizado a predição intra-quadro 16x16. Então os 16 elementos DC dos 16 blocos do macrobloco irão formar a matriz 4x4 $\mathbf{W_D}$ de entrada para a Hadamard direta 4x4.

$$Y_D = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} & W_D & \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) / 2 \qquad (2.12)$$

Como pode ser observado em (2.12), os elementos das matrizes possuem apenas valores -1 e 1. Deste modo, apenas somas e subtrações são necessárias para realizar os cálculos relativos a esta transformada. A divisão por dois realizada em todos os

elementos da matriz de resultados é um simples deslocamento de uma casa binária para a direita.

A transformada Hadamard 2x2 é aplicada apenas para amostras DC de crominância. Esta transformada é utilizada tanto para Cb quanto para Cr. Em função da relação de entrada 4:2:0 para Y, Cb e Cr, cada macrobloco possui 16x16 amostras de luminância, 8x8 amostras de Cb e 8x8 amostras de Cr. As amostras de crominância passam pela FDCT 2-D em blocos de 4x4 amostras. Então, cada matriz 8x8 de entrada é formada por quatro matrizes 4x4. Os quatro elementos DC das quatro matrizes resultantes da FDCT 2-D passam pela Hadamard 2-D 2x2.

O cálculo da Hadamard 2x2 definido pelo padrão H.264/AVC está apresentado em (2.13).

$$W_{QD} = \left(\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \\ W_D \\ \\ \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \right) \qquad (2.13)$$

Pode ser observado na fórmula acima que os cálculos da Hadamard 2x2 sobre as amostras DC de Cb e Cr são muito simples, consistindo de poucas somas e subtrações.

O módulo T, sendo formado pelas três transformadas que estão apresentadas em (2.10), (2.12) e (2.13), deve sincronizar a operação entre elas, de modo a gerar o fluxo correto de dados na sua saída. A ordem de processamento está apresentada na Figura 3.16. Inicialmente, o macrobloco de luminância (**Y** na Figura 2.8) passa pela FDCT 2-D. Se o modo é intra-quadro 16x16, então, como foi explicado, os coeficientes DCs das matrizes 4x4 resultantes da FDCT 2-D passam pela Hadamard 4x4 direta. Neste caso, primeiramente é enviado para saída o bloco **–1** na Figura 2.8, com os resultados da Hadamard 4x4 direta, e depois os elementos AC são enviados para a saída (blocos **0** a **15** na Figura 2.8). Se o modo não é intra-quadro 16x16, então os blocos **0** a **15** são enviados diretamente para a saída e a Hadamard 4x4 não é aplicada. Seguindo os 16 blocos de luminância, são processados quatro blocos de crominância Cb e quatro blocos de crominância Cr. Os blocos de crominância passam pela FDCT 2-D e, sob os elementos DC dos resultados, é aplicada a Hadamard 2-D 2x2. Então é enviado para a saída o bloco **16** na Figura 2.8, com os resultados da Hadamard 2x2 para o componente Cb, depois é enviado o bloco **17**, com os resultados da Hadamard 2x2 para Cr. Finalmente, os coeficientes AC de crominância são enviados para a saída, primeiro os de Cb (blocos **18** a **21**) e depois os de Cr (blocos **22** a **25**).

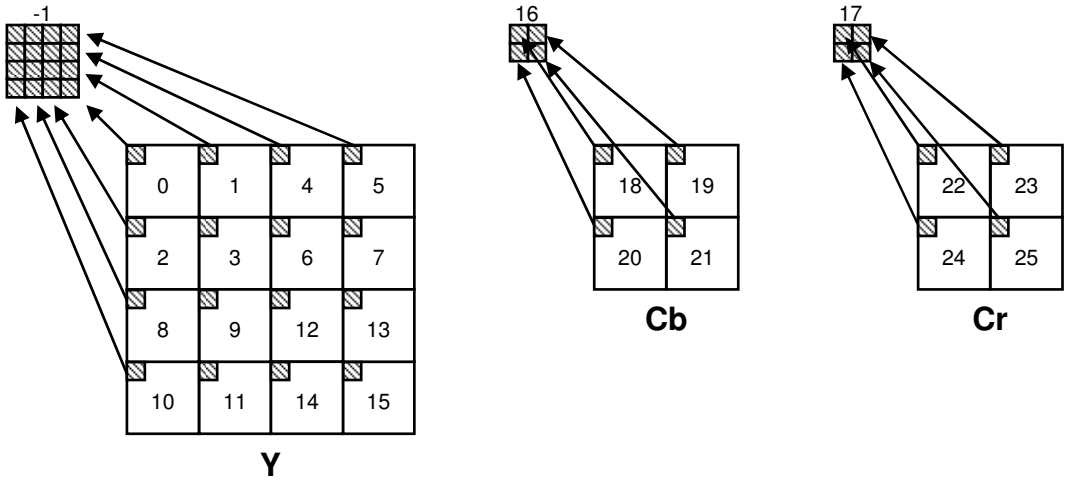


Figura 2.8: Ordem de processamento de amostras pelo módulo T

2.3.5 O Módulo da Quantização Direta (Q)

O módulo Q na Figura 3.4 realiza a quantização direta e a correção da escala do cálculo das transformadas. Este módulo está presente apenas no codificador H.264/AVC. Dependendo do modo de predição utlizado e se o elemento é de crominância ou luminância, os cálculos realizados pelo módulo Q são modificados mas, genericamente, as operações realizadas são uma multiplicação da entrada por uma constante, a soma do resultado com uma outra constante e um deslocamento no resultado da soma controlado por uma terceira constante. Estas constantes são influenciadas diretamente pelo parâmetro de quantização (QP) que é uma entrada externa que informa ao módulo Q qual é o passo de quantização (Qstep) que deve ser utilizado. QP pode variar de 0 a 51 e para cada QP existe um Qstep. Os primeiros seis valores de Qstep, relativos aos seis primeiros QP, são definidos pelo padrão como está apresentado na Tabela 2.1. Os demais Qsteps podem ser derivados dos seis primeiros, pois o Qstep dobra de valor a cada variação de 6 em QP. Então o $Qstep_{(6)}$ é igual $Qstep_{(0)} \times 2$.

Tabela 2.1: Relação entre QP e Qstep

QP	0	1	2	3	4	5	6	...	12
Qstep	0,625	0,6875	0,8125	0,875	1	1,125	1,25	...	2,5

Para os elementos de luminância ou crominância AC e para elementos de luminância DC que não tenham sido codificados no modo intra-quadro 16x16, ou seja, para os elementos que foram processados apenas pela FDCT 2-D no módulo T, a quantização é definida por (2.14), já considerando o uso de números em ponto fixo:

$$\begin{aligned} |Z_{(i,j)}| &= (|W_{(i,j)}| \cdot MF + f) >> qbits \\ sign(Z_{(i,j)}) &= sign(W_{(i,j)}) \end{aligned} \qquad (2.14)$$

Em (2.14), $\mathbf{W_{ij}}$ é o coeficiente resultante da DCT 2-D, **MF** é uma constante gerada a partir do fator de escala e do parâmetro de quantização (**QP**), **f** é uma constante definida pelo padrão em função da predição ter sido gerada pelo modo inter-quadros ou intra-quadro e do parâmetro de quantização utilizado. Por fim, **qbits** indica o deslocamento que deve ocorrer antes do cálculo ser finalizado. É importante salientar que o sinal do resultado deve ser o mesmo sinal da amostra de entrada e que o cálculo é realizado apenas considerando o módulo da amostra de entrada.

A constante **MF** é definida como está apresentado em (2.15).

$$MF = \frac{PF}{Qstep} << qbits \qquad (2.15)$$

Em (2.15), **PF** é o fator de escala, **Qstep** é o passo de quantização e **qbits** é o mesmo deslocamento que estava apresentado em (2.14). O fator de escala **PF** depende da posição da amostra no módulo e pode ser $\mathbf{a^2}$, **ab/2** ou $\mathbf{b^2/4}$, sendo que **a** e **b** são as mesmas constantes definidas para o módulo T em (2.11).

O cálculo de **qbits** é função de **QP** e está apresentado em (2.16). É importante destacar o arredondamento para baixo do fator **QP/6** que está apresentado em (2.16).

$$qbits = 15 + \lfloor QP/6 \rfloor \qquad (2.16)$$

Por fim, a constante **f** é definida em (2.17).

$$\begin{aligned} f &= 2^{qbits}/3 && \text{se a predição for intra} \\ f &= 2^{qbits}/6 && \text{se a predição for inter} \end{aligned} \qquad (2.17)$$

A quantização para amostras **DC** de crominância ou para amostras de luminância que foram codificados segundo a predição intra-quadro no modo 16x16 é definida por (2.18).

$$\begin{aligned} |Z_{D(i,j)}| &= (|Y_{D(i,j)}| \cdot MF_{(0,0)} + 2f) >> (qbits + 1) \\ sign(Z_{D(i,j)}) &= sign(Y_{D(i,j)}) \end{aligned} \qquad (2.18)$$

Este cálculo é bastante similar ao apresentado em (2.14), sendo que as constantes **MF**, **qbits** e **f** são definidas de modo idêntico ao descrito (2.15), (2.16) e (2.17).

2.3.6 O Módulo das Transformadas Inversas (IT)

O módulo IT do padrão H.264/AVC, apresentado nas Figuras 3.4 e 3.5, por fazer as operações inversas ao módulo T, possui muitas características idênticas a este módulo. O módulo IT está presente nos codificadores e nos decodificadores H.264/AVC. O curioso é que o padrão H.264/AVC definiu que a operação do módulo IT seja fracionada em duas partes. A primeira é realizada diretamente sobre os resultados da quantização direta (Q) e envolve as transformadas Hadamard 2x2 e 4x4 inversas. Este resultado é, então, entregue para a quantização inversa (IQ). Então, os resultados da quantização inversa são entregues novamente para o módulo IT, que realiza a última etapa de seus cálculos, aplicando a DCT 2-D inversa (IDCT 2-D).

Para coeficientes DC das informações de crominância ou das informações de luminância cuja predição tenha sido do tipo intra-quadro 16x16 é aplicada a transformada Hadamard 2-D inversa. Nos coeficientes DC dos blocos de luminância codificados no modo intra-quadro 16x16 é aplicada a Hadamard 4x4 inversa, enquanto que para os coeficientes DC de crominância é aplicada a Hadamard 2x2 inversa.

O cálculo da Hadamard 4x4 inversa está apresentado em (2.19).

$$W_{QD} = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} \\ Z_D \\ \\ \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) \qquad (2.19)$$

Como pode ser observado em (2.19), o cálculo da Hadamard 4x4 inversa é muito semelhante ao cálculo da Hadamard 4x4 direta que foi apresentada em (2.12), na descrição do módulo T. As matrizes possuem apenas valores 1, positivos e negativos, como na Hadamard 4x4 direta. Esta característica implica na realização apenas de operações de somas e subtrações. A diferença entre a Hadamard 4x4 direta e inversa está na divisão por dois realizada sob todos os valores de saída, que não existe na Hadamard 4x4 inversa.

A transformada Hadamard 2x2 inversa é aplicada apenas para amostras DC de crominância. Esta transformada é utilizada tanto para Cb quanto para Cr. O cálculo da Hadamard 2x2 inversa é idêntico ao cálculo da Hadamard 2x2 direta, que foi apresentado em (2.13).

A transformada IDCT 2-D definida pelo padrão H.264/AVC está apresentada em (2.20), onde **X** é a matriz 4x4 de entrada, $\mathbf{C_i}$ é a matriz da IDCT inteira em uma dimensão, $\mathbf{C_i^T}$ é a transposta de da matriz da IDCT em uma dimensão e $\mathbf{E_i}$ é a matriz de fatores de escala. O símbolo ⊗ na equação indica uma multiplicação escalar. As letras **a** e **b** na matriz $\mathbf{E_i}$ são as mesmas constantes definidas para a DCT 2-D direta.

$$Y = C_i^T (X \otimes E_i) C_i = \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \left(\begin{bmatrix} & X & \end{bmatrix} \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \right) \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{bmatrix} \quad (2.20)$$

O cálculo da IDCT 2-D já recebe em suas entradas o cálculo relativo à multiplicação escalar por $\mathbf{E_i}$. O módulo de quantização inversa (IQ), do mesmo modo que para o cálculo da FDCT 2-D, realiza esta operação. Novamente, esta tarefa adicional não implica em aumento na complexidade do módulo IQ.

Como pode ser observado em (2.20), as matrizes $\mathbf{C_i^T}$ e $\mathbf{C_i}$ possuem apenas valores 1 e ½, positivos e negativos. Isso implica na realização de operações de somas e subtrações e um deslocamento de uma casa para a direita, no caso do cálculo utilizar o valor ½.

O cálculo da IDCT 2-D é aplicado sobre todos os dados que são entregues pela quantização inversa (IQ).

O módulo IT, sendo formado pelas três transformadas inversas que estão apresentadas em (2.13), (2.19) e (2.20) deve sincronizar a operação entre elas, de modo a gerar o fluxo correto de dados na sua saída. A ordem de processamento é exatamente a mesma apresentada na descrição do módulo T.

2.3.7 O Módulo da Quantização Inversa (IQ)

O módulo IQ, apresentado nas Figuras 2.1 e 2.2, está presente nos codificadores e nos decodificadores H.264/AVC. Este módulo realiza a quantização inversa e a correção da escala do cálculo das transformadas.

Como já mencionado na seção anterior, por uma definição do padrão, antes da quantização inversa são realizados partes dos cálculos relativos ao módulo IT. As tranformadas Hadamard inversas 4x4 e 2x2, quando utilizadas (DCs de crominância ou DCs de luminância para o modo intra-quadro 16x16), são aplicadas antes da quantização inversa.

Dependendo do modo de predição utilizado e se o elemento é de crominância ou de luminância, os cálculos realizados pelo módulo IQ são modificados, do mesmo modo que ocorre no módulo Q. Também no módulo IQ as operações realizadas são uma multiplicação da entrada por uma constante, a soma do resultado com uma outra constante e um deslocamento no resultado da soma que é controlado por uma terceira constante.

Como na quantização direta, estas constantes são influenciadas diretamente pelo parâmetro de quantização (QP) que é uma entrada externa que informa ao módulo IQ

qual é o passo de quantização (Qstep) que foi utilizado na codificação. Os Qsteps utilizados na quantização inversa são os mesmos que são usados na quantização direta.

Para os elementos de luminância ou crominância AC e para elementos de luminância DC que não tenham sido codificados no modo intra-quadro 16x16, a quantização inversa é definida por (2.21).

$$W'_{(i,j)} = Z_{(i,j)} \cdot V_{(i,j)} \cdot 2^{\lfloor QP/6 \rfloor} \qquad (2.21)$$

Em (2.21), $\mathbf{Z_{(i,j)}}$ é o coeficiente quantizado, $\mathbf{V_{(i,j)}}$ é uma constante gerada a partir do fator de escala ($\mathbf{PF_{(i,j)}}$) e do parâmetro de quantização (**QP**) e $2^{\lfloor QP/6 \rfloor}$ é um deslocamento definido por QP e que deve ocorrer na saída antes do cálculo ser finalizado.

A constante $\mathbf{V_{(i,j)}}$ está definida em (2.22), onde **Qstep** é o passo de quantização e $\mathbf{PF_{(i,j)}}$ é o fator de escala.

$$V_{(i,j)} = (Qstep \cdot PF_{(i,j)} \cdot 64) \qquad (2.22)$$

O fator de escala $\mathbf{PF_{(i,j)}}$ é diferente na quantização inversa em relação à quantização direta, mas também depende da posição da amostra no bloco. Na quantização inversa $\mathbf{PF_{(i,j)}}$ pode assumir os valores $\mathbf{a^2}$, **ab** ou $\mathbf{b^2}$ sendo que **a** e **b** são as mesmas constantes definidas para o módulo T em (2.11). A multiplicação por 64, que é um deslocamento de seis casas binárias para a esquerda, é utilizada para prevenir erros de arrendodamento.

A quantização inversa para elementos **DC** de luminância que foram codificados segundo a predição intra-quadro no modo 16x16 é definida em (2.23).

$$\begin{aligned} W'_{D(i,j)} &= W_{QD(i,j)} \cdot V_{(0,0)} \cdot 2^{floor(QP/6)} && (QP \geq 12) \\ W'_{D(i,j)} &= \left[W_{QD(i,j)} \cdot V_{(0,0)} \cdot 2^{1-floor(QP/6)}\right] >> \lfloor QP/6 \rfloor && (QP < 12) \end{aligned} \qquad (2.23)$$

Este cálculo é bastante similar aos anteriores, sendo que a constante $\mathbf{V_{(i,j)}}$ é definida de modo idêntico ao citado acima.

A quantização para elementos **DC** de crominância é definida por (2.24).

$$\begin{aligned} W'_{D(i,j)} &= W_{QD(i,j)} \cdot V_{(0,0)} \cdot 2^{\lfloor QP/6 \rfloor - 1} && (QP \geq 6) \\ W'_{D(i,j)} &= \left[W_{QD(i,j)} \cdot V_{(0,0)}\right] >> 1 && (QP < 6) \end{aligned} \qquad (2.24)$$

2.3.8 O Módulo do Filtro Redutor do Efeito de Bloco

O módulo Filtro nas Figuras 2.1 e 2.2 é o filtro redutor de efeito de bloco normatizado pelo padrão H.264/AVC e que está presente nos codificadores e nos decodificadores que seguem este padrão.

O objetivo deste filtro é suavizar o efeito de blocos do quadro reconstruído antes de ele ser usado para fazer a predição de um novo macrobloco do tipo inter-quadros. Este filtro é adaptativo, distinguindo uma aresta real da imagem, que não deve ser filtrada, de um artefato gerado por um elevado passo de quantização, que deve ser filtrado.

A filtragem é aplicada para bordas verticais e horizontais dos blocos 4x4 de um macrobloco, de acordo com os passos 1 a 4 apresentados abaixo e na Figura 2.9.

1) Filtrar as quatro bordas verticais do componente de luminância (**a**, **b**, **c** e **d** na Figura 2.9);
2) Filtrar as quatro bordas horizontais do componente de luminância (**e**, **f**, **g** e **h** na Figura 2.9);
3) Filtrar as duas bordas verticais de cada componente de crominância (**i** e **j** na Figura 2.9);
4) Filtrar as duas bordas horizontais de cada componente de crominância (**k** e **l** na Figura 2.9).

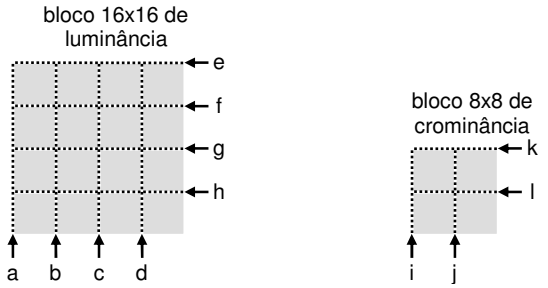


Figura 2.9: Ordem de filtragem de bordas em um macrobloco

Cada operação de filtragem afeta até três amostras de cada lado da borda, como está apresentado na Figura 2.10. Neste caso, são considerados dois blocos 4x4 adjacentes chamados de **p** e **q**, e apenas quatro amostras de cada bloco estão presentes em cada exemplo.

A "força" de filtragem a ser realizada depende da quantização utilizada no bloco, do modo de codificação dos blocos vizinhos e do gradiente das amostras da imagem através da borda. A aplicação do filtro proporciona um aumento significativo da qualidade subjetiva do vídeo reconstruído.

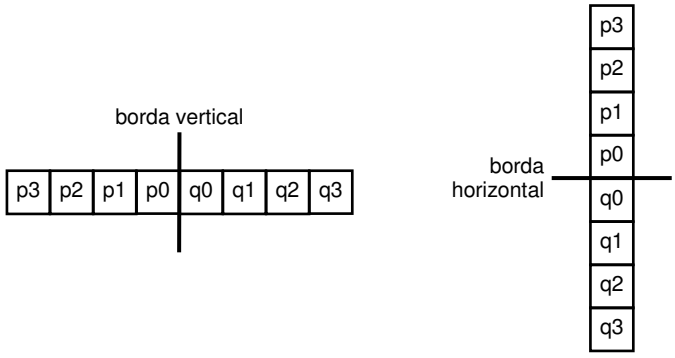


Figura 2.10: Amostras adjacentes para bordas verticais e horizontais

Existem cinco diferentes forças de filtragem, sendo definidos pelo parâmetro ***b*S** (*boundary strength*). O parâmetro ***b*S** pode variar de zero a quatro e, portanto, existem cinco diferentes forças de filtragem, onde um ***b*S**=4 define uma filtragem com força máxima e um ***b*S**=0 define que nenhuma filtragem é realizada. A definição de qual tipo de filtragem deve ser realizada segue as seguintes regras, considerando uma borda entre os blocos **q** e **p**:

- ***b*S=4** : é utilizado se os blocos **q** e/ou **p** foram codificados no modo intra-quadro e se a borda é uma borda de macrobloco.

- ***b*S=3** : é utilizado se os blocos **q** e/ou **p** foram codificados no modo intra-quadro e se a borda não é uma borda de macrobloco.
- ***b*S=2** : é utilizado se os blocos **q** e/ou **p** não foram codificados no modo intra-quadro e se **p** e **q** possuem coeficientes codificados.
- ***b*S=1** : é utilizado se os blocos **q** e/ou **p** não foram codificados no modo intra-quadro; se **p** e **q** não possuem coeficientes codificados; se **p** e **q** usam quadros de referência diferentes ou diferentes números de quadros de referência ou possuem valores de vetores de movimento que se diferenciam por uma ou mais amostras de luminância;
- ***b*S=0** : é utilizado se nenhuma das situações anteriores for verdadeira.

Mas a decisão de realizar a filtragem não depende apenas do parâmetro ***b*S**. Considerando um grupo de amostras (p2, p1, p0, q0, q1, q2), a filtragem acontece se *b*S>0 e se a condição apresentada em (2.25) for verdadeira.

$$|p0 - q0| < \alpha \quad \text{e} \quad |p1 - p0| < \beta \quad \text{e} \quad |q1 - q0| \leq \beta \qquad (2.25)$$

Em (2.25), α e β são definidos pelo padrão e crescem de acordo com a média do parâmetro de quantização (QP) dos blocos **p** e **q**. Se QP possuir um valor baixo, então α e β também terão valores baixos. Neste caso, o efeito de bloco gerado tem importância menor e, por isso, o filtro permanecerá inativo durante mais tempo. Por outro lado, se QP possuir um valor elevado, então a quantização gerará mais perdas, gerando efeitos de blocos mais significativos. Neste caso, os valores de α e β serão maiores, fazendo com que o filtro fique mais tempo ativo e, assim, mais amostras da borda são filtradas.

2.3.9 O Módulo de Codificação de Entropia

Na codificação de entropia, que está presente nos codificadores e decodificadores (Figuras 2.1 e 2.2) o padrão H.264/AVC introduz algumas ferramentas que aumentam bastante a sua eficiência de codificação. Nos níveis hierárquicos superiores (quadros, etc.), os elementos sintáticos são codificados usando códigos binários fixos ou de comprimento variável. A partir do nível de *slices* ou abaixo (macroblocos, blocos, etc.), os elementos sintáticos são codificados usando a codificação aritmética adaptativa ao contexto (CABAC) (RICHARDSON, 2003) ou códigos de comprimento variável (VLC) (SALOMON, 2000). No caso do uso de VLC, a informação residual (coeficientes das transformadas quantizados) é codificada usando a codificação de comprimento variável adaptativa ao contexto (CAVLC) (RICHARDSON, 2003), enquanto que as demais unidades de codificação são codificadas usando códigos Exp-Golomb (SALOMON, 2000). A opção pelo CABAC não está disponível nos perfis *baseline* e *extended*.

A principal inovação introduzida na codificação de entropia do padrão H.264/AVC, tanto na codificação aritmética quanto na codificação VLC é o uso de codificação adaptativa baseada em contextos. Nela, a maneira com que são codificados os diversos elementos sintáticos depende do elemento a ser codificado, da fase em que se encontra o algoritmo de codificação e dos elementos sintáticos que já foram codificados.

Os resíduos resultantes da quantização devem ser reordenados antes de serem utilizados pela codificação de entropia. Os blocos 4x4 com coeficientes do módulo das transformadas, após a quantização, são reordenados em forma de ziguezague, como está

apresentado na Figura 2.11. Quando a predição intra-quadro 16x16 é utilizada o bloco -1 da Figura 2.8, contendo os resultados quantizados da Hadamard 4x4 direta, é também ordenado em ziguezague. Os blocos 0 a 15 na Figura 2.8 irão possuir 15 elementos ao invés de 16, em função da aplicação da Hadamard 4x4 sobre os elementos DC de cada bloco. Então os 15 coeficientes AC que restam em cada bloco são reorganizados do mesmo modo que está apresentado na Figura 2.11, mas iniciando da segunda posição. Os blocos 16 e 17 da Figura 2.8, que foram processados pela Hadamard 2x2 direta, são também ordenados em ziguezague, mesmo contendo apenas 2x2 amostras. Finalmente, os 15 coeficientes AC restante nos blocos 4x4 de crominância são, então, reorganizados seguindo a ordem da Figura 2.11, mas iniciando da segunda posição.

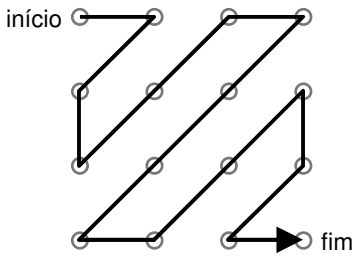


Figura 2.11: Ordem ziguezague de leitura dos blocos 4x4
para a codificação de entropia

As próximas subseções do texto irão apresentar resumidamente, a codificação Exp-Golomb, a codificação de comprimento variável adaptativa ao contexto (CAVLC) e a codificação aritmética binária adaptativa ao contexto (CABAC).

2.3.9.1 Códigos Exp-Golomb

Os códigos Exp-Golomb (*Exponencial* Golomb) (SALOMON, 2000) são códigos de comprimento variável com uma construção regular. Um número inteiro não negativo *N* é codificado usando a estrutura apresentada em (2.26):

$$Código = [\ M\ zeros\ ][\ 1\ ][\ INFO\ ] \qquad (2.26)$$

Em (27), **M** representa o número de zeros que antecedem o primeiro valor 1 no código. O valor de **M** é dado por (2.27)

$$M = \lfloor \log_2 \cdot (num\_cod + 1) \rfloor \qquad (2.27)$$

A variável **num_cod** em (27) indica qual é o número do código. Códigos com maior probabilidade de ocorrência possuem um número de código mais baixo.

O campo INFO em (28) possui **M** bits e indica qual é a informação codificada. O valor do campo INFO é dado por (2.28).

$$INFO = num\_cod + 1 - 2^M \qquad (2.28)$$

O primeiro código de Exp-Golomb não possui os campos **M zeros** e **INFO**, sendo sempre representado apenas pelo valor ‘1’. A Tabela 2.2 apresenta os seis primeiros códigos de Exp-Golomb.

No padrão H.264/AVC, para cada elemento sintático **k** a ser codificado com o código Exp-Golomb, há uma regra que mapeia o valor de **k** para valores inteiros não negativos, isto é, a regra indica como o valor do elemento sintático **k** deve ser mapeado para um valor de **num_cod**. São quatro as possibilidades de mapeamento, dependendo do tipo de elemento sintático codificado (ITU-T, 2005).

Tabela 2.2: Seis primeiros códigos de Exp-Golomb

num_cod	Código
0	1
1	010
2	011
3	00100
4	00101
5	00110
. . .	. . .

2.3.9.2 Codificação de Comprimento Variável Adaptativa ao Contexto – CAVLC

A codificação de comprimento variável adaptativa ao contexto (CAVLC) é utilizada para codificar os resíduos resultantes da quantização, já ordenados em ziguezague. 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 CAVLC foi desenvolvida para explorar as características dos blocos quantizados, quais sejam (RICHARDSON, 2003):

- O resultado da quantização produz, tipicamente, matrizes esparsas. Então a CAVLC utiliza RLE (SALOMON, 2000) para representar compactamente as seqüências de zeros.
- Os coeficientes não zeros de alta freqüência, depois da leitura em ziguezague, são freqüentemente seqüências de ±1. Então a CAVLC sinaliza o número dos coeficientes ±1 de alta freqüência de uma forma compacta.
- O número de coeficientes não zero em blocos vizinhos são correlacionados. O número de coeficientes é codificado usando uma tabela e a escolha de qual valor da tabela deve ser usado depende do número de coeficientes não zero nos blocos vizinhos.
- O nível (magnitude) dos coeficientes não zero tende a ser maior para os primeiros coeficientes lidos em ziguezague (baixas freqüências) e menores para as freqüências mais elevadas. Então o CAVLC tira vantagem desta característica adaptando a escolha da tabela de VLC que será utilizada para o parâmetro de nível dependendo dos níveis de magnitude dos coeficientes recentemente codificados.

A CAVLC está disponível para todos os perfis do padrão H.264/AVC e é uma alternativa menos complexa ao codificador aritmético adaptativo ao contexto (CABAC), que está disponível nos perfis *main* e *high*. A escolha pelo CAVLC ao invés do CABAC conduz a uma implementação de menor complexidade, mas gera um impacto negativo na eficiência de codificação.

2.3.9.3 Codificação Aritmética Binária Adaptativa ao Contexto (CABAC)

A codificação aritmética binária adaptativa ao contexto (CABAC) é uma ferramenta de codificação disponível apenas para os perfis *main* e *high* do padrão H.264/AVC. A CABAC atinge elevadas taxas de compressão através da seleção dos 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.

A codificação pelo CABAC envolve os seguintes estágios:

1. **Binarização**: O CABAC utiliza codificação aritmética binária, o que significa que apenas decisões binárias (0 ou 1) são codificadas. Então os símbolos que não possuem valores binários são binarizados ou convertidos em um código binário antes da codificação. Esse processo é similar ao de converter um símbolo de dados em um código de comprimento variável (VLC), mas o código binário é codificado adicionalmente pelo codificador aritmético antes de ser transmitido. Cada posição de um dígito binário é chamada de um **bin**. Os passos 2, 3, e 4 são repetidos para todos **bins**.
2. **Seleção dos modelos probabilísticos:** Um modelo de contexto é um modelo probabilístico para um ou mais **bins** do símbolo binarizado. A escolha do modelo de contexto a ser utilizado depende dos modelos disponíveis e das estatísticas dos símbolos recentemente codificados. O modelo de contexto armazena a probabilidade de cada **bin** ser 0 ou 1. No H.264/AVC são usados 398 contextos diferentes (ITU-T, 2005; MARPE, 2003).
3. **Codificação aritmética**: Um codificador aritmético codifica cada **bin** de acordo com o modelo probabilístico selecionado. A codificação aritmética será apresentada com mais detalhes no decorrer desta seção, mas é importante notar que, de acordo com os princípios da codificação aritmética, existem somente duas sub-faixas possíveis para cada **bin**, correspondentes a ‘0’ e ‘1’.
4. **Atualização de probabilidades:** O modelo de contexto selecionado é atualizado com base no valor atual codificado. Por exemplo, se o valor do **bin** atual é ‘0’, a contagem de ocorrências de zeros é incrementada.

Como foi apresentado no item 3 acima, a operação do CABAC utiliza a codificação aritmética para codificar os **bins**. Na codificação aritmética, uma mensagem é representada por um intervalo contido entre ‘0’ e ‘1’. Considerando que a tabela de probabilidades de ocorrência dos símbolos já tenha sido construída, a codificação aritmética segue os seguintes passos:

1. Definir a faixa inicial de probabilidades contendo todos os símbolos. Para N símbolos o intervalo é dividido em N subintervalos tal que o intervalo correspondente a um símbolo qualquer possui largura igual à sua probabilidade;
2. Ler o próximo símbolo da entrada;
3. Encontrar a sub-faixa do símbolo atual;
4. Expandir esta sub-faixa para que seja usada como nova faixa, entre ‘0’ e ‘1’;
5. Repetir os passos 2 a 4 até que o valor de um dígito pára de variar (intervalo pequeno o suficiente);

6. Transmitir algum valor que esteja na faixa de valores do último símbolo codificado.

A Figura 2.12 apresenta um exemplo de codificação aritmética, considerando a seqüência de símbolos (0, -1, 0, 2). A tabela de probabilidades para este exemplo está apresentada na Tabela 2.3. Na Figura 2.12, os números maiores sem parênteses indicam a divisão da faixa inicial. Os números grandes entre parênteses indicam qual é o símbolo atual que está sendo codificado. Os números menores indicam a distribuição de probabilidades para cada passo da codificação.

Para o exemplo da Figura 2.12, o intervalo final ficou entre 0,3928 e 0,396 e o valor 0,394 foi o escolhido para ser transmitido e representar o conjunto de valores (0, -1, 0, 2). Com este valor e com o conhecimento dos dados apresentados na Tabela 2.3, o decodificador será capaz de reconstruir os símbolos.

Tabela 2.3: Tabela de probabilidades e sub-faixas para os símbolos do exemplo

Símbolo	Probabilidade	Sub-faixa
-2	0,1	0 – 0,1
-1	0,2	0,1 – 0,3
0	0,4	0,3 – 0,7
1	0,2	0,7 – 0,9
2	0,1	0,9 – 1

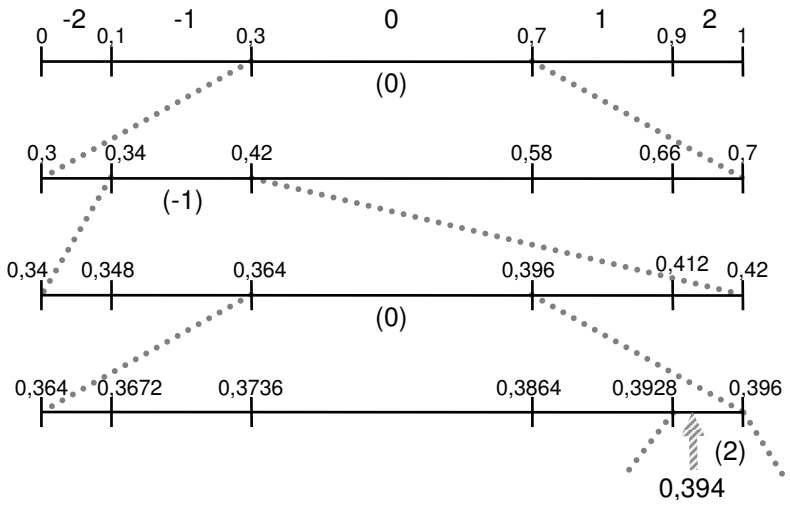


Figura 2.12: Exemplo de codificação aritmética

É importante destacar que o modelo probabilístico usado é independente do codificador aritmético. Isto faz com que possam ser usados modelos probabilísticos diferentes dependendo do estado em que o codificador se encontre. Por exemplo, as probabilidades dos símbolos em cada modelo podem ser atualizadas à medida que os símbolos vão sendo codificados/decodificados.

A codificação/decodificação aritmética requer operações aritméticas para gerar as faixas, o que faz com que a codificação aritmética seja bem mais complexa computacionalmente do que os códigos de comprimento variável. Esta complexidade

elevada deve ser tratada tanto pelo codificador quanto pelo decodificador H.264/AVC e é um custo adicional associado ao uso do CABAC. Por outro lado, os ganhos em eficiência de codificação com a utilização do CABAC são significativos. Novamente o compromisso entre complexidade e eficiência de codificação deve ser avaliado para a tomada de decisão de qual tipo de codificador será utilizado na codificação de entropia dos codecs H.264/AVC.

2.3.10 Modo de Decisão

A especificação do padrão H.264/AVC define apenas a sintaxe do *bitstream* e o processo de decodificação. O processo de codificação é deixado de fora do escopo do padrão para permitir maior flexibilidade para as implementações. Mas o modo de decisão é um problema chave para a compressão, pois ele determina quais as decisões de codificação serão tomadas para cada vídeo processado (WIEGAND, 2003).

No codificador H.264/AVC existem muitas decisões que devem ser tomadas pelo codificador, incluindo a definição do tipo de predição (intra-quadro ou inter-quadros), a definição do tamanho de bloco usado nas predições intra-quadro ou inter-quadros, a definição se o modo skip será usado ou não, a definição se será usada bi-predição, etc. Como são muitas as possibilidades é muito importante que as decisões tomadas sejam as melhores possíveis, pois com escolhas sub-ótimas, alguns dos benefícios propiciados pelo H.264/AVC podem ser perdidos (PURI, 2004).

O problema do modo de decisão é causado porque seqüências de vídeo típicas contêm grande variação de conteúdo e movimento, sendo necessário selecionar entre diferentes opções de codificação com variação na eficiência da relação taxa-distorção para diferentes partes do vídeo. A tarefa do controle do codificador é determinar um grupo de parâmetros de codificação de modo que um certo compromisso na relação taxa-distorção seja atingido (WIEGAND, 2003).

Mesmo que não sejam normatizadas pelo padrão H.264/AVC, técnicas de otimização taxa-distorção (WIEGAND, 2003; SULLIVAN, 1998) devem ser usadas para todas as decisões tomadas pelo codificador. Com este tipo de controle do codificador, significativos ganhos em termos de eficiência de codificação podem ser obtidos. Mas o uso deste tipo de critério não evita que todas as possíveis combinações de modos de codificação tenham que ser calculadas para que se possa atingir a escolha ótima. Entretanto, ao se fazer a escolha que minimiza o custo **J** de cada decisão, esta escolha vai tender a maximizar a eficiência do codificador.

O parâmetro $\boldsymbol{\lambda}$ é usado para controlar a taxa obtida e está relacionado com o passo de quantização **Qstep**. Para cada decisão, é medido o custo **J**, de acordo com (2.29).

$$J = D + \lambda \cdot R \qquad (2.29)$$

Em (2.29), **R** é a taxa de bits e **D** é a distorção. Neste caso, para $\boldsymbol{\lambda}$ fixo, é possível gerar um custo **J** para cada diferente predição.

3 A ARQUITETURA DE COMPENSAÇÃO DE MOVIMENTO - *HP422-MOCHA*

Esse capítulo é dedicado a apresentar a arquitetura HP422-MoCHA, uma arquitetura para compensação de movimento do padrão H.264/AVC com suporte às ferramentas do perfil *High 4:2:2* desenvolvida para decodificar, em tempo real, vídeos HDTV 1920x1080 à uma taxa de 30 quadros por segundo.

A seção 3.1 apresenta a compensação de movimento e todas suas ferramentas no padrão H.264/AVC. Na seção 3.2 é apresentada, em detalhes, a arquitetura HP422-MoCHA. Resultados de síntese e comparações com trabalhos relacionados aparecem na seção 3.3 seguidas pelo método de verificação apresentado na seção 3.4. A seção 3.5 traz as considerações finais a respeito da arquitetura proposta.

3.1 Compensação de Movimento (MC)

Esta seção apresenta a Compensação de movimento (MC) abordando também a Estimação de movimento (ME), uma vez que estes módulos estão fortemente relacionados.

A estimação de movimento está presente apenas no codificador e têm como função encontrar a região dos quadros de referência que mais se assemelha ao macrobloco atual. Este módulo é o que apresenta a maior complexidade computacional dentre todos os módulos de um codificador H.264/AVC (PURI, 2004). Este grande custo computacional é função das inovações inseridas neste módulo do padrão, que tiveram o objetivo de atingir elevadas taxas de compressão. Reside nos módulos da ME e MC as principais fontes de ganhos do H.264/AVC em relação aos demais padrões de compressão de vídeo (WIEGAND, 2003, RICHARDSON, 2003).

A estimação de movimento deve prover as ferramentas de codificação capazes de localizar, nos quadros de referência, qual macrobloco mais se assemelha ao macrobloco atual. Assim que é encontrado este macrobloco, a ME deve gerar um vetor indicando a posição deste macrobloco 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 o componente de luminância do macrobloco. Para sub-amostragem 4:2:0 as componentes de crominância possuem a metade da resolução horizontal e vertical do componente de luminância, então os componentes horizontal e vertical de cada vetor de movimento são divididos por dois para serem aplicados aos blocos de crominância. No caso de sub-

amostragem 4:2:2 apenas a dimensão horizontal do vetor é dividida por dois. O mesmo vetor da luminância é utilizado pra crominância quando utilizando amostragem 4:4:4.

A compensação de movimento deve ser capaz de, a partir dos vetores de movimento e dos quadros de referência, reconstruir a predição do quadro atual. Durante a fase de codificação, este quadro será subtraído do quadro atual para gerar o quadro de resíduos que passará pela transformada. Na decodificação, o resultado da compensação é somado aos resíduos gerados pela transformada inversa, para gerar o quadro atual.

O módulo de compensação de movimento deve ter as ferramentas necessárias para decodificar todos os modos de codificação utilizados pela ME.

3.1.1 Tamanho de blocos variável

A principal inovação do H.264/AVC está na possibilidade de utilização de tamanhos de blocos variáveis para realizar a compensação de movimento. Ao invés de usar um macrobloco inteiro na estimação de movimento, o padrão H.264/AVC 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 (WIEGAND, 2003a, RICHARDSON, 2003).

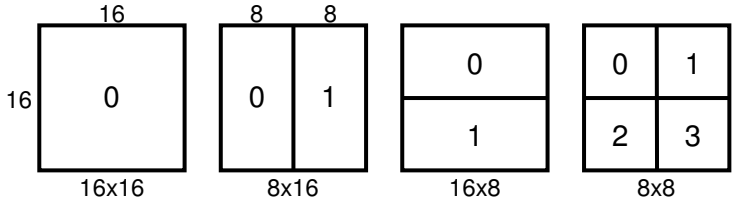


Figura 3.1: Divisão do macrobloco em partições de macroblocos

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 que podem ter o tamanho 8x8, 4x8, 8x4 ou 4x4, como está apresentado na Figura 3.2.

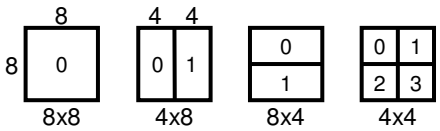


Figura 3.2: Divisão de uma partição de macrobloco em partições de sub-macroblocos

Cada bloco de crominância é dividido da mesma maneira que os blocos de luminância, exceto pelo tamanho da partição, que para subamostragem 4:2:0, terá exatamente a metade da resolução horizontal e vertical da partição de luminância. Por exemplo, uma partição de 8x16 de luminância corresponde a uma partição de 4x8 de crominância.

3.1.2 Múltiplos quadros de referência

Outra característica importante do padrão H.264/AVC é o uso de múltiplos quadros de referência. No padrão H.264/AVC 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 mostrado na Figura 3.3. 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/AVC 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.

O padrão H.264/AVC define um tipo de armazenamento especial nas listas, o quadro **long_term** (ITU-T, 2003). 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.

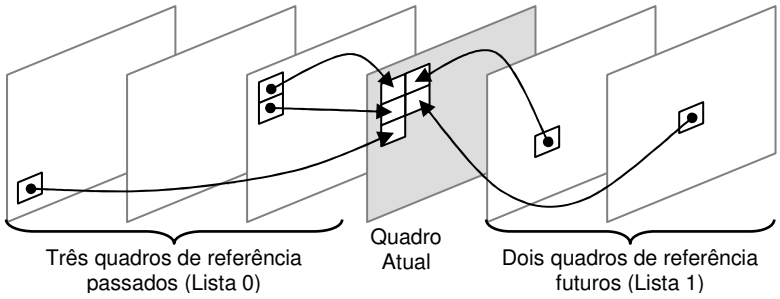


Figura 3.3: Uso de múltiplos quadros de referência.

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

3.1.3 Vetores apontando para fora das bordas do quadro

A possibilidade de utilizar vetores de movimento que apontam para fora dos limites dos quadros também faz parte do padrão H.264/AVC. Neste caso é realizada uma extrapolação dos quadros nas bordas. A Figura 3.4 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 vetor (vertical e horizontal) não pertencem às dimensões do quadro.

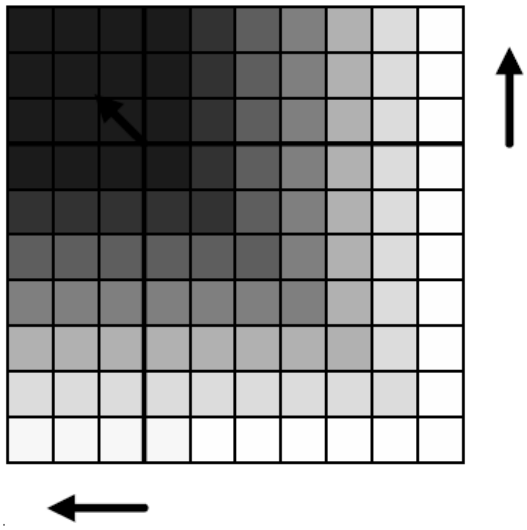


Figura 3.4: Extrapolação da borda superior esquerda.

3.1.4 Precisão de quarto de pixel

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

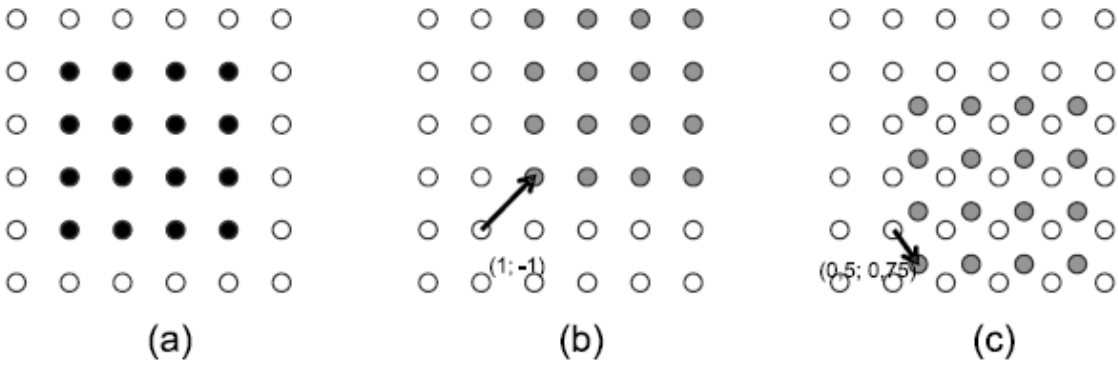


Figura 3.5: (a) quadro atual a ser predito; (b) predição com vetor de movimento inteiro com valores inteiros (1, -1); (c) vetor de movimento com valores fracionários (0,5 ; 0,75)

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

No H.264/AVC, a compensação de movimento usando ¼ de pixel é obrigatória. A interpolação para ¼ de pixel de pixel é realizada em dois passos. No primeiro passo, é realizada a interpolação dos quadros para ½ 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 ½ pixel por letras minúsculas.

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

$$b = (E - 5 \cdot F + 20 \cdot G + 20 \cdot H - 5 \cdot I + J)/32 \qquad (3.1)$$

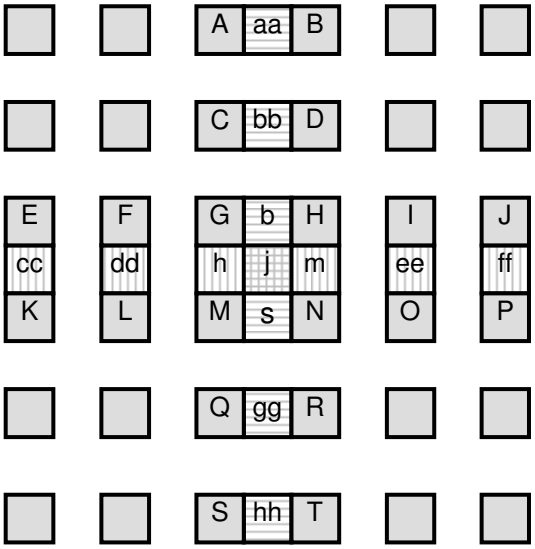


Figura 3.6: Interpolação para posições de ½ pixel para o componente de luminância.

No segundo passo, são interpoladas as posições de ¼ 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).

$$a = (G + b)/2 \qquad (3.2)$$

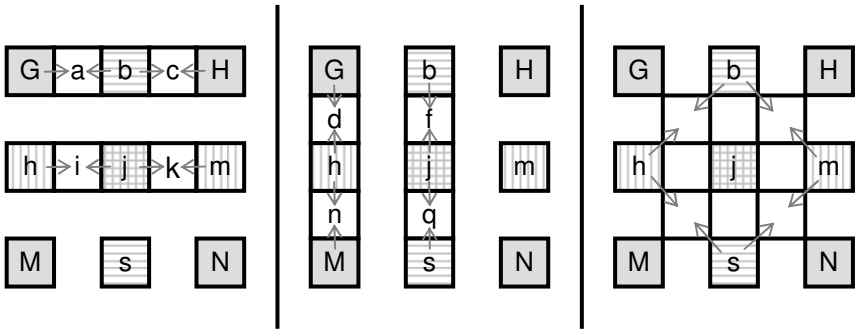


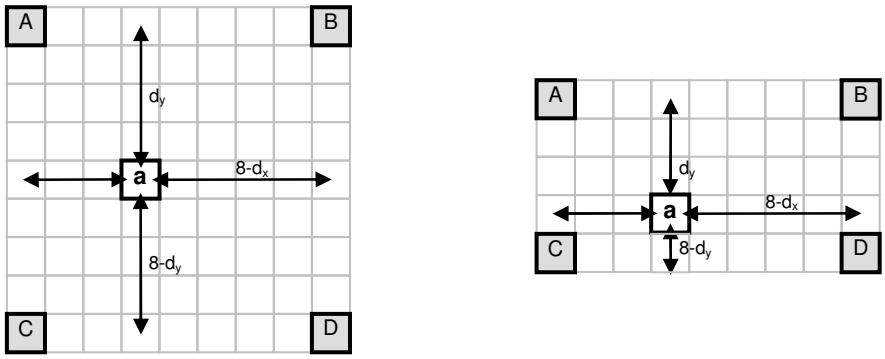
Figura 3.7: Interpolação para posições de ¼ de pixel

Para os vetores de crominância, que utilizam uma resolução de até 1/8 de pixel, usa-se interpolação linear. As amostras interpoladas são geradas em intervalos de oito amostras entre amostras inteiras se utilizando subamostragem e em intervalos de quatro amostras se a amostragem naquela dimensão for completa. 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 Figura 3.8(a) representa a interpolação para sub-amostragem de cores 4:2:0 enquanto a Figura 3.8(b) representa a interpolação para sub-amostragem de cores 4:2:2.

A combinação linear das amostras inteiras para 4:2:0 é definida pela equação (3.3) e para 4:2:2 pela equação (3.4).

$$a = round\left\{\left[(8-d_x)\cdot(8-d_y)\cdot A + d_x\cdot(8-d_y)\cdot B + (8-d_x)\cdot d_y\cdot C + d_x\cdot d_y\cdot D\right]/64\right\} \quad (3.3)$$

$$a = round\left\{\left[(8-d_x)\cdot(8-2d_y)\cdot A + d_x\cdot(8-2d_y)\cdot B + (8-d_x)\cdot 2d_y\cdot C + d_x\cdot 2d_y\cdot D\right]/64\right\} \quad (3.4)$$



(a) (b)

Figura 3.8: Interpolação para componentes de crominância em (a) sub-amostragem 4:2:0 e (b) sub-amostragem 4:2:2

3.1.5 Bi-predição e Predição Ponderada

Essa funcionalidade permite que um bloco seja predito a partir de dois quadros de referência, um quadro da **lista 0** e um quadro da **lista 1**.

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 módulo de predição é calculada como uma média entre as amostras dos quadros das listas 0 e 1. Este cálculo é apresentado em (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.

$$pred(i,j) = \lfloor (pred0(i,j) + pred1(i,j) + 1) >> 1 \rfloor \qquad (3.5)$$

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/AVC. 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 com base na distância relativa entre o quadro corrente e seus quadros de referência. 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 equação 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.

$$pred(i,j) = \left\lfloor ((pred0(i,j) * wX + 2^{\log WD - 1}) >> \log WD) + oX \right\rfloor \qquad (3.6)$$

Quando uma compensação do tipo bi-preditiva é aplicada, a predição ponderada utiliza a equação 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.

$$pred(i,j) = \left\lfloor ((pred0(i,j) * w0 + pred1(i,j) * w1 + 2^{\log WD}) >> (\log WD + 1)) + ((o0 + o1 + 1) >> 1) \right\rfloor \qquad (3.7)$$

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 predição ponderada é especialmente útil em *fades* nas seqüências de vídeos.

3.1.6 Predição Skip e Direta

O padrão H.264/AVC também aprimora o conceito de macrobloco *skip* em relação aos padrões anteriores. Diferente dos outros padrões, um macrobloco *skip* no padrão H.264/AVC usa uma predição de vetores de movimento, ao invés de simplesmente copiar o bloco co-localizado. O bloco co-localizado, no padrão H.264/AVC, 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 direta se divide em direta espacial e direta temporal, sendo que a direta espacial é uma inovação do H.264 enquanto a direta temporal é uma simplificação da predição encontrada no MPEG-4 parte 2. A predição direta utiliza informações presentes no quadro co-localizado que, segundo o padrão H.264, é o quadro de referência índice 0 da **lista 1**. Por bloco co-localizado entende-se ao bloco do quadro co-localizado que se encontra na mesma posição que o bloco corrente.

3.1.7 Predição de Vetores de Movimento

Devido à elevada correlação entre blocos vizinhos, o padrão H.264/AVC define que os vetores de movimento, assim como os índices dos quadros de referência, sejam inferidos com base nos vetores de movimento e índices de referência dos blocos vizinhos. Esses vetores são chamados de vetores de movimento preditos (*predictive motion vectors* - PMV) (ITU-T, 2003).

Desta forma, apenas os vetores de movimento diferenciais são codificados no *bitstream*, para depois serem somados aos PMVs, obtendo assim, os vetores de movimento do bloco atual. Os PMVs são normalmente obtidos aplicando a mediana aos vetores dos blocos vizinhos. No entanto, macroblocos *skip* e macroblocos que sofrem predição direta são tratados de maneiras diferentes.

Os vetores de movimento calculados apresentam precisão de ¼ de amostra e são utilizados para interpolação de luminância. A partir desses vetores são derivados os vetores de movimento para crominância com precisão de até 1/8 de amostra.

A ordem de predição segue a ordem de duplo Z como ilustra a Figura 3.9.

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

Figura 3.9: Ordem de predição de vetores no MB

No H.264/AVC, o bloco corrente, seja ele uma partição ou sub-partição, é denominado bloco ‘E’. As partições ou sub-partições contíguas ao bloco corrente são nomeadas com as letras ‘A’, ’B’, ’C’ e ’D’. Sendo ‘A’ o bloco localizado à esquerda de ‘E’, ‘B’ é o bloco localizado imediatamente acima de ‘E’, ‘C’ é a partição acima e à direita de ‘E’ e finalmente, ‘D’ se localiza acima e a esquerda do bloco corrente ‘E’. A Figura 3.10 exemplifica essa relação entre os blocos vizinho.

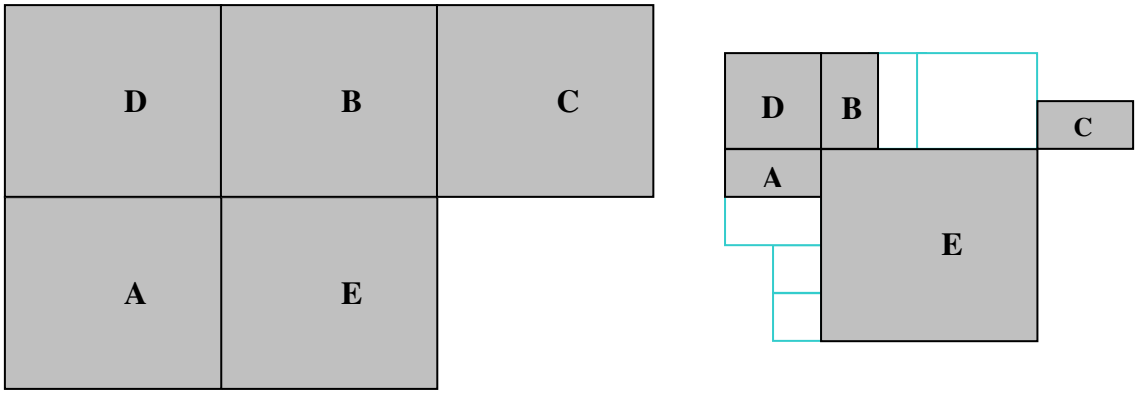


Figura 3.10: Relação entre blocos vizinho (a) em formatos idênticos e (b) em formatos diferentes

Partições já processadas dentro do macrobloco corrente também são consideradas partições vizinhas e, portanto, são consideradas na predição das demais partições.

A operação padrão para o cálculo do PMV é a mediana aplicada a três vetores de movimento dos blocos vizinhos. Normalmente são utilizados os vetores dos blocos ‘A’, ‘B’ e ‘C’. Caso ‘C’ não esteja disponível, é utilizado o vetor do bloco ‘D’. No caso de apenas um dos blocos ter a mesma referência do bloco corrente, seu vetor é setado como sendo o PMV. Este tipo de predição pode ser aplicado tanto a macroblocos do tipo P quanto do tipo B.

Além da predição padrão, existem ainda dois tipos de predição direta, a predição direta espacial e a predição direta temporal. As predições diretas apenas são utilizadas em quadros do tipo B.

Na predição direta espacial os vetores são derivados considerando os vetores de movimento do bloco co-localizado, além dos vetores de blocos vizinhos como na predição padrão.

A predição direta temporal utiliza a referência do bloco co-localizado para calcular as referências do bloco atual. De acordo com a distância entre as referências e o quadro corrente, aplica-se um fator de ponderação aos vetores do bloco co-localizado para se obter os vetores do bloco corrente.

Macroblocos *skip* são preditos de acordo com o tipo de *slice* ao qual pertencem. Em um slice tipo P os vetores do macrobloco *skip* podem assumir o valor zero ou ainda podem ser tratados pela predição padrão. Já em *slices* tipo B, esses macroblocos são preditos pela predição direta, espacial ou temporal de acordo com o modo utilizado na codificação.

3.2 Arquitetura HP422-MoCHA

Esse capítulo apresenta a arquitetura do compensador de movimento para o perfil High 422 do padrão H.264/AVC denominada HP422-MoCHA. Esse nome de deve ao fato de se tratar de uma extensão da arquitetura MoCHA (AZEVEDO, 2006) que suporta o Perfil Main do H.264/AVC.

A arquitetura foi projetada para alcançar taxa de processamento suficiente para decodificar vídeos de alta definição (HDTV), no formato de até 1920x1080, 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 HP422-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 um quarto de pixel, a predição ponderada e a média dos blocos preditos, para o caso da bi-predição. A Figura 3.11 apresenta o modelo do compensador seguido nessa dissertação.

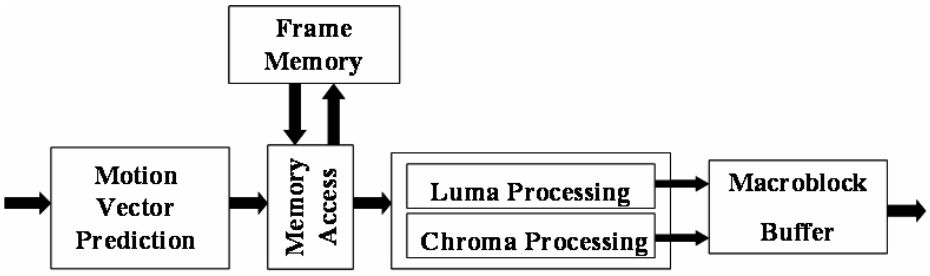


Figura 3.11: MoCHA Architecture

A arquitetura 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 *bitsream*, 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. 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/AVC.

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 Xilinx (XILINX: THE PROGRAMMABLE LOGIC COMPANY, 2008). A família de dispositivos escolhida foi a Virtex II Pro (VIRTEX SERIES, 2008), 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, 2008), tendo como sintetizador o Symplify Pro (SYNPLICITY: PRODUCTS: SYNPLIFY PRO, 2008) da Synplicity (SYNPLICITY: HOME, 2008). 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 exceção da síntese da arquitetura completa.

O compensador de movimento para o decodificador de vídeo H.264/AVC 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 mapeiam 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 cells*. 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 presente versão da arquitetura possui uma limitação no que se refere à implementação completa das funcionalidades do padrão H.264/AVC no perfil High. A 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.

Este capítulo está dividido em quatro seções. A primeira seção apresenta o bloco preditor de vetores de movimento, a segunda seção apresenta o processador de amostras, a terceira seção apresenta o bloco de acesso à memória e por fim a última seção apresenta os resultados e comparações com trabalhos relacionados.

3.2.1 Preditor de Vetores de Movimento (MVP)

Essa seção irá apresentar a arquitetura e a descrição do módulo de hardware responsável pela predição dos vetores de movimento. A linguagem utilizada para a descrição da arquitetura foi o VHDL.

Com base em software desenvolvido em C e no conhecimento adquirido sobre o processo da predição de vetores, passou-se para a descrição do hardware para a predição dos vetores de movimento. A exemplo do software, o hardware também foi desenvolvido de maneira incremental, seguindo três passos. Primeiramente foi desenvolvida a arquitetura para predição padrão para *slices* tipo P, seguido da descrição para predição direta espacial e, finalmente, o desenvolvimento da predição direta temporal. A predição de vetores para vídeos entrelaçados não foi abordada nesse trabalho.

A modelagem do MVP em hardware foi realizada usando, por base, máquinas de estados finitos (FSM), uma vez que o algoritmo para predição de vetores de movimento se mostra extremamente seqüencial.

A arquitetura foi descrita em VHDL utilizando uma descrição mista entre VHDL estrutural e comportamental. Essa abordagem foi tomada para permitir uma implementação mais eficaz das estruturas utilizadas em C++ e para propiciar ao sintetizador maior liberdade no reuso de componentes de hardware utilizados.

A predição padrão para *slices* P, como descrito anteriormente, começa pela verificação da disponibilidade dos vizinhos selecionando seus vetores de movimento. É, então, obtida a mediana dos blocos vizinhos e esta é somada ao vetor de movimento diferencial. Essa soma dá origem aos MVs da partição ou sub-partição corrente.

Para a realização deste cálculo, é necessário que todos macroblocos da linha superior sejam armazenados pela arquitetura. Foram então descritos dois *buffers* em memória, um para armazenar os vetores e outro para armazenar os índices de referência dos blocos vizinhos. Como essa implementação visa decodificar HDTV com 1920 pixels de largura, o buffer que armazena os vetores de movimento deve armazenar 480 vetores, para isso foi dimensionado em 512 palavras de 32 bits. O *buffer* de índices de referência deve armazenar 240 índices e, portanto, foi dimensionado em 60 palavras de 32 bits. A Figura 3.12 representa todos os blocos cujas informações devem ser armazenadas.

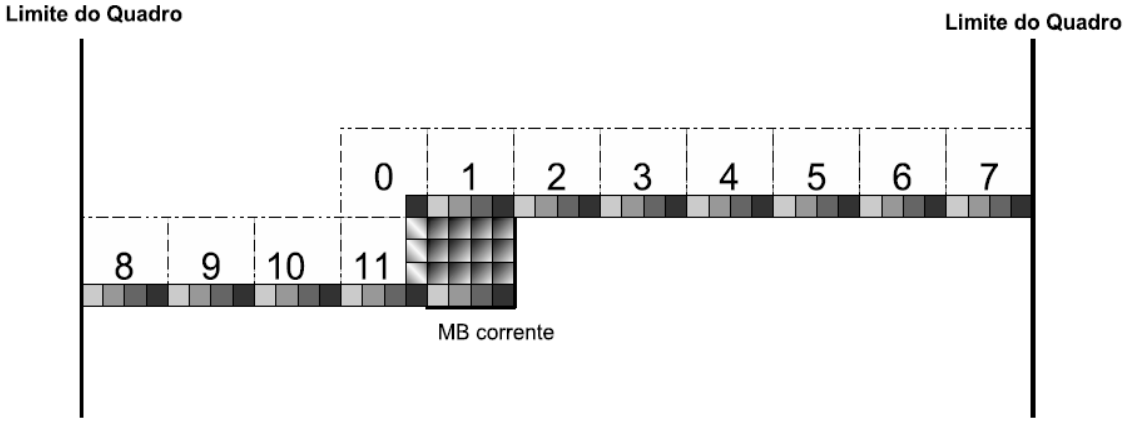


Figura 3.12: Blocos necessários ao cálculo do MVP

A arquitetura também conta com 10 conjuntos de registradores para o armazenamento dos vetores vizinhos e 16 conjuntos para armazenamento de vetores internos do macrobloco corrente. São tratados como conjuntos de registradores por armazenarem informação da lista 0 e da lista 1. A Figura 3.13 representa a disposição dos conjuntos de registradores que armazenam os vetores de movimento. As caixas com letras, na Figura 3.13, representam os vetores de movimento dos blocos vizinhos. As caixas com fundo em cinza representam os vetores de movimento do próprio macrobloco. Dois conjuntos com 4 registradores armazenam os vetores de movimento imediatamente à esquerda e acima do macrobloco corrente. Esses vetores são referentes aos vizinhos A e B do macrobloco corrente, respectivamente (A e B na Figura 3.13). Dois registradores armazenam os vetores do bloco vizinho superior esquerdo e do bloco superior direito, referentes aos vizinhos D e C, respectivamente (D e C na Figura 3.13). Um conjunto de 16 registradores armazena os vetores de movimento dos blocos do macrobloco corrente (0 a 15 na Figura 3.13).

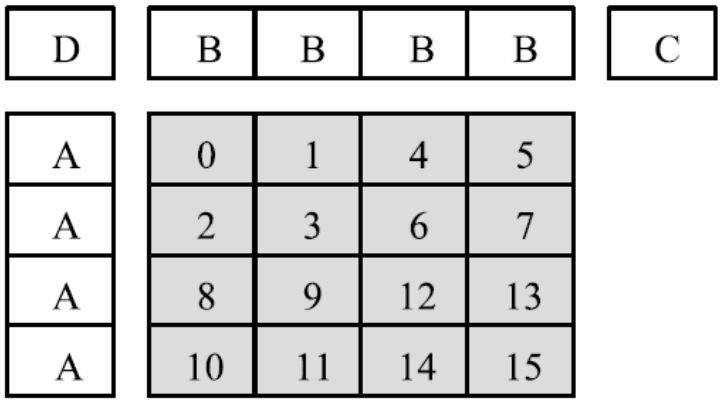


Figura 3.13: Estrutura de registradores para armazenamento de vetores internos ao macrobloco (0 a 15) e vetores de macroblocos vizinhos (A, B, C e D)

Os outros cinco conjuntos de registradores armazenam os índices de referência dos blocos vizinhos. Como cada partição de macrobloco (de até 8x8) compartilha as mesmas referências, são necessários menos registradores para armazenar os índices de referência dos blocos vizinhos, tal qual representado na Figura 3.14.

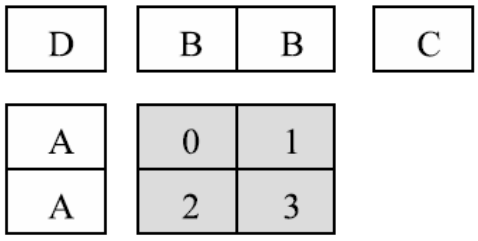


Figura 3.14: Estrutura de registradores para armazenamento de índices de referência internos ao macrobloco (0 a 3) e índices de referência de macroblocos vizinhos (A, B, C e D)

As operações sobre esses conjuntos de registradores foram descritas na forma de uma máquina de estados. Essa máquina de *Mealy* deve processar um macrobloco em, no máximo, 384 ciclos de relógio. Cada macrobloco pode ser dividido em até 16 blocos 4x4, que são processados de forma serial pela máquina de estados, para tanto, cada bloco deve ser processado em até 24 ciclos de relógio. A Figura 3.15 apresenta o diagrama da máquina de estados hierárquica do preditor de vetores de movimento. Cada macroestado, destacado com preenchimento na Figura 3.15, será desdobrado e seus estados serão detalhados ao longo desse capítulo.

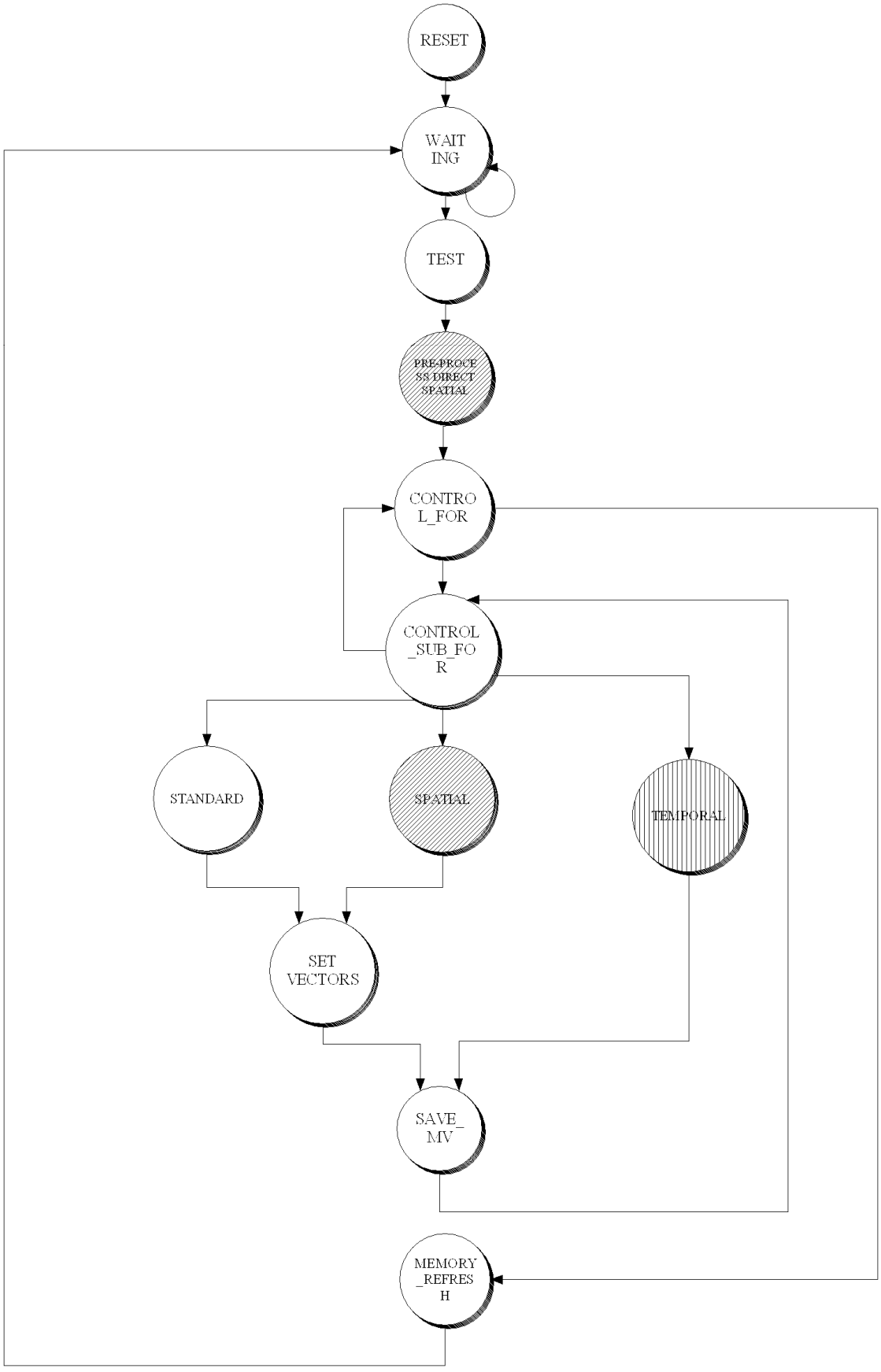


Figura 3.15: Diagrama de estados da FSM do MVP

3.2.1.1 Preditor Com Suporte à Predição Direta Temporal

Tomando por base a o MVP utilizado para o MoCHA original, desenvolveu-se arquitetura que suportasse também a predição direta temporal, fez-se a inserção de um novo caminho de dados dentro dos laços principais, além de um modulo para o cálculo do fator de escala.

Foram inseridos seis estados adicionais para o tratamento da predição direta temporal. Esses estados são representados na Figura 3.15 pelo macroestado TEMPORAL (grifado com linhas verticais) e apresentados na Figura 3.16.

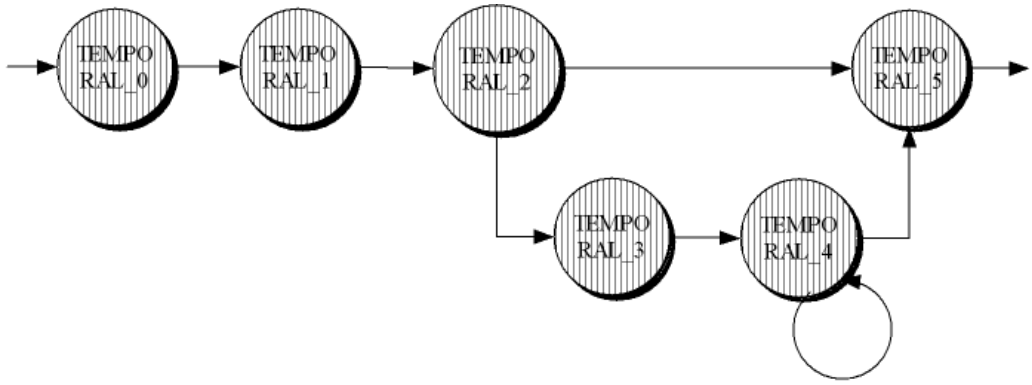


Figura 3.16: Diagrama de estados do macroestado TEMPORAL

O estado TEMPORAL_0 se encarrega de calcular a posição do bloco co-localizado a ser lido pelo estado seguinte, o estado TEMPORAL_1.

Como o fator de escala, **DistScaleFactor**, é calculado com base na distância dos quadros de referência e do quadro atual e cada partição pode referenciar quadros diferentes, esse fator deve ser calculado apenas uma vez a cada partição. Ou seja, não é necessário calcular o fator de escala para cada sub-partição. Como esse cálculo está dentro do laço de sub-partição, é realizado um teste para saber esse calculo já foi realizado antes, evitando a repetição do mesmo. É no estado TEMPORAL_2 que esse teste é feito. Quando é necessário calcular o fator de escala, o fluxo de dados segue para o estado TEMPORAL_3, senão, é desviado para o estado TEMPORAL_5.

Também no estado TEMPORAL_2 é selecionado o índice de referência do bloco co-localizado. Esse índice é enviado ao gerenciador de listas para que este, por sua vez, devolva o índice de referência re-mapeado, além dos valores de POC para os quadros de referência e o quadro atual. Esses valores são necessários para a obtenção do fator de escala.

Foi desenvolvida uma arquitetura para realizar o cálculo do fator de escala. Essa arquitetura será detalhada nos próximos parágrafos. O estado TEMPORAL_3 tem a função de disparar o cálculo, enquanto o estado TEMPORAL_4 entra em *loop* esperando que o fator de escala tenha sido determinado.

O último estado inserido para processamento da predição direta temporal, o estado TEMPORAL_5, efetua o cálculo dos vetores de movimento. Vetores da **lista 0** são calculados através da soma da constante ‘128’ ao produto entre fator de escala e vetores

co-localizados. Vetores da **lista 1** são encontrados subtraindo os vetores co-localizados dos vetores da **lista 0**.

Como citado acima, foi criada uma arquitetura especificamente para calcular o fator de escala utilizado na predição direta temporal. Essa arquitetura, apresentada na Figura 3.17, foi implementada em um *pipeline* de 3 estágios. O primeiro estágio é composto por dois subtratores de 18 bits, dois operadores de *clip* (utilizados para manter as saídas dos subtratores no intervalo [0, 255]), e um somador que adiciona a constante 16384 ao resultado de uma das subtrações. No segundo estágio existe um divisor de 16 bits. O último estágio conta com um multiplicador de 18 bits, um somador com a constante 32, um deslocador (*shifter*) e um operador de *clip*.

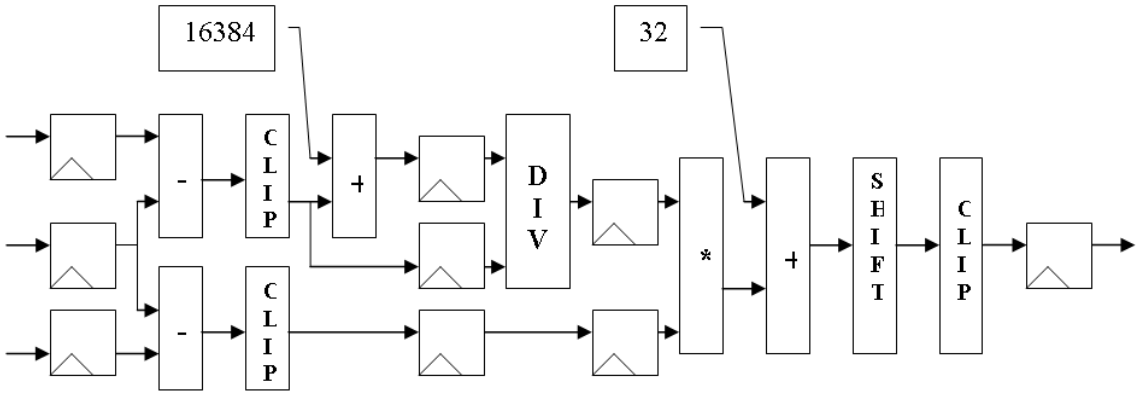


Figura 3.17: Caminho de dados para o cálculo do fator de escala

A inserção da predição direta temporal concluiu a arquitetura proposta do preditor de vetores de movimento. Essa arquitetura é capaz de tratar tanto a predição padrão quanto os dois tipos de predição direta. Essa arquitetura tem latência mínima de 17 ciclos e uma latência máxima de 226 ciclos. Somando-se os 16 ciclos finais para envio de resultados, a arquitetura completa gasta, no máximo, 242 ciclos para processar um macrobloco, mantendo-se dentro dos 384 ciclos disponíveis para esse processo.

O preditor de vetores, em sua versão final, conta com 28 sinais de entrada e 23 sinais de saída.

De acordo com os resultados de síntese do preditor de vetores, este atingiu uma freqüência máxima de operação de 133 MHz. A Tabela 3.1 mostra os dados de síntese e de ocupação do dispositivo Xilinx Virtex-II PRO VP30.

Tabela 3.1: Síntese do MVP com Suporte a *Slices* B e Predição Direta Espacial e Temporal

	HP422-MoCHA MVPr	
Slices	3.494	33%
Flip Flops	3.358	16%
LUTs	5.961	18%
BRAM	3	2%
Multipliers	0	0%

Como o número de ciclos gastos para processar cada macrobloco varia de acordo com o tipo do macrobloco e do *slice* ao qual pertence, optou-se por avaliar o desempenho do preditor por meio de simulação. A arquitetura foi testada utilizando como entrada os dados extraídos do vídeo "Foreman.yuv" e o número médio de ciclos gasto no processamento de um macrobloco foi obtido.

Para 100 frames codificados apenas com *slices* P, o preditor apresentou uma média de 64 ciclos de relógio para processar um macrobloco. Esse número fica bastante abaixo do pior caso que é de 236 ciclos.

Quando o mesmo vídeo é codificado com uma seqüência de quadros I-P-B-B-P utilizando predição direta espacial, o número médio de ciclos aumenta bastante atingindo 127 ciclos, enquanto o pior caso sobe para 242 ciclos. Se forem considerados somente os quadros B tem-se uma média de aproximadamente 161 ciclos para decodificar um macrobloco.

Mantendo a seqüência de quadros I-P-B-B-P, porém utilizando predição direta temporal, o número médio fica em cerca de 88 ciclos por macrobloco. Considerando apenas quadros tipo B com predição direta temporal, o número médio será de 100 ciclos de relógio para processar cada macrobloco.

Todos os números médios foram arredondados e os valores obtidos podem ser alterados de acordo com o vídeo de entrada ou mesmo com o codificador utilizado. Neste experimento foi utilizado o codificador do *software* de referência do padrão H.264/AVC para gerar os dados de estímulo da arquitetura.

A Tabela 3.2 agrupa os dados analisados acima e mostra, para cada tipo de quadro, o número de ciclos gastos para processar um macrobloco no pior caso e no caso médio. Também é apresentada uma estimativa de quantos quadros em HDTV podem ser decodificados, por segundo, pela arquitetura de predição de vetores, considerando pior caso e caso médio. Esses dados supõem uma freqüência de operação de 133 MHz (freqüência máxima de operação do módulo).

Tabela 3.2: Taxa de processamento do MVP para HDTV

Seqüência de Quadros	**Pior caso (ciclos)**	**Nº de quadros HDTV no pior caso (quadros/s)**	**Caso médio (ciclos)**	**Nº de quadros HDTV no caso médio(quadros/s)**
P	236	69	64	254
B c/ pred. direta espacial	242	67	161	101
B c/ pred. direta temporal	236	69	100	162
I-P-B-B-P c/ direta espacial	242	67	127	128
I-P-B-B-P c/ direta temporal	236	69	88	185

A arquitetura de preditor de vetores de movimento desenvolvida neste trabalho apresenta um desempenho superior ao necessário para decodificar quadros HDTV no padrão H.264/AVC em tempo real.

Não foi encontrada, na literatura nenhuma implementação completa desse módulo em hardware para o perfil *Main* ou *High* do padrão H.264/AVC. Em (WANG, 2003), esse módulo do compensador de movimento foi implementado em software. Em (DENG, 2004) nada é afirmado a respeito do MVP. No artigo (CHEN, 2004), onde é apresentada uma arquitetura para ME, é implementado um preditor de vetores incompleto. No artigo (WANG, 2005b) é apresentada uma arquitetura completa para MC, no entanto essa implementação considera apenas o perfil *Baseline*. Em (LI, 2006) nada é afirmado com relação a completude do preditor de vetores de movimento desenvolvido. No trabalho apresentado em (CHEN, 2006) um software MVP é utilizado enquanto em (LI, 2007) esse módulo não foi implementado.

3.2.2 Processador de 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 saída é de 93 milhões de amostras por segundo com sub-amostragem 4:2:0 e 124 milhões de amostras por segundo com sub-amostragem 4:2:2. Essa taxa de saída deve ser alcançada com um relógio a 100 MHz.

Esse objetivo exige a interpolação de 244,8 mil MBs por segundo sendo que se considerarmos bi-predição, o interpolator precisa processar o dobro, 489,6 mil MBs por segundo. Esses requisitos de desempenho exigem uma grande exploração de paralelismo.

Para a filtragem de luminância, essa solução usa, para calcular a amostras de meio pixel, uma nova organização de filtros baseada na propriedade da separabilidade onde o filtro 2D é separado em filtros unidimensionais, verticais e horizontais. Enquanto a versão original do MoCHA, (AZEVEDO, 2005) e (AZEVEDO, 2007), utilizava uma organização de filtros baseada em (WANG, 2005b), o HP-422 MoCHA utiliza uma nova estruturas de filtros.

O interpolador foi definido para operar sobre blocos de 4x4 amostras, que é o menor tamanho de bloco definido pelo H.264/AVC. Esta definição foi feita com o objetivo de poupar recursos de hardware e para simplificar o controle do interpolator uma vez em cada macrobloco sempre pode ser facilmente quebrado em blocos 4x4. Enquanto no interpolador de luma os dados são processados em blocos de 4x4 amostras no interpolador de croma os dados são processados em blocos de 2x2 amostras quando utilizando sub-amostragem 4:2:0 ou blocos de 4x2 amostras quando utilizando sub-amostragem 4:2:2.

Um novo filtro de luminância foi desenvolvido visando reduzir os recursos de hardware mantendo o desempenho equivalente às outras implementações encontradas na literatura e ao filtro implementado na MoCHA original. As amostras de meio pixel

são geradas utilizando um filtro FIR bidirecional de 6 *taps* enquanto as amostras de quarto de pixel são geradas com uma interpolação bilinear simples. Na organização de filtros proposta, quatro filtros verticais e oito horizontais geram as amostras de meio pixel e quatro filtros bilineares geram as amostras de 1/4 de pixel. A arquitetura gera 4 amostras por ciclo com *pipeline* cheio. A organização dos filtros é apresentada na Figura 3.18.

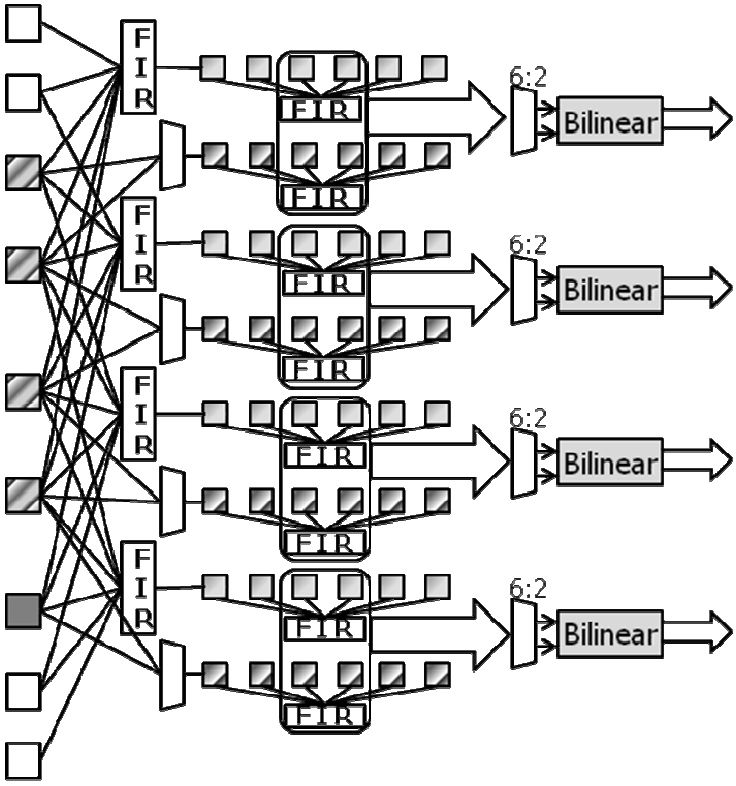


Figura 3.18: Interpolador de Luminância

Nove amostras inteiras [0..8] são recebidas por ciclo de relógio e processadas pelos quatro filtros verticais (representados PR caixas verticais com o texto 'FIR'), como mostrado na Figura 4.8. O primeiro filtro mais acima aplica a interpolação sobre as amostras [0..5], o segundo sobre as amostras [1..6] enquanto o terceiro e quarto filtros sobre as amostras [2..7] e [3..8] respectivamente. Esse quatro filtros operam completamente em paralelo e permitem q toda a filtragem 1-D seja feita em apenas um ciclo. Obviamente, essa interpolação é desnecessária se a interpolação vertical não seja exigida.

As amostras interpoladas verticalmente são, então, enviadas para uma cadeia de registradores que deslocam os dados organizando-os para a interpolação horizontal. As amostras inteiras não interpoladas [2, 3, 4, 5, 6] são direcionadas a quatro outras cadeias de registradores de deslocamento. De acordo com o vetor de movimento, as amostras são selecionadas para entrar nas cadeias de deslocamento e, conseqüentemente, serem interpoladas horizontalmente, podendo ser [2, 3, 4, 5] ou [3, 4, 5, 6] as amostras interpoladas. Isso reduz o número de filtros horizontais de nove para oito. Para controlar a seleção, são usados quatro multiplexadores 2:1 antes das cadeias de registradores responsáveis por deslocar amostras inteiras.

Depois de aplicar a interpolação horizontal, as amostras adequadas são filtradas pelos quatro filtros bilineares. Nesse momento a informação dos vetores de movimento é novamente utilizada para decidir quais dentre as seis amostras candidatas passarão pelos filtros bilineares. As amostras candidatas são representadas na Figura 4.8 pelas caixas conectadas às setas que apontam para uma lógica de seleção 6:2. Essa lógica seleciona duas dentre as seis amostras candidatas.

Os filtros FIR de 6 *taps* tem coeficientes (1, -5, 20, 20, -5, 1) conforme equação 3.8. Para implementação em hardware algumas simplificações podem ser feitas visando reduzir o consumo de recursos de hardware. A equação 3.9 representa a equação otimizada para implementação em hardware. Multiplicações e divisões por potência de dois são implementadas através de deslocamentos para esquerda e para a direita, respectivamente.

$$Y = (E - 5F + 20G + 20H - 5I + j)/32 \qquad (3.8)$$

$$Y = \{(E+J)+(G+H)+4[4(G+H)-(F+I)]+[4(G+H)-(F+I)]\}/32 \quad (3.9)$$

Todos os filtros FIR utilizados nessa arquitetura seguem a organização interna representada na Figura 3.19, embora possam ter diferentes larguras de dados.

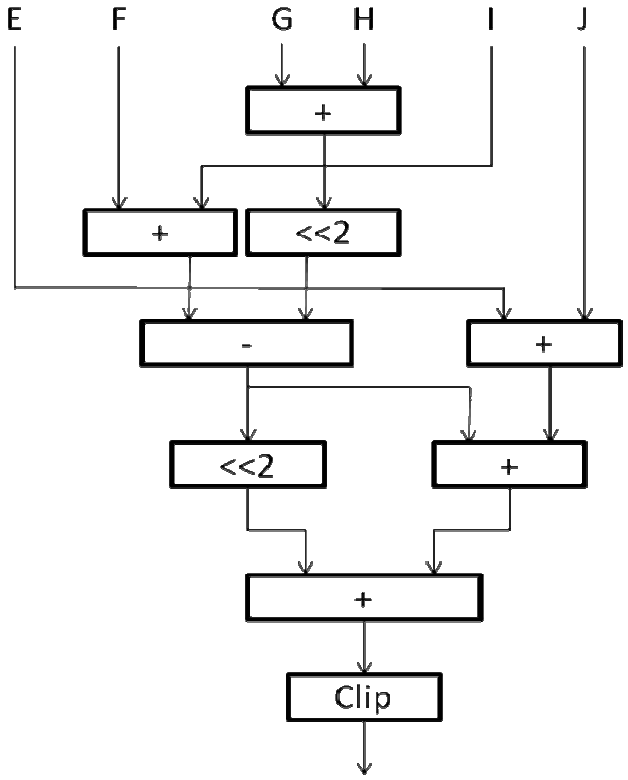


Figura 3.19: Organização do Filtro FIR 6-taps

Os filtros são compostos por cinco somadores, um subtrator e dois deslocadores. Ao fim, o resultado passa por uma unidade de *clipping*, operação também conhecida como saturação, responsável por manter o valor de saída dentro dos valores limite.

O módulo de *clipping* é composto por três blocos (Figura 3.20) onde cada um deles é responsável pela saturação para uma diferente largura de amostra. Primeiramente, todas amostras passam pelo bloco clip10 que limita o valor entre [0..1023]. Depois disso, essa amostra é enviada para o clip9 e clip8 que limitam os valores [0..511] e

[0..255], respectivamente. Por fim, um multiplexador seleciona entre uma das três amostras geradas tendo como valor de seleção a largura de amostra para o vídeo a ser decodificado no momento.

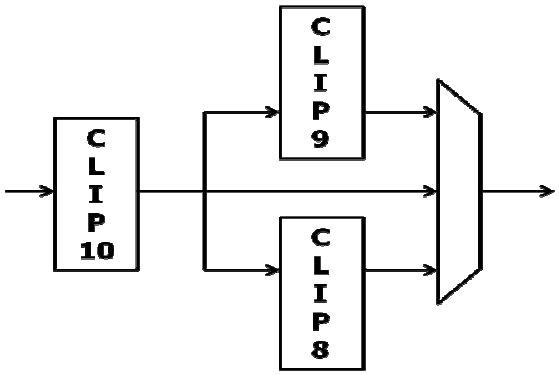


Figura 3.20: Unidade de Clipping

A latência do interpolador de luminância é de 7 ciclos e sua vazão máxima processa um bloco 4x4 em 9 ciclos e um bloco 4x4 bi-preditivo em 18 ciclos.

O interpolador de crominância é um módulo menos complexo computacionalmente que o interpolador de luminância, devido à simplicidade da filtragem bilinear com precisão de 1/8 de pixel. No entanto o controle complica um pouco devido à necessidade de suportar mais de um formato de sub-amostragem de cores. Esse módulo trabalha sobre blocos de 2x2 amostras se a sub-amostragem for 4:2:0 ou blocos de 4x2 para sub-amostragem 4:2:2.

A arquitetura usa um único caminho de dados para processar os dois canais de cores (Cb e Cr) serialmente. O interpolador é composto por dois filtros bilineares organizados conforme mostra Figura 3.21. O filtro recebe duas amostras (A e B) e as coordenadas X e Y do vetor de movimento.

A latência para processar um bloco preditivo é de 4 ciclos. A vazão máxima é de um blcoco 4x4 a cada 5 ciclos. Um bloco Cb e um Cr processados serialmente gastam 11 ciclos de relógio.

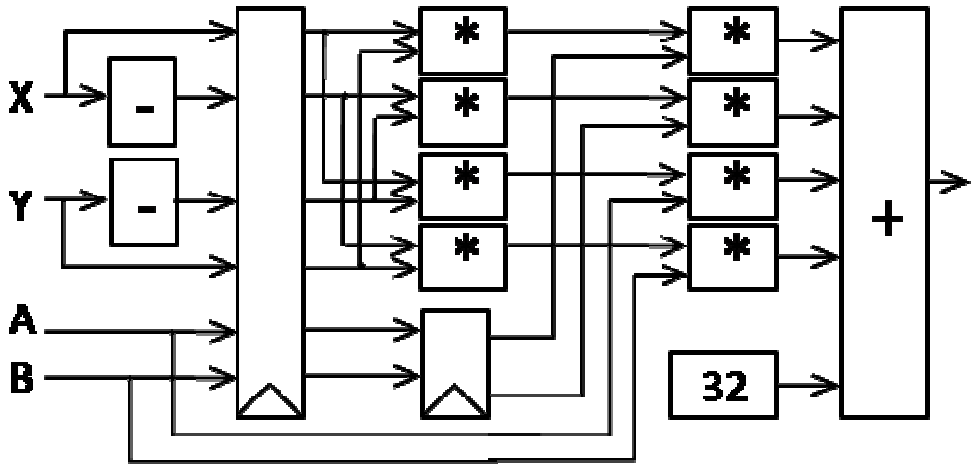


Figura 3.21: Filtro de Crominância

3.2.2.1 Processador de Amostras no Perfil Main

Ao desenvolver uma nova estrutura para o processador de amostras, foi possível, como mencionado anteriormente, reduzir um filtro FIR vertical com relação à organização anterior do filtro de luminância. Para fazer uma comparação justa, essa nova arquitetura foi descrita para, desta vez, para suportar apenas as ferramentas do perfil *Main.*

Uma síntese para ASIC foi feita utilizando a ferramenta *Leonardo Spectrum* (MENTOR, 2008) possibilitando a comparação da área utilizada pela arquitetura proposta com as demais soluções presentes na literatura. Os resultados de síntese gerados com a tecnologia TSMC 0.18 µm (TSMC, 2008) estão apresentados na Tabela 3.3. Esses resultados não incluem dados de memória.

Tabela 3.3: Resultados de Síntese para Standard Cells TSMC 0.18 µm

	Interpolador Chroma	**Interpolador Luma**	**Total**
Num. de Gates	3.980	8.365	12.345
Frequencia	1696,9 MHz	129,9 MHz	129,9 MHz

A Tabela 3.4 apresenta um resumo das arquiteturas comparáveis, listando os perfis em que opera, o número e tipos dos filtros, o número de g*ates* utilizados, o número de ciclos para processar um MB e a mínima freqüência necessária para processar vídeos HDTV 1080p.

Tabela 3.4: Resultados Comparativos com outras Arquiteturas de Interpolação

	Wang'03	**Chen'04**	**Wang'05**	**Chen'06**	**Azevedo'07**	**Li'07**	**Proposto**
Perfil	*Baseline*	*Baseline*	*Baseline*	*Main*	*Main*	*Baseline*	*Main*
Arquitetura	1-D	Separada 1-D	Separada 1-D	Separada 1-D	Separada 1-D	Separada 1-D	Separada 1-D
Componentes do Interpolador	FIR x 2	Horiz. FIR x5	Horiz. FIR x9	Horiz. FIR x4 (bilineares embarcados)	Horiz. FIR x9	FIR x4	Horiz. FIR x8
		Vert. FIR x11	Vert. FIR x4	Vert. FIR x8 (bilineares embarcados)	Vert. FIR x4		Vert. FIR x4
	Bilinear	Bilinear	Bilinear x4	Zero (12 filtros bilineares embarcados)	Bilinear x4	Bilinear x4	Bilinear x4
	1/8 filtro x3	n/a	1/8 filtro x2	n/a	1/8 filtro	none	1/8 filtro
Num. de *gates* do Interpolador	11,172	23,872 (0.25um)	20,686 (0.18um)	15,000 (0.18um)	18,846 (0.18um)	13,027 (0.18um)	**12,345** (0.18um)
Ciclos para Interpolação	1280 ciclos/MB (pior caso)	n/a	560 ciclos/MB (pior caso)	320 ciclos/MB (caso médio)	304 ciclos/MB (pior caso)	304 ciclos/MB (pior caso)	**304 ciclos/MB** (pior caso)
Freq. requerida p/ HD 1080	n/a	n/a	100MHz	87MHz	82MHz	100MHz	**82MHz**

A Tabela 3.4 apresenta comparações entre o interpolador proposto e outros trabalhos encontrados na literatura. Os trabalhos (WANG, 2003), (WANG, 2005b), (CHEN, 2004) e (LI, 2007) apresentam soluções para o perfil *Baseline* que exige menor performance. Quando comparando com trabalhos para o perfil *Main*, o interpolador proposto reduz em 17% o número de *gates* com relação à (CHEN, 2006), utilizando menor número de ciclos para processar um MB e, portanto, exige uma freqüência de operação inferior para decodificar HDTV 1080p. Esse trabalho ainda reduz 34% de hardware sem nenhuma perda de desempenho se comparado ao trabalho proposto na arquitetura MoCHA original (AZEVEDO, 2007).

Quanto ao número de filtros, o interpolador proposto reduz um filtro FIR se comparado à (WANG, 2005) e à (AZEVEDO, 2007). Também reduz oito filtros bilineares em relação ao trabalho em (CHEN, 2006) que, embora não apresente filtros bilineares explícitos, eles estão embarcados nos filtros FIR. O trabalho de (LI, 2007) utiliza menos filtros mas processa apenas vídeos *Baseline*, ou seja, não suporta bi-predição.

O interpolador e os resultados apresentados nessa sub-seção foram tema de um artigo apresentado no SBCCI 2008, chamado “High Throughput Architecture for H.264/AVC Motion Compensation Sample Interpolator for HDTV” (ZATT,2008a), com autoria de Bruno Zatt, Luciano Agostini, Altamiro Susin e Sergio Bampi.

3.2.3 Acesso à Memória

Foi definida uma memória cache tridimensional, conforme apresentado na Figura 3.22, 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.

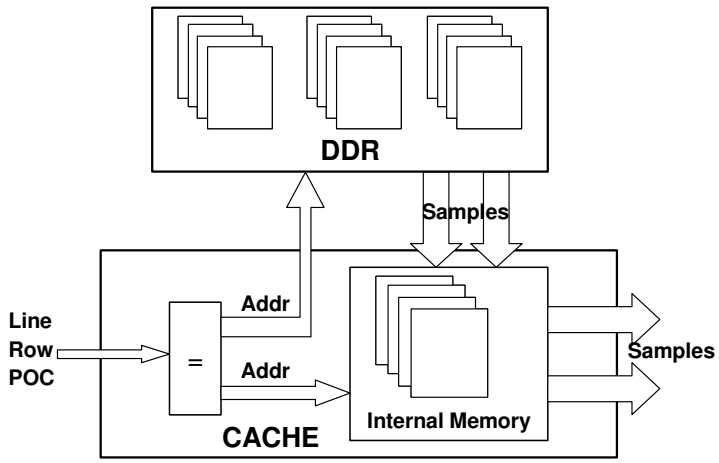


Figura 3.22: Esquemático da Memória Cache

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 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 à 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 benefício 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 à 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 à 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 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.

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 *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 16 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 32 primeiras posições registradas.

Na versão original do MoCHA, a configuração da cache foi baseada nos resultados de experimentos realizados (ZATT, 2006). 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).

A definição do tamanho da *cache* foi definido de acordo com os estudos feitos para o MoCHA original (ZATT, 2006). O número de amostras de luminância foi mantido o mesmo, e para isso algumas mudanças foram aplicadas. O número de posições para amostras de crominância foram duplicadas para poder acomodar vídeos com sub-amostragem 4:2:2. Essa alteração implicou em um aumento de 33% no número de bits dessa cache. Em seguida, cada uma das amostras foi aumentada de 8 para 10 bits uma vez que o perfil High 422 suporta vídeos com amostras de até 10 bits. Essa mudança implicou ainda um aumento de 25% no tamanho da *cache*. Ao fim das modificações, para o mesmo número de amostras de luminância, a *cache* teve um aumento de 66,25% em seu tamanho em bits. O número total de bits utilizados na *cache* foi de 400 Kb. A redução da largura de banda necessária para transmissão foi de 60% em média, enquanto o número de ciclos para acesso à memória reduziu em 85% para sub-amostragem 4:2:2 considerando um ciclo de para troca de linha da memória.

3.3 Resultados de Síntese e Comparação

A síntese da arquitetura utilizou, tal qual as demais sínteses apresentadas, o dispositivo XC2VP30-7 da família Virtex II Pro (VIRTEX SERIES, 2008) da Xilinx (XILINX: THE PROGRAMMABLE LOGIC COMPANY, 2008). A ferramenta utilizada para a síntese foi o ISE (PLATFORM STUDIO AND THE EDK, 2008), tendo como sintetizador o Symplify Pro 8.1 (SYNPLICITY: PRODUCTS: SYNPLIFY PRO, 2008) da Synplicity (SYNPLICITY: HOME, 2008). A Tabela 3.5 apresenta os resultados de síntese para cada um dos blocos da arquitetura, bem como para o MoCHA original e o HP422-MoCHA. A última coluna representa o aumento no uso de recursos de hardware do HP422-MoCHA com relação ao MoCHA perfil *Main*. A freqüência máxima de operação reportada foi de 99.9 MHz.

Tabela 3.5: Resultados de Síntese

	MVP	MA	SP	MoCHA Main 4:2:0	HP422-MoCHA	Increase %
Slices	3,494	951	5,511	8,465	10,530	24%
Flip Flops	3,358	723	5,904	5,671	6,313	11%
LUTs	5,961	1,175	6,742	10,835	13,894	28%
BRAM	3	24	0	21	27	28%
Multipliers	0	0	8	12	12	0%

Considerando uma freqüência de 100 MHz, a arquitetura apresentada pode processar até 152 milhões de amostras para vídeos inteiramente bi-preditivos e 290 milhões de amostras para vídeos inteiramente preditivos. Essa vazão permite exibir 37 quadros bi-preditivos HDTV ou 69 quadros preditivos HDTV por segundo.

Na literatura atual não foi encontrada nenhuma solução para a compensação de movimento no perfil *High 4:2:2* do padrão H.264/AVC. Podem ser encontrados apenas trabalhos que resolvem este mesmo problema nos perfis Main e Baseline. Por esse motivo, nossas comparações usarão apenas esses trabalhos como referência.

O compensador de movimento apresentado em (WANG, 2005b) foi implementado para HDTV no perfil *Baseline*. O MVP foi completamente implementado em hardware, mas por ser *Baseline* não suporta predição direta. O interpolador de luminância utiliza um filtro FIR a mais que o HP422-MoCHA.

Em (LIE, 2005), é apresentado um interpolador para o perfil *Main*, mas nenhum dado a respeito de desempenho do hardware é mostrado. (WANG, 2005) utiliza filtros diagonais de 4-taps no lugar dos filtros definidos pelo padrão, inserindo erros aos vídeos.

Em (CHEN, 2006), encontramos uma arquitetura para predição inter e intra-quadro que utiliza uma implementação em software do MVP. Essa solução não apresenta filtros bilineares explicitamente, no entanto esses filtros estão embarcados nos filtros FIR, totalizando 12 filtros bilineares contra apenas 4 do HP422-MoCHA.

Finalmente (LI, 2007) que não implementa o MVP, processa HDTV (1920x1080) @ 30 fps mas não suporta o perfil *High*.

3.4 Verificação

Esta seção descreve o método de verificação utilizado para verificar o correto funcionamento da arquitetura proposta englobando a obtenção dos casos de teste, a simulação, a escrita do *test bench* e a simulação do VHDL descrito.

Para verificar o funcionamento adequado da arquitetura descrita em VHDL, foi desenvolvida uma bateria de testes para verificação desta descrição. Como uma verificação formal se faz completamente inviável, partiu-se para a abordagem de verificação através de simulações exaustivas.

Para esta abordagem de verificação, é necessária a existência de uma grande quantidade e variedade de casos de teste, casos que devem apresentar dados de entrada e resultados comprovadamente corretos. A busca desses casos foi feita através do

software de referência do padrão H.264/AVC (IMAGE, 2008), versão JM12.2. As variáveis localizadas para verificação da arquitetura MoCHA original não puderam ser usadas uma vez que o *software* utilizado naquela oportunidade foi o JM9.5 que não contava com suporte aos Perfis *High*.

Obtidos os casos de teste teve início a fase de simulação, onde se utilizou o software ModelSim (MENTOR,2008b) da Mentor Graphics para inserir os dados obtidos na arquitetura desenvolvida. Mas antes de rodar a simulação, 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 preditor de vetores foram salvos em outros arquivos de texto.

Para utilizar os dados já capturados e convertidos em uma simulação no ModelSim, foi escrito um *test bench*. 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 binários dos arquivos de texto, identificando qual tipo de bloco será tratado e repassando para o preditor de vetores as informações de controle, vetores diferenciais e índices de referência necessários.

Além de inserir os estímulos de entrada à 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, nomeado como "mb_ready", é ativo em nível lógico 1 (um) e indica a existência de vetores já processados e prontos para serem lidos. Os vetores e referências processadas são, então, salvas em arquivos de texto acompanhadas de um contador de macroblocos.

Apenas depois desses processos de busca, conversão de dados e descrição do *test bench* pôde-se executar as simulações que validariam o compensador de movimento.

Para esta simulação, como já citado, foi escolhido o software ModelSim desenvolvido pela empresa Mentor Graphics em sua versão "ModelSim SE 6.0b" (MENTOR, 2008b). 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 através de formas de onda. Além disso, utilizando ModelSim pode-se fazer simulações não apenas comportamentais, mas também simulações da arquitetura já mapeada em diversas famílias de FPGA. Neste caso, o FPGA utilizado pra prototipação e simulação foi da família Xilinx Virtex-II Pro (XILINX, 2008b).

Durante as simulações foram detectados alguns pequenos erros de codificação em VHDL. A localização desses erros exigiu um esforço considerável e foi realizada através da observação do comportamento e dos valores assumidos pelos sinais internos da arquitetura. Depois de eliminar os *bugs* encontrados, puderam-se rodar as simulações exaustivas, capazes de comprovar com grande confiabilidade o funcionamento do módulo descrito.

Inicialmente, diante da indisponibilidade de vídeos 4:2:2 com larguras de amostra de 9 e 10 bits, foram utilizados vídeos 4:2:0 com amostras de 8 bits para a verificação. Nossa estratégia foi replicar as amostras de crominância sintetizando assim um vídeo com sub-amostragem 4:2:2. Para gerar vídeos com larguras de 9 ou 10 bits por amostra, utilizamos replicação dos bits menos significativos. Essa abordagem nos deu uma boa

verificação inicial, confirmada posteriormente com o uso de vídeos 4:2:2 com amostras de 9 e 10 bits, obtidos com a ajuda dos integrantes do JVT (*Joint Video Team*) por meio da lista de discussão daquele grupo.

As simulações foram primeiramente aplicadas para o vídeo citado codificado apenas com *slices* do tipo P, além de um *slice* tipo I que deve ser o primeiro frame obrigatoriamente. Assim foram obtidos os resultados para validar o processamento apenas de *slices* tipo P. 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 *slices* tipo P e *slices* tipo B intercalados entre si, de acordo com a seqüência I-P-B-B-P. Nessa segunda bateria de simulação, os slices B foram codificados utilizando predição direta espacial. A seguir, os *slices* B foram codificados com predição direta temporal e simulados. Completada essa etapa, foram obtidos dados para comprovar completamente o correto funcionamento do preditor de vetores em simulações de caráter comportamental.

Os dados obtidos por meio de simulação foram considerados corretos após comparação com os dados gerados pelo software de referência. Essa comparação foi feita através de um programa descrito em C++.

Esse processo de simulações aplicado em nível comportamental também foi aplicado para a arquitetura em fase de pós "place *and route*", repetindo os resultados já encontrados na primeira fase de verificação.

O método de verificação é abordado em mais detalhes em (ZATT, 2006), publicado no *Iberchip Workshop*.

3.5 Considerações Finais sobre o HP422-MoCHA

Esse capítulo detalhou a arquitetura HP422-MoCHA, uma arquitetura de compensação de movimento para o perfil *High 4:2:2* do padrão H.264/AVC com desempenho suficiente para decodificar, em tempo real, vídeos HDTV com resolução de 1920x1080 pixels.

A arquitetura HP422-MoCHA foi apresentada em um artigo publicado no *International Symposium on Circuits and Systems* (ISCAS 2008) e foi a primeira capaz de decodificar H.264/AVC Perfil *High* a ser publicada com detalhes específicos sobre a implementação em (ZATT, 2008).

Além disso, foi proposta uma nova arquitetura para o interpolador de amostras que mostrou ser mais econômica em recursos de *hardware* ocupados, sem degradação de desempenho. Esta arquitetura é apresentada em artigo publicado (ZATT, 2008a) no *Symposium on Integrated Circuits and Systems Design*.

Apesar dos ótimos resultados acadêmicos gerados por esse trabalho, o mesmo ainda não foi integrado a um sistema completo de decodificação. Embora o desenvolvimento de todos os módulos do decodificador já tenha sido concluído em nosso grupo de pesquisa, a tarefa de integrá-los, formando assim um sistema completo, tem se mostrado complexa. As principais dificuldades detectadas são a sincronização dos módulos e a utilização correta dos inúmeros sinais que compõem a interface de cada grande módulo.

Até o momento, já foram integrados os módulos de predição intra-quadro, as transformadas, a quantização e o filtro redutor de efeitos de bloco. O compensador de movimento tem se mostrado de difícil integração por ser o maior módulo de *hardware* desenvolvido e contar com um grande número de sinais de interface que devem ser devidamente sincronizados com o controle do sistema de decodificação.

A dificuldade de integração já era esperada no início deste desenvolvimento, quando se optou por particionar o trabalho de desenvolvimento em módulos separados, com uma especificação de controle global ainda não definida. Etapas iniciais de definição do módulo global foram suprimidas, fazendo-se diretamente a implementação dos módulos de hardware em uma abordagem *bottom-up*. Então, para estudar a temporização e sincronização de todos os módulos do decodificador, de forma a obter um sistema de decodificação funcional, o grupo decidiu retornar algumas etapas e desenvolver um modelo completo, em alto nível, do decodificador H.264/AVC.

Com base nessa experiência, decidiu-se utilizar uma estratégia de desenvolvimento diferente para o projeto do codificador H.264/AVC. Uma abordagem *meet-in-the-middle* foi considerada mais adequada para este novo projeto. Nessa abordagem um modelo de alto nível de abstração é desenvolvido em paralelo com o projeto de alguns dos módulos de hardware. É nesse contexto que se desenvolveu o modelo do codificador H.264/AVC, descrito em SystemC, que é assunto do capítulo 4 dessa dissertação.

4 MODELAGEM SYSTEMC DO CODIFICADOR H.264/AVC

Nesse capítulo será apresentado o desenvolvimento, em linguagem SystemC, da modelagem de uma arquitetura para um codificador H.264/AVC. Essa arquitetura foi projetada para codificar vídeos em resolução HDTV 1920x1080 pixels em tempo real. O codificador foi projetado para ser compatível com o perfil *Main*, mas utiliza apenas um subconjunto das ferramentas disponíveis no perfil. O método utilizado, a linguagem de descrição, a plataforma de desenvolvimento bem como os detalhes da arquitetura proposta serão descritos ao longo deste capítulo.

O capítulo que se inicia está disposto da seguinte forma: a seção 4.1 trata das principais características da modelagem, de seus níveis de abstração e da metodologia de desenvolvimento de sistemas envolvida, além de uma introdução sobre a linguagem SystemC e a plataforma de desenvolvimento utilizada. A arquitetura modelada é apresentada na seção 4.3. Os resultados são apresentados na seção 4.4 e as considerações finais na seção 4.5.

4.1 Modelagem

Conforme justificado no capítulo 3, a experiência no desenvolvimento do decodificador H.264/AVC fez com que algumas mudanças na estratégia de desenvolvimento tenham sido tomadas. Ao invés de utilizar uma abordagem *bottom-up* como no primeiro projeto, decidiu-se utilizar uma abordagem *meet-in-the-middle* permitindo dessa forma um estudo do sistema como um todo ao mesmo tempo em que são codificados em HDL os módulos de *hardware*. Nessa abordagem, foi desenvolvido um modelo em alto nível do codificador, utilizando a linguagem SystemC, que utiliza informações de módulos de hardware e devolve aos desenvolvedores de hardware informações arquiteturais e sistêmicas. Dessa forma, é formado um ambiente de desenvolvimento interativo que busca reduzir as dificuldades de integração entre os módulos e reduzir o tempo total de projeto.

Além dos ganhos que essa modelagem traz ao processo de desenvolvimento do codificador, não é desprezível sua importância no momento da verificação dos módulos de hardware. Com um modelo funcional é possível gerar casos de teste para os módulos e ainda proceder a verificação por meio de co-simulação entre modelo e descrição de HW.

A modelagem desse sistema de codificação busca seguir conceitos de *System-Level Design* e *Transaction Level Modelling* (TLM) definidos em (KEUTZER, 2000) e (CAI, 2003), respectivamente. No entanto, é importante salientar que a abordagem *meet-in-the-middle* utilizada nesse projeto difere daquela definida em (KEUTZER, 2000) por não partir uma plataforma de HW previamente fixada, mas, interativamente, definir sistema e módulos de HW de forma cooperativa.

A metodologia de desenvolvimento de sistemas através de TLM tem sido amplamente utilizada no meio acadêmico (AMER, 2004) (AMER, 2005) (YU, 2007) e industrial (ROSE, 2002). Como principais vantagens dessa abordagem, em relação à metodologia tradicional, podem-se salientar: tempo de desenvolvimento curto, fácil exploração do espaço de projeto, avaliação precoce do sistema, curto tempo de simulação, interoperabilidade entre diversos níveis de abstração, refinamento entre níveis de abstração. Essas vantagens permitem uma avaliação arquitetural e avaliação de desempenho em alto nível, onde esse processo se torna mais rápido e barato do que em RTL.

4.1.1 Níveis de Abstração TLM

Conforme definido em (CAI, 2003), existem diferentes níveis de abstração que podem ser utilizados em uma modelagem de acordo com conceitos TLM, onde cada modelo é dedicado a um objetivo específico.

Uma das principais características do TLM é a possibilidade de tratar comunicação e processamento como dois problemas distintos, sendo que os níveis de abstração possíveis são definidos de acordo com o nível de precisão da comunicação e do processamento. O gráfico mostrado na Figura 4.1 representa os modelos possíveis, onde o eixo *X* representa o nível de precisão da computação e o eixo *Y* representa a precisão da modelagem da comunicação.

Outra importante característica da modelagem TLM é a possibilidade de refinamento entre os diferentes níveis de abstração, ou seja, pode-se iniciar por um modelo de especificação, mais simplificado, e refiná-lo até alcançar, por exemplo, um modelo de implementação com precisão de ciclo para comunicação e para computação.

A modelagem TLM permite também a convivência, em um mesmo modelo, de diversos níveis de abstração dos módulos. Essa possibilidade foi utilizada no modelo a ser apresentado a seguir, assim como a possibilidade de sucessivos refinamentos.

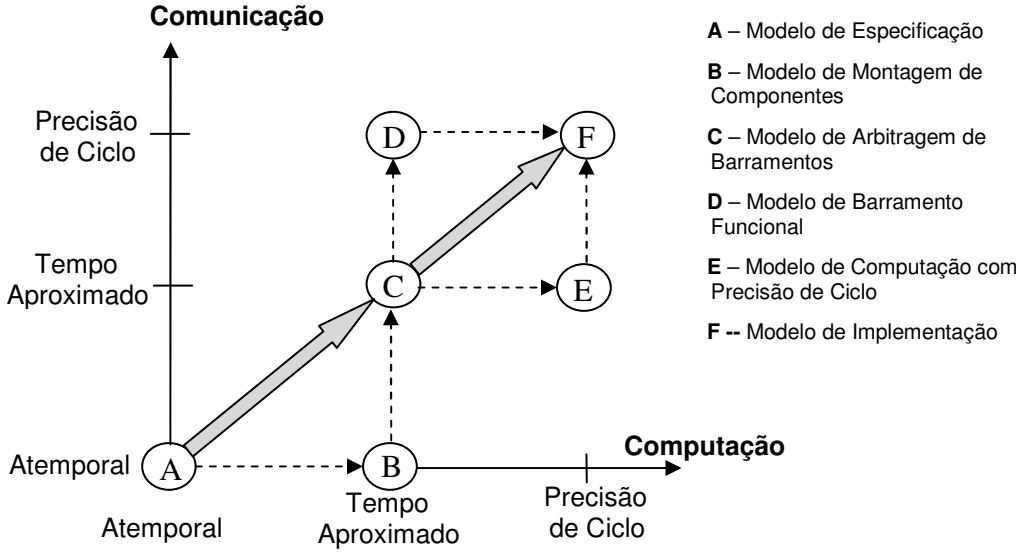


Figura 4.1: Níveis de Modelagem (CAI, 2003)

4.1.2 Linguagem SystemC

O SystemC (OSCI, 2008) é uma biblioteca de classes e macros para C++ que, quando criada em 1999 pela OSCI (*Open SystemC Initiative*), tinha a finalidade de modelar sistemas. Como a linguagem herdou conceitos de linguagens proprietárias como o Scenic da Synopsys e o N2C da CoWare que serviam para modelar sistemas em um nível de abstração próximo ao de hardware, ela é hoje uma biblioteca de classes reconhecida pela IEEE voltada para a modelagem de hardware.

Tal biblioteca permite modelar hardware em diversos níveis de abstração graças a características como concorrência, orientação a eventos e tipos de dados próprios para esse tipo de modelagem. A Figura 4.2 mostra um diagrama com a composição da linguagem SystemC. Atualmente é possível através de ferramentas especificas como o Cynthesizer da Forte ou o CoCentric SystemC da Synopsys, em SystemC gerar hardware, desde que se restrinja as construções ao subconjunto reduzido da biblioteca dito sintetizável.

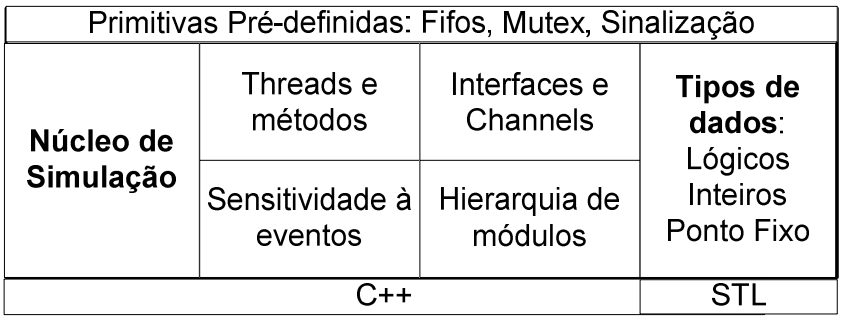


Figura 4.2: Arquitetura do SystemC

Por essa característica, o SystemC veio para preencher a lacuna existente entre a especificação do projeto de sistemas computacionais e a sua concepção, permitindo modelar hardware e software em um ambiente homogêneo, criando uma "especificação executável" do sistema em questão. Além disso, SystemC oferece total suporte à modelagem TLM, como criação de canais de comunicação, refinamentos sucessivos e convivência, em um mesmo modelo, de módulos descritos em diferentes níveis de abstração. Além disso, a linguagem permite co-simulação entre o modelo e descrições de hardware feitas em outras linguagens como VHDL e Verilog, por exemplo.

Por essas razões SystemC foi adotado para desenvolver o modelo do codificador H.264/AVC. Em nossa implementação foi utilizada a versão mais atual da linguagem no momento do início do desenvolvimento, o SystemC 2.2.

4.1.3 Plataforma de Desenvolvimento

A biblioteca SystemC funciona em conjunto com qualquer compilador C++ e, como uma biblioteca *opensource,* a OSCI recomenda o uso do GNU g++ (GNU, 2008) em Linux, fornecendo os devidos *makefiles* para que tudo funcione corretamente. No caso do Windows são fornecidas as instruções para compilar a biblioteca para Visual C++.

Uma vez compilado um programa em SystemC, é gerado um executável no qual é possível verificar os valores gerados pelo programa desenvolvido por meio da operação de saída *cout* do ANSI C++. Para testar o programa em questão é necessário criar funções que geram estímulos para as entradas do módulo e uma função *main()*, conforme o padrão ANSI C, que se encarrega de chamar os processos de execução. Nela, é necessário especificar a criação de um arquivo de *trace* que é um arquivo gerado pelo programa na extensão VCD (*Value Change Dump*) que contém as formas de ondas especificadas pelo comando *sc_trace*.

Essas formas de onda podem ser visualizadas no GTKWave (GTKWave, 2008), também *opensource*, que possibilita a visualização de arquivos VCD.

Esse software, porém, não possui uma interface muito amigável, tornando confusa a análise das formas de onda e dificultando a simulação, além de mostrar as formas de onda somente para as portas de entrada e saída do módulo em teste, ocultando os sinais internos.

Por essas limitações, foi avaliada a utilização do Modelsim, da Mentor Graphics (Mentor, 2007), não gratuito, mas disponível na universidade. O Modelsim possui suporte a SystemC e utiliza uma boa interface de simulação para gerar as formas de onda do módulo em teste, além de possibilitar a abertura de arquivos VCD em sua interface.

Utilizando esse último software é possível visualizar os sinais internos do sistema em teste uma vez que o Modelsim trata a simulação de forma diferente da dupla: compilador C++ e GTKWave.

Isso implica em uma hierarquia diferente de simulação por parte do Modelsim, por exemplo, não há uma função *main()*, mas um módulo de hierarquia superior que incorpora o gerador de estímulos (*test bench*) e o módulo em teste. Não há modificações significativas no módulo a ser simulado, salvo que as diretivas exigidas pelo Modelsim

são opcionais no modelo OSCI, por exemplo, a definição dos nomes das portas no construtor do módulo em questão. O Modelsim exige esses nomes para que as portas possam ser identificadas durante a simulação. No caso do modelo OSCI isso é um parâmetro opcional na função *sc_trace*.

O fato das diretivas necessárias ao Modelsim serem opcionais ao modelo padrão, implica que o Modelsim está de acordo com as normas do SystemC, existindo portabilidade entre os dois modelos graças a diretivas de compilação *ifdef* tal como a *MTI_SYSTEMC* que permite que determinada faixa de código seja compilada apenas se o usuário estiver usando o compilador *sccom* do Modelsim.

Versões anteriores do ModelSim tinham uma importante desvantagem com relação ao modelo OSCI. Os tipos de dados definidos por uma *struct* não podiam ser visualizados na forma de onda, pois o simulador não reconhecia a forma de onda resultante do conjunto de sinais membros da struct, podendo ser visto apenas no *debugger*. No entanto, essa deficiência foi corrigida na última versão do *software*, ModelSim 6.3f, permitindo assim, ampla liberdade no uso dessa plataforma.

4.2 Arquitetura e Modelagem do Codificador H.264/AVC

Esta seção apresenta a arquitetura e a modelagem do sistema de codificação proposto detalhando cada um de seus blocos componentes. Detalhes dos algoritmos implementados por cada um dos módulos podem ser encontrados no capítulo 2.

O codificador foi modelado utilizando uma metodologia baseada em TLM (*Transaction Level Modeling*) permitindo, assim, uma melhor separação entre detalhes de implementação da computação e da comunicação. Esse modelo utiliza diferentes níveis de abstração de acordo com as exigências de cada bloco desenvolvido, aproveitando assim, a capacidade da linguagem SystemC de suportar a convivência, em um único modelo, de diferentes níveis de abstração.

O objetivo desse modelo é definir uma arquitetura capaz de codificar, em tempo real, seqüências de vídeo HDTV 1080p a 30 quadros por segundo. O perfil abordado é o perfil Main do padrão H.264/AVC, no entanto, como o codificador (diferentemente do decodificador) não é normatizado, algumas restrições foram feitas para poder aliar taxa de compressão à utilização de hardware e ao desempenho desejado. Além disso, restrições na implementação se fazem mandatórias para que o trabalho caiba no escopo de um trabalho de mestrado, tendo em vista a grande complexidade do H.264/AVC.

O modelo do codificador é composto por 6 blocos principais: Preditor Intra-quadro; Estimador de Movimento (ME); Transformadas e Quantização Diretas (T/Q); Transformadas e Quantização Inversas (IT/IQ); CAVLC e Modo de Decisão (MD). A arquitetura apresentada na Figura 4.2 foi desenvolvida em forma de um *pipeline* hierárquico composto por dois macro-estágios. A granularidade deste *pipeline* é de um macrobloco, portanto a cada N (sendo N o número de ciclos de um macro-estágio) ciclos de relógio um macrobloco é processado por um macro-estágio e repassado para o seguinte. O primeiro estágio é responsável por toda parte de predição intra-quadro e inter-quadros e pelas transformadas e quantização, tanto diretas como inversas. Todos esses processos devem ser executados em um único macro-estágio devido às

dependências de dados entre os blocos 4x4 dentro de um mesmo macrobloco para a predição intra-quadro. Essa dependência será mais bem compreendida na próxima subseção.

O segundo macro-estágio se encarrega de fazer a codificação de entropia, que neste caso se resume ao CAVLC, e de armazenar os quadros reconstruídos na memória de referência para posterior uso por parte predição inter-quadros.

Muitas das opções arquiteturais feitas no desenvolvimento desse modelo levaram em conta blocos de HW já desenvolvidos, como é o caso do preditor intra-quadro, e módulos em desenvolvimento e pesquisa, como o codificador CAVLC e a ME.

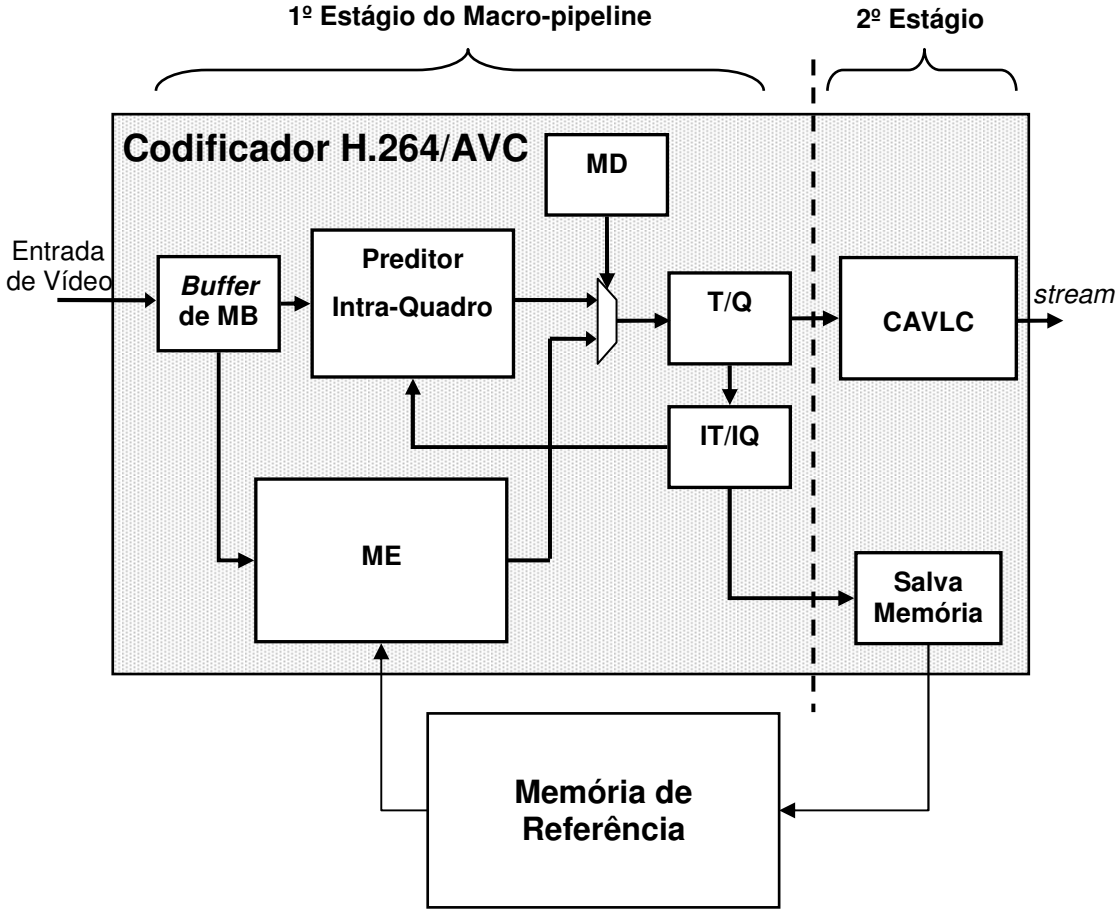


Figura 4.3: Arquitetura e Estrutura do Modelo do Codificador

4.2.1 Preditor Intra-Quadro

Esse bloco é encarregado de todo o processo de predição intra-quadro e está representado pela Figura 4.4. A arquitetura se divide em quatro módulos principais: Intra Vizinhos, Preditor de Amostras, SAD/MD I4MB e Memória de Predição. Cada módulo listado foi modelado como um SC_MODULE cuja lógica de computação foi implementada através de SC_METHODs e/ou SC_THREADs. O modelo de computação utilizado foi com precisão de ciclo. O modelo de comunicação também tem precisão de ciclo para comunicação interna ao preditor e tempo aproximado para comunicação com demais blocos. A comunicação com demais blocos é feita através de FIFOs.

É importante salientar que esse módulo é totalmente aplicável à HW, uma vez que foi inteiramente baseado em uma implementação VHDL verificada e sintetizada cujo relatório pode ser encontrado como apêndice desta dissertação. Essa abordagem foi adotada para garantir que o modelo tivesse forte relação com um HW real e obedecesse a restrições de tempo reais. Dessa forma podemos garantir que o modelo pode sim ser levado até o nível de circuito integrado, seja através de implementação em FPGA ou ASIC. Pelo mesmo motivo, esse módulo serve como base para a definição do número de ciclos dos macro-estágios de *pipeline* e, portanto, tem importância fundamental na definição da arquitetura do codificador como um todo.

Devido à dependência de dados entre blocos I4MB, o módulo de predição ficaria ocioso enquanto este bloco estivesse sendo processado pelas etapas T/Q, esperando este ser reconstruído para obter seus vizinhos para a próxima predição. Nesta solução há somente um caminho de dados para a predição. Quando um bloco 4x4 é predito e já está na etapa T/Q, pode ser calculado ao mesmo tempo a predição deste bloco considerando modos I16MB. Logo, a predição I16MB é feita em partições de blocos 4x4, no intervalo entre as predições dos 9 modos I4MB. Como são 4 modos somente, em paralelo pode ser feita a predição das crominâncias, de forma intercalada (Cb ou Cr) para aproveitar o caminho de dados. Apesar disto é preciso criar buffers de predição para guardar as predições dos blocos I4MB e I16MB enquanto a decisão ainda não foi realizada.

Na Figura 4.4, a caixa cinza denominada "*Preditor Intra-Quadro"* representa todo o Preditor Intra, as caixas brancas simbolizam os módulos e as elipses representam os métodos e threads. Os cilindros ilustram os canais de comunicação e as caixas ligadas à estes representam as interfaces entre módulos e canais. Essa simbologia será utilizada ao longo de todo esse trabalho em todos os diagramas que representam modelos SystemC.

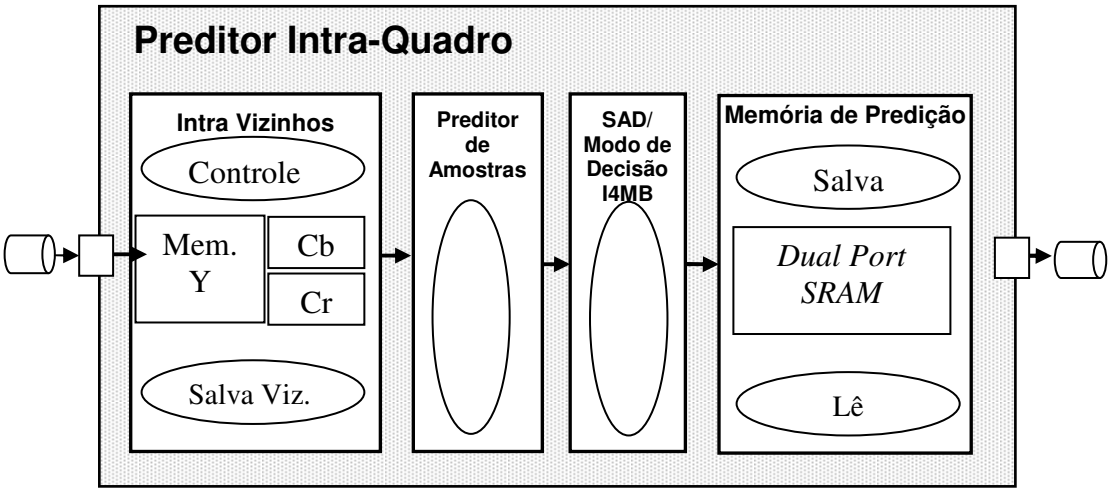


Figura 4.4: Diagrama do Codificador Intra-Quadro

4.2.1.1 Módulo Intra Vizinhos

Este módulo contém a memória que guarda a vizinhança necessária para codificar os macroblocos de uma linha do quadro, bem como possui duas máquinas de estados que controlam o preditor.

A memória, ilustrada na Figura 4.5, guarda uma linha de amostras vizinhas a serem usadas na próxima linha de macroblocos para predição. No exemplo mostrado na Figura 4.5, o conteúdo da memória é composto pelos vizinhos dos macroblocos 0-7, da linha de macroblocos anterior, e dos macroblocos 8-11, da linha atual que já foram processados. Cada vez que um macrobloco é reconstruído, a linha superior usada como referência neste passo é substituída pela inferior. Esta memória é formada, na verdade, por três blocos de memória (SC_MODULE), um para cada canal de cor. Para suportar vídeos de resolução 1920x1080 (1080p) e amostras de 8 bits a memória de luminância (Y_memory) possui 1920 bytes enquanto as memórias de crominância (Cb_memory e Cr_ memory) possuem 960 bytes cada.

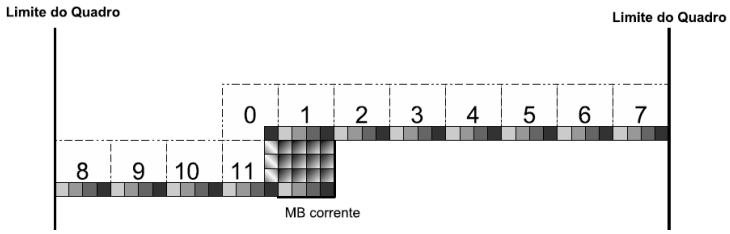


Figura 4.5: Memória de Vizinhos

Cada máquina de estados (FSM) foi implementada através de um SC_METHOD. O processo denominado "control" implementa uma máquina de estados composta por 44 estados que faz a leitura dos vizinhos da memória, cálculo dos valores DC e cálculo dos parâmetros 'a', 'b' e 'c' do modo planar, bem como disparar o processo de decodificação. Esse módulo dispara a predição dos blocos intercalando seus tipos de acordo com a seguinte ordem: T0 = I4MB; T1 = (I16MB + Cb); T2 = I4MB; T3 = (I16MB + Cr). A técnica de intercalar os tipos de blocos busca minimizar o efeito da dependência de dados entre os blocos I4MB e garantir uma maior utilização das unidades operativas.

A outra máquina de estados 'Neighbor Save' aguarda até que os resíduos dos blocos preditos percorram todo o laço das transformadas e quantização (T/Q e IT/IQ) para atualizar as memórias das amostras vizinhas. Essa máquina foi descrita em 25 estados. A comunicação entre os métodos é feita por meio de sinais (sc_signal) e tem precisão de ciclo.

4.2.1.2 Módulo Preditor de Amostras

O preditor de amostras opera com paralelismo de linha, ou seja, a cada ciclo é produzida a predição de quatro amostras na saída para todos os modos, de forma que, ao final de quatro ciclos é feita a predição de um bloco 4x4 para todos os modos. A Figura 4.6 mostra um diagrama do preditor.

Esse módulo é composto por 31 PEs (elementos de processamento) controlados por uma máquina de estados descrita através de um SC_METHOD. Esses PEs utilizam as informações de vizinhança recebidas do módulo "Intra Vizinhos" para gerar a predição dos 9 modos I4MB ou os 4 modos I16MB e os 4 modos de croma.

Existem 4 tipos de PEs: 11 PEs são utilizadas para aplicar a Eq. 2.7, 2 PEs para a Eq. 2.8, 10 PEs para a Eq. 2.9 e 8 PEs para o modo planar (Eq. 2.6), sendo 4 para I16MB e 4 para croma. Em todo ciclo todas as predições possíveis são geradas em paralelo e multiplexadas de maneira a gerar a predição adequada. Os multiplexadores são controlados pela FSM.

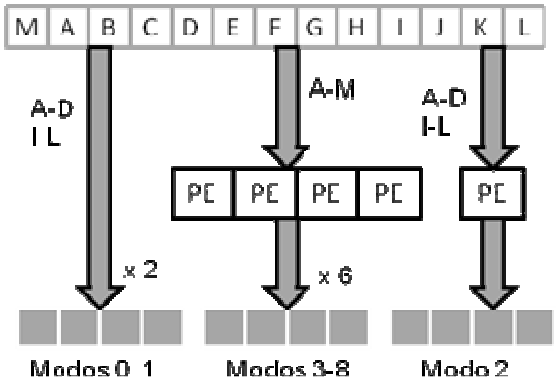


Figura 4.6: Preditor de Amostras

4.2.1.3 Módulo de cálculo do SAD e Decisão I4MB

O módulo de cálculo do SAD, apresentado na Figura 4.7, possui 9 elementos de cálculo de SAD (PE0 até PE8) implementados como árvores de somadores, que faz a soma das quatro amostras da saída do preditor, um registrador para guardar o resultado da soma e mais um somador para adicionar este valor com as próximas amostras. Este módulo trabalha em *pipeline* com o modulo de predição: assim que as 4 amostras foram preditas, o módulo de cálculo do SAD dispara a árvore de somadores, e o módulo preditor passa a calcular as próximas 4 amostras. Este módulo foi modelado com apenas uma máquina de estados algorítmica (ASM – *Algorithmic State Machine*) com 5 estados utilizando um SC_METHOD, onde cada estado é responsável por acumular o SAD de uma linha além do estado de espera.

O módulo de decisão 4x4 consiste numa árvore de comparadores, que tem como entrada o SAD de cada modo e a informação da disponibilidade do modo, calculada com base na disponibilidade dos vizinhos, fornecido pelo módulo que calcula os vizinhos. A saída é a informação do modo escolhido. A implementação SystemC utilizou um SC_THREAD.

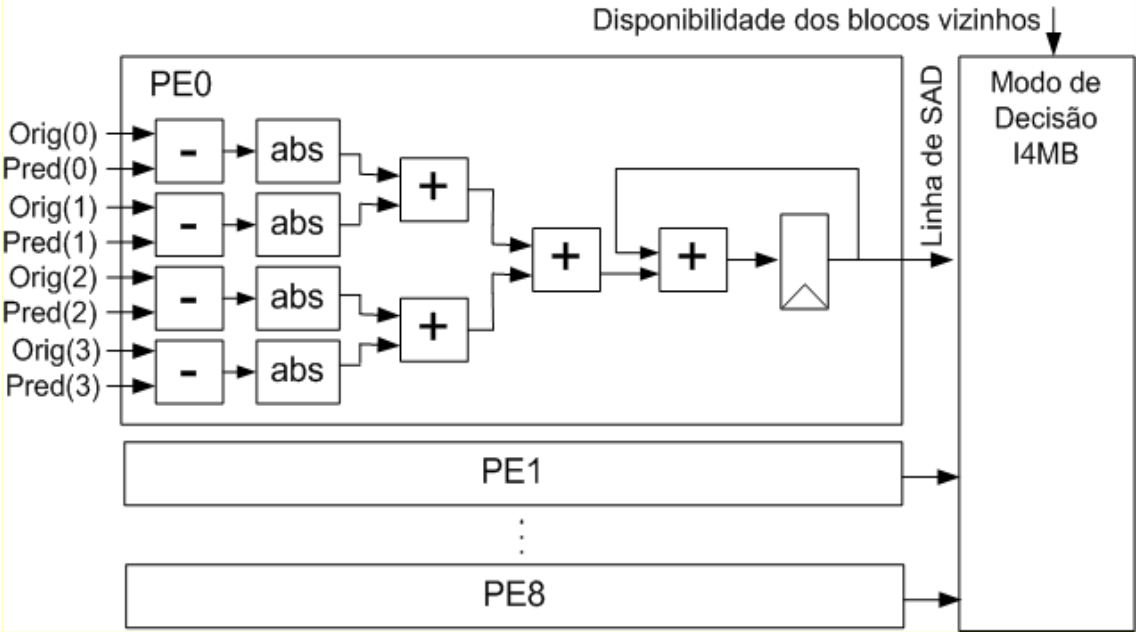


Figura 4.7: Módulo de Cálculo do SAD e Decisão I4MB

4.2.1.4 Memória de predição

A memória de predição guarda as predições para todos os modos, feito pelo preditor de amostras. Esta memória é necessária, pois devem ser calculados os resíduos da predição para o modo selecionado, que é decidido somente quando todo o processamento de um macrobloco termina.

Apenas depois de gerar as predições possíveis e calcular o SAD (ou outra métrica) pode-se escolher o tipo do macrobloco a ser codificado. Nesse momento, então, é necessário que se tenha disponível os resíduos para serem enviados para a codificação de entropia. Para tanto, os valores da predição são necessários para geração dos resíduos e para posterior reconstrução do macrobloco. Para obter os valores preditos pode-se gerá-los novamente ou, como optado nessa arquitetura, armazená-los em uma memória interna.

O módulo chamado de "memória de predição" contém uma memória interna SRAM *dual port* organizada em 68 linhas de 36 amostras ou 288 bits cada. Essa memória armazena a predição de um bloco I4MB, um macrobloco I16MB com os canais de croma Cb e Cr. Sua organização interna é da seguinte forma: as primeiras quatro (0..3) linhas armazenam os 9 tipos de predição para um bloco I4MB; as linhas pares 4 e 35 armazenam os 4 modos I16MB e os 4 modos de predição Cb; as linhas ímpares 4 e 35 armazenam os 4 modos I16MB e os 4 modos Cr; da linha 36 até 67 é armazenada apenas predição para I16MB.

Além do modelo da SRAM *dual port*, o módulo contém dois processos do tipo SC_METHOD. O primeiro processo, "write", é uma FSM com 11 estados que armazena as predições geradas. O segundo processo, "read" é uma FSM com 6 estados que aguarda a escolha do tipo do macrobloco e faz a leitura da memória, enviando a predição ao subtrator e transformadas. A comunicação entre os processos é feita por meio de sinais *sc_signal.*

4.2.2 T/Q

O módulo chamado T/Q engloba em um único bloco as tarefas do subtrator, que gera os resíduos a partir das amostras originais e preditas, das transformadas diretas e da quantização direta. A arquitetura pode ser dividida em duas partes principais, o subtrator e o caminho de dados que executa as transformadas e a quantização. No entanto, a modelagem não diferencia esses dois processos como sendo diferentes SC_MODULEs, e sim implementa como processos distintos dentro de um mesmo SC_MODULE.

Esse módulo utiliza paralelismo de uma linha de um bloco 4x4 e é baseada nas arquiteturas de quantização direta, de transformada direta parcialmente paralela e Hadamard direta apresentadas em (AGOSTINI, 2007).

A modelagem desse módulo foi feita inicialmente em um nível de abstração mais alto, facilitando assim sua verificação, mas posteriormente foi refinado a ponto de atingir precisão de ciclo tanto na computação como na comunicação entre módulos. Essa precisão é necessária por esse módulo fazer parte do laço de reconstrução intra, composto pelo preditor intra, pelas transformadas diretas e inversas e pela quantização direta e inversa.

A Figura 4.8 simboliza o módulo T/Q e sua descrição SystemC, obedecendo a simbologia descrita na seção anterior.

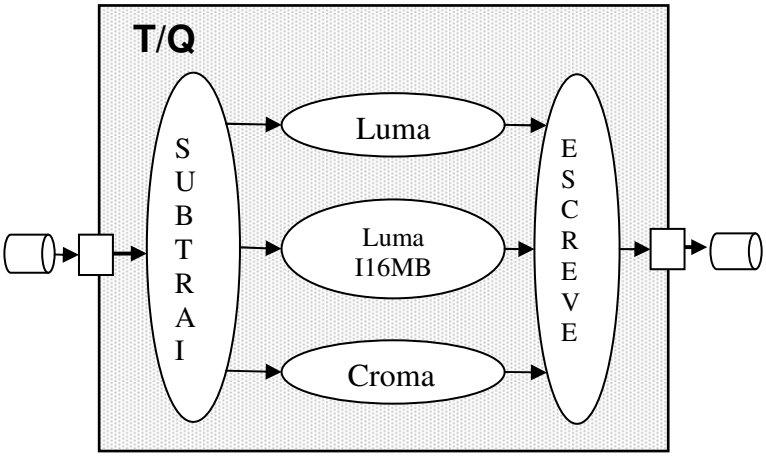


Figura 4.8: Estrutura de Modelagem do Bloco T/Q

4.2.2.1 Subtrator

Esse módulo é responsável pela leitura das FIFOs que contêm as amostras originais e as amostras preditas, subtraí-las gerando os resíduos a serem transformados pelo bloco seguinte.

A modelagem foi feita com o uso de uma SC_THREAD no módulo T/Q. Essa SC_THREAD também é responsável por disparar os métodos que executam as transformadas e a quantização utilizando eventos *sc_event.*

4.2.2.2 Transformadas e Quantização

O bloco das transformadas e quantização propriamente dito implementa a arquitetura apresentada na Figura 4.9. Ele recebe os resíduos vindos do subtrator, aplica a FDCT e, de acordo com o tipo do bloco a ser tratado, encaminha às transformadas Hadamard ou diretamente à quantização adequada.

No campo da modelagem, essa arquitetura foi implementada utilizando 3 diferentes SC_METHODs, um para o processamento completo das amostras de luminância I16MB, outro para crominância e um terceiro para os demais modos de luminância. Ou seja, por simplicidade na modelagem foram implementados 3 diferentes caminhos de dados, um para cada um dos possíveis caminhos da Figura 4.9. Isso em nada altera a precisão do modelo e sua correspondência com a arquitetura de HW.

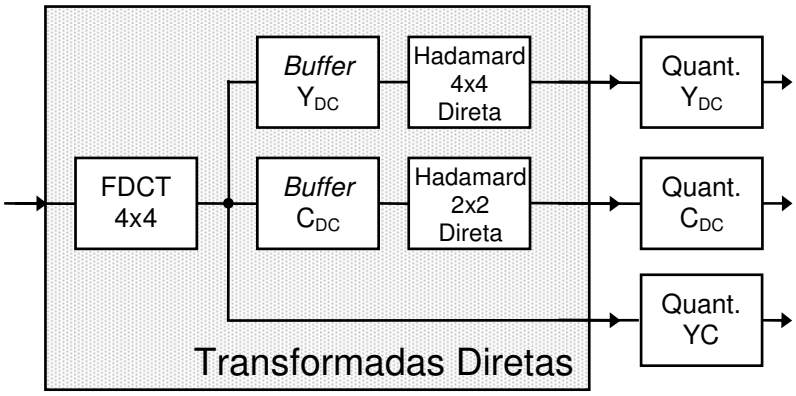


Figura 4.9: Arquitetura das Transformadas e Quantização

A escrita dos dados após fica a cargo de outro processo, SC_THREAD, que nas FIFOs de saída, sendo que uma envia dados à codificação de entropia e outra para o bloco da quantização e transformada inversas (IT/IQ).

4.2.3 IT/IQ

Esse bloco processa as transformadas e quantização inversas. Sua arquitetura e modelagem seguiram as mesmas diretrizes utilizadas no módulo T/Q, partindo de uma descrição alto nível e sofrendo refinamentos sucessivos para dar ao modelo precisão de ciclo. Como o bloco IT/IQ não implementa exatamente a operação inversa ao bloco T/Q, este apresenta algumas diferenças arquiteturais com relação a àquele, no entanto a estrutura de modelagem manteve-se a mesma. O paralelismo utilizado é de 4 amostras, ou uma linha, por ciclo.

Dois blocos principais compõem o módulo IT/IQ, o responsável pelas transformadas e pela quantização e o responsável por implementar o somador.

4.2.3.1 Transformadas e Quantização Inversa

Esse bloco recebe os dados vindos do bloco T/Q por meio de uma FIFO e seleciona o caminho de dados adequado de acordo com o tipo do bloco a ser processado. Os dados podem então ser encaminhados às transformadas Hadamard 2x2 ou 4x4 se forem coeficientes DC (ver capítulo 2), ou diretamente a quantização inversa. Após a quantização inversa é aplicada a DCT inversa (IDCT). Esse processo reconstrói os resíduos que são encaminhados ao somador.

A exemplo do bloco T/Q, essa arquitetura foi implementada utilizando 3 diferentes SC_METHODs, um para o processamento completo das amostras de luminância I16MB, outro para crominância e um terceiro para os demais modos de luminância. Ou seja, por simplicidade na modelagem foram implementados 3 diferentes caminhos de dados, um para cada um dos possíveis caminhos da Figura 4.10. A leitura dos dados vindos do bloco T/Q é feita por um processo SC_THREAD.

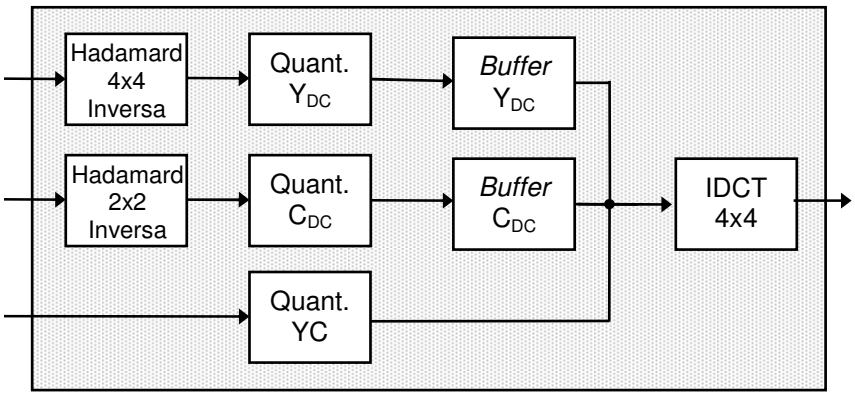


Figura 4.10: Arquitetura das Transformadas e Quantização Inversas

4.2.3.2 Somador

O somador gera as amostras reconstruídas efetuando a soma dos resíduos já inversamente quantizados e transformados com as amostras preditas. As amostras preditas são lidas de FIFOs com amostras enviadas pelos módulos de predição intra-quadro ou inter-quadros. Sua modelagem foi feita por um processo do tipo SC_THREAD disparado por eventos.

4.2.4 ME

A estimação de movimento é a tarefa mais computacionalmente complexa do codificador e permite grande liberdade para cada diferente implementação. Diferentes algoritmos de busca podem ser usados, diferentes áreas de buscas, técnicas de subamostragem, bipredição, busca em múltiplos quadros de referência, predição ponderada, precisão de quarto de pixel, entre outras possibilidades.

Nessa implementação optou-se por uma ME relativamente simples e de acordo com trabalhos realizados em paralelo pelo grupo de pesquisa. Essa implementação utiliza *Full Search* como algoritmo de busca, uma área de busca de até 48x48 pixels considerando apenas amostras de luminância, SAD como critério de escolha e suporta todos os tamanhos de blocos permitidos pelo padrão. A busca é feita em apenas um quadro de referência, portanto não suporta bi-predição.

As simplificações inseridas nesse módulo têm como principal motivo alcançar um codificador completo e funcional o mais breve possível, adicionando complexidade com o andamento do projeto. Outro importante motivo é a necessidade de tal modelo caber no escopo e no tempo de um trabalho de mestrado. No entanto modificações são de fácil implementação e análise devido à maneira em que esse bloco foi modelado.

O modelo desse módulo foi descrito em alto nível, mas refinamentos fizeram com que pudesse ser emulado o comportamento do HW com precisão de ciclo. As restrições de tempo são impostas pelo tempo de um macro-estágio de *pipeline* definido pelo tempo do laço de predição intra-quadro, composto pelos módulos: Preditor Intra-quadro, T/Q e IT/IQ.

A modelagem para o bloco da ME está representada pela Figura 4.11 e se divide em quatro principais blocos internos: o módulo de leitura da memória externa, o controle da busca, o cálculo do SAD e os acumuladores de SAD. O nível de paralelismo utilizado é

de um macrobloco por ciclo, ou seja, um macrobloco candidato pode ser avaliado a cada ciclo.

Arquiteturas desenvolvidas pelo grupo de pesquisa para ME podem ser consultadas em (AGOSTINI, 2007), (PORTO, 2008). As arquiteturas de (AGOSTINI, 2007) utilizam algoritmo de busca *Full Search* enquanto o trabalho em (PORTO, 2008) explora algoritmos rápidos e implementa duas arquiteturas baseadas no algoritmo *Diamond Search* utilizando a técnica de sub-amostragem de amostras. Esses trabalhos não se preocupam com diferentes tamanhos de bloco, utilizando apenas blocos de 16x16 amostras. Atualmente está sendo desenvolvida uma arquitetura, baseada em *Full Search*, que explora diferentes tamanhos de blocos. É nessa arquitetura que esse modelo se baseia.

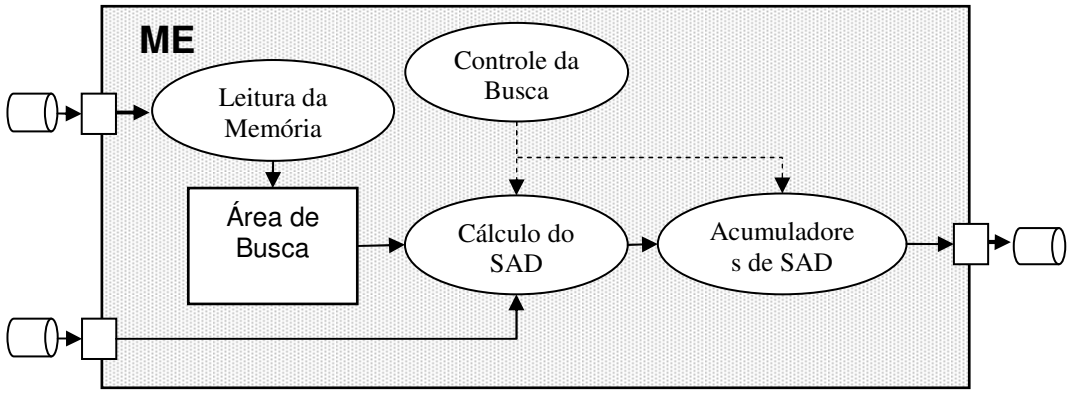


Figura 4.11: Estrutura de Modelagem da ME

4.2.4.1 Leitura da Memória

A região do quadro de referência a ser pesquisada, chamada de área de busca, primeiramente deve ser lida da memória de quadros de referência antes de ser explorada pelo algoritmo de busca. Essa memória de quadros de referência é tipicamente hospedada em memória externa pelo grande volume de dados que armazena, basta considerar que apenas um quadro de referência 1920x1080 utiliza 2,96 MB de memória.

Um processo SC_THREAD foi então criado para gerenciar a comunicação com a memória externa e armazenar em uma memória interna a área de busca para o macrobloco atual. Essa memória interna para área de busca armazena uma área de 3x3 macroblocos da imagem de referência, o que representa uma área de busca de 48x48 amostras. Os 9 macroblocos trazidos da memória correspondem ao macrobloco na mesma posição relativa ao macrobloco atual a ser processado e os oito macroblocos a sua volta (superior, superior esquerdo, esquerdo, inferior esquerdo, inferior, inferior direito, direito e superior direito), conforme representado na Figura 4.12. As amostras de luminância são lidas no primeiro momento, pois na arquitetura proposta somente o canal de luminância é utilizado na decisão da ME. Somente após a leitura das amostras de luminância, são lidas as amostras de crominância para serem enviadas aos demais estágios de codificação. O modelo considera que possa ler, em média, uma palavra de 64 bits da memória externa a cada ciclo.

Embora a área de busca, de 48x48 amostras, seja muito grande para que o HW de busca possa fazer a busca completa em tempo real para HDTV, ela foi projetada nesssa

dimensões para permitir maior qualidade na busca quando trabalhando com resoluções menores ou quando tempo real não é exigido.

A organização da memória externa foi projetada de forma a diminuir o número de ciclos para leitura da memória, para isso buscou-se reduzir o número de trocas de linha na memória. Com isso as amostras de luminância de cada quadro de referência foram posicionadas de forma contígua entre si, posicionando as amostras dos canais Cb e Cr logo a seguir, conforme ilustrado pela Figura 4.13.

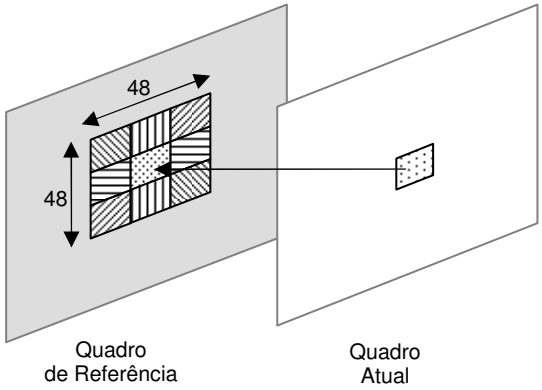


Figura 4.12: Área de Busca Lida da Memória Externa

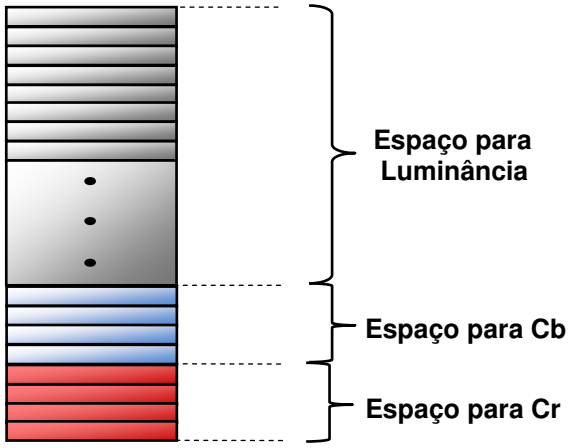


Figura 4.13: Organização da Memória Externa

4.2.4.2 Controlador do Algoritmo de Busca

Esse bloco é responsável por todo o controle da busca da ME. Da forma em que foi descrita, toda a lógica de busca da ME pode ser alterada fazendo modificações apenas nesse módulo. O controlador seleciona qual a posição da área de busca deve ser comparada em cada instante de tempo, disparando a comparação de um bloco de 16x16 amostras a cada ciclo de relógio. Através de parâmetros externos, é possível definir a área de busca a ser pesquisada de acordo com as restrições de desempenho até um limite de 48x48 amostras, podendo ser um retângulo como 32x48, por exemplo.

A modelagem foi feita por meio de um SC_THREAD que dispara os elementos de processamento de cálculo de SAD e comparação por meio de eventos *sc_event.*

4.2.4.3 Cálculo de SAD

Embora o módulo ME trabalhe com paralelismo de 16x16 amostras, o cálculo do SAD é feito, em paralelo, de forma independente para cada bloco 4x4. Dessa forma, o uso de tamanhos variáveis de bloco se resume a forma em que os blocos 4x4 são selecionados de acordo com o SAD calculado. A estrutura interna do elemento de processamento que acumula o SAD utiliza uma árvore de somadores.

4.2.4.4 Acumulador de SAD

O acumulador de SAD seleciona o melhor o melhor bloco 4x4 para aquela posição no macrobloco e acumula o valor total do SAD para aquele macrobloco. Existe um acumulador para cada possível particionamento de um macrobloco, ou seja, existe um acumulador para 16x16, um para 16x8, um para 8x16 e finalmente um para 8x8. De forma análoga, existe um acumulador para cada partição com seus possíveis particionamentos: 8x4, 4x8 e 4x4. Isso gera todos os possíveis particionamentos para um macrobloco predito, deixando a cargo do modo de decisão escolher o melhor dentre eles.

Os acumuladores funcionam da seguinte forma. Quando novos valores de SAD são gerados, esses novos valores são comparados aos valores dos blocos selecionados anteriormente. Caso sejam menores, o valor de SAD daquela partição é substituído e o valor do acumulador é atualizado. Além do valor do SAD, o acumulador também armazena o vetor de movimento e a amostras do bloco escolhido para ser processado pelas próximas etapas da codificação.

4.2.5 CAVLC

Este é o único bloco completo a ser modelado que ocupa o segundo estágio do macro-pipeline. Sua arquitetura foi dividida em dois estágios principais, um que consiste na obtenção dos parâmetros para a geração da codificação, tais como *Total_Coeff,Total_Zeros, Trailing_Ones* e *Run_Before*, e outro que é a geração dos palavras de código que compõe o bitstream. Essas duas etapas distintas foram descritas como descritas como sendo dois SC_MODULEs que integram o SC_MODULE chamado CAVLC, conforme Figura 4.14.

Como o primeiro módulo interno do CAVLC, que obtém os parâmetros, gera dados a uma taxa constante enquanto o outro módulo consome em taxa variável, a comunicação entre eles foi modelada por meio de um FIFO para conformação de tráfego. A Figura 4.14 mostra a FIFO que comunica os módulos.

O CAVLC lê toda informação do macrobloco processado pelas transformadas e quantização e processa um bloco 4x4 por vez seguindo uma ordem predefinida apresentada no capítulo 2, ver Figura 2.8.

O módulo topo do CAVLC foi modelado com dois métodos SC_METHODs, um responsável pela entrada de dados, descrito como uma FSM com 9 estados e outro, uma FSM de 3 estados, para disparar o processamento dos módulos de obtenção de parâmetros e geração do *bitstream* de saída.

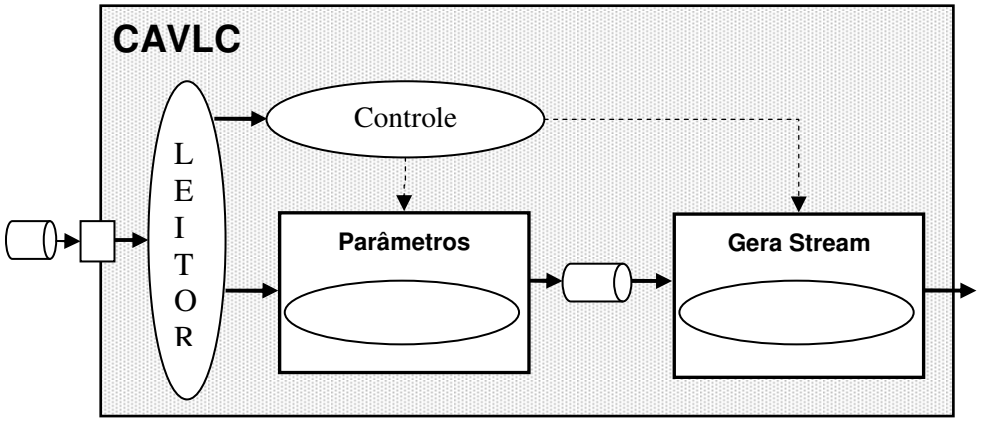


Figura 4.14: Estrutura de Modelagem do CAVLC

4.2.5.1 Cálculo de Parâmetros

Conforme mostrado pela Figura 4.14, quatro módulos em paralelo foram definidos para obter os quatro parâmetros necessários para a codificação: *Total_Coeff, Total_Zeros, Trailing_Ones* e *Run_Before*. Além disso, esses módulos também montam dois vetores com valores utilizados na codificação: o *Levels* que é um vetor contendo os valores de resíduos válidos e com módulo maior que '1'; e o *Trailing_one_signal* contendo os sinais de cada valor das corridas de resíduos com módulo '1' encontradas nos resíduos.

Cada um dos módulos percorre o vetor com todos os valores dos resíduos de forma serial, portanto essa etapa pode consumir até 16 ciclos de relógio uma vez que esses vetores têm comprimento máximo de 16 valores. As informações obtidas são então escritas em uma FIFO que faz a comunicação com o outro módulo interno do CAVLC.

Todos os módulos que operam em paralelo foram modelados utilizando uma única SC_THREAD de forma a simplificar a descrição.

4.2.5.2 Geração do Stream

Com base nos valores e parâmetros recebidos por meio da FIFO que comunica os dois módulos internos do CAVLC, esse módulo pode gerar uma parte do *Bitstream.* O *stream* gerado pelo CAVLC é apenas uma parte do *bitstream* H.264/AVC, na verdade é uma parte da camada de macrobloco. As demais informações, como vetores de movimento, tipos de macrobloco, informações de seqüência e *slice,* são processadas utilizando códigos *Exp-Golomb*, também conhecidos no padrão como UVLC (*Universal Variable Length Code*).

O módulo que gera o *stream* do CAVLC pode gerar um byte por ciclo de relógio. Sua modelagem foi feita utilizando um SC_METHOD que descreve máquina de estados composta por oito estados.

4.2.6 Controle

Como cada um dos módulos foi desenvolvido de forma bastante completa, o controle do codificador pôde ser bastante simplificado, resumindo-se a uma máquina de

estados que controla o andamento dos macroblocos sendo processados e dispara o processamento de cada estágio do macro-pipeline. Sua modelagem utilizou apenas um processo SC_METHOD descrevendo uma máquina de estados composta de sete estados.

O módulo de controle foi desenvolvido com a possibilidade de controlar o bloco da ME de forma a alterar a área de busca de acordo com o desempenho requerido e o vídeo a ser codificado. Se o codificador trabalha sobre vídeos de alta definição, a área de busca deve ser restringida, no entanto, quando trabalhando sobre vídeos de menor resolução, pode-se optar entre baixar a freqüência de operação do codificador em busca de reduzir o consumo de energia ou ainda manter a freqüência e aumentar a área de busca para alcançar melhores resultados em termos de codificação. O aumento da área de busca pode ser utilizado também sobre vídeos HDTV que não exigem codificação em tempo real.

4.2.7 Modo de Decisão (MD)

Conforme já discutido no capítulo 2 desta dissertação, esse é um problema muito complexo, pois deve selecionar, entre inúmeras opções de codificação, aquela que apresentar uma melhor codificação considerando distorção e número de bits gerados. O *software* de referência do padrão H.264/AVC utiliza uma técnica chamada RDO (*Rate Distortion Optimization*), que utiliza como informações para cálculo a distorção do macrobloco reconstruído e o número total de bits gerados para cada uma das possíveis codificações. É fácil perceber que isso insere um enorme aumento na complexidade computacional do codificador.

Para implementações em *hardware* que buscam tempo real para HDTV, o RDO representa um grande aumento na área, pois exigiria blocos de codificação de entropia trabalhando em paralelo a fim de obter o número de bits gerados para cada uma das inúmeras alternativas de codificação. Além disto, o RDO introduziria um grande aumento na complexidade de controle do codificador. Dessa forma, essa arquitetura prevê um método simplificado para o modo de decisão, no entanto a proposta do mesmo não está no escopo deste trabalho. O modelo do codificador desenvolvido, se propõe a servir de plataforma para experimentos que possam propor um modo de decisão simplificado, porém eficiente, que melhor atenda as necessidades de um codificador em HW. Diversos trabalhos que buscam propor um modo de decisão simplificado para o H.264/AVC podem ser encontrados na literatura, como, por exemplo (PAN, 2005) (WU, 2004).

O bloco responsável pelo modo de decisão foi modelado como um SC_MODULE embora o algoritmo do modo de decisão não tenha sido definido. Ao longo da modelagem a lógica de decisão foi sendo alterada para permitir a validação das diversas formas de codificação.

4.3 Resultados

Embora a modelagem de um sistema não traga resultados tão concretos como o desenvolvimento do hardware propriamente dito, esse trabalho tem grande importância no desenvolvimento do codificador como um todo, permitindo caracterizar a temporização do sistema, a comunicação entre os diferentes módulos, verificar o funcionamento do sistema completo além de servir como plataforma para verificação dos módulos de *hardware* e para experimentos que explorem diferentes alternativas de

codificação, principalmente sob o ponto de vista da estimação de movimento e do modo de decisão.

A Figura 4.15 apresenta três diagramas de tempo que caracterizam a temporização do codificador obtida através da modelagem SystemC. A figura detalha a temporização do primeiro estágio do *pipeline*, onde o eixo Y apresenta as unidades funcionais enquanto o eixo X apresenta o número de ciclos utilizados em cada tarefa. Esse estágio é responsável pela geração de todas as formas de predição possíveis, tanto intra quanto inter-quadros, para um macrobloco. As unidades representadas são: Geração de Parâmetros Intra e Atualização de Vizinhos, que geram os parâmetros DC e Planar e ao fim do processamento do MB atualizam a vizinhança; Preditor Intra, responsável por gerar a predição intra-quadro tanto para I4MB quanto para I16MB (representados na figura em cores diferentes entre si); T/Q, bloco das transformadas e quantização direta que respeita, na figura, a mesmo padrão de cores relativos aos tipos de dados processados, aplicando cor azul ao dados de Cb e vermelho aos dados de Cr; IT/IQ, bloco das transformadas e quantização inversa; MD, modo de decisão; Memória de Referência, que se refere ao tempo utilizado para leitura da memória de referência, em cinza luminância, em azul Cb e vermelho Cr; Busca ME, que representa o tempo de busca para estimação de movimento.

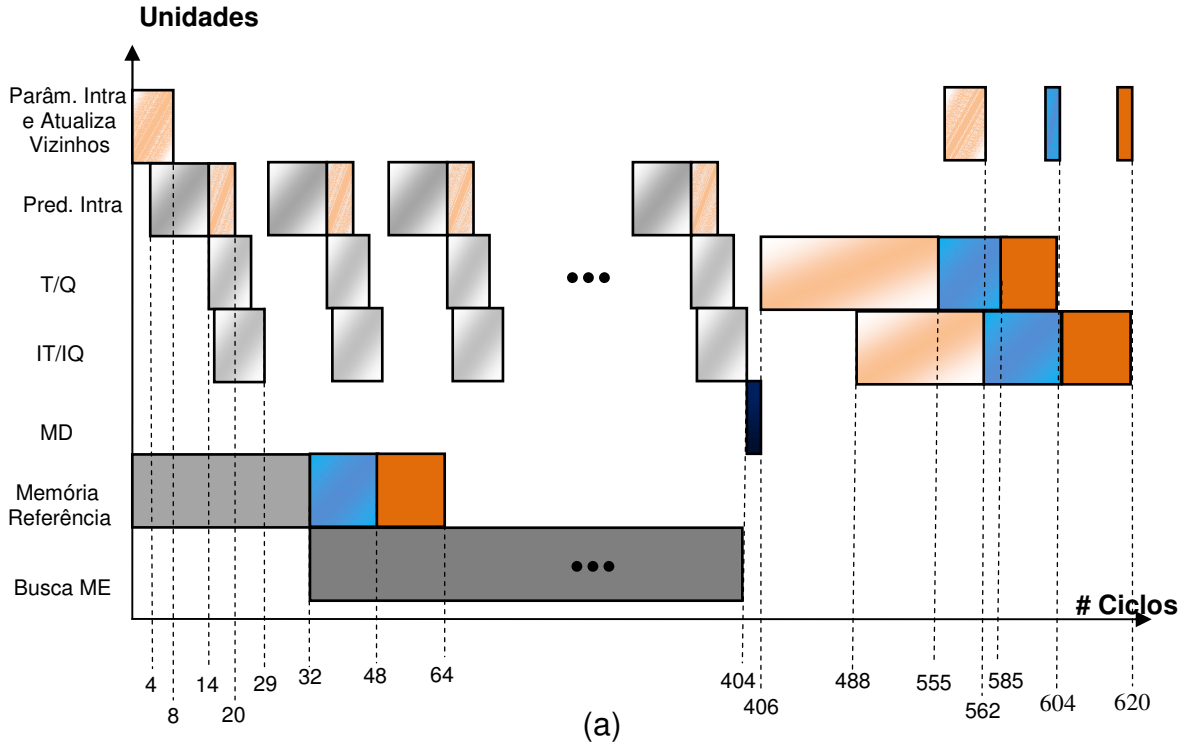
Na Figura 4.15(a) é apresentado o diagrama de tempo no pior caso, onde o modo de decisão escolhe, como melhor alternativa de codificação, o modo I16MB. Essa escolha representa o pior caso pois, como mostrado na Figura 4.15(a), utiliza 620 ciclos de relógio após a escolha do modo de decisão devido à latência inserida pelas transformadas Hadamard 4x4. Nesse modo, é necessário esperar o fim do processamento pelo bloco T/Q para que possa se disparar o bloco IT/IQ. Conforme apontado pela figura, cada macro-estágio de pipeline consome 620 ciclos de relógio, isso aponta para uma freqüência mínima de operação, para atingir codificação em tempo real de vídeos 1920x1080, de aproximadamente 150 MHz.

A Figura 4.15(b) mostra o diagrama de tempo do codificador quando o modo de decisão seleciona a predição I4MB. Esse é o melhor caso em termos de número de ciclo já que todos os blocos de luminância já foram transformados e quantizados previamente. Nesse caso 471 ciclos são necessários para processar um macrobloco, exigindo assim uma freqüência de operação de 114,5 MHz para codificar HDTV 1080p em tempo real.

Na Figura 4.15(c) o diagrama de tempo mostra o caso intermediário, onde a predição inter-quadros é selecionada. Embora todas as amostras tenham de ser transformadas e quantizadas novamente, a não necessidade de aplicar as transformadas Hadamard 4x4 reduzem a latência do laço T/Q – IT/IQ. Para este terceiro caso são necessários 544 ciclos para codificação de um macrobloco enquanto a freqüência exigida é de 132 MHz.

A Figura 4.16 apresenta uma forma de operação que não se preocupa com o melhor desempenho possível, adequada para ser utilizada com vídeos de menor resolução ou para codificação que não exige tempo real. É possível notar que a ME continua operando mesmo após a conclusão da busca intra-quadro, dessa forma mais ciclos de relógio são gastos em troca de uma melhor qualidade de predição. O número de ciclos depende na área de busca explorada até um limite de 48x48, lembrando que cada bloco candidato utiliza 1 ciclo de relógio para o processamento, uma vez que a latência já fora preenchida.

Na Figura 4.17 é apresentada a temporização do segundo macro-estágio de *pipeline*, O tempo de processamento do CAVLC é variável de acordo com os resíduos a serem codificados. É importante perceber que esse estágio não é crítico em termos de desempenho se comparado ao primeiro estágio e opera com sobra de ciclos de relógio. Conforme definido pelo padrão, o número máximo de bits utilizados para um macrobloco é de 3200 bits, mesmo se considerarmos que todos são gerados pelo CAVLC, isso consumiria 400 ciclos de relógio, já que a arquitetura proposta gera um byte por ciclo. A unidade de escrita na memória, mostrada na Figura 4.17, é responsável por atualizar a memória de referência. Esse módulo não foi detalhado ao longo do texto devido a sua simplicidade. Esse módulo foi projetado para futuramente implementar o filtro de redução de efeitos de bloco, cuja implementação é facultativa no codificador e, portanto, foi uma das simplificações adotadas.



(a)

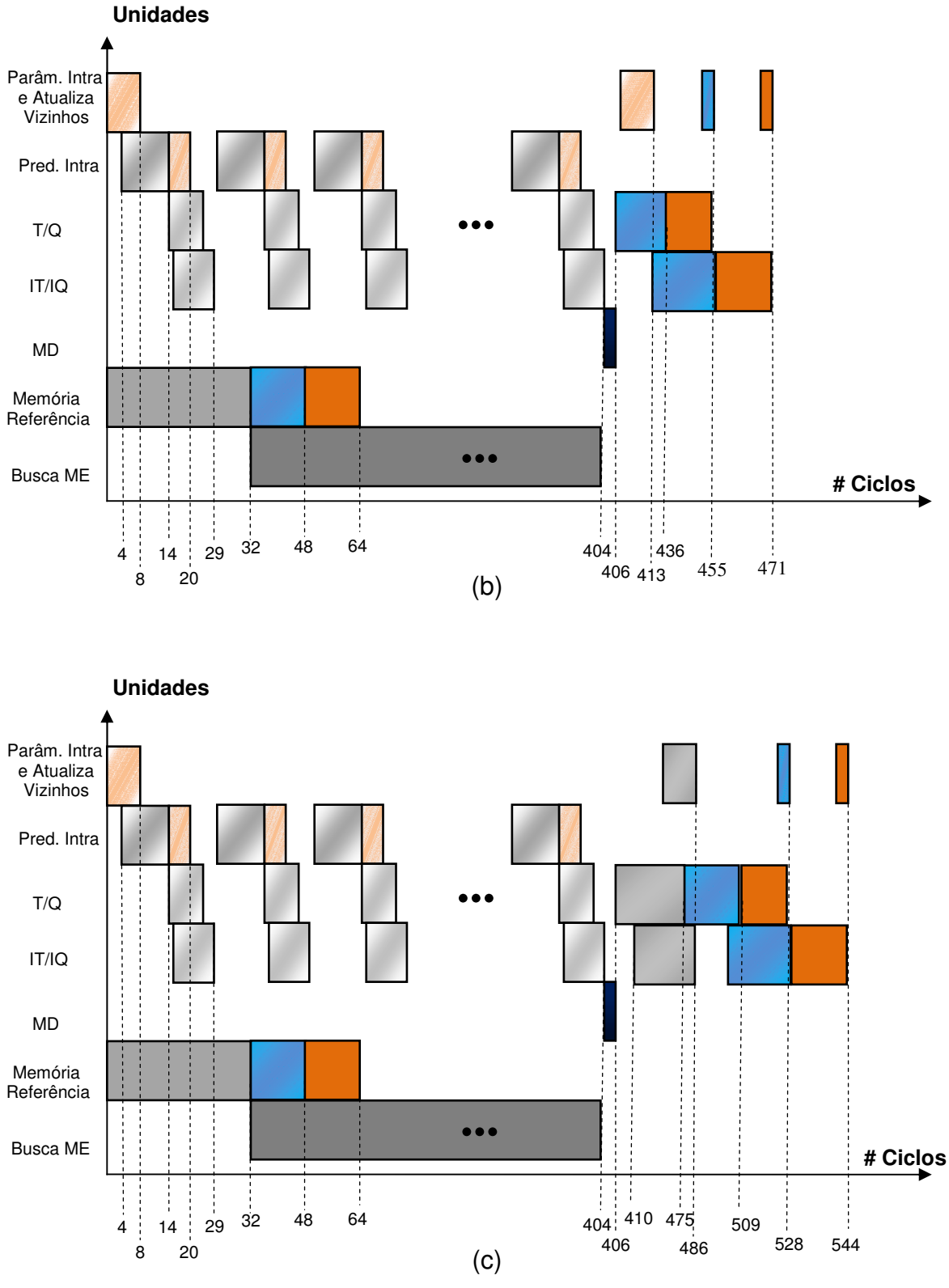


Figura 4.15: Diagrama de Tempo do Primeiro Macro-Estágio de *Pipeline* do Codificador quando Escolhido (a) Modo I16MB, (b) Modo I4MB, (c) Inter

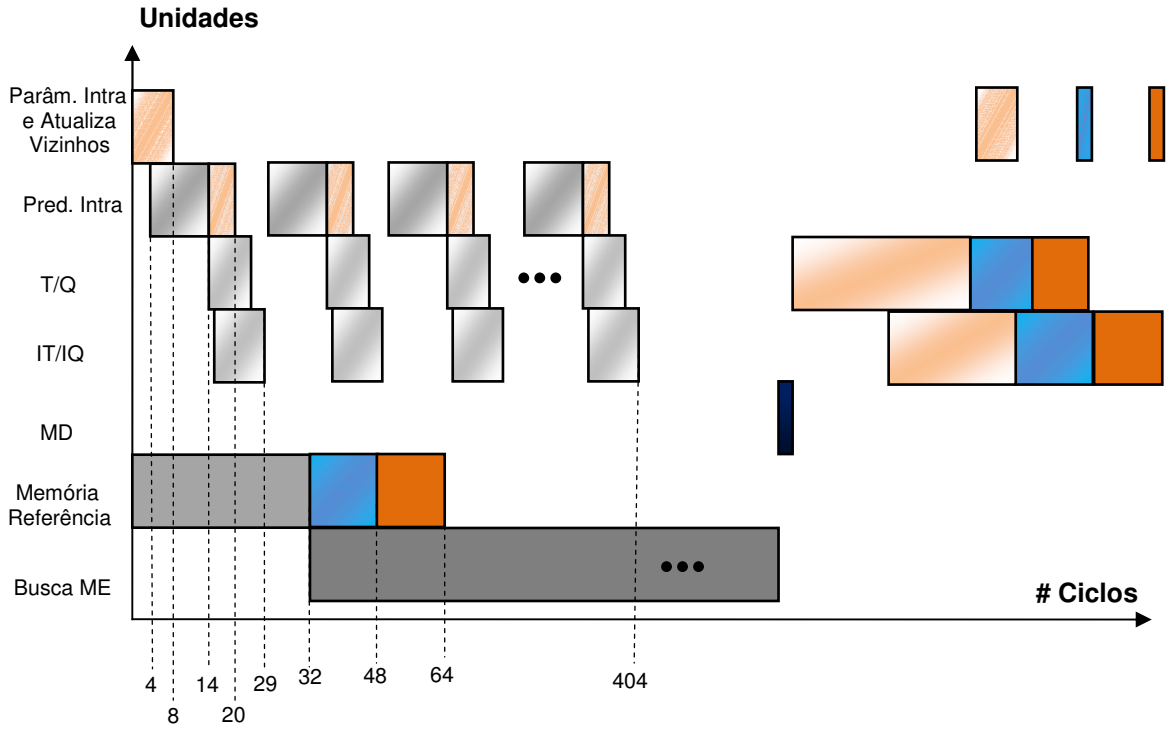


Figura 4.16: Diagrama de Tempo do Primeiro Macro-Estágio de *Pipeline* do Codificador Explorando maior Área de Busca da ME

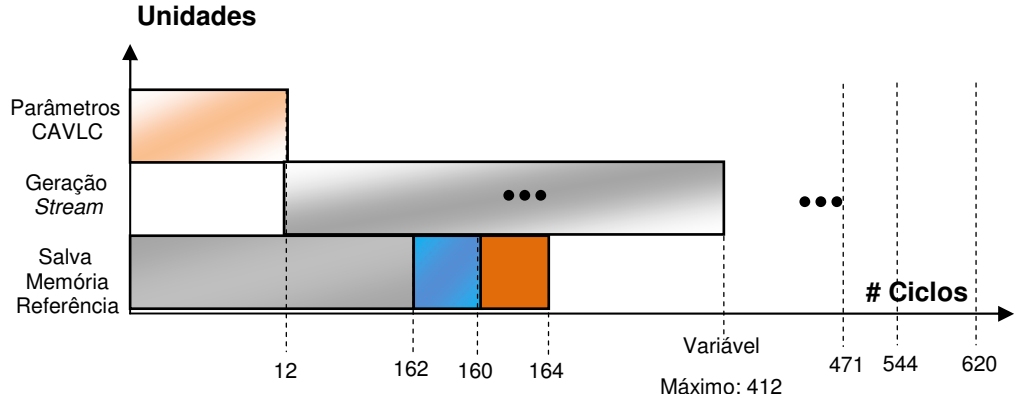


Figura 4.17: Diagrama de Tempo do Segundo Macro-Estágio de *Pipeline* do Codificador

Sob o ponto de vista de comunicação entre os diferentes módulos, foram levantados os tráfegos de dados em cada ponto de comunicação. A Figura 4.18 apresenta as conexões analisadas sendo identificadas por uma letra. A Tabela 4.1 apresenta a vazão de dados em cada um desses pontos para codificação de vídeos HDTV 1080p a 30 quadros por segundo. É previsível que o ponto crítico em termos de comunicação encontra-se entre a ME e a memória externa, ainda mais se considerarmos a possibilidade de utilizar outras ferramentas do padrão, como os múltiplos quadros de referência, que aumentam em muito a vazão de dados nesse ponto. A intensa comunicação sugere o projeto de uma solução de hierarquia de memória que reduza a largura de banda de memória, esse projeto, no entanto, é encarado como um trabalho futuro que utilizará o modelo desenvolvido como plataforma para experimentos.

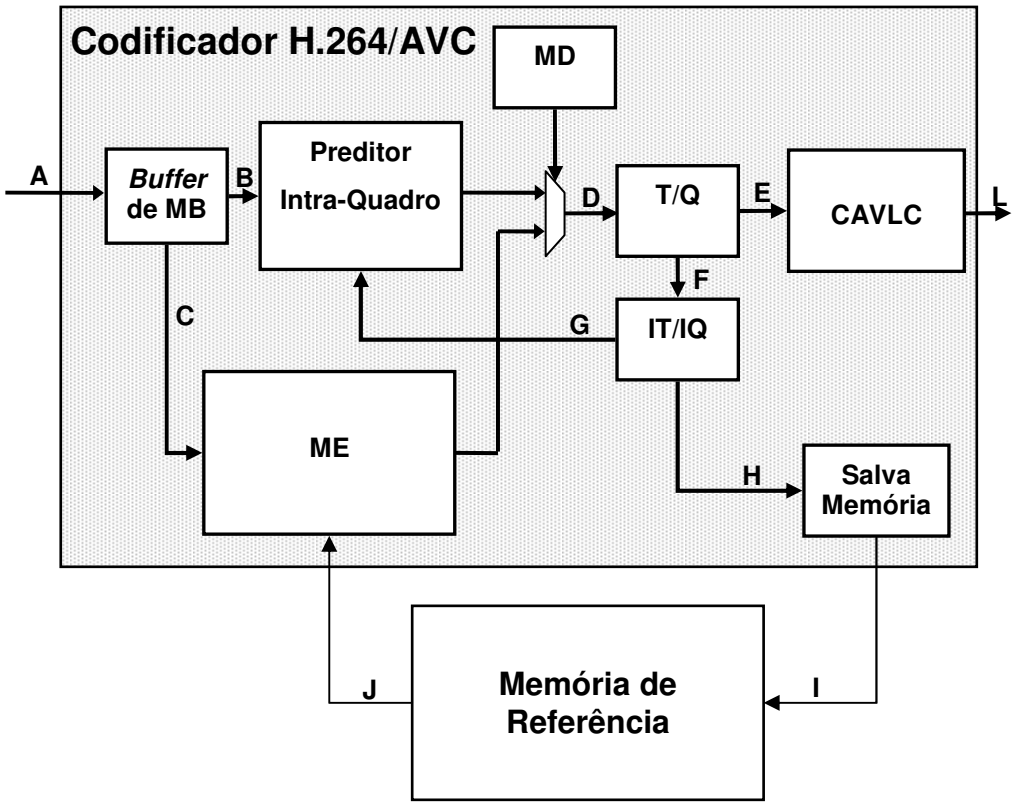


Figura 4.18: Arquitetura do Codificador com as Conexões Analisadas

Tabela 4.1: Resultados de Tráfego de Dados para Vídeos 1920x1080 @ 30fps

Conexão	Tráfego de Dados
A	93,3 MB/s
B	93,3 MB/s
C	93,3 MB/s
D	93,3 – 155,5MB/s
E	93,3 MB/s
F	93,3 MB/s

Conexão	Tráfego de Dados
G	29.6 – 36,9 MB/s
H	93,3 MB/s
I	93,3 MB/s
J	812,8 MB/s
L	Variável

Não foi encontrado na literatura, até o presente momento, nenhum trabalho que descreva uma modelagem em alto nível de um *hardware* para o codificador, ou mesmo para o decodificador, de vídeo H.264/AVC. Um trabalho associado foi encontrado em (AMER, 2004), no entanto o mesmo se preocupa unicamente em analisar a comunicação entre os diferentes módulos do codificador descritos como *software*. Além disso, (AMER, 2004) utiliza um particionamento dos módulos do codificador totalmente diferente da abordagem proposta nessa dissertação. Seu particionamento utiliza a estrutura utilizada pelo *software* de referência do padrão. Trabalhos que apresentam a implementação de *hardware* do codificador são encontrados na literatura, como em (HUANG, 2005).

4.4 Considerações Finais sobre a Modelagem do Codificador H.264/AVC

Este capítulo apresentou, em detalhes, a modelagem desenvolvida pra uma arquitetura de *hardware* dedicada à codificação de vídeos H.264/AVC projetada para codificar seqüências de vídeo HDTV 1920x1080 em tempo real com uma taxa de 30 quadros por segundo.

A modelagem foi descrita utilizando a linguagem SystemC em cerca de 15.000 linhas de código. Essa modelagem se encaixa na nova abordagem definida pelo grupo de pesquisas para desenvolvimento do codificador de vídeo, conforme abordado no início deste capítulo.

A modelagem alcançou o objetivo principal de chegar a uma implementação funcional de um codificador, embora assumindo diversas restrições de codificação, que serve de auxílio no estudo do sistema como um todo e no desenvolvimento do *hardware* final, tanto dos módulos isolados como de sua integração, desde o projeto da arquitetura até a verificação de seu funcionamento.

Além dos resultados práticos obtidos, a modelagem rendeu uma publicação no VLSI-Soc 2008 entitulada “SystemC Modeling of an H.264/AVC Intra Frame Encoder Architecture” (ZATT, 2008b). Nesse texto são apresentados os resultados da modelagem do preditor intra-quadro e dos blocos T/Q e IT/IQ.

5 CONCLUSÃO E TRABALHOS FUTUROS

Esta dissertação foi descrita em duas partes principais e apresentou, em sua primeira parte, o desenvolvimento da arquitetura do compensador de movimento para decodificadores de vídeo segundo o padrão H.264/AVC. A segunda parte apresentou a modelagem de uma arquitetura de *hardware* para codificação de vídeo segundo o padrão H.264/AVC. Foram apresentados os conceitos básicos da codificação e decodificação de vídeo digital segundo o padrão H.264/AVC.

Uma arquitetura para a compensação de movimento para decodificadores de vídeo foi desenvolvida. A arquitetura, denominada HP422-MoCHA (*High Profile 4:2:2 Motion Compensation Hardware Architecture*), baseada na arquitetura MoCHA (*Motion Compensation Hardware Architecture*) (AZEVEDO, 2007), suporta a compensação de movimento para o perfil *High 4:2:2* 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 à 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 largura de banda com a memória externa de aproximadamente 60%.

A arquitetura desenvolvida tem algumas limitações na implementação de todas as funcionalidades descritas no padrão H.264 para o perfil *High 4:2:2*. A arquitetura do compensador considera que um quadro não contém mais que um *slice*.

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.

Na literatura atual não foi encontrada nenhuma solução para a compensação de movimento no perfil *High 4:2:2* do padrão H.264/AVC, apenas podemos encontrar trabalhos que resolvem este mesmo problema nos perfis *Main* e *Baseline*. Comparações com os trabalhos associados foram devidamente apresentadas.

Uma nova estrutura para interpolação de amostra na compensação de movimento foi proposta, sendo que sua versão para o Perfil *Main* se mostra 17% mais compacta, em

termos de *gates*, que a solução mais compacta encontrada na literatura sem degradar sua performance.

As dificuldades de integração do decodificador de vídeo, para o qual a arquitetura MoCHA foi originalmente desenvolvida, foram abordadas dando base a uma nova abordagem no projeto do codificador H.264/AVC. Essa nova abordagem, baseada no conceito *meet-in-the-middle*, deu origem ao trabalho de modelagem do codificador, apresentado nessa dissertação.

O modelo da arquitetura de codificação foi detalhado, bem como cada um de seus módulos componentes. A descrição utilizou a linguagem SystemC e consumiu aproximadamente 15.000 linhas de código, apesar das diversas restrições impostas ao modelo. Seu projeto foi desenvolvido com o objetivo de codificar vídeo H.264/AVC segundo o perfil *Main* do padrão com desempenho para codificar vídeos 1920x1080 em tempo real, a 30 quadros por segundo.

A modelagem alcançou o objetivo principal de chegar a uma implementação funcional de um codificador, embora assumindo diversas restrições de codificação, permitindo a caracterização temporal e de comunicação do codificador. Dessa forma, o modelo se mostra uma poderosa ferramenta para o desenvolvimento do sistema de codificação em HW, desde a etapa de projeto até a verificação final.

Não foi encontrado na literatura, até o presente momento, nenhum trabalho que descreva uma modelagem em alto nível de um *hardware* para o codificador, ou mesmo para o decodificador, de vídeo H.264/AVC. Um trabalho associado foi encontrado, no entanto o mesmo se preocupa unicamente em analisar a comunicação entre os diferentes módulos do codificador descritos como *software*.

A contribuição desta dissertação, além dos dois trabalhos implementados anteriormente sugeridos, inclui o desenvolvimento de uma plataforma que poderá contribuir muito com o grupo de pesquisa e com outros experimentos futuros.

Como trabalhos futuros podem ser listados uma vasta gama de possíveis explorações a serem feitas no modelo do codificador sob os pontos de vista algorítmico e arquitetural. Sob o ponto de vista algorítmico, a estimação de movimento e o modo de decisão são os blocos com maior número de opções de projeto a serem discutidas. A exploração de diferentes algoritmos de busca, novas ferramentas de predição, diferentes critérios de similaridade impulsionam a pesquisa na área da ME, enquanto no modo de decisão têm-se buscado simplificações a fim de reduzir a complexidade do RDO, principalmente simplificações voltadas a *hardware.* No que diz respeito a alternativas arquiteturais, é importante lembrar que a arquitetura modelada representa apenas uma alternativa entre tantas possíveis, de forma que muitas outras alternativas podem ser exploradas e diversas modificações podem ser feitas mesmo sobre a arquitetura proposta. Por exemplo, a separação da ME em diferentes estágios de *pipeline* onde um ou mais são responsáveis pela busca intera (amostras originais da imagem) e outro pela busca nos pixels fracionários, já encontra-se nos planos como trabalho futuro. Além disso, existem blocos codificador ainda não implementados que deverão ser modelados, como o CABAC e o filtro redutor de efeitos de bloco.

REFERÊNCIAS

AGOSTINI, L. V. **Desenvolvimento de Arquiteturas de Alto Desempenho Dedicadas à Compressão de Vídeo Segundo o Padrão H.264/AVC.** 2007. 172 f. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

AGOSTINI, L. V. ; AZEVEDO, A.; STAEHLER, W. ; ROSA, V. ; ZATT, B.; PINTO, A. C.; PORTO, R.; BAMPI, S. ; SUSIN, A. Design and FPGA Prototyping of a H.264/AVC Main Profile Decoder for HDTV. **Journal of the Brazilian Computer Society**, Rio de Janeiro, v. 12, n. 4, p. 25-36, jan. 2002.

AMER, I.; BADAWY, W.; JULLIEN, G**.** A Design Flow For An H.264 Embedded Video Encoder. In: INTERNATIONAL CONFERENCE ON INFORMATION AND COMMUNICATION TECHNOLOGY, ICICT, 3., 2005, Cairo. **Enabling Technology Technology for the New Knowledge Society**: proceedings. Piscataway: IEEE, 2005. p. 505-513.

AMER, I. et al**.** On The Way to an H.264 HW/SW Reference Model: a SystemC Modeling Strategy to Integrate Selected IP-Blocks with the H.264 Software Reference Model. In: IEEE WORKSHOP ON SIGNAL PROCESSING SYSTEMS DESIGN AND IMPLEMENTATION, 2005. **Proceedings…**[S.l.]: IEEE, 2005. p.178-181.

AZEVEDO, A.; ZATT, B.; AGOSTINI, L. V.; BAMPI, S. MoCHA: a Bi-Predictive Motion Compensation Hardware for H.264/AVC Decoder Targeting HDTV. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2007, New Orleans. **Proceedings. . .** New York: IEEE, 2007. p.1617-1620.

AZEVEDO, A.; ZATT, B.; AGOSTINI, L.; BAMPI, S. Motion Compensation Sample Processing for HDTV H.264/AVC Decoder. In: IEEE NORCHIP, 23., 2005. **Proceedings. . .** [S.l.: s.n.], 2005. p.110-113.

BHASKARAN, V.; KONSTANTINIDES, K. **Image and Video Compression Standards:** Algorithms and Architectures. 2nd ed. Boston: Kluwer Academic Publishers, 1997.

CAI, L.; GAJSKI, D. Transaction Level Modeling: An Overview. In: IEEE/ACM/IFIP INTERNATIONAL CONFERENCE ON HARDWARE/SOFTWARE CODESING AND SYSTEM SYNTHESIS AND CONFERENCE FOR SYSTEM-LEVEL DESING, CODES+ISSS, 2003. **Proceedings…** [S.l.:s.n.], 2003. p.19-24.

CHEN J. et al. Low Complexity Architecture Design of H.264 Predictive Pixel Compensator for HDTV Applications. In: IEEE INTERNATIONAL CONF. ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, 3., 2006. **Proceedings…** France: IEEE, 2006. p.932-938.

CHEN, T.; HAUNG, Y.; CHEN, L. Fully Utilized And Reusable Architecture For Fractional Motion Estimation of H.264/AVC. In: IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, 2004. **Proceedings…** [S.l.:s.n.], 2004. p.9-12.

CHU, C. et al. Hierarchical Global Motion Estimation/Compensation in Low Bitrate Video Coding. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 1997. **Proceedings…** New York: IEEE, 1997. p. 1149-1152.

DENG, L. et al. An Efficient VLSI Architecture for MC Interpolation in AVC Video Coding. In: INTERNATIONAL MULTICONFERENCE IN COMPUTER SCIENCE AND COMPUTER ENGINEERING, 2004. **Proceedings…** [S.l.:s.n.], 2004.

GNU G++. GCC, the GNU Compiler Collection. Disponível em: <http://gcc.gnu.org/> Acesso em: nov. 2008.

GTKWave Electronic Waveform Viewer. Disponível em: <http://home.nc.rr.com/gtkwave/>. Acesso em: nov. 2008.

HUANG, Y. W. et al. Analysis, Fast Algorithm, and VLSI Architecture Design for H.264/AVC Intra Frame Coder. **IEEE Transactions Circuits and Systems for Video Technology**, v. 15, n. 3, p. 378-401, Mar. 2005.

HUANG, Y.W. et al. A 1.3TOPS H.264/AVC Single-Chip Encoder for HDTV Applications**.** In: IEEE INTERNATIONAL SOLID-STATE CIRCUITS CONFERENCE, ISSCC, 2005. **Proceedings…** [S.l.]: IEEE, 2005.

IMAGE PROCESSING – IP Homepages. Disponível em: <http://iphome.hhi.de/suehring/tml/>. Acesso em: jun. 2008.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO/IEC 11172 - MPEG-1 (11/1993)**: coding of moving pictures and associated audio for digital storage media up to about 1.5Mbit/s – part 2: video. [S.l.], 1993.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO/IEC 14496-2 - MPEG-4 Part 2 (01/1999)**: coding of audio visual objects – part 2: visual. [S.l.], 1999.

INTERNATIONAL TELECOMMUNICATION UNION. **H.264 AVC Fidelity Range Extensions**. JVT-L050. [S.l.], 2004.

INTERNATIONAL TELECOMMUNICATION UNION. ITU-T Home. Disponível em: <www.itu.int/ITU-T/>. Acesso em: jun. 2008.

INTERNATIONAL TELECOMMUNICATION UNION. **ITU-T Recommendation H.261 v1 (11/90):** video codec for audiovisual services at px64 kbit/s. [S.l.], 1990.

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

INTERNATIONAL TELECOMMUNICATION UNION. **ITU-T Recommendation H.263 v3 (11/00):** video coding for low bit rate communication. [S.l.], 2000.

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

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

INTERNATIONAL TELECOMMUNICATION UNION. Joint Video Team (JVT). Disponível em: <http://www.itu.int/ITU-T/studygroups/com16/jvt/>. Acesso em: jun. 2008.

JAIN, J.; JAIN, A. Displacement Measurement and Its Application in Interframe Image Coding. **IEEE Transactions on Communications**, [S.l.], v. 29, n. 12, p. 1799-1808, Dec. 1981.

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

KEUTZER, K. et al. System-Level Design: Orthogonalization of Concerns and Platform-Based Design. **IEEE Transactions on Computer-Aided Design Of Integrated Circuits And Systems**, [S.l.], v. 19, n. 12, p. 1523-1543, Dec. 2000.

KROUPIS, N. et al. A Modified Spiral Search Motion Estimation Algorithm and its Embedded System Implementation. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2005. **Proceedings…** [S.l.]: IEEE, 2005. p. 3347-3350.

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

KWON, S. et al. Overview of H.264/AVC / MPEG-4 Part 10. In: INTERNATIONAL CONFERENCE ON MULTIMEDIA, INTERNET AND VIDEO TECHNOLOGIES, WSEAS, 2005. **Proceedings…** [S.l.:s.n.], 2005.

LI, C.; CHEN, C.; SU, W. A Unified Systolic Architecture for Combined Inter and Intra Predictions in H.264/AVC Decoder. In: INTERNATIONAL CONFERENCE ON COMMUNICATIONS AND MOBILE COMPUTING, 2006. **Proceedings…** New York: ACM, 2006. p. 73-78.

LI, R. et al. A New Three-Step Search Algorithm for Block Motion Estimation. **IEEE Transactions on Circuits and Systems for Video Technology,** [S.l.], v. 4, n. 4, p. 438-442, Aug. 1994.

LI, Y. et al. Memory Cache Based Motion Compensation Architecture for HDTV H.264/AVC Decoder. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, 2007. **Proceedings…** [S.l.:s.n.], 2007. p. 2906-2909.

LIE W.-N. et al. Hardware-Efficient Computing Architecture for Motion Compensation Interpolation in H.264 Video Coding. In: IEEE INTERNATIONAL SYMPOSIUM SYMPOSIUM ON CIRCUITS & SYSTEMS, 2005. **Proceedings…** [S.l.:s.n.], 2005. p. 2136-2139.

MALVAR, H. et al. 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.

MARPE, D. et al. Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 13, n. 7, p. 620-636, July 2003.

MENTOR GRAPHICS. **Leonardo Spectrum**. Disponível em: <http://www.mentor.com/products/fpga_pld/synthesis/leonardo_spectrum/>. Acesso em: jun. 2008.

MENTOR GRAPHICS. **ModelSim**. Disponível em: <http://www.model.com>. Acesso em: jun. 2008b.

MIANO, J. **Compressed Image File Formats:** JPEG, PNG, GIF, XBM, BMP. New York: Addison-Wesley, 1999.

OSCI: Open SystemC Initiative. Disponível em: <http://www.systemc.org/>. Acesso em: jun. 2008.

PAN, F. et al. Fast Mode Decision Algorithm for Intraprediction in H.264/AVC Video Coding. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 15, n. 7, p. 813-822, July 2005.

PORTO, M. S.; BAMPI, S.; SUSIN, A. A.; AGOSTINI, L. V. Architectural design for the new QSDS with dynamic iteration control motion estimation algorithm targeting HDTV. In: ANNUAL SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEM DESIGN, 2008. **Proceedings…** [S.l.:s.n.], 2008. p. 216-221.

PURI, A. et al. Video Coding Using the H.264/MPEG-4 AVC Compression Standard. **Elsevier Signal Processing: Image Communication**, [S.l.], n. 19, p.793–849, 2004.

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

ROSE, A. et al. Transaction Level Modeling in SystemC. In: Synopsys User Group Conference, 2002. Diponível em: httpdyn.cs.huji.ac.il/moodles/old/file.php?file=/94/Papers/tlm_whitepaper.pdf.

SAHAFI, L. **Context-Based Complexity Reduction Applied to H.264/AVC Video Compression**. 2005. 70 f. Thesis (Master in Applied Science) – School of Engineering Science, Simon Frase University, Canada.

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

SULLIVAN, G. et al. The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions. In: CONFERENCE ON APPLICATIONS OF DIGITAL IMAGE PROCESSING, 27., 2004. **Proceedings…** [S.l.:s.n.], 2004.

SULLIVAN, G.; WIEGAND, T. Rate-Distortion Optimization for Video Compression. **IEEE Signal Processing Magazine**, [S.l.], v. 15, p. 74-90, Nov. 1988.

TOURAPIS, M. et al. Fast Motion Estimation Using Circular Zonal Search. In: SPIE SYMPOSIUM OF VISUAL COMMUNICATIONS AND IMAGE PROCESSING, VCIP, 1999. **Proceedings…** [S.l.:s.n.], 1999. v. 2, p. 1496-1504.

WANG R. et al. High Throughput And Low Memory Access Sub-Pixel Interpolation Architecture For H.264/Avc Hdtv Decoder. **IEEE Trans. on Consumer Electronics**, [S.l.], v. 51, n. 3, p. 1006- 1013, Aug. 2005.

WANG R. et al. Motion Compensation Memory Accesss Optimization Strategies For H.264/AVC Decoder. In: IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING, 2005. **Proceedings…** [S.l.], 2005a. p. 97-100.

WANG S. et al. A Platform Based MPEG-4 Advanced Video Coding (AVC) Decoder with Block Level Pipelining. In: INTERNATIONAL CONFERENCE ON INFORMATION AND COMMUNICATIONS SECURITY, 2003. **Proceedings…** [S.l.:s.n.], 2003.

WANG, S.-Z. et al. 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.], 2005b. p.4558-4561.

WIEGAND, T. et al. 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. et al. 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 2003a.

WU, D. et al. Block INTER Mode Decision for Fast Encoding Of H.264. In: IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, 2004. **Proceedings…** [S.l.: s.n.], 2004. p.181-184.

XILINX INC. **Platform Studio and the EDK.** Disponível em: <http://www.xilinx.com/ise/embedded_design_prod/platform_studio.htm>. Acesso em: jun. 2008.

XILINX INC. **Virtex-II Pro and Virtex-II Pro X Platform FPGAs:** Complete Data Sheet. [S.l.], 2006. Disponível em: <www.xilinx.com>. Acesso em: jun. 2008a.

XILINX INC. **Xilinx University Program – Virtex-II Pro Development System – Hardware Reference Manual**. [S.l.], 2005. Disponível em: <www.digilentinc.com>. Acesso em: jun. 2008b.

XILINX INC. **Xilinx**: The Programmable Logic Company. Disponível em: <www.xilinx.com>. Acesso em: jun. 2008c.

YI, X.; LING, N**.** Rapid Block-Matching Motion Estimation Using Modified Diamond Search Algorithm. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2005. **Proceedings…** [S.l.]: IEEE, 2005. p. 5489-5492.

YU, L.; ABDI, S.; GAJSKI, D. H.264 TLM in SystemC for Shared Bus Platform. Irvine, UC, 2007. Base de dados fechada do Gigascale System Research Center, acesso ao artigo diretamente com os autores.

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, 12., 2006, San Jose, Costa Rica. **XII Workshop IBERCHIP**. San Jose: Universidad de Costa Rica S.R,L., 2006. p.83-86.

ZATT, B.; AGOSTINI, L.; SUSIN, A.; BAMPI, S. HP422-MoCHA: An H.264/AVC High Profile Motion Compensation Architecture for HDTV. In: INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2008, Seattle. **Proceedings. . .** [S.l.]:IEEE, 2008. p.25-28.

ZATT, B.; AGOSTINI, L.; SUSIN, A.; BAMPI, S. High Throughput Architecture for H.264/AVC Motion Compensation Sample Interpolator for HDTV. In: SBCCI, 2008. **Proceedings. . .** New York: ACM, 2008a. p.228-232.

ZATT, B.; DINIZ, C.M.; AGOSTINI, L.; SUSIN, A.; BAMPI, S. SystemC Modeling of an H.264/AVC Intra Frame Encoder Architecture. In: VLSI-SoC, 2008. **Proceedings. . .** [S.l.: s.n.], 2008b.

APÊNDICE A RELATÓRIO DE DISCIPLINA SOBRE UMA ARQUITETURA DE PREDIÇÃO INTRA-QUADRO

Arquitetura de Alto Desempenho de um Codificador Intra-Quadro para vídeos de alta definição conforme o padrão H.264/AVC

Bruno Zatt, Cláudio Machado Diniz, Sergio Bampi
Universidade Federal do Rio Grande do Sul - UFRGS
Instituto de Informática
Porto Alegre, RS - Brazil
{bzatt, cmdiniz, bampi}@inf.ufrgs.br

Universidade Federal do Rio Grande do Sul - UFRGS
Departamento de Eng. Elétrica - DELET
Porto Alegre, RS - Brazil
altamiro.susin@ufrgs.br

***Abstract*—Este trabalho apresenta o desenvolvimento de um codificador intra-quadro conforme o padrão H.264/AVC. Uma solução arquitetural é proposta e para o codificador intra-quadro, sendo esta implementada em VHDL e sintetizada para Xilinx FPGA. A arquitetura foi validada e resultados de PSNR foram avaliados para vídeos de resoluções QCIF e HD 720p. A síntese para FPGA mostra que a arquitetura atinge o throughput necessário para processar vídeos de alta definição (HDTV 1080p) em tempo real, a 30 quadros por segundo.**

1. Introdução

O H.264/AVC é o mais novo padrão de compressão de video, proposto por membros da ITU-T e ISO/IEC, formando o JVT. Este padrão pode produzir uma taxa de compressão de até 50% em comparação com o MPEG-2, ao preço de um acréscimo de complexidade de aproximadamente quatro vezes. A grande contribuição à elevada taxa de compressão do padrão H.264/AVC é atribuída à predição inter-quadros, também conhecida como estimação de movimento. Este tipo de predição procura reduzir a redundância temporal, procurando padrões entre blocos de quadros diferentes. A estimação de movimento é aplicada somente em macroblocos tipo P (preditivo), contidos em slices tipo P (preditivo).

A predição intra-quadro é uma das inovações do padrão H.264/AVC, por ser realizada no domínio espacial. Ela é feita com base nas amostras vizinhas espaciais do macrobloco, que foram anteriormente processadas. Este tipo de predição busca eliminar a redundância espacial presente em um quadro do vídeo, reduzindo significantemente os valores que serão transformados e quantizados. A predição intra-quadro atua principalmente em slices tipo I, onde não possuem macroblocos que são preditos usando a estimação de movimento.

A complexidade do codificador intra-quadro, somada à necessidade de codificar vídeos de alta-definição em tempo real, justifica o projeto de uma arquitetura de alto desempenho, em hardware, para o codificador intra-quadro em conformidade com o padrão H.264/AVC.

Na seção II, os conceitos sobre a predição intra-quadro são introduzidos. A seção III apresenta a arquitetura proposta para um codificador intra-quadro. A seção IV mostra a estratégia de validação desta arquitetura, bem como alguns resultados de quadros de vídeo codificados utilizando a arquitetura. Na seção V são mostrados os resultados síntese e comentários. A

seção VI apresenta as conclusões e sugestões de trabalhos futuros.

2. Predição Intra-Quadro

No H.264/AVC, o vídeo é representado no espaço de cores YCbCr (luminância, crominância azul e crominância vermelha). A relação de sub-amostragem de cor mais comum é o 4:2:0, onde para cada 4 amostras de luminância existe uma amostra de cada crominância. O macrobloco é formado por um bloco de 16x16 amostras de luminância e dois blocos de 8x8 amostras de crominância. A predição intra-quadro para luminância é feita sobre os macroblocos (Intra 16x16) ou sobre os 16 blocos de 4x4 amostras (Intra 4x4) contidos em um macrobloco. Os dois tipos de predição são ilustrados na figura 1. Na figura 1.b, os números indicam a ordem de codificação dos blocos 4x4, conhecida como duplo-Z. A predição para croma, no caso da sub-amostragem 4:2:0 é feita sobre blocos de 8x8 amostras. Cada tipo de predição contém modos que geram predições para diferentes padrões contidos no quadro. A predição é feita para cada bloco, baseado nas amostras vizinhas já processadas do quadro.

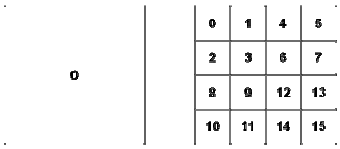


Figure 1. Modos de predição. a) Intra 16x16 b) Intra 4x4

A. Predição para blocos 4x4

O padrão define 9 modos de predição para blocos 4x4, para padrões horizontais, verticais e diagonais com diferentes ângulos de inclinação. A predição 4x4 é muito adequada para blocos com maior nível de detalhe. A figura 2 ilustra as direções dos 9 modos. Os blocos em cinza correspondem ao bloco 4x4 que está sendo predito no momento, a partir dos vizinhos, representados pelas amostras A-M. Nos modos 0 (vertical) e 1 (horizontal) o bloco predito é construído pela cópia das amostras vizinhas A-D e I-L, respectivamente. O modo 2 (DC) faz uma média das amostras A-D e I-L e copia o resultado para todas amostras do bloco predito. Os modos 3 a 8 são calculados por médias ponderadas das amostras vizinhas, dependendo das direções das setas e da posição da amostra no bloco predito.

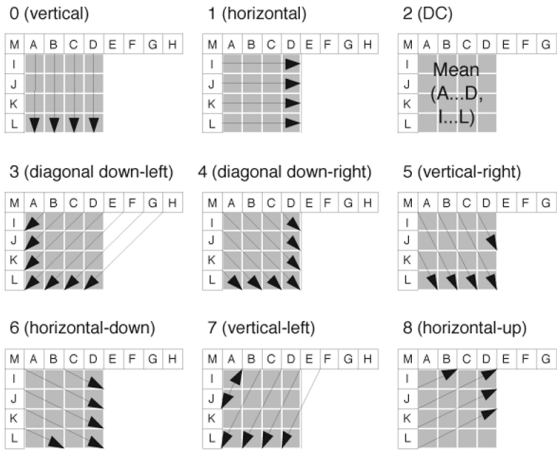


Figure 2. Modos de predição 4x4

B. Predição para blocos 16x16 e 8x8

Para blocos de luminância 16x16 são definidos 3 modos: vertical (0), horizontal (1), DC (2) e planar (3). Para blocos 8x8 de crominância os modos são os mesmos, porém a numeração é diferente: DC (0), horizontal (1), vertical (2) e planar (3). Este tipo de predição é adequada para regiões mais homogêneas, pois o bloco predito é 16 vezes maior que na predição 4x4. A figura 3 ilustra as direções dos 3 modos. Os modos vertical, horizontal e DC são semelhantes aos modos definidos para blocos 4x4, com a diferença que são 32 amostras vizinhas para luminância e 16 amostras vizinhas para crominância, ao contrário das 8 amostras no Intra 4x4.

A predição Intra 16x16 usa somente os vizinhos acima e à esquerda do macrobloco, ao contrário dos modos 4x4 que usam vizinhos na diagonal direita acima do bloco. O modo planar é o mais complexo de ser calculado, por necessitar do cálculo de três parâmetros (a,b e c) como mostrado em (1), (2) e (3) sendo que o cálculo de (2) e (3) dependem dos parâmetros H e V definidos em (4) e (5)

$$a = 16 \cdot (p[-1,15] + p[15,-1]) \quad (1)$$

$$b = (5 \cdot H + 32) \gg 6 \quad (2)$$

$$c = (5 \cdot V + 32) \gg 6 \quad (3)$$

$$H = \sum_{x'=0}^{7}(x'+1)\cdot(p[-1,8+x'] - p[-1,6-x']) \quad (4)$$

$$V = \sum_{y'=0}^{7}(y'+1)\cdot(p[8+y',-1] - p[6-y',-1]) \quad (5)$$

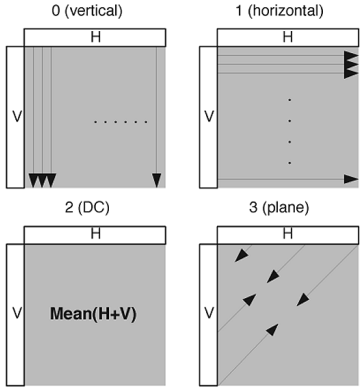


Figure 3. Modos de predição 16x16

A posição [y,x] representa a amostra contida na linha y e coluna x do macrobloco, sendo que [0,0] representa o canto superior esquerdo do macrobloco. A amostra p[y,x] representa a amostra vizinha, sendo que os índices [y,x] estão numerados relativos à amostra do canto superior esquerdo do macrobloco. A predição do modo planar é calculada, a partir dos parâmetros previamente calculados, como

$$P[y,x] = Clip((a + b\cdot(x-7) + c\cdot(y-7) + 16) \gg 5 \quad (6)$$

onde P[y,x] é a predição da amostra [y,x] do macrobloco e Clip é uma função de saturação para adequar os valores na faixa de representação da imagem (0 a 255, para amostras de 8 bits).

C. Disponibilidade dos vizinhos

A predição intra-quadro é feita com base nos vizinhos espaciais do macrobloco (quando a predição é Intra 16x16) ou do bloco 4x4 (quando a predição é Intra 4x4). O padrão H.264/AVC define os vizinhos de um macrobloco ou bloco de acordo com a figura 4, onde CurrMbAddr é o macrobloco ou bloco que está sendo predito, mbAddrD é a amostra vizinha do canto superior esquerdo (corresponde ao vizinho M da figura 2), mbAddrB é a vizinhança superior (amostras A-D para blocos 4x4, como na figura 1 ou H para macroblocos, como na figura 3) , mbAddrA é a vizinhança da esquerda (I-L para blocos 4x4, como na figura 1 ou V para macroblocos, como na figura 3) e mbAddrC é a vizinhança da diagonal direita, usada somente para predição 4x4 (amostras E-H, como na figura 2).

Existem situações em que a vizinhança A, B, C ou D pode não estar disponível, como nas fronteiras do quadro ou quando o bloco 4x4 precisa de um vizinho que ainda não foi processado. Na maioria dos casos, quando a vizinhança necessária para o processamento de um modo de predição não está disponível, esta predição não é realizada. As exceções a esta regra são:

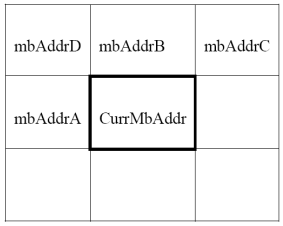


Figure 4. Vizinhos de um macrobloco ou partição 4x4

- Na predição DC, quando somente uma vizinhança não é disponível (A ou B), a média é feita sobre as amostras da outra vizinhança. Se nenhuma das vizinhanças é disponível, todas amostras recebem o valor 128 (centro da escala, 0-255);
- Nas predições diagonais que usam a vizinhança C, para blocos 4x4 diferentes de 3 ou 11 (ver figura 1), se a vizinhança C não é disponível, a amostra mais à direita da vizinhança B é replicada para as amostras da vizinhança C.

D. Modo de decisão

Depois de realizada a predição de um macrobloco ou bloco, este deve comparado ao macrobloco ou bloco correspondente no vídeo original, para obtenção dos resíduos de predição. Os resíduos são a diferença ponto a ponto entre as amostras originais e as preditas. Os resíduos da predição é que serão enviados posteriormente para as etapas de transformada e quantização, para posteriormente serem processados pela codificação de entropia. No decodificador, a imagem original será reconstruída com base nos resíduos da predição e no modo de predição escolhido. Como se deseja ter uma boa compressão, é preciso enviar a menor quantidade de informação (resíduos e informação de modo) possível para o decodificador. Quanto mais

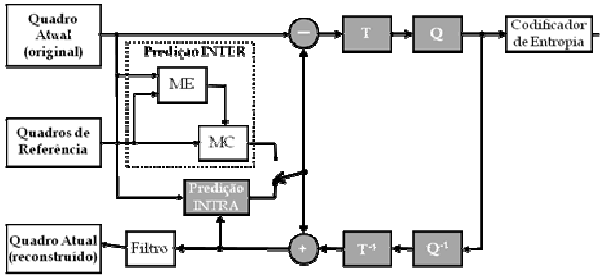
similar o bloco predito é do bloco original, menor a quantidade de resíduos gerada pela predição.

Portanto, no codificador, deve-se escolher o modo de predição que produza um bloco predito mais semelhante possível com o bloco correspondente no vídeo original. Para isto, todas as predições são processadas para que o modo de decisão avalie qual a predição que gera o menor quantidade de informação na saída. Isto é uma das tarefas mais complexas do codificador, que fazem parte do módulo de controle do codificador. Uma métrica muito usada para avaliar a quantidade de informação gerada por um modo de predição é o SAD (*Sum of Absolute Differences).* Ela consiste no somatório das diferenças ponto a ponto entre as amostras do bloco original e do bloco predito. Pode-se avaliar, por exemplo, qual dos modos 4x4 gera a menor quantidade de resíduo, avaliando qual o menor SAD entre eles.

Quando à decisão se o macrobloco será predito com os modos 16x16 ou com os modos 4x4 é um pouco mais complicada. Esta decisão não pode ser resolvida com o SAD, pois os modos 4x4 sempre irão gerar menos resíduo que os modos 16x16, devido à granularidade dos modos 4x4 ser 16 vezes menor e possuir mais modos de predição. Porém, pelo fato da imagem estar particionada em 4x4, é preciso enviar a informação do modo escolhido para cada bloco 4x4, ou seja, são 16 informações de modo a serem enviadas para o decodificador. Ainda, pelo fato de possuir 9 modos, cada informação de modo possui 4 bits para ser representada, ao contrário dos 2 bits necessários para representar a informação de modo na predição 16x16. Ou seja, em regiões homogêneas o volume de informações na predição 16x16 é significativamente maior que na predição 4x4. Portanto, esta decisão deve ser baseada na saída da codificação de entropia, onde as informações de resíduo e de modo já foram codificadas.

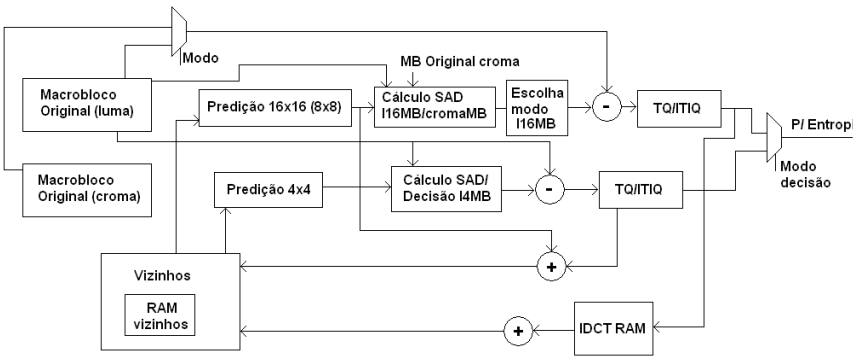
E. Laço de Codificação

Assim que o modo de um bloco foi escolhido, os resíduos para este modo devem passar pelas etapas hachuradas na figura 5, para que este bloco sirva de referência (vizinho) para a predição do próximo bloco. Isto forma um laço de codificação, onde existe uma dependência de dados entre blocos 4x4 vizinhos. Para atingir a taxa de dados na saída é preciso garantir que a latência dos blocos que estão no laço respeite a taxa de blocos na saída do bloco Q, após completada a codificação. Quando a predição é Intra 4x4, depois do modo de decisão 4x4 os resíduos deste bloco podem passar para a etapa T e Q. O caso crítico é quando a escolha se dá pelo Intra 16x16, e esta escolha só pode ser feita depois de todo loop dos 16 blocos 4x4. Neste caso se deve transformar os resíduos do modo 16x16 escolhido, em partições de 4x4 (são 256 valores de resíduo.

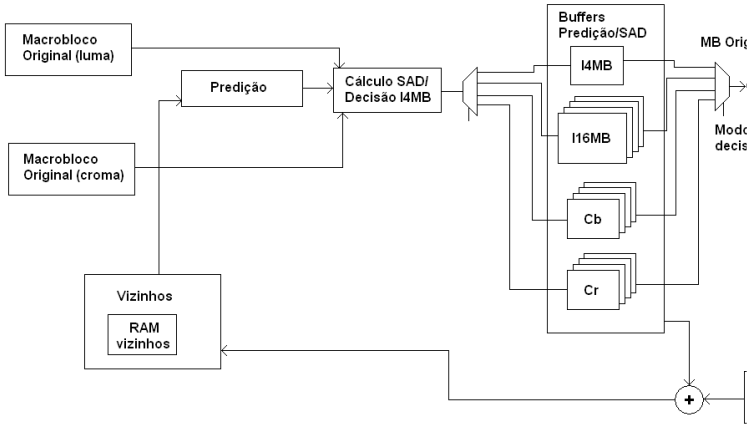


3. Arquitetura do Codificador Intra-Quadro

A arquitetura do codificador intra-quadro partiu da definição dos seguintes requisitos: O codificador intra-quadro deve realizar a codificação, para vídeos com resolução 1080p a 30 quadros/s; o hardware deve ter a menor área que atinja este desempenho. Em seguida, partiu-se para a definição da arquitetura e posteriormente para o desenvolvimento dos seguintes módulos: Preditor, que realiza o cálculo da predição intra-quadro; Cálculo do SAD, que faz o cálculo das diferenças entre as amostras do bloco original e do bloco predito e as acumula; Escolha do modo, que compara os SADs calculados para e escolhe o modo que possui menor SAD; Módulo Intra Vizinhos, que possui uma memória RAM para guardar as amostras já processadas, que servem de vizinhas para predição dos próximos blocos. Neste último módulo também está implementada a parte controle do codificador intra-quadro, pois ele possui informações de leitura e escrita dos vizinhos e faz parte do início e final do processo de codificação intra.



a)Primeira solução arquitetural



b)Arquitetura proposta definida

Figure 6. Arquiteturas

A. Investigação de Arquiteturas

A primeira arquitetura proposta para o codificador intra-quadro baseia-se em 2 caminhos de dados, um para cada tipo de predição (Intra 4x4 e Intra 16x16), como mostrado na figura 6.a. Previa-se também um módulo de T/Q mais eficiente para a parte crítica do laço de codificação, que processa blocos 4x4.

Avaliando o funcionamento desta arquitetura, percebe-se que, devido à dependência de dados entre blocos 4x4, o módulo de predição 4x4 ficaria ocioso enquanto este bloco estivesse sendo processado pelas etapas T/Q, esperando este ser reconstruído para obter seus vizinhos para a próxima predição. Logo, uma nova alternativa de arquitetura seria mais eficiente. Esta é mostrada na figura 6.b.

Nesta solução há somente um caminho de dados para a predição. Quando um bloco 4x4 é predito e já está na etapa T/Q, pode ser calculado ao mesmo tempo a predição deste bloco considerando modos 16x16. Logo, a predição Intra 16x16 é feita em partições de blocos 4x4, no intervalo entre as predições dos 9 modos 4x4. Como são 3 modos somente, em paralelo pode ser feita a predição das crominâncias, de forma intercalada (Cb ou Cr) para aproveitar o caminho de dados. Apesar disto é preciso criar buffers de predição para guardar as predições dos 4x4 e 16x16 enquanto a decisão ainda não foi realizada. Ainda assim, a redução de área de um caminho de dados completo é significativa, sendo que não há perda de desempenho.

B. Módulo Preditor

O módulo preditor realiza a predição de blocos 4x4 ou partições 4x4 de blocos 16x16. A cada ciclo é produzida a predição de quatro amostras na saída para os 9 modos, de forma que, ao final de quarto ciclos é feita a predição de um bloco 4x4 para todos os modos. Isto é atingido devido ao grau de paralelismo utilizado, pois a predição para cada amostra pode ser feita independentemente. A figura 7 mostra um diagrama do preditor. Para modos horizontais e

verticais (0, 1) as amostras da vizinhança são copiadas para a linha predita (4 amostras em cinza). Modos diagonais necessitam de cálculo para a predição, feitas pelos blocos PEs. Os blocos PEs implementam uma das equações mostradas em (7), (8) e (9). O modo planar necessita de uma unidade PE especial, que implementa as operações mostradas em (1) a (6).
A equação depende da amostra que está sendo calculada no momento, de forma que foram utilizados o mínimo de PEs necessárias para calcular todos os modos em paralelo e muxes foram colocados nas entradas das PEs para selecionar as amostras, dependendo da linha (0, 1, 2 ou 3) do bloco que está sendo tratada. Os muxes são controlados por uma simples máquina de estados.

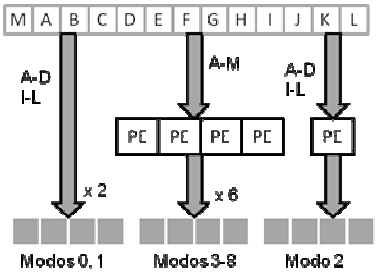


Figure 7. Diagrama do Preditor

$$p = (a + 2 \cdot b + c + 2) \gg 2 \quad (7)$$

$$p = (g + 3 \cdot h + 2) \gg 2 \quad (8)$$

$$p = (l + k + 1) \gg 1 \quad (9)$$

C. Módulo de cálculo do SAD e Decisão 4x4

O módulo de cálculo do SAD foi implementado com uma árvore de somadores, que faz a soma das quatro amostras da saída do preditor, um registrador para guardar o resultado da soma e mais um somador para adicionar este valor com as próximas amostras. Este módulo trabalha em pipeline com o modulo de predição: assim que as 4 amostras foram preditas, o módulo de cálculo do SAD dispara a árvore de somadores, e o módulo preditor passa a calcular as próximas 4 amostras. Este módulo é feito em um pipeline de 3 estágios para redução do atraso combinacional dos somadores. Foram instanciados 9 módulos de cálculo de SAD (PE0 a PE8), um para cada modo.
O módulo de decisão 4x4 consiste numa árvore de comparadores, que tem como entrada o SAD de cada modo e a informação da disponibilidade do modo, calculada com base na disponibilidade dos vizinhos, fornecido pelo módulo que calcula os vizinhos. A saída é a informação do modo escolhido.

D. Módulo Intra Vizinhos

Este módulo contém a memória que guarda a vizinhança necessária para codificar os macroblocos de uma linha do quadro, bem como possui as máquinas de estados que controlam o sistema. A memória, ilustrada na figura 9, guarda uma linha de amostras vizinhas a serem usadas na próxima linha de macroblocos para predição. No estágio mostrado na figura, o conteúdo da memória é composto pelos vizinhos dos macroblocos 0-7, que ainda não foram processados, e dos macroblocos 8-11, que já foram processados. Cada vez que um macrobloco é reconstruído, a linha superior usada como referência neste passo é substituída pela inferior. Esta memória, para vídeos de resolução 1920x1080 (1080p) e amostras de 8 bits possui um tamanho de 1920 bytes.

O módulo de controle do codificador intra-quadro possui duas máquinas de estados (FSM). Uma FSM faz a leitura dos vizinhos da memória, cálculo dos valores DC e cálculo dos parâmetros a, b e c do modo planar, bem como disparar o processo de decodificação. Esta máquina possui 44 estados, porém os estados são bem simplificados e a máquina tem um caráter seqüencial. A outra FSM possui a função de salvar as amostras reconstruídas na memória de vizinhos, e possui 13 estados.

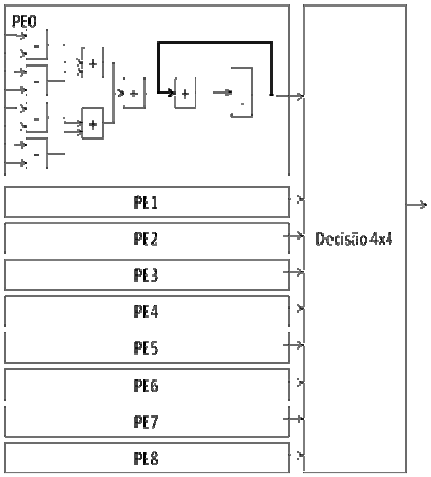


Figure 8. Diagrama dos Módulo de Cálculo do SAD e Decisão 4x4

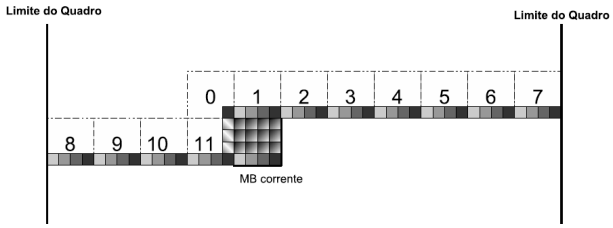


Figure 9. Memória de vizinhos

E. Memória de predição

A memória de predição (figura 10) guarda as predições para todos os modos, feito pelo preditor. Esta memória é necessária, pois devem ser calculados os resíduos da predição para o modo selecionado, que é decidido somente quando todo o processamento de um macrobloco termina.

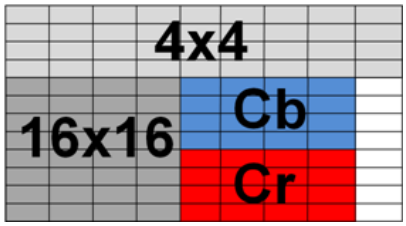


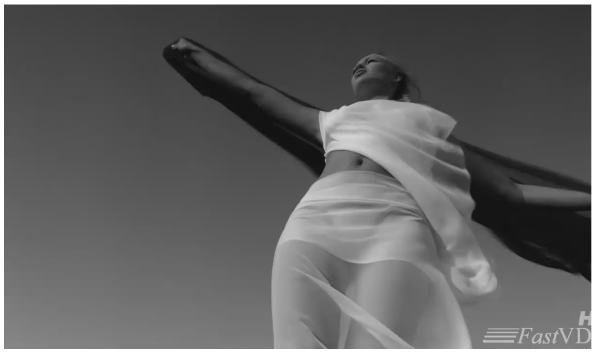
Figure 10. Organização da memória de predição

4.Validação

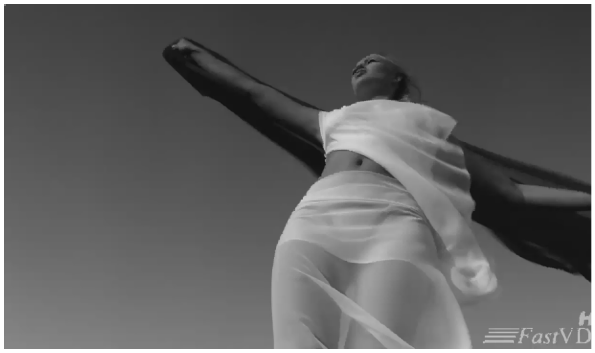
A validação da arquitetura foi realizada através de comparações entre as entradas e saídas da arquitetura, geradas por simulação, e as entradas e saídas de um modelo em software (MATLAB), que foi desenvolvido baseado na norma do padrão H.264/AVC. A simulação da arquitetura foi feita utilizando o software ModelSim, da Mentor. Além disto, dois quadros de tamanhos QCIF (176x144) e HDTV 720p (1280x720) foram codificados utilizando a arquitetura (por simulação) e o modelo, e as imagens reconstruídas foram avaliadas visualmente e através dos resultados de PSNR (*Peak Signal-Noise Ratio*) entre o quadro original e o predito.

a) b) c)

Figure 11. Vídeo Foreman QCIF a) Original b) Intra 4x4 c) Intra 16x16



a)



b)

c)

Figure 12. Vídeo Girl FVDO 720p a) Original b) Intra 4x4 c) Intra 16x16

Nas figuras 11.b, 11.c, 12.b e 12.c são mostradas as predições para 2 quadros de seqüências de vídeo QCIF e 720p utilizando o modo Intra 4x4 ou Intra 16x16, respectivamente. A decisão entre Intra 4x4 e 16x16 não foi implementada por depender de outros módulos do codificador que não estavam disponíveis em hardware e fogem do escopo deste trabalho. A tabela 1 mostra os resultados de PSNR para os dois vídeos. Nota-se que a predição 4x4 possui um PSNR maior, para os 2 vídeos, pois ela possui uma granularidade menor e acompanha melhor os detalhes das imagens. Percebe-se ainda que, para vídeos de mais alta resolução, a predição intra-quadro atua de forma melhor, resultando em um PSNR mais alto, produzindo menor quantidade de ruído relativo ao tamanho do quadro.

TABLE I. RESULTADOS DE PSNR

Video	**Predição utilizada**	
	Intra 4x4	***Intra 16x16***
Foreman QCIF	27,89 dB	17,74 dB
Girl FVDO 720p	41,96 dB	28,86 dB

5.Resultados de Síntese

Os resultados de síntese para Xilinx Virtex-II Pro FPGA, utilizando Xilinx ISE, para os módulos do codificador intra-quadro são mostrados na tabela II. Percebe-se que o módulo que possui maior área em lógica (LUTs) é o de vizinhos, devido às máquinas de controle e cálculos dos parâmetros DC e plano. A memória de vizinhos utiliza somente 3 BRAMs para armazenar os vizinhos de luma e croma. No modo de predição são usados 5 multiplicadores contidos no FPGA, para o cálculo do modo plano. A memória de predição é a maior, utilizando 8 BRAMs para armazenamento de todos os modos de predição para um bloco 4x4, um macrobloco 16x16

TABLE II. SÍNTESE DO MÓDULOS DO CODIFICADOR INTRA

Módulo	**Utilização do dispositivo**				
	Slices	***Slice Flip Flops***	***4 Input LUTs***	***BRAMs***	***Mult 18x18***
Vizinhos	2231	2049	3965	3	-
Predição	938	591	1661	-	5
SAD	660	603	1071	-	-
Mem. Pred	193	314	71	8	-

Device: Xilinx Virtex-II Pro VP30 FPGA

A tabela III mostra os resultados de síntese do codificador intra-quadro completo. Ele ocupou somente 31% da área em slices de um FPGA Virtex-II Pro VP30, e utilizou 8% das BRAMs disponíveis. O módulo completo atingiu uma freqüência de 112,818 MHz para este dispositivo. Foi feita uma síntese para *standard-cells,* utilizando o software Leonardo

Spectrum, da Mentor, utilizando a tecnologia TSMC 0.18um. O resultado de área foi de 48765 gates e atingiu uma freqüência de 133,4 MHz.

Na figura 13 é mostrado o diagrama de tempo, para o caso crítico do codificador. Para codificar vídeos de alta resolução (1080p), é necessário processar cada macrobloco em 464 ciclos em uma freqüência de 100 MHz. O diagrama mostra que a arquitetura processa um macrobloco a cada 451 ciclos, para o caso crítico onde a Intra 16x16 é escolhido, sendo preciso transformar todo macrobloco (16 blocos 4x4).

TABLE III. SÍNTESE DO CODIFICADOR INTRA COMPLETO

Módulo	Utilização do dispositivo		
	Usado	*Disponível*	*Utilização*
Slices	4308	13696	31%
Slice Flip Flops	3354	27392	12%
4 Input LUTs	7723	27392	28%
BRAMs	11	136	8%
Mult 18x18	10	136	7%

Device: Xilinx Virtex-II Pro VP30 FPGA

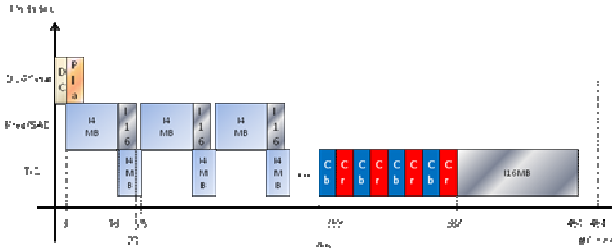


Figure 13. Diagrama de tempo

O diagrama ilustra o processamento no tempo de cada unidade, como cálculo do DC e parâmetros do modo planar, feito pelo módulo de controle (vizinhos), o processamento de um bloco 4x4 (I4MB) pela predição e cálculo de SAD e o processamento intercalado das partições 4x4 de macroblocos.

Com a freqüência atingida e o número de ciclos mostrado no diagrama (451), o codificador intra-quadro atinge um throughput de 248337 macroblocos/s. O throughput necessário para processar vídeos de alta definição HDTV 1920x1080 @ 30 Hz é de 243000 macroblocos/s. Sendo assim, a arquitetura atinge o desempenho necessário para processar vídeos de alta definição Full HDTV em tempo real.

6. Conclusões e Trabalhos Futuros

Este trabalho mostrou a proposta de uma arquitetura para o codificador intra-quadro do padrão H.264/AVC. Todos os módulos desta arquitetura foram implementados em VHDL e sintetizados para FPGA da Xilinx e para standard-cells, tecnologia TSMC 0.18um. A validação foi realizada utilizando simulações e comparação de arquivos com um modelo em software, bem como a avaliação PSNR dos quadros preditos pela arquitetura.

A arquitetura é toda realizada em pipeline e processa um macrobloco a cada 451 ciclos. A síntese para FPGA e standard-cells mostrou que a freqüência atingida, tendo com base o número de ciclos, possui o throughput necessário para processar vídeos HDTV 1920x1080 (1080p) em tempo real, a 30 quadros/s. Os resultados de área são suficientes para integrar o codificador intra-quadro em um codificador H.264/AVC completo em hardware, em FPGAs mais atuais, como Xilinx Virtex-4.

Como trabalhos futuros pretende-se otimizar a arquitetura proposta, com o objetivo de diminuir a área e potência, mantendo o desempenho para HDTV.

7. References

[1] Huang, Y.-W., Bing-Yu Hsieh, Tung-Chien Chen, and Liang-Gee Chen. Analysis, Fast Algorithm, and VLSI Architecture Design for H.264/AVC Intra Frame Coder. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, VOL. 15, NO. 3, MARCH 2005.

[2] Kibum Suh, Seongmo Park, and Hanjin Cho; An Efficient Hardware Architecture of Intra Prediction and TQ/IQIT Module for H.264 Encoder.

[3] *Genhua Jin, Jin-Su Jung and Hyuk-Jae Lee;* An Efficient Pipelined Architecture for H.264/AVC Intra Frame Processing.

[4] Yu-Wen Huang, Bing-Yu Hsieh, Tung-Chien Chen, and Liang-Gee Chen; HARDWARE ARCHITECTURE DESIGN FOR H.264/AVC INTRA FRAME CODER;

[5] Jeyun Lee and Byeungwoo Jean; Fast mode decision for H.264.

[6] Yu-Wen Huang, Bing-Yu Hsieh, Tung-Chien Chen, and Liang-Gee Chen*;* Analysis, Fast Algorithm, and VLSI Architecture Design for H.264/AVC Intra Frame Coder.

[7] Chao-Chung Cheng, Tian-Sheuan Chang; Fast Three Step Intra Prediction Algorithm for 4x4 blocks in H.264.

[8] JVT Editors. ITU-T H.264/AVC Recommendation (03/2005), 2005