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

VAGNER SANTOS DA ROSA

Arquiteturas de Hardware Dedicadas para Codificadores de Vídeo H.264 – Filtragem de Efeitos de Bloco e Codificação Aritmética Binária Adaptativa a Contexto

Tese apresentada como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação

Prof. Dr. Sergio Bampi
Orientador

Porto Alegre, Outubro de 2010.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Rosa, Vagner Santos da

Arquiteturas de Hardware Dedicadas para Codificadores de Vídeo H.264 – Filtragem de Efeitos de Bloco e Codificação Aritmética Adaptativa ao Contexto [manuscrito] / Vagner Santos da Rosa. – 2010.

171 f.:il.

Orientador: Sergio Bampi.

Tese (Doutorado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2010.

1. H.264. 2. Hardware 3.FPGA 4. ASIC, Deblocking Filter, Filtro de Deblocagem 5. CABAC e CABAD. 6. Prototipação. I. S. Bampi, Orientador da. III. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos Alexandre Netto
Vice-Reitor: Prof. Rui Vicente Oppermann
Pró-Reitor de Pós-Graduação: Prof. Aldo Bolten Lucion
Diretor do Instituto de Informática: Prof. Flávio Rech Wagner
Coordenador do PPGC: Prof. Álvaro Freitas Moreira
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

AGRADECIMENTOS

O período de doutorado foi para mim um período de muitas dificuldades, porém cheio de conquistas e realizações que vão muito além desta tese e o apoio recebido da família, dos amigos e colegas foram muito importantes. Dedico este espaço para agradecer a cada uma destas pessoas que em muito contribuíram para minha formação, meu caráter e minha vida.

Agradeço aos meus pais, José e Rosane, que desde muito cedo investiram na minha educação, me deram suporte para chegar onde estou; que me guiaram nos momentos de decisão e me incentivaram no momento das conquistas. Agradeço a minha avó Celia, por toda preocupação, carinho e dedicação.

Agradeço também a minha esposa, Giovana, que me deu amor, carinho e dois filhos maravilhosos, Otávio e Laura, que amo muito.

Ao meu orientador Prof Sergio Bampi pelos conselhos, orientação e apoio nas mais diversas situações inclusive as não relacionadas com a realização deste trabalho.

Ao meu amigo Prof. Luciano Agostini que me apoiou, incentivou e influenciou em muitos momentos, que me fez olhar muitas coisas de uma nova perspectiva.

Aos professores Altamiro Susin, Ricardo Reis e Eduardo Costa, que sempre me apoiaram e acreditaram no desenvolvimento deste trabalho.

Aos meus tantos amigos e colegas que estiveram envolvidos neste trabalho: Dieison Deprá, Cláudio Diniz, Bruno Zatt, Thaísa Leal, Ana Pinto, Marcelo Porto, Roger Porto, Tatiana Santos, Leandro Max, Franco Valdez, dentre outros que tiveram participações importantes e decisivas neste trabalho, através de discussões, conselhos e revisões deste trabalho.

Aos professores do Departamento de Matemática da Universidade Federal do Rio Grande, pelo apoio dado para a conclusão deste trabalho através do afastamento para doutorado.

Finalmente agradeço a todos aqueles que direta ou indiretamente contribuíram para este trabalho de forma menor, porém não menos importante e peço que me perdoem se, por omissão esqueci alguém importante.

Muito obrigado a todos!

SUMÁRIO

LISTA DE FIGURAS 9
LISTA DE ABREVIATURAS 15
RESUMO 21
ABSTRACT 23
1 Introdução 25
1.1 Proposta de trabalho 26
1.2 Trabalhos relacionados no grupo 27
2 Conceitos Básicos de Compressão de Vídeo Digital 29
2.1 O Olho Humano 29
2.2 Representação de imagens digitais 31
2.3 Espaço de Cores 31
2.4 Subamostragem de cores 35
2.5 Redundância de Dados na Representação de Vídeos 36
2.6 Codificadores/decodificadores de vídeo para compressão 38
3 O padrão H.264 51
3.1 Introdução ao Padrão H.264 51
3.1.1 Histórico do Padrão 51
3.1.2 Definições 52
3.1.3 Perfis e Níveis 54
3.1.4 Formato de Dados Codificados 56
3.1.5 Comparação com Padrões Anteriores 56
3.2 O Codec H.264 59
3.2.1 Os Blocos da Estimação e Compensação de Movimento (ME/MC) 62
3.2.2 O Bloco de Predição Intra 65
3.2.3 O Bloco das Transformadas Diretas (T) e Transformadas Inversas (T^{-1}) 66
3.2.4 O Bloco da Quantização Direta (Q) e Quantização Inversa (Q^{-1}) 67

3.2.5 O Bloco do Filtro de Deblocagem 68
3.2.6 O Bloco de Codificação de Entropia 70
3.3 Escalabilidade no H.264 75
3.3.1 Dimensões da escalabilidade 77
3.4 Múltiplas visões no H.264 78
3.5 Principais Desafios 80
4 ARQUITETURA PARA O Filtro de deblocagem 87
4.1 Algoritmo do Filtro de Deblocagem 87
4.2 Arquitetura proposta 89
4.3 Operação do filtro 92
4.4 Saída RGB 94
4.5 Suporte ao perfís HIGH, SVC e MVC 95
4.6 Implementação 95
4.7 Resultados em FPGA 96
4.8 Resultados em ASIC 97
4.9 Comparação com a literatura 99
4.10 Conclusões 100
5 Arquiteturas para o codificador e decodificador aritméticos do CABAC 101
5.1 Visão geral do CABAC 101
5.1.1 O processo de binarização e debinarização 103
5.1.2 Modelagem de probabilidade 104
5.1.3 O codificador aritmético binário (BAC) 105
5.1.4 O decodificador aritmético binário (BAD) 108
5.1.5 Paralelização acima do nível de *Slice* para o CABAC e CABAD 109
5.2 Arquitetura para o decodificador aritmético binário adaptativo ao contexto (CABAD) 110
5.2.1 Análise estatística do fluxo de dados no CABAD 111
5.2.2 Proposta arquitetural para os motores de decodificação do CABAD .. 113
5.2.3 Resultados experimentais 115
5.2.4 Conclusões 116
5.3 Proposta arquitetural para o CABAC 116
5.3.1 Proposta arquitetural para o codificador aritmético do CABAC 118
5.3.2 Proposta arquitetural para o codificador aritmético do CABAC 122
5.3.3 Resultados 129
5.3.4 Codificador CABAC completo 131

5.3.5 Conclusões do CABAC 133
6 Plataforma de validação 135
6.1.1 Prototipação dos módulos do H.264 136
6.1.2 Validação funcional 136
6.1.3 Validação na frequência nominal de operação 137
6.1.4 Detalhamento da arquitetura da plataforma de validação 138
6.1.5 Aplicação da metodologia 140
6.1.6 Conclusões 142
7 Conclusões 143
Referências 147
Apêndice A Estudo do Hardware de Prototipação 155
A.1 Hardware do kit da Digilent. 155
A.1.1 Configurando o software para a placa Digilent 157
A.2 Sistema Sundance 157
A.2.1 Módulos 158
A.2.2 Placas Carrier 159
A.2.3 Mecanismos de interconexão 161
A.2.4 Placa SMT395E-VP70 164
A.2.5 Instalação do software Sundance 167

LISTA DE FIGURAS

Figura 1.1: Decodificador H.264 – Módulos desenvolvidos pelo grupo. 28
Figura 1.2: Codificador H.264 – Módulos desenvolvidos pelo grupo. 28
Figura 2.1: Vista em corte de um olho humano. 29
Figura 2.2: Visão de uma seção da retina. 30
Figura 2.3: (a) Imagem digital e (b) região de 16×16 pixels. 31
Figura 2.4: Decomposição de uma imagem RGB em suas componentes. 32
Figura 2.5: *Gamut* possível no espaço de cores sRGB. 33
Figura 2.6: Componentes de cor no formato RGB e YCbCr para uma região de uma mesma imagem. 34
Figura 2.7: Localização da informação de crominância nos formatos sub-amostrados de representação de cores. 35
Figura 2.8: Representação YCbCr no formato 4:2:0. 36
Figura 2.9: Um codificador elementar genérico: Apenas codificação de entropia... 38
Figura 2.10: Exemplo de cena natural. 38
Figura 2.11: Transformada DCT. 40
Figura 2.12: Codificador com Transformada DCT. 40
Figura 2.13: Comparação resultado da quantização (a) sem quantização (a); (b) com passo de quantização elevado. 41
Figura 2.14: Codificador com Transformada e Quantização. 41
Figura 2.15: Codificação diferencial: Produção de resíduo. 42
Figura 2.16: Histogramas de duas imagens subsequentes de uma sequência de vídeo com resolução de 176x144 pixels. 43
Figura 2.18: Vetores de movimento gerados pela busca de blocos de 8x8 pixels no quadro anterior da sequência de vídeo sobrepostos na imagem atual. 44
Figura 2.19: Resíduo calculado utilizando estimação/compensação de movimento. 45
Figura 2.20: Resíduo calculado utilizando estimação de movimento. 45
Figura 2.21: Histograma do resíduo usando o processo de ME. 46
Figura 2.22: Troca de cena na codificação diferencial. 46

Figura 2.23: Codificador de vídeo DPCM com ME. 47

Figura 2.24: Modelo inicial de compressor de vídeo híbrido. 47

Figura 2.25: Compressor de vídeo híbrido com laço de reconstrução 48

Figura 2.26: Modelo completo de um codificador de vídeo. 48

Figura 2.27: Modelo de um decodificador de vídeo. 49

Figura 3.1: Perfis *Baseline*, *Main*, *Extended* e *High* do padrão H.264 55

Figura 3.2: Comparação (a) dos ganhos na taxa de bits e (b) na economia na taxa de bits comparada o MPEG-2, para o vídeo "Foreman" considerando resolução de vídeo para *streaming* 58

Figura 3.3: Comparação (a) dos ganhos na taxa de bits e (b) na economia na taxa de bits comparada o MPEG-2, para o vídeo "Mother & Daughter" considerando resolução de vídeo para vídeo conferência (QCIF). 58

Figura 3.4: Comparação (a) dos ganhos na taxa de bits e (b) na economia na taxa de bits comparada o MPEG-2, para o vídeo "Entertainment" considerando resolução de vídeo padrão (SD). 58

Figura 3.5: Diagrama em blocos de um codificador H.264. 60

Figura 3.6: Diagrama em blocos de um decodificador H.264. 60

Figura 3.7: Divisão do macrobloco em partições de macroblocos. (RICHARDSON, 2003) 62

Figura 3.8: Divisão de uma partição de macrobloco em partições de sub-macroblocos. (RICHARDSON, 2003) 62

Figura 3.9: Estimação de movimento com precisão de frações de pixel (AGOSTINI, 2007). 63

Figura 3.10: Uso de múltiplos quadros de referência (AGOSTINI, 2007). 64

Figura 3.11: Identificação das amostras para a predição intra (AGOSTINI, 2007). 65

Figura 3.12: Nove modos da predição intra para blocos de luma 4x4 (AGOSTINI, 2007) 65

Figura 3.13: Quatro modos da predição intra para blocos de luma 16x16 (AGOSTINI, 2007) 66

Figura 3.14: Ordem de processamento de amostras pelo bloco T. 67

Figura 3.15: Ordem de filtragem de bordas em um macrobloco. 69

Figura 3.16: Amostras adjacentes para bordas verticais e horizontais. 69

Figura 3.17: Matriz de elementos da imagem (a) amostras (b) após processamento pelo bloco ME. 70

Figura 3.18: Ordem ziguezague de leitura dos blocos 4x4 para a codificação de entropia. 71

Figura 3.19: Exemplo de codificação aritmética 75

Figura 3.20: Estrutura de um codificador escalável (SVC) com 3 camadas. 76

Figura 3.21: Codificação hierárquica diádica com GOP de 8 e 4 níveis temporais. 77

Figura 3.22: Relação genérica entre uma camada base e uma de enriquecimento... 78

Figura 3.23: Estrutura de predição temporal entre visões usando quadros B hierárquicos. 79

Figura 3.24: Esquema de ordenamento *Time-First Coding Order*. 79

Figura 3.25: Estrutura de codificação um laço ME/MC com ICA baseado em macroblocos (VETRO, 2008). 80

Figura 3.26: Análise para diferentes implementações de estimação de movimento [reproduzido de Huang (2006)]. 83

Figura 4.1: Posições das bordas para um dado bloco de 4x4 pixels em um macrobloco de 16x16 pixels. 88

Figura 4.2: Convenções do Filtro de Deblocagem. 88

Figura 4.3: Enumeração dos blocos de luma/croma e diagrama de cores de macroblocos. 90

Figura 4.4: Sequência de blocos esperada na entrada do Filtro de Deblocagem. 90

Figura 4.5: Filtro de borda. 90

Figura 4.6: Arquitetura do filtro de borda. 91

Figura 4.7: Encapsulamento do filtro de borda. 91

Figura 4.8: Arquitetura proposta para o Filtro de Deblocagem. 92

Figura 4.9: Sequência de blocos de saída do filtro. 94

Figura 4.10: Saída RGB e para a memória de referência a partir do Filtro de Deblocagem desenvolvido. 94

Figura 4.11: Saída do Filtro de Deblocagem para um quadro da sequência FOREMAN_QCIF, codificada como Intra. 96

Figura 4.12: Layout do Filtro de Deblocagem, sem memórias (SILVA, 2010). 98

Figura 4.13: Layout final do Filtro de Deblocagem (SILVA, 2010). 99

Figura 5.14: Fluxo de dados no CABAC. 102

Figura 5.1: Origem dos elementos sintáticos. 103

Figura 5.2: Algoritmo para cálculo dos valores iniciais do contexto 105

Figura 5.3: Intervalo inicial (a) e após vários passos de codificação (b). 105

Figura 5.4: Processo de codificação aritmética para um *bin*. 106

Figura 5.5: Processo de divisão de intervalo. 107

Figura 5.6: Processo de renormalização. 108

Figura 5.7: Processo de codificação Bypass. 108

Figura 5.8: Processo de inserção de bits no *bitstream*. 108

Figura 5.9: Organização do BAD com os três tipos de motores produzindo 1 *bin* por ciclo. 110

Figura 5.10: Arquitetura para os motores de decodificação conforme concepção de Yu (2005). 113
Figura 5.11:Arquitetura proposta para o BAD. 114
Figura 5.12: Arquitetura do motor regular (DEPRÁ, 2009). 115
Figura 5.13: Proposta arquitetural para o codificador CABAC. 117
Figura 5.14: Proposta arquitetural o núcleo do CABAC. 118
Figura 5.15: Sequenciamento da inicialização dos contextos. 119
Figura 5.16: Pipeline para o cálculo do valor inicial dos contextos. 120
Figura 5.17: Alternativas de implementação do CABAC, destacando alternativas para "Um *Bin*". 122
Figura 5.18: Pipeline do rLPS. 124
Figura 5.19: Interface do codificador aritmético proposta. 124
Figura 5.20: Arquitetura para a divisão do intervalo. 125
Figura 5.21: Arquitetura sequencial para a renormalização. 125
Figura 5.22: Arquitetura "*Pipeline Outstanding*" para aceleração da renormalização. 126
Figura 5.23: Histograma da produção de bits para um conjunto de *bins* de uma amostra de video real. 127
Figura 5.24: Histograma da produção de bits para um conjunto de *bins* de uma amostra de video real – desprezando o motor *bypass*. 128
Figura 5.25: Arquitetura de **Ciclo Único** para a aceleração da renormalização. 128
Figura 5.26: Arquiretura do Binarizador-Contextualizador. 132
Figura 5.27: Estrutura hierárquica do RTL do CABAC, usando o codificador aritmético **Ciclo Único**. 132
Figura 6.1: Abordagem para a validação funcional do protótipo (ROSA, 2007). 137
Figura 6.2: Validação em velocidade plena (ROSA, 2007). 137
Figura 6.3: Detalhes da arquitetura da plataforma de prototipação (ROSA, 2007). 138
Figura 6.4: Prototipação do módulo da compensação de movimento. 140
Figura 6.5: Exemplo de prototipação da integração de alguns módulos do codificador H.264 (ROSA, 2007). 141
Figura 6.6: Prototipação do Filtro de Deblocagem. 141
Figura A.1: Foto da placa XUP-V2P. 156
Figura A.2: Um sistema típico usando placas Sundance. 158
Figura A.3: Diagrama em blocos de uma placa TIM de processador. 159
Figura A.4: Diagrama simplificado de uma placa Carrier. 160
Figura A.5: Foto de um TIM ilustrando os conectores de interfaceamento. 161

Figura A.6: Completando a matriz de interconexão de *ComPorts* através de cabos e conectores FMS. 162
Figura A.7: Conector SDB. 163
Figura A.8: Conexão através de um cabo SHB. 163
Figura A.9: Conectores RSL. 164
Figura A.10: Diagrama em blocos da SMT395E. 165
Figura A.11: Foto da placa SMT395E. 166
Figura A.12: Sundance Wizard. 167
Figura A.13: Configuração do Code Composer Studio. 168
Figura A.14: Configurando a placa *carrier*, passo 1. 169
Figura A.15: Configurando a placa *carrier*, passo 2. 169
Figura A.16: Configuração do Code Composer Studio 170
Figura A.17: Tela inicial do Code Composer Studio quando há mais de uma placa de processador conectada à placa *carrier* 170
Figura A.18: Tela principal do Flash Programming Utility. 171
Figura A.19: Flash Programming Utility – Blocos de usuário. 171

LISTA DE ABREVIATURAS

AC	*Alternate Current* – Corrente Alternada
AC97	Padrão para codecs de áudio em computadores
ASIC	*Application Specific Integrated Circuit* – Circuito Integrado de Aplicação Específica
ASO	*Arbitrary Slice Order* – Ordem Arbitrária de *Slices*
AVC	*Advanced Video Coding* – Codificação de Vídeo Avançada
B	*Blue* - Azul
B	*Bidirectional frame* - Quadro Bidirecional
BAC	*Binary Arithmetic Coder* – Codificador Aritmético Binário
BAD	*Binary Arithmetic Decoder* – Decodificador Aritmético Binário
Bin	Bit após ser binarizado
BS	*Boundary Strenght* – Força da borda
CABAC	*Context-adaptive Binary Arithmetic Coder* – Codificador Aritmético Binário Adaptativo ao contexto
CABAD	*Context-adaptive Binary Arithmetic Decoder* – Decodificador Binário Adaptativo ao Contexto
CAVLC	*Context-adaptive Variable-Length Coder* – Codificador de comprimento variável adaptativo ao contexto
CAVLD	*Context-adaptive Variable-Length Decoder* – Decodificador de comprimento variável adaptativo ao contexto.
CIF	*Common Interchange Format* – Formato de Intercâmbio Padrão
Cb	Crominância em azul
CBF	*Coded Block Flag* – Sinalizador de Bloco Codificado
CBP	*Coded Block Pattern* – Padrão de bloco codificado
CCS	*Code Composer Studio*
CGS	*Coarse-Grain Scalability* – Escalabilidade de Grão Grosso
CODEC	*COder and DECoder* – Codificador e Decodificador
CMOS	*Complementary Metal Oxide Semiconductor* – Semicondutor de óxido metálico complementar

Cr	Crominância em vermelho
DC	*Direct Current*- Corrente Contínua
DCE	*Data Communication Equipment* – Equipamento de comunicação de dados
DCT	*Discrete Cossine Transform* – Transformada Discreta de Cosseno
DDR	*Double Data Rate* – Taxa de Dados Duplicada
DPCM	*Differential Pulse Code Modulation* – Modulação por Código de Pulso Diferencial
DRAM	*Dynamic RAM* – RAM Dinâmica
DSP	*Digital Signal Processor* – Processador Digital de Sinais
DSP48E	Módulo MAC de 48 bits dos dispositivos FPGA da XILINX
DTE	*Data Terminal Equipment* – Equipamento de Terminal de Dados
DVIC	*Difference Value of Illumination Change* – Valor da Diferença de Mudança de Iluminação
DVD	*Digital Video Disc* – Disco de Vídeo Digital
EDK	*Embebbed Development Kit*
FDCT	*Forward Discrete Cossine Transform* - Transformada Discreta de Cosseno Direta
FGS	*Fine-Grain Scalability* - Escalabilidade de Grão Fino
FIFO	*First-in, First-out* - Primeiro a Entrar, Primeiro a Sair
FPGA	*Field-Programmable Gate Array* – Arranjo de Portas programáveis em campo
FRExt	*Fidelity Range Extensions* – Extensões da faixa de fidelidade
G	*Green* - Verde
GOP	*Group Of Pictures* – Grupo de Figuras
H422P	*High Profile*, 10 bits por amostra, subamostragem padrão 4:2:2
HI10	*High Profile*, 10 bits por amostra
HD	*High Definition* – Alta Definição
HDL	*Hardware Description Language* – Linguagem de Descrição de Hardware
HDTV	*High Definition Television* – Televisão de Alta Definição
HP	*High Profile* - Perfil *High*
HSV	Espaço de cores matiz (*Hue*), saturação (*Saturation*), valor (*Value*)
HW	*Hardware*
ICA MC	*Illumination Change Adaptive Motion Compensation* – Compensação de Movimento Adaptativa a Mudanças de Iluminação

ISO	*International Organization for Standardization* – Organização Internacional de Padronização
I	*Intra frame* – Quadro Intra
I_PCM	*Intra Pulse Code Modulation* – Modulação por Código de Pulso Intra
IP	*Intellectual Property* – Propriedade Intelectual
ISE	*Integrated Synthesis Enviroment* – Ambiente de Síntese Integrada
ITU	*International Telecommuniations Union* – União Internacional de Telecomunicações
ITU-T	*International Telecommuniations Union – Standarization Sector* – Setor de Padronizações da ITU
JVM	*Joint Video Model*
JMVM	*Joint Multi-view Video Model*
JSVM	*Joint Scalable Video Model*
JTAG	*Joint Test Action Group*
JVT	*Joint Video Team*
LED	*Light Emmitter Diode* – Diodo Emissor de Luz
LOP	*Line Of Pixels* - Linha de Pixels
LPS	*Least Probable Symbol* – Símbolo Menos provável
LUT	*Look-Up Table* – Tabela de Consulta
LZC	*Leading Zero Count* - Contagem dos Zeros à esquerda
MAC	*Multiply and Accumulate* – Multiplica e Acumula (operação de DSP)
MAE	*Mean Absolute Error* – Erro Médio Absoluto
MB	*MacroBloco*
MBAFF	*MacroBlock Adaptive Frame/Field* - Quadro/Campo adaptativo no nível de macrobloco
MC	*Motion Compensation* - Compensação de Movimento
MCTF	*Motion Compensated Temporal Filtering* – Filtro Temporal com Compensação de Movimento
ME	*Motion Estimation* - Estimação de Movimento
MPEG	*Motion Pictures Experts Group* – Grupo de Peritos em Figuras em Movimento
MPS	*Most Probable Symbol* – Símbolo Mais Provável
MSE	*Mean Square Error* - Erro Médio Quadrático
MV	*Motion Vector* - Vetor de Movimento
MVC	*Multi-view Video Coder* – Codificador de Vídeo para Múltiplas Vistas
MVD	*Motion Vector Difference* – Vetor de Movimento Diferencial

NAL	*Network Abstraction Layer* – Camada de Abstração de Rede
NDA	*Non-Disclosure Agreement* – Acordo de não revelação
P	*Predicted Frame* – Quadro Predito
P	*Previews Block* – Bloco Anterior
PC	*Personal Computer* – Computador Pessoal
PCI	*Periferal Component Interconnect* – Interconexão de Componentes Periféricos
PSNR	*Peak Signal-to-Noise Ratio* – Relação sinal-ruído de pico
Q	*Quantization* - Quantização
Q	*Current Block* - Bloco Atual (no filtro de deblocagem)
Q^{-1}	*Inverse Quantization* - Quantização Inversa
QCIF	*Quarter CIF* – Quarta parte de um CIF
QP	*Quantization Parameter* - Parâmetro de Quantização
QPY	*Quantization Parameter for Y samples* – Parâmetro de Quantização para as amostras Y
R	*Red* - Vermelho
R	*Range* - intervalo
RAM	*Random Access Memory* – Memória de Acesso Aleatório
RGB	*Red, Green, Blue* - Espaço de cores Vermelho (*Red*), Verde (*Green*) e Azul (*Blue*)
rLPS	*Range of LPS* – Faixa do LPS
ROM	*Read-Only Memory* – Memória apenas de leitura
RSL	*Rocket Serial Link*
RTL	*Register Transfer Level* – Nível de transferência entre registradores
SAD	*Sum of Absolute Differences* – Soma das diferenças absolutas
SAE	*Sum of Absolute Errors* – Soma dos Erros Absolutos
SD	*Standard Definition* - Definição (resolução de imagem) Padrão
SDRAM	*Syncronous DRAM* – DRAM Síncrona
SDB	*Sundance Digital Bus* – Barramento Digital proprietário da *Sundance*
SDTV	*Standard Definition Television* – Televisão com Definição (resolução da imagem) padrão
SHB	*Sundance High-speed Bus* – Barramento de Alta Velocidade proprietário da *Sundance*
SE	*Syntax Element* – Elemento Sintático
SI	*Switching I frame* – Quadro I de chaveamento
SP	*Switching P frame* – Quadro P de chaveamento

SNR	*Signal-to-Noise Ratio* – Relação Sinal-Ruído
SRAM	*Static RAM* – RAM estática
sRGB	*Standard RGB* – RGB padrão
STDIN	*Standard Input* – Entrada Padrão
STDOUT	*Standard Output* – Saída Padrão
SVC	*Scalable Video Coder* - Codificador de Vídeo Escalável
SW	*Software*
T	*Transform* - Transformada
T^{-1}	*Inverse Transform* – Transformada Inversa
TIM	*Texas Instruments Module* – Módulo da Texas Instruments
UART	*Universal Asyncronous Receiver-Transmitter* – Transmissor e Receptor Assíncrono Universal
ULA	Unidade Lógica e Aritmética
USART	*Universal Syncronous and Asyncronous Receiver-Transmitter* – Transmissor e Receptor Síncrono e Assíncrono Universal
USB	*Universal Serial Bus* – Barramento Serial Universal
YCbCr	Luma (Y) e diferenças de crominância em azul (Cb) e vermelho (Cr)
Y	Luminância
VCEG	*Video Coding Experts Group* – Grupo de Peritos em codificação de vídeo
VCL	*Video Coding Layer* - Camada de Codificação de Vídeo
VGA	*Video Graphics Array*
VLC	*Variable Length Coder* – Codificador de Comprimento Variável
VHDL	*Very high speed integrated circuit hardware description language* – Linguagem de descrição de hardware para circuitos integrados de alta velocidade
XSGA	*Extended Super VGA* – Super VGA estendido

RESUMO

Novas arquiteturas de hardware desenvolvidas para blocos chave do padrão de codificação de vídeo ISO/IEC 14496-10 são discutidas, propostas, implementadas e validades nesta tese. Também chamado de H.264, AVC (*Advanced Video Coder*) ou MPEG-4 parte 10, o padrão é o estado da arte em codificação de vídeo, apresentando as mais altas taxas de compressão possíveis por um compressor de vídeo padronizado por organismos internacionais (ISO/IEC e ITU-T). O H.264 já passou por três revisões importantes: na primeira foram incluídos novos perfis, voltados para a extensão da fidelidade e aplicações profissionais, na segunda veio o suporte a escalabilidade (SVC – *Scalable Video Coder*). Uma terceira revisão suporta fontes de vídeo com múltiplas vistas (MVC – *Multi-view Video Coder*). Nesta tese são apresentadas arquiteturas para dois módulos do codificador H.264: o CABAC e o Filtro de Deblocagem (*Deblocking Filter*). O CABAC (*Context-Adaptive Binary Arithmetic Coder*) possui desafios importantes devido às dependências de dados de natureza bit-a-bit. Uma revisão das alternativas arquiteturais e uma solução específica para a codificação CABAC é apresentada nesta tese. O filtro de deblocagem também apresenta diversos desafios importantes para seu desenvolvimento e foi alvo de uma proposta arquitetural apresentada neste trabalho. Finalmente a arquitetura de uma plataforma de validação genérica para validar módulos desenvolvidos para o codificador e decodificador H.264 também é apresentada. Os módulos escolhidos estão de acordo com os demais trabalhos realizados pelo grupo de pesquisa da UFRGS, que têm por objetivo desenvolver um decodificador e um codificador completos capazes de processar vídeo digital de alta definição no formato 1080p em tempo real.

Palavras-chave: Codificação de Vídeo; H.264; AVC;SVC;MVC; Hardware; CABAD; CABAC; Filtro de Deblocagem; Prototipação.

Dedicated Hardware Architectures for H.64 Video Encoders – Deblocking Filter and Context Adaptive Binary Arithmetic Coding

ABSTRACT

New hardware architectures developed for key blocks of the ISO/IEC 14496-10 video coding standard are discussed, proposed, implemented, and validated in this thesis. The standard is also called H.264, AVC (Advanced Video Coder) or MPEG-4 part 10, and is the state-of-the-art in video coding, presenting the highest compression ratios achievable by an internationally standardized video coder (ISO/IEC and ITU-T). The H.264 has already been revised three times: the first included new profiles for fidelity extension and professional applications. The second brought the scalability support (SVC – Scalable Video Coder). The third revision supports video sources with multiple views (MVC – Multi-view Video Coder). The present work developed high performance architectures for CABAC (Context-Adaptive Binary Arithmetic Coder), which were challenging because of the bitwise data dependencies. A through revision of the alternative architectures and a specific architectural solution for CABAC encoding are presented in this thesis. A dedicated hardware architecture for a HIGH profile Deblocking Filter is also presented, developed, validated and synthesized for two different targets: FPGA and ASIC. The validation methodology is presented and applied to three different modules of the H.264 encoder. The H.264 blocks dealt with in this thesis work complement those developed by other works in the UFRGS research group and contribute to the development of complete encoders for real-time processing of high definition digital video at 1080p.

Keywords: Video Coding; H.264; AVC; SVC; MVC; Hardware; CABAD; CABAC; Deblocking Filter; Prototyping.

1 INTRODUÇÃO

O processamento de sinais de vídeo digital sempre demandou bastante atenção da academia e da indústria devido à grande quantidade de dados a ser armazenado ou transmitido. Esta grande quantidade de dados advém da natureza tridimensional do sinal de vídeo: é necessária a transmissão de uma sequência de imagens para que se obtenha a sensação de movimento; uma imagem possui duas dimensões espaciais e o tempo é a terceira dimensão do sinal de vídeo. A compressão de vídeo sempre foi uma necessidade para todas as aplicações que envolvem vídeo, visando à redução da quantidade de dados a ser transmitida ou armazenada. O sucesso de muitas aplicações que envolvem vídeo depende da capacidade de compressão do sinal de vídeo através de codificação.

A presença de regiões com gradientes suaves de cores e intensidade luminosa nas imagens de uma sequência de vídeo (redundância espacial) e a similaridade entre imagens sequenciais (redundância temporal) são algumas características que se destacam na maior parte dos sinais de vídeo naturais, obtidos através de uma câmera de vídeo. Os algoritmos de compressão de vídeo são então desenvolvidos para tentar explorar estas características do sinal. Algumas fontes de vídeo, entretanto, são comprimidas com mais facilidade do que outras. Uma sequência de estúdio de jornalismo, por exemplo, possui muita redundância temporal já que as imagens da sequência de vídeo são idênticas exceto em uma região onde há movimento do apresentador, enquanto que uma cena de filme de ação possui diferenças significativas entre quadros sucessivos. Quando a aquisição de vídeo é feita a partir de cenas naturais, é tolerável algum grau de distorção (degradação) na qualidade das imagens adquiridas, possibilitando a compressão com perda de informação (RICHARDSON, 2002). A análise de qual informação será descartada no processo de compressão com perdas é um tema bastante abrangente que envolve o desenvolvimento de modelos de percepção do olho humano e algoritmos para descartar o máximo de informação com o mínimo de impacto visual durante o processo de codificação. Aplicações que usam vídeo para aplicações científicas (cenas não-naturais) devem usar com cuidado os compressores com perdas, uma vez que as perdas podem levar a distorções numéricas significativas, mesmo quando imperceptíveis ao olho humano.

A complexidade de um compressor de vídeo está diretamente ligada à quantidade e a complexidade das ferramentas (conjuntos de algoritmos) empregadas para analisar a sequência de vídeo e identificar informação redundante ou irrelevante para que a informação possa ser transmitida de forma comprimida. Pode-se com isso construir um compressor que seja arbitrariamente complexo com o objetivo de aumentar a eficiência do processo de compressão desde que se disponha de capacidade de processamento suficiente. Como maioria das aplicações exige que os dados sejam comprimidos à medida que são capturados, em tempo real, há uma limitação da capacidade de

processamento em operações por unidade de tempo disponível. A crescente disponibilidade de processamento oferecida pelo contínuo avanço da tecnologia microeletrônica e avanços na eficiência dos algoritmos empregados no processo de compressão têm possibilitado o desenvolvimento de compressores de vídeo cada vez mais eficientes, porém cada vez mais complexos (ITRS, 2010). Contudo a compressão de vídeo é e será por muito tempo um gargalo para as aplicações de vídeo na medida em que resoluções cada vez maiores e exigências de consumo de energia cada vez menores associados a algoritmos cada vez mais complexos são exigidos, tornando necessário o desenvolvimento de arquiteturas de hardware específicas para a compressão de vídeo.

1.1 Proposta de trabalho

A proposta de trabalho de tese estará focada no desenvolvimento e prototipação de arquiteturas de alto desempenho para a codificação e a decodificação de vídeo em Hardware no padrão H.264. Este trabalho inclui as seguintes atividades:

- Proposta arquitetural para o Filtro de Deblocagem
- Propostas arquiteturais para o codificador e decodificador CABAC
- Proposta de uma estratégia de prototipação para módulos do codec (Codificador/Decodificador) H.264

Nesta tese a metodologia de apresentação do processo de codificação é descrita no Capítulo 2, onde, em essência, todas as técnicas de compressão são desenvolvidas com o objetivo de aumentar a entropia do *bitstream* codificado. Um breve estudo do padrão H.264 (ITU, 2010), que é o estado da arte atual, é apresentado no Capítulo 3.

No Capítulo 4, é apresentada uma proposta arquitetural para o Filtro de Deblocagem do padrão H.264. O projeto desenvolvido integra todos os mecanismos necessários ao funcionamento do Filtro de Deblocagem em hardware. Após ser descrito em linguagem VHDL e sintetizado em FPGA e ASIC, o desempenho obtido supera o requisito para a resolução de vídeo 1080p (1920x1080 a 30 quadros por segundo). O layout de um circuito integrado em tecnologia 0.18µm também é apresentado. Uma extensão deste trabalho para reordenar os dados de saída para o formato *raster-scan* YCbCr 4:4:4 e RGB é apresentado. Uma modificação no projeto implementa o padrão Hi10p para o formato sub-amostrado 4:2:0 com 10 bits por amostra.

No Capítulo 5 são apresentadas propostas arquiteturais para o Codificador Aritmético Adaptativo ao Contexo (CABAC) e Decodificador Aritmético Adaptativo ao Contexto (CABAD). No CABAD, uma análise de fluxo de dados nos sub-módulos é apresentada, seguido de propostas arquiteturais para os decodificadores aritméticos do CABAD. Uma proposta arquitetural para o CABAC é apresentada. Três arquiteturas para o codificador aritmético são apresentadas, baseadas em uma mesma interface. Resultados de eficiência de processamento e, após descrição e síntese, resultados de área, potência, eficiência energética e eficiência em área. Uma arquitetura para o núcleo do CABAC (codificador aritmético + acesso, inicialização e atualização dos contextos) é proposta e desenvolvida capaz de se conectar a qualquer dos codificadores aritméticos propostos. Finalmente, a proposta arquitetural do CABAC completo é apresentada, incluindo a hierarquia da descrição RTL.

No Capítulo 5, uma estratégia de prototipação usando o ambiente de desenvolvimento de sistemas embarcados *Embebbed Development Kit* (EDK) (PLATFORM, 2008) é apresentada, para ser utilizada em FPGAs da Xilinx (XILINX,

2008). A estratégia foi proposta para a validação de módulos do H.264 desenvolvidos no grupo, mas é genérica e pode ser aplicada a qualquer outro módulo de hardware que possua entradas e saídas que necessite de estímulos e comparação de resultados.

O apêndice A apresenta um estudo sobre as plataformas de prototipação que podem ser utilizadas para a validação dos módulos IP desenvolvidos nesta tese.

1.2 Trabalhos relacionados no grupo

O desenvolvimento desta tese está relacionado com diversos outros trabalhos de projeto de arquiteturas para circuitos digitais, síntese e validação de hardware, sob orientação do orientador desta tese. A seguir serão citados alguns trabalhos relevantes para a contextualização desta proposta. Esta lista não é completa, e está posta aqui apenas para contextualizar o escopo deste trabalho com os interesses do grupo.

Agostini (2007) implementou o bloco de transformadas (T), quantização (Q), transformadas inversas (T^{-1}), quantização inversa (Q^{-1}) e algumas arquiteturas para a estimação de movimento (ME) para o codificador. Os módulos T^{-1} e Q^{-1} podem ser empregados tanto no decodificador quanto no codificador. Os trabalhos em desenvolvimento pelo autor estendem a exploração do espaço de projeto destes módulos, com alternativas que consideram alta taxa de processamento, baixa latência (necessária no codificador, devido a dependências de dados presentes no laço de codificação intra/transformadas) e também baixo consumo.

Azevedo (2006) desenvolveu uma proposta arquitetural para o bloco compensador de movimento (MC) do decodificador capaz de atingir alta definição em um FPGA Virtex-II Pro. O trabalho incluiu o projeto de uma memória cache para a busca de informações na memória principal.

Porto (2008) estendeu o trabalho proposto por Agostini (2007) para a estimação de movimento usando *pel-subsampling* e múltiplos tamanhos de bloco, com melhorias arquiteturais e resultados de desempenho tanto para FPGA quanto para ASIC.

Staehler (2006) apresentou uma proposta para bloco intra no decodificador e Diniz (2009) uma proposta para o codificador intra.

O trabalho de Zatt (2008) trouxe melhorias na arquitetura para compensação de movimento originalmente desenvolvida por Azevedo (2006) e também realizou uma modelagem completa de um codificador H.264 na linguagem SystemC.

Vários outros trabalhos de pós-graduação encontram-se em andamento no grupo. A Figura 1.1 apresenta um diagrama dos módulos que compõe o decodificador H.264 e a Figura 1.2 apresenta os módulos que compõe o Codificador H.264. Nestas figuras, os módulos em cinza claro já foram alvo de pesquisa e desenvolvimento de arquiteturas por algum membro do grupo. O CABAD (DEPRA, 2008; DEPRA, 2008a) foi desenvolvido em co-autoria com o autor da presente tese e a parte do trabalho em que houve co-autoria direta está apresentada na Seção 5.2 desta tese. O Filtro de Deblocagem (ROSA, 2007; ROSA 2009), destacado em cinza escuro na Figura 1.1 e na Figura 1.2, foi desenvolvido pelo autor e está descrito no Capítulo 4 desta tese.

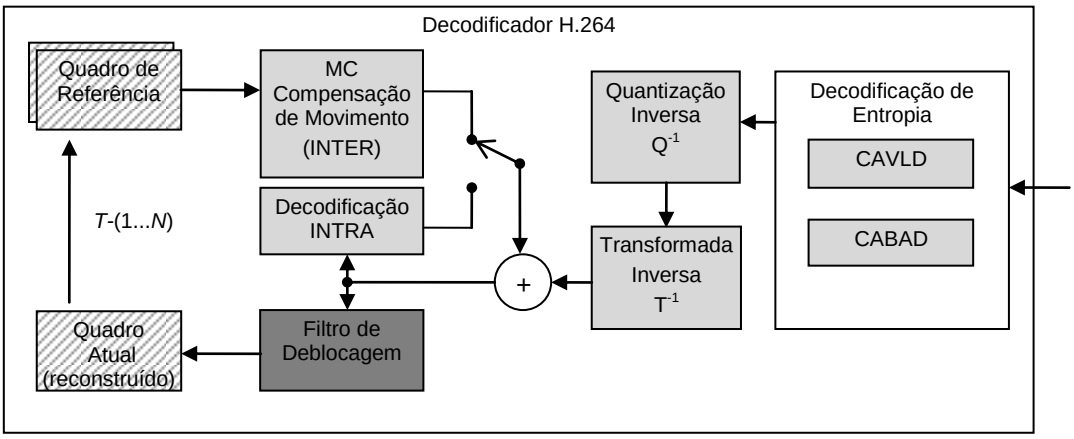

Figura 1.1: Decodificador H.264 – Módulos desenvolvidos pelo grupo.

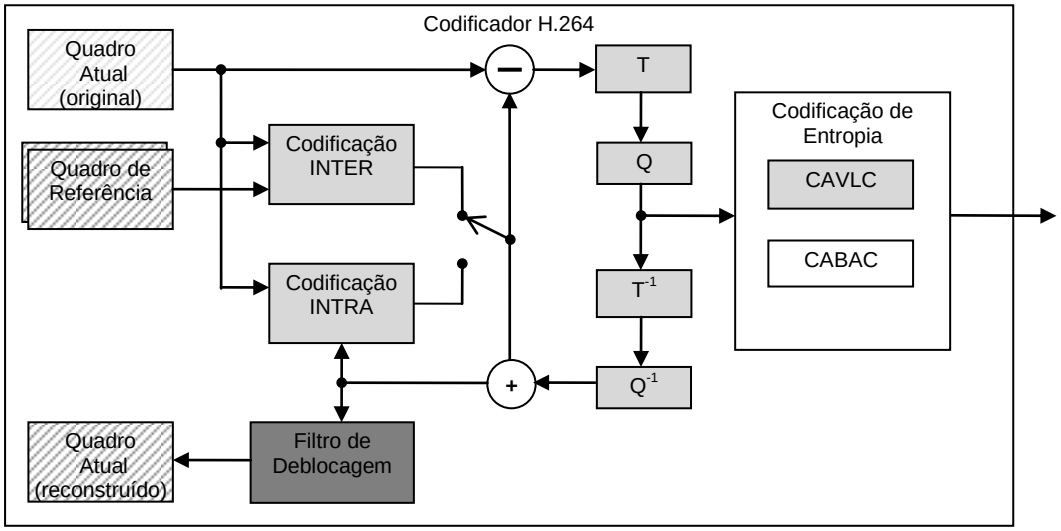

Figura 1.2: Codificador H.264 – Módulos desenvolvidos pelo grupo.

Como é possível perceber na Figura 1.2, o módulo CABAC é o único que ainda não foi alvo de desenvolvimento arquitetural por algum membro do grupo, sendo este abordado no Capítulo 5 desta tese.

2 CONCEITOS BÁSICOS DE COMPRESSÃO DE VÍDEO DIGITAL

As aplicações de entretenimento ou de consumo geral são o foco dos padrões de compressão de vídeo normatizado pelo ITU-T (*International Telecommunications Union – Standardization Sector* – Setor de padronização da União Internacional de Telecomunicações) ou pela ISO (International Organization for Standardization – *Organização Internacional de Padronização*). Nestas aplicações, o objetivo final do sinal de vídeo é a sua visualização. Os algoritmos de compressão de vídeo para posterior visualização tentam tirar proveito de características do olho humano que possibilitem uma maior compressão do sinal de vídeo, quase sempre através de perda controlada de informação (RICHARDSON, 2003). Neste capítulo uma breve apresentação sobre o sistema visual humano é apresentada na seção 2.1. Na seção 2.2 os espaços de cores relevantes e a possibilidade de sub-amostragem da informação de cromaticidade é apresentado. A seção 2.3 estratifica os tipos de redundâncias que podem ser exploradas em compressores de vídeo e finalmente a seção 2.4 apresenta um modelo simplificado de um compressor de vídeo híbrido similar ao modelo usado no H.264.

2.1 O Olho Humano

O olho humano é composto por diversas estruturas que permitem a visualização de objetos em uma ampla região do espaço através de um posicionamento preciso do globo ocular e com regulagem de iluminação e foco. A Figura 2.1 apresenta uma ilustração de um corte de um olho humano.

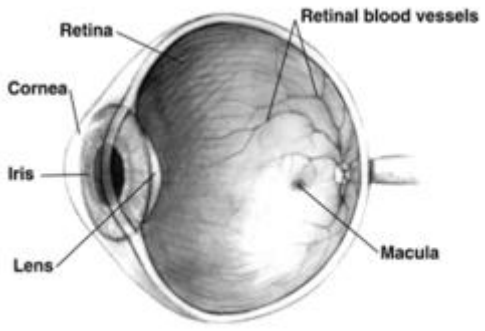

Figura 2.1: Vista em corte de um olho humano.

Extraído de (EYE, 2008).

A córnea é a estrutura externa que permite a entrada de luz para o interior do olho, que é preenchindo com um líquido transparente. A íris provê um mecanismo de regulagem da quantidade de luz que pode entrar no olho. A luz é focada na retina

através do cristalino (*Lens*, na Figura 2.1). A retina ocupa cerca de 70% da área da parede interna do olho (EYE, 2008). Conjuntos de músculos não representados na Figura 2.1 são responsáveis pelos movimentos da íris, do cristalino e do próprio globo ocular.

A retina é uma estutura complexa onde existem células sensíveis à luz conectadas a neurônios que processam e canalizam a informação até o nervo óptico. O nervo óptico finalmente leva a informação até o cerebro. As células sensíveis à luz são divididas em dois tipos básicos: bastonetes e cones. Os bastonetes capturam uma ampla região do espectro visível enquanto os cones se dividem em três tipos que capturam três diferentes regiões mais estreitas do espectro visível. Estes dois diferentes tipos de células estão distribuídas de forma heterogênea na retina (RETINA, 2008). A periferia da retina contém uma maior concentração de células tipo bastonete enquanto que em uma pequena região denominada Mácula (MACULA, 2008) há uma grande concentração de células do tipo cone. Uma pequena região da mácula, chamada fóvea, é onde se encontra a maior densidade de células sensíveis a luz e, por isso, esta região é a que dá maior definição à visão. A Figura 2.2 mostra uma seção de uma região da retina ilustrando a conectividade entre as células que produzem a informação visual que é canalizada ao nervo ótico para processamento pelo cérebro.

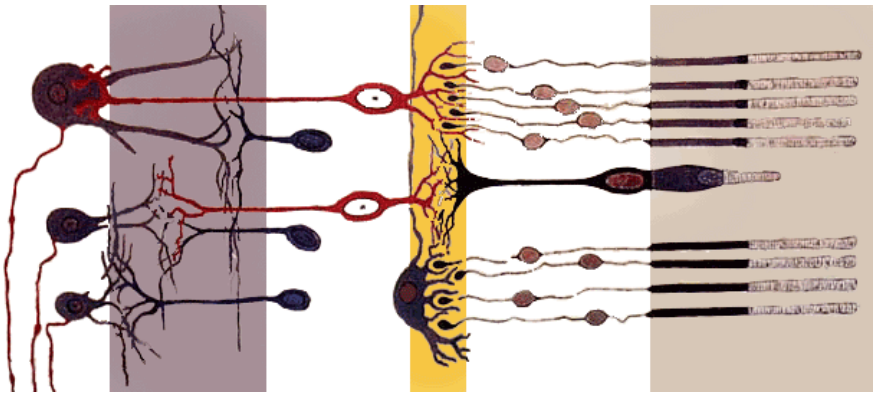

Figura 2.2: Visão de uma seção da retina.

Extraído de (RETINA, 2008).

Os receptores luminosos (bastonetes e cones) estão na parte à direita da figura. As camadas intermediárias são compostas por neurônios que realizam algum pré-processamento nos estímulos recebidos de forma a reduzir a quantidade de informação a ser transportada pelo nervo óptico até o cérebro. A compreensão do funcionamento destas estruturas leva ao desenvolvimento de compressores de vídeo que devem explorar estas características do sistema visual humano aumentando a taxa de compressão através da eliminação de informação que de qualquer forma será descartada.

A retina possui cerca de 240 milhões de bastonetes e 13 milhões de cones (GONZALEZ, 2003). Os bastonetes são responsáveis pela recepção de luz de baixa intensidade (visão noturna), e apresentam-se em estado saturado quando em condições normais de iluminação. Em contrapartida, os cones são menos sensíveis, sendo responsáveis pela visão diurna. Por esta razão temos dificuldades de discernir as cores dos objetos em ambientes com muito pouca iluminação.

Apesar da dominância dos cones em ambientes bem iluminados, o processamento realizado pelos neurônios da camada intermediária da retina (Figura 2.2) na região da

fóvea faz com que a resolução visual de intensidade luminosa seja maior que a resolução visual das cores. Deste modo, o sistema visual humano é mais sensível a informações de luminância do que a informações de crominância.

2.2 Representação de imagens digitais

As imagens digitais são representadas através de uma matriz retangular de elementos de cor. Esta matriz é mapeada para um dispositivo de exibição onde cada um dos elementos desta matriz representa uma informação de brilho ou cor de uma pequena região do dispositivo de exibição. A cada um destes elementos é dada a denominação pixel (em alusão à palavra inglesa "picture element"). A Figura 2.3 apresenta uma imagem digital.

Figura 2.3: (a) Imagem digital e (b) região de 16×16 pixels.

A Figura 2.3(a) apresenta uma imagem digital representada por uma matriz com *x* elementos de largura e *y* elementos de altura. A Figura 2.3(b) apresenta um detalhe desta imagem onde apenas uma região de 16 por 16 pixels é exibida. Observando que a informação espacial é discreta e cada pixel ocupa uma área no dispositivo de representação, diz-se que quanto maior for a quantidade de pixels de uma imagem, maior será sua resolução. Para a exibição em um dispositivo onde a imagem terá tamanho fixo, uma maior resolução representa uma redução no tamanho dos pixels.

2.3 Espaço de Cores

A representação de um vídeo colorido está associada à interpretação das cores pelo sistema visual humano. Como mencionado na seção 2.1, a retina possui estruturas sensíveis à luz chamados bastonetes e cones, sendo os bastonetes sensores de amplo espectro que nos dá visão monocromática em ambientes de baixa iluminação e os cones, menos sensíveis, porém sensíveis a um espectro mais estreito. No olho humano existem três tipos de cones que respondem a três faixas de comprimentos de onda distintos. Através destas estruturas, o sistema visual humano é capaz de perceber luz em um espectro que vai desde 700nm até 380nm (GAMUT, 2008). A capacidade dos cones de discernir regiões do espectro visível torna possível a percepção de cores. Luz com comprimento de onda logo abaixo de 700nm será percebida como vermelho. Comprimentos de onda menores levarão a percepção de amarelo, verde, azul e finalmente violeta próximo aos 380nm. O universo de cores perceptíveis pelo olho humano é chamado de gama ou gamut (GAMUT, 2008).

Como temos três sensores de cores, a representação de qualquer cor deveria ocorrer em um espaço tridimensional. Em 1931 a Comissão Internacional sobre Iluminação (CIE - *Commission internationale de l'éclairage*) criou uma representação bidimensional, onde dois eixos (*x* e *y*) representam a cromaticidade no plano e um terceiro eixo (*Y*) representa o brilho. Esta é a representação utilizada para definir a resposta padrão do sistema visual humano, referida como CIE 1931.

A Figura 2.5 apresenta um diagrama de cromaticidade padrão CIE 1931 (*xyY*) destacando o *gamut* possível em um monitor padrão sRGB (representação triangular colorida). O triângulo é formado por três componentes primárias: vermelho (R – *red*), verde (G – *green*) e azul (B – *blue*). A representação de cores na região fora do triângulo mas dentro do contorno que define o *gamut* possível para o olho humano não foram representadas, pois não são possíveis de serem representadas com fidelidade por um monitor sRGB. De fato, o espaço de cores sRGB foi criado a partir da resposta de diferentes tipos de fósforos utilizados em tubos de raios catódicos (CRT – *Cathodic Ray Tube*) utilizados para a criação da TV colorida e usada até hoje em televisores e monitores de computador. Embora a área representável pelo espaço de cores sRGB pareça ser pequena em relação ao total percebido pelo olho humano no gráfico da Figura 2.5, não há relação nesta representação entre a área e o número de cores distintas percebidas pelo olho humano, sendo muito menor na área fora do triângulo sRGB (GAMUT, 2008). O contorno do gráfico representa a região de visão monocromática, sendo o comprimento de onda apresentado em para alguns pontos, indo desde 700nm (limite inferior do vermelho) passando pelo verde em 520nm e chegando ao violeta em 380nm. A magnitude dos eixos *x* e *y* não têm significado físico.

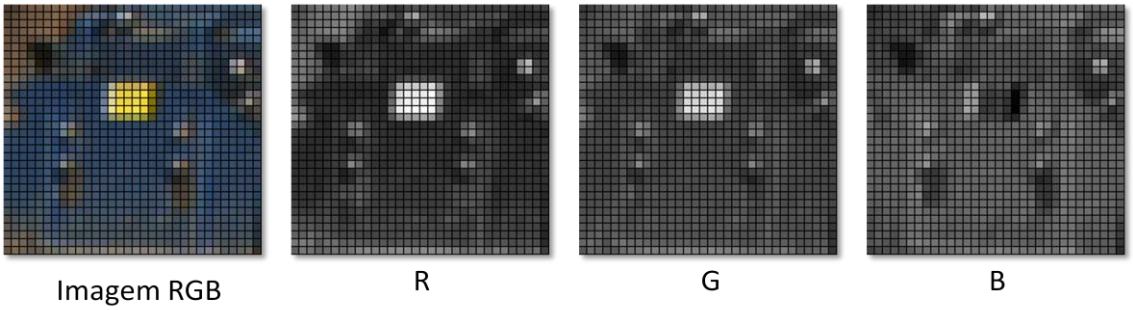

Figura 2.4: Decomposição de uma imagem RGB em suas componentes.

A Figura 2.4 apresenta uma pequena região de uma imagem digital RGB e a intensidade de cada um dos componentes R (Red – Vermelho), G (Green – Verde) e B(Blue – Azul) mapeada para uma escala de cinzas. Nesta imagem cada ponto da matriz R, G e B é chamado de amostra e apresenta uma faixa dinâmica de 256 valores possíveis (de 0 a 255). A combinação destas três amostras para cada elemento da matriz irá gerar os pixels da imagem RGB à esquerda. Verificando-se todas as possíveis combinações de valor de cada um dos elementos R, G e B que compõe cada pixel da imagem, obtem-se uma possibilidade de representação de cores de 16,7 milhões (256x256x256).

Para cada um dos componentes de cor da imagem da Figura 2.4, deverá haver um mapeamento entre o valor numérico da matriz que representa os elementos numericamente e o dispositivo que converterá os valores desta matriz em informação de brilho em cada uma das cores para compor a imagem colorida. No espaço de cores sRGB (*standard* RGB – RGB padronizado), uma função de mapeamento não-linear

(função gama) é utilizada, de forma que a variação de brilho entre valores sucessivos da matriz que representa a imagem não é constante. Os vetores principais do espaço de cores sRGB são obtidos a partir do espaço de cores *XYZ*, que foi obtido através da caracterização da sensibilidade do olho humano às cores, onde os vetores *X* e *Y* representam a cromaticidade e o vetor *Z* representa a luminosidade. A Figura 2.5 apresenta um gráfico bidimensional para a cromaticidade, com a luminoside (eixo *Z*, vertical ao plano de projeção *X*,*Y*) no seu valor máximo.

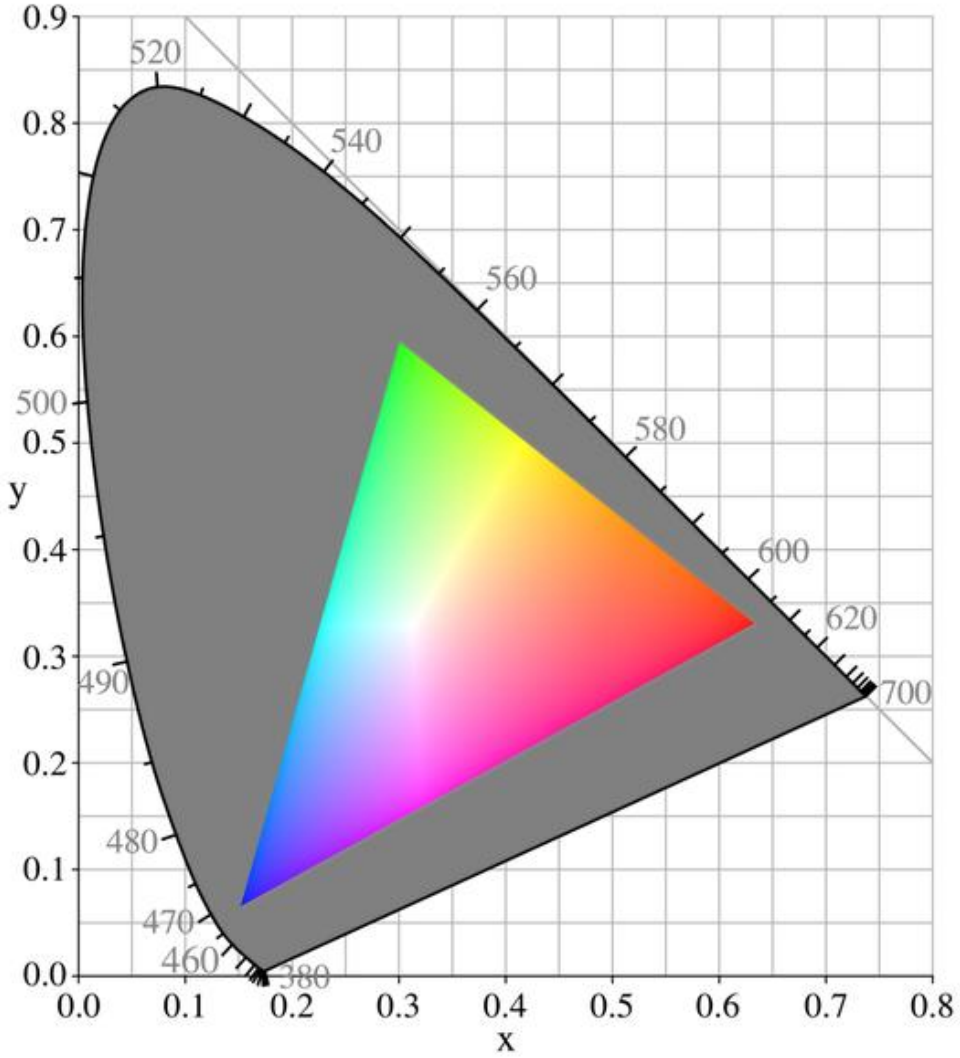

Figura 2.5: *Gamut* possível no espaço de cores sRGB[1].

Extraído de (GAMUT, 2008).

Além do espaço de cores sRGB, cuja gama de cores está representada pelo triângulo na Figura 2.5, existem muitas outras formas de se representar cores de acordo com a quantidade de vetores e do seu posicionamento no espectro visível. A definição da

[1] Embora algumas impressoras coloridas possam representar de forma mais ampla o Gamut de cores que pode ser percebido pelo olho humano, a imagem inserida neste texto está no espaço de cores sRGB, não sendo possível representar de forma acurada cores fora do triângulo. Quando impresso em escala de cinzas, os tons de cinza não terão significado expressivo. Ainda poderão ocorrer distorções por ajuste incorreto do fator gama entre a imagem e o dispositivo de exibição/impressão.

posição destes vetores dentro do *gamut* pode levar a exploração de certas características de sensibilidade a cores do olho humano e é essencial para a eficiência da codificação de vídeos digitais.

Alguns destes espaços vetoriais são suportados por órgãos de padronização para determinadas aplicações. Para representar imagens digitais, existem padrões como: sRGB, HSV e YCbCr (SHI, 1999). O espaço de cores sRGB é um dos mais comuns, tendo em vista que é este o espaço de cores utilizado para a definição das cores primárias nos monitores coloridos controlados por microcomputadores comuns. O sRGB representa, em três matrizes distintas, as três cores primárias captadas pelo sistema visual humano: vermelho, verde e azul. No espaço de cores YCbCr, as três componentes utilizadas são luminância (Y), que define a intensidade luminosa ou o brilho; crominância azul (Cb) e crominância vermelha (Cr) (MIANO, 1999).

Os componentes R, G e B possuem um elevado grau de correlação com a informação de luminância. A informação de luminância é percebida pelo olho humano com muito mais acuidade (resolução) do que a informação de cor. Por esta razão, um espaço de cores que tenha a luminância como uma das componentes principais é importante, pois permite que diferentes resoluções espaciais sejam aplicadas para luminância e para crominância, sendo o espaço de cores denominado YCbCr o preferido para aplicações envolvendo vídeo (RICHARDSON, 2002). A Figura 2.6 compara a forma RGB com a YCbCr de representação de cores para uma mesma imagem.

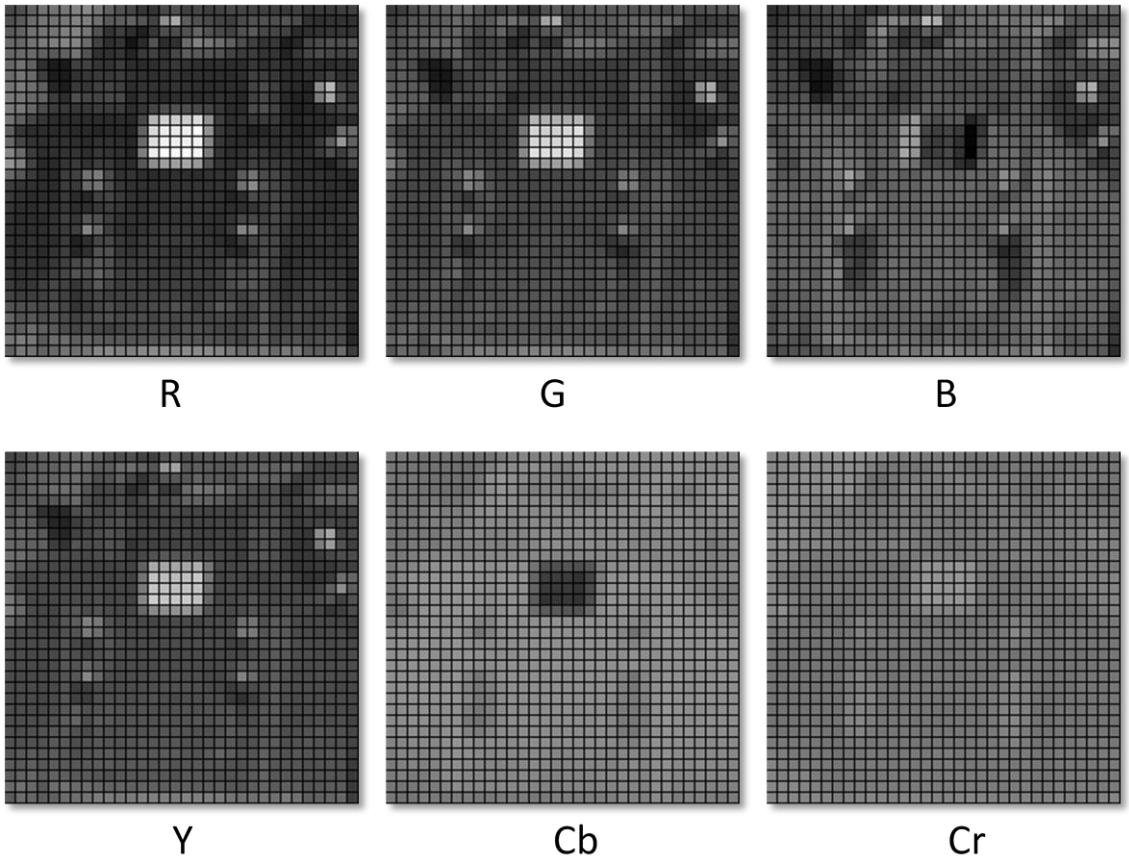

Figura 2.6: Componentes de cor no formato RGB e YCbCr para uma região de uma mesma imagem.

Além disso, no espaço de cores YCbCr a informação de cor (componentes Cb e Cr) está separada da informação de intensidade luminosa (componente Y). Deste modo, estas informações podem ser tratadas de forma diferenciada pelos compressores de imagens e vídeos. Se este texto estiver sendo exibido em um dispositivo monocromático (impresso, por exemplo), é possível que a representação Y coincida com a "imagem RGB" da Figura 2.4. Isto ocorre porque em dispositivos monocromáticos, apenas a informação de luminância pode ser representada.

2.4 Subamostragem de cores

Os compressores de vídeo podem explorar a característica psicovisual humana de menor sensibilidade espacial às cores para aumentar a eficiência de codificação através da redução da taxa de amostragem dos componentes de crominância em relação aos componentes de luminância. Esta operação é chamada de sub-amostragem de cores e é realizada sob o espaço de cores YCbCr nos padrões de compressão de vídeos atuais (RICHARDSON, 2003).

Existem várias formas de relacionar os componentes de crominância com o componente de luminância para realizar a sub-amostragem. Os formatos mais comuns são o 4:4:4, o 4:2:2 e o 4:2:0. No formato 4:4:4, para cada quatro amostras de luminância (Y), existem quatro amostras de crominância azul (Cb) e quatro amostras de crominância vermelha (Cr). Por isso, os três componentes de cor possuem a mesma resolução e existe uma amostra de cada elemento de cor para cada pixel da imagem e, assim, a sub-amostragem não foi aplicada. No formato 4:2:2, para cada quatro amostras de Y na direção horizontal, existem apenas duas amostras de Cb e duas amostras de Cr. Neste caso, as amostras de crominância possuem a mesma resolução vertical das amostras de luminância, mas possuem metade da resolução horizontal. No formato 4:2:0, para cada quatro amostras de Y, existe apenas uma amostra de Cb e uma amostra de Cr. Neste caso, as amostras de crominância possuem metade da resolução horizontal e metade da resolução vertical do que as amostras de luminância. A nomenclatura 4:2:0 é usada por motivos históricos, pois os números não representam a relação lógica entre os componentes de cor, que seria naturalmente 4:1:1 (RICHARDSON, 2003). A Figura 2.7 apresenta a localização geográfica da informação de crominância em um bloco de 2x2 pixels da imagem, que formam o padrão 4:2:0, 4:2:2 e 4:4:4. Para imagens de qualquer tamanho, este padrão de 2x2 pixels se repete.

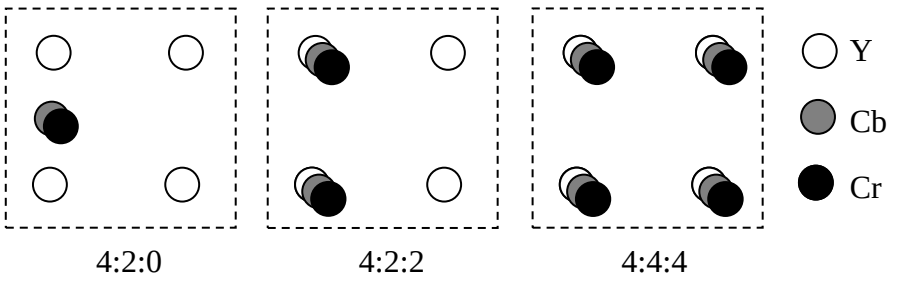

Figura 2.7: Localização da informação de crominância nos formatos sub-amostrados de representação de cores.

A sub-amostragem de cor aumenta significativamente a eficiência da compressão, uma vez que parte da informação da imagem é simplesmente descartada, sem causar impacto visual perceptível. Considerando o formato 4:2:0 como exemplo, uma vez que cada componente de crominância possui exatamente um quarto das amostras presentes

no componente de luminância, então um vídeo YCbCr no formato 4:2:0 irá utilizar exatamente a metade das amostras necessárias para um vídeo RGB ou YCbCr no formato 4:4:4. Isso implica em redução de 50% da informação que precisa ser processada e/ou transmitida. Na Figura 2.6 as amostras estão no formato 4:4:4. A Figura 2.8 apresenta as mesmas amostras no formato 4:2:0.

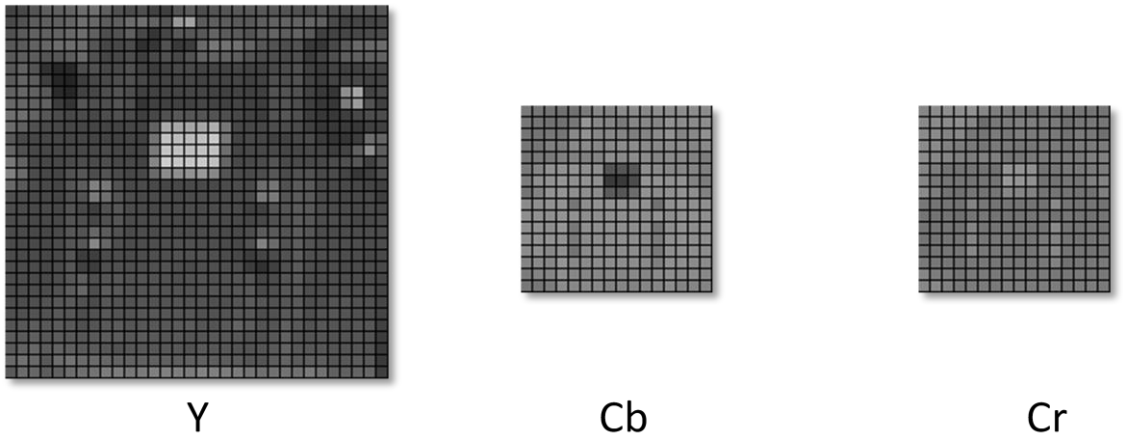

Figura 2.8: Representação YCbCr no formato 4:2:0.

Observe que no formato 4:2:0, os componentes Cb e Cr possuem a metade do número de elementos em ambas as dimensões, sendo a quantidade total de elementos de cada uma das componetes de cor ¼ do total de elementos do componente Y. Para recompor a imagem colorida 4:4:4 a partir da subamostragem 4:2:0, os componentes de cor devem ser sobreamostrados, de forma que para cada elemento de crominânica (Cb e Cr) existam 4 elementos de luminânia correspondentes.

O padrão H.264, foco deste trabalho, considera que os dados do vídeo de entrada estão no espaço de cores YCbCr. Maiores discussões sobre outros formatos de representação de cores estão fora do escopo deste trabalho. Sub-amostragens de cor nos formatos 4:2:0, 4:2:2 e 4:4:4 são permitidos, mas o formato mais usado é o 4:2:0. A resolução típica para cada componente de vídeo é 8 bits, entretanto, no perfil HIGH 10 (ITU, 2005), as componentes de vídeo podem conter até 10 bits.

2.5 Redundância de Dados na Representação de Vídeos

A principal atividade dos algoritmos de compressão de dados de qualquer natureza é tentar identificar redundâncias que se manifestam através de correlações, repetições e distribuições estatísticas não-uniformes existentes em um conjunto de dados e encontrar um novo conjunto (o menor possível) que represente o conjunto original através de regras definidas. A compressão de vídeo é uma aplicação específica da compressão de dados onde uma série de características do conjunto de dados é conhecida, facilitando o trabalho dos algoritmos de compressão. Basicamente, existem quatro tipos diferentes de redundâncias exploradas na compressão de vídeos: redundância espacial, redundância temporal, o modelo do sistema visual humano (redundância psicovisual) e redundância entrópica. Existem controvérsias entre os autores a respeito da definição dos tipos de redundância. A classificação apresentada neste trabalho, com quatro diferentes tipos de redundância, é baseada em (SHI, 1999) e (GONZALEZ, 2003). Cada uma destas redundâncias será resumidamente explicada nos próximos parágrafos e, na próxima seção do texto, será explicado como cada redundância é tratada em um CODEC de vídeo genérico.

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

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

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

- **Modelo do sistema visual humano** – As características do sistema visual humano fazem com que não consigamos captar alguns tipos de informações na imagem e são úteis para a compressão de vídeo. A própria representação do espectro visível com apenas 3 componentes de cor é possível pela limitação do sistema visual humano em distinguir cores. Além disso, algumas informações da imagem, como o brilho, por exemplo, são mais importantes para o sistema visual humano do que outras, como a cor. Alguns autores consideram que o modelo do olho humano leva a produção de um tipo de redundância chamada redundância psicovisual (GONZALEZ, 2003). Para explorar este tipo de redundância, parte da informação original da imagem é eliminada de forma irreversível pelo codificador. Existem dois tipos principais de processos utilizados para este fim. O primeiro, chamado de "sub-amostragem" (que foi definida na seção anterior) é utilizado na entrada do vídeo e elimina parte das informações de cores. O segundo, conhecido por "quantização", normalmente é aplicado no domínio das

frequências e elimina ou atenua as frequências de menor importância para o sistema visual humano. Por isso, este tipo de processo de codificação é chamado de "codificação com perdas". É importante destacar que a eliminação de informações de acordo com as características de percepção do sistema visual humano contribui para que o codificador atinja elevadas taxas de compressão, com pequeno impacto na qualidade subjetiva da imagem.

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

2.6 Codificadores/decodificadores de vídeo para compressão

Há diversas formas de codificar um sinal de vídeo com o objetivo de compressão. A forma mais direta seria a aplicação de um codificador de entropia como o ilustrado na Figura 2.9 para codificar a matriz de amostras de cada uma das imagens que compõem a sequência de vídeo em seu formato original.

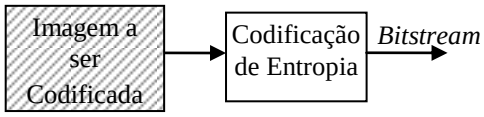

Figura 2.9: Um codificador elementar genérico: Apenas codificação de entropia.

Codificadores desta natureza funcionam bem para alguns tipos de imagens estáticas. Por exemplo, o padrão GIF (*Graphics Interchange Format*) (MIANO, 1999), utiliza um compressor de entropia baseado na contagem de sequências de valores repetidos RLE (*Run-Lenght encoding*) na matriz de amostras da imagem. Possui boa taxa de compressão para imagens que contenham grandes blocos com cores sólidas (sem variação de tonalidade) e é amplamente utilizado, mas não possui desempenho aceitável para cenas naturais, onde variações suaves de cores precisam ser codificadas como a imagem da Figura 2.10.

Figura 2.10: Exemplo de cena natural.

Diversas outras técnicas de codificação de entropia também possuem resultados ruins para a codificação de vídeo, pois a redundância em uma sequência de vídeo não está claramente evidente para os métodos clássicos como (VLC, huffman, etc.) pois estes algoritmos consideram os dados de entrada como brutos (não claramente organizados) e, mesmo que possível, a busca da redundância por um codificador de entropia para aplicações genéricas pode levar a uma complexidade computacional excessivamente elevada. Por isso, o conhecimento de maiores detalhes da fonte e de redundâncias a serem exploradas, tais como as descritas na seção anterior, podem levar

ao projeto de um codificador mais eficiente tanto em capacidade de compressão quanto em tempo de execução.

Além disso, os compressores de vídeo geralmente comprimem **com perdas**. A compressão com perdas requer uma análise mais criteriosa do sinal de vídeo com o objetivo de descartar informações que sejam irrelevantes ou não representativas para a compressão, mas não para a visão. Por esta razão, algoritmos muito mais especializados devem ser empregados para a codificação de imagens estáticas e vídeo, especialmente quando se deseja explorar as redundâncias apresentadas na seção anterior, inclusive a compressão com perdas. A literatura apresenta diversas alternativas para a compressão de vídeo, entretanto os codificadores híbridos (RICHARDSON, 2003) como o H.264 são aqueles que atualmente possuem o melhor compromisso entre capacidade de compressão e complexidade computacional. Outros tipos de codificadores estão fora do escopo deste trabalho. Os codificadores híbridos são assim ditos, pois implementam tanto ferramentas de predição do sinal fonte (temporal e espacial) quanto ferramentas que convertem e operam o sinal no domínio das frequencias (RICHARDSON, 2003). **A partir deste ponto**, o termo ***Codificador*** ou ***Decodificador*** de vídeo será utilizado para caracterizar um codificador ou decodificador com o objetivo de **compressão** no modelo **híbrido**.

Com relação às redundâncias enunciadas na seção anterior, o objetivo de um codificador de vídeo é essencialmente reorganizar as informações da matriz de pixels com os seguintes objetivos principais:

- A redundância entrópica fique evidente para um codificador de entropia;
- Informações consideradas irrelevantes para a visão humana possam ser destacadas e descartadas em algum grau (codificação com perdas).

Como foi mencionado anteriormente neste capítulo, o olho humano possui mais resolução na distinção iluminação do que em cores, por esta razão o espaço de cores YCbCr com sub-amostragem na informação de crominância (Cb e Cr) (Figura 2.8) é uma alternativa tipicamente utilizada e reduz a quantidade de informação de entrada do codificador propriamente dito. A literatura em geral não considera a conversão para o espaço de cores YCbCr com crominância subamostrada como o início do processo de codificação de vídeo, embora perdas irreversíveis na informação de crominância sejam inseridas neste processo. Nesta seção todas as ilustrações levam em consideração apenas a informação de luminância. A informação de crominância, quando presente, poderá (e geralmente será) tratada como uma imagem adicional a ser codificada. Desta forma, o termo **amostra** será utilizado como o equivalente de uma das componentes de cor de um **pixel**, e será assumido que a amostra é de **luminância** (componente Y) quando não for especificado qual das componentes está sendo operada.

Para a exploração da redundância espacial uma alternativa amplamente utilizada é a conversão da imagem para o domínio das frequências. Para isto, uma matriz de transformação é utilizada com tamanho típico de 8x8. Cada imagem da sequência de vídeo é dividida em blocos do tamanho da matriz de transformada e esta é aplicada para cada um dos blocos da imagem. A transformada mais utilizada para a codificação de cenas naturais estáticas e sequências de vídeo é a DCT (*Discrete Cossine Transform*, Transformada Discreta de Cosseno). A Figura 2.11 ilustra a aplicação da transformada DCT em um bloco 8×8 da imagem da Figura 2.10.

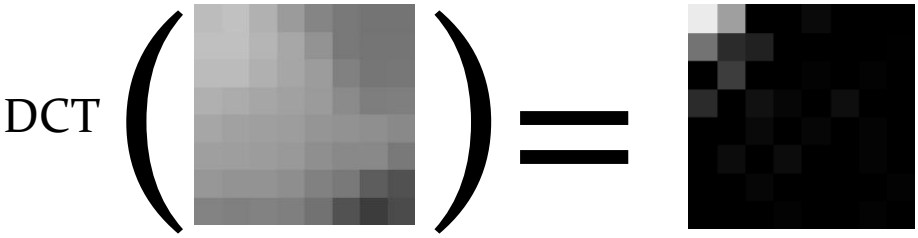

Figura 2.11: Transformada DCT.

Pela ampliação é possível distinguir claramente cada uma das amostras da matriz original e da matriz transformada. Na matriz original, as amostras são de 8 bits, enquanto que na matriz transformada são de 16 bits, onde apenas os 8 bits mais significativos foram convertidos para informação de intensidade do resultado. Na transformada DCT 2D utilizada neste exemplo, a informação de mais baixa frequência se concentra na direção do canto superior esquerdo. O valor do elemento do canto superior esquerdo é chamado de componente DC (em alusão a *Direct Current*, em eletricidade) e representa o valor médio do bloco original de amostras. O canto inferior direito concentra as frequências mais elevadas tipicamente presentes quando há bordas agudas na imagem.

Apesar da maior faixa dinâmica, o resultado tranformado reorganiza a informação de forma que o resultado típico para cenas naturais é o apresentado na Figura 2.11. O codificador de entropia pode lidar com a informação de cada ponto espectral de forma particular, aumentando a capacidade de compressão do codificador. A Figura 2.12 ilustra o processo de codificação quando a transformada é empregada.

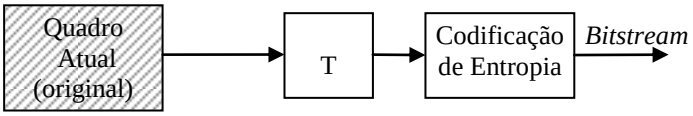

Figura 2.12: Codificador com Transformada DCT.

O principal benefício da transformada, entretanto, é a exploração da redundância psico-visual através da inserção de perdas controladas. Da mesma forma que o olho humano possui menor acuidade para cores, também é sabido que o olho humano é menos sensível a altas frequências (RICHARDSON, 2003). As amostras originais são **quantizadas** através da multiplicação ou divisão por uma matriz de quantização (na verdade requantização, uma vez que a informação digital é discreta e quantizada) e, com isso, inserir perdas. Para explorar as limitações do sistema visual humano, os coeficientes da matriz de quantização são escolhidos de forma a reduzirem mais intensamente a faixa dinâmica dos valores transformados de alta frequência do que dos de baixa frequência. Para aplicações típicas (DVD – Digital Video Disc, por exemplo), os valores do canto inferior direito da matriz resultante terão grande probabilidade de terem valor zero.

A taxa de produção de bits do codificador é variável e está diretamente ligada a entropia possível para a sequência de vídeo de entrada e da capacidade do codificador de aumentar a entropia (através da eficiência da codificação). Com o uso da quantização é possível controlar a taxa de produção de bits na saída do codificador variando os coeficientes da matriz de quantização. Com isso, é possível reduzir a taxa de bits

produzida pelo codificador em detrimento à qualidade de vídeo. Na prática, deve-se estabelecer matrizes de quantização que propiciem a maior redução na taxa de bits na saída do codificador (maior compressão) com uma deterioração aceitável na qualidade do vídeo. A Figura 2.13 ilustra duas imagens que são parte de uma sequência de vídeo onde a primeira (a) não sofreu compressão enquanto a segunda (b) sofreu muita compressão através de uma matriz de quantização com coeficientes muito elevados.

(a) Não comprimida (b) Muito comprimida

Figura 2.13: Comparação resultado da quantização (a) sem quantização (a); (b) com passo de quantização elevado.

Na Figura 2.13(b) os efeitos quadriculados apresentados na imagem são chamados efeitos de bloco e são típicos de compressores que operam as imagens da sequência de vídeo usando blocos. Este efeito se manifesta com mais intensidade quando os coeficientes da matriz de quantização são muito elevados.

Para tornar o envio das matrizes de codificação mais eficiente para o decodificador, as matrizes são tipicamente geradas por um algoritmo especifico para este fim, comum tanto ao codificador quanto ao decodificador. A entrada é um valor chamado parâmetro de quantização (QP – *Quantization Parameter*) e a saída é a matriz de quantização que será usada para multiplicar cada um dos coeficientes de transformada.

Para aplicações que exijam taxa de bits constante na saída do codificador (*broadcast*, por exemplo), pode-se empregar um algoritmo de controle de taxa que modifique durante o processo de codificação o valor do QP. Certamente o valor do QP que está sendo usado para codificar um dado bloco deve chegar ao decodificador através de uma informação específica para este fim. A Figura 2.14 ilustra o modelo de um codificador que usa transformada e quantização.

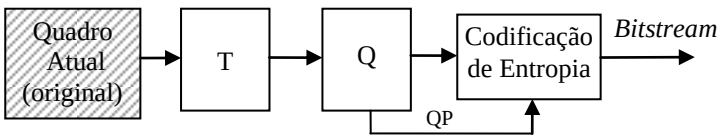

Figura 2.14: Codificador com Transformada e Quantização.

Outra forma de explorar a redundância espacial é através do uso de predição intra-quadro, que será discutida mais adiante nesta seção.

Para a exploração da redundância temporal um módulo de codificação diferencial é empregado, produzindo como saída informação residual.

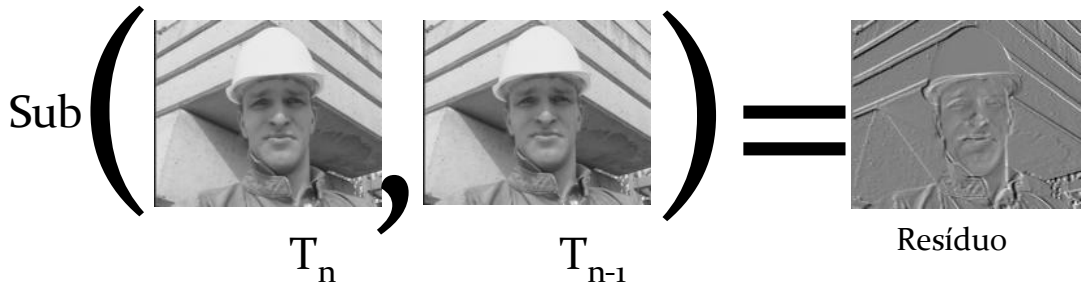

Figura 2.15: Codificação diferencial: Produção de resíduo.

A Figura 2.15 ilustra a codificação diferencial. Tipicamente duas imagens vizinhas no tempo são muito semelhantes. A subtração da informação (operador **Sub**) dos valores dos pixels colocalizados da imagem em um instante de tempo T_n pelos da imagem anterior (T_{n-1}) produz o resultado apresentado como resíduo. As imagens fonte T_n e T_{n-1} possuem faixa dinâmica de 8 bits (256 cores – intervalo [0, 255]) enquanto que a imagem resíduo possui faixa dinâmica maior (9 bits), pois a operação de subtração de dois valores de 8 bits pode gerar valores no intervalo [-255, 255]. Para a exibição do resíduo na Figura 2.15 a matriz de valores foi manipulada para permanecer no intervalo [0, 255] possível das imagens originais:

$$Resíduo = \frac{Rm}{2} + 128 \quad , \tag{2.1}$$

onde *Resíduo* é a matriz que forma a imagem exibida e *Rm* é a matriz de resíduos calculada.

Em um codificador hipotético que use codificação diferencial, a primeira imagem a ser codificada seria a imagem do tempo T_0 e como nenhuma imagem anterior a esta existe (T_{-1}), esta poderia ser assumida como sendo uma matriz onde todos os elementos possuem valor zero. Desta forma, o primeiro "resíduo" seria a própria imagem do instante T_0. Com essa premissa, o decodificador será capaz de reconstruir a sequência de imagens originais, fazendo a operação reversa da codificação diferencial, isto é, somando o resíduo do instante de tempo T_n com a imagem reconstituída do instante de tempo T_{n-1}, obtendo assim a imagem do instante de tempo T_n.

O benefício da codificação diferencial é mais bem compreendido quando se analisa o histograma das imagens originais e do resíduo gerado pela sua subtração. A Figura 2.16 apresenta de forma sobreposta o histograma das duas imagens originais utilizadas para o exemplo de codificação diferencial apresentado na Figura 2.15 e a Figura 2.17 apresenta o histograma do resíduo.

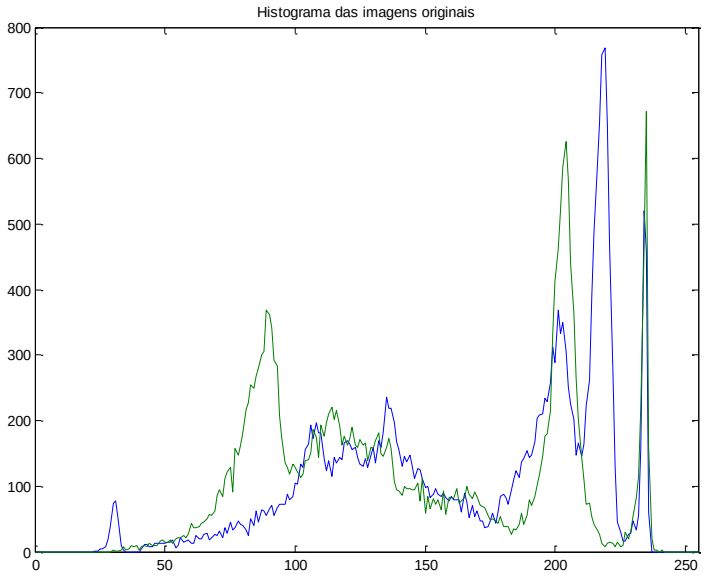

Figura 2.16: Histogramas de duas imagens subsequentes de uma sequência de vídeo com resolução de 176x144 pixels.

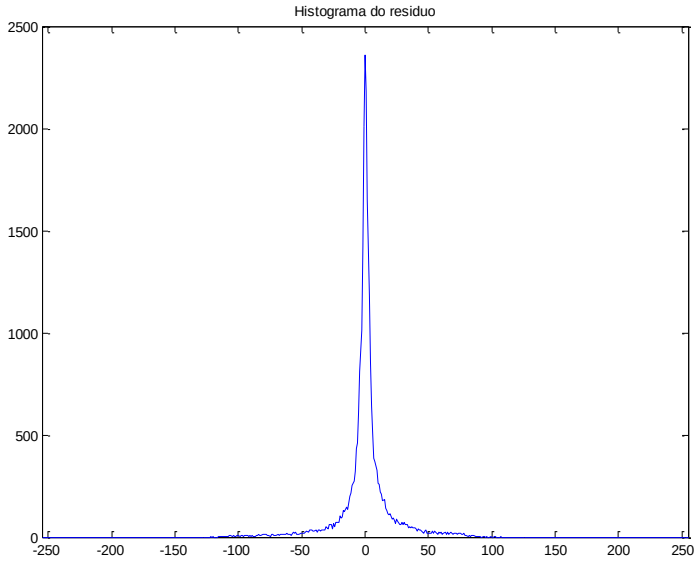

Figura 2.17: Histograma do resíduo produzido pela subtração de duas imagens subsequentes de uma sequência de vídeo com resolução de 176x144 pixels.

Observe que enquanto na Figura 2.16 os valores das amostras de ambas as imagens estão distribuídos de forma desigual ao longo do espaço de representação [0,255] (8 bits), na Figura 2.17, o valor dos resíduos encontra-se distribuído muito próximo ao valor zero com uma grande ocorrência do valor zero. Isso facilita o processo de codificação de entropia, onde em um codificador com códigos de comprimento variável (VLC - *variable length codes*) pode atribuir o código de menor tamanho para o valor zero e códigos de tamanhos maiores para os valores que ocorrem menos frequentemente, gerando uma grande compressão entrópica, não evidente nas imagens originais.

Embora a sequência de duas imagens analisadas produza um resíduo de boa qualidade (isto é, facilita a compressão entrópica), pode haver movimento de câmera ou de objetos na cena entre imagens subsequentes. Por isso, incorporar um estimador/compensador de movimento no processo de exploração da redundância temporal pode ajudar significativamente na produção de resíduos de boa qualidade para a codificação de entropia. Como os objetos da cena podem possuir movimento em direções distintas e o codificador híbrido não conhece os objetos da cena, apenas os pixels nela representados, é típico repartir a imagem atual em pequenos blocos de pixels e buscar vetores de movimento que apontem a posição de cada um destes blocos na imagem de referência, usando uma função de minimização do resíduo. O bloco pode ter tamanho fixo ou seu tamanho pode ser variável e escolhido de acordo com critérios que levem a uma codificação mais eficiente. No primeiro caso, o codificador terá que enviar para o decodificador a informação residual para cada um dos blocos e também os vetores de movimento; no segundo caso, a informação de tamanho de bloco também deve ser informada ao decodificador. A Figura 2.18 ilustra uma porção de uma imagem que foi dividida em blocos de 8x8 sobreposta pelos vetores de movimento calculados por um critério de minimização de resíduo chamado soma das diferenças absolutas (SAD – *Sum of absolute differences*).

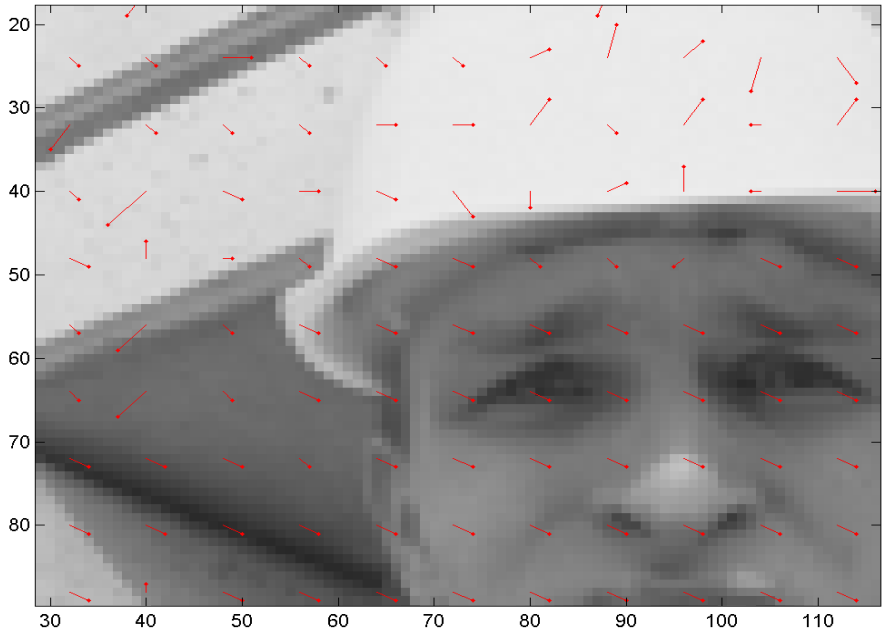

Figura 2.18: Vetores de movimento gerados pela busca de blocos de 8x8 pixels no quadro anterior da sequência de vídeo sobrepostos na imagem atual.

Na Figura 2.18, a origem dos vetores de movimento está colocalizada com o canto inferior direito do bloco a eles associado e a haste aponta o ponto colocalizado na imagem anterior (marcado por um ponto na haste) onde pode ser encontrado um bloco de igual tamanho que minimiza o resíduo segundo critério SAD com área de busca limitada a uma região de [-7,8] pixels em ambas as direções neste exemplo. Observe que o critério SAD é um critério de minimização de erro que não necessariamente irá selecionar um vetor na direção do movimento: em alguns casos a minimização do erro ocorre com vetores que não apontam na direção do movimento. O codificador, em

busca de eficiência, pode usar codificação diferencial também para os vetores de movimento, transmitindo apenas a diferença entre o vetor do bloco atual e os vetores vizinhos já enviados ao decodificador. Um critério de minimização que leve em conta a quantidade de informação contida no vetor de movimento pode produzir melhores resultados em termos de compressão, mesmo que produza um pouco mais de informação residual.

O processo de busca dos vetores de movimento, segundo um critério de busca qualquer, é chamado de Estimação de Movimento (ME - *motion estimation* – o termo *Estimativa de Movimento* é outra tradução possível, porém a estimativa é o resultado da aplicação do processo de estimação, não sendo usado na área). A operação reversa, a ser realizada pelo decodificador, é chamada de compensação de movimento e reconstitui a imagem original a partir da informação de resíduo e dos vetores de movimento. A Figura 2.19 ilustra o resíduo obtido a partir do processo de Estimação de Movimento (ME) com blocos de 8x8 amostras usando o critério SAD.

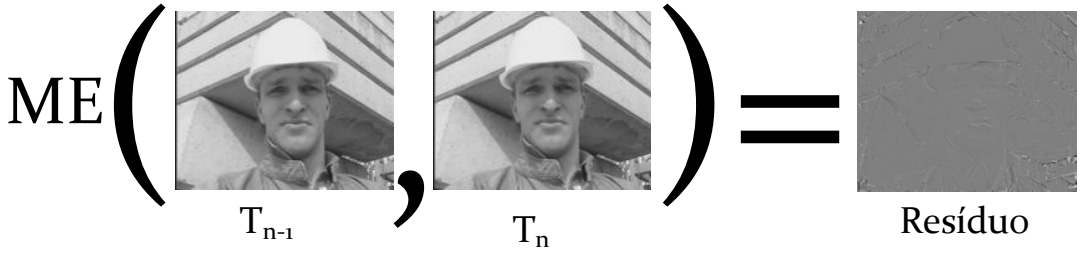

Figura 2.19: Resíduo calculado utilizando estimação/compensação de movimento.

As imagens da Figura 2.19 são as mesmas utilizadas para a operação de subtração ilustrada na Figura 2.15. A Figura 2.20 compara os resíduos produzidos pelos dois processos.

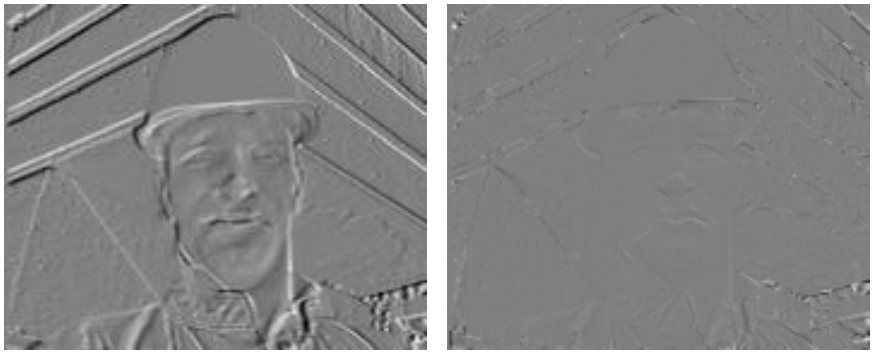

Figura 2.20: Resíduo calculado utilizando estimação de movimento.

O ganho através da redução da informação residual é evidente com a utilização da ME, apesar da diferença visual entre as duas imagens originais ser muito pequena. O ganho final de codificação dependerá da forma de codificação que for utilizada para os vetores de movimento e do ajuste entre esta forma de codificação e o algoritmo de busca dos vetores de movimento. Como o codificador pode ser projetado de forma a não enviar vetores de movimento em situações onde não há ganho no envio destes, no pior caso, um codificador que implementa ME será igual a um que usa apenas a codificação

diferencial simples. Finalmente a Figura 2.21 ilustra o histograma do resíduo ilustrado na Figura 2.19 e Figura 2.20(b).

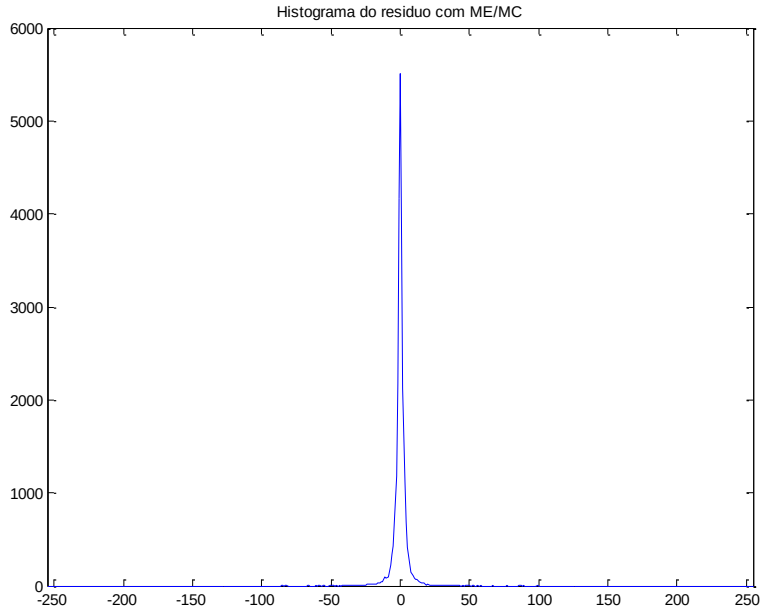

Figura 2.21: Histograma do resíduo usando o processo de ME.

Comparando o histograma da Figura 2.21 com o da Figura 2.17, observa-se no último que a ocorrência de valores próximos a zero é ainda maior, aumentando os benefícios já descritos anteriormente para a codificação de entropia.

Há situações, como a ilustrada na Figura 2.22 em que há uma troca de cena onde a codificação diferencial definitivamente não produz bons resultados, mesmo se usada com Estimação de Movimento.

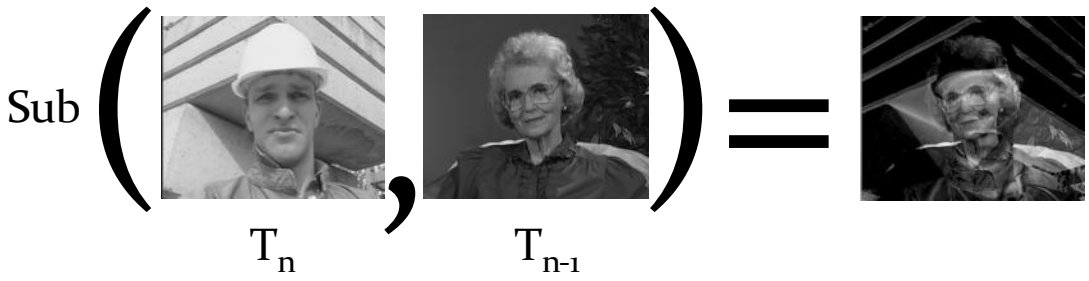

Figura 2.22: Troca de cena na codificação diferencial.

Um codificador que use exclusivamente da exploração da redundância temporal irá produzir muito mais bits em sua saída (codificação menos eficiente) quando não houver redundância temporal ou quando esta não for evidenciada pelos algoritmos de estimação de movimento usados no codificador. A Figura 2.23 ilustra o modelo de um codificador DPCM com estimação/compensação de movimento.

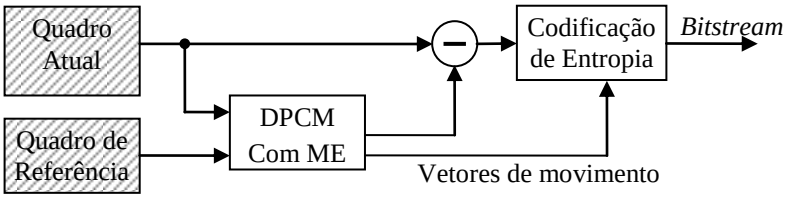

Figura 2.23: Codificador de vídeo DPCM com ME.

Até o momento foram apresentados codificadores que:

A. Exploram a redundância entrópica (Figura 2.9);
B. Utilizam transformadas em bloco para operar no domínio das frequências explorando a redundância espacial e facilitando a entropia (Figura 2.12)
C. Eliminam de forma controlada a informação considerada irrelevante para o olho humano através da quantização dos coeficientes transformados, explorando a redundância psicovisual (Figura 2.14)
D. Exploram a redundância temporal através de um laço DPCM com estimação de movimento (Figura 2.23)

A Figura 2.14 apresenta as técnicas dos itens A, B e C já integradas. A integração da técnica do item D no codificador consiste em inserir a transformada e a quantização após a produção da informação residual e antes da codificação de entropia, como proposto no modelo da Figura 2.24.

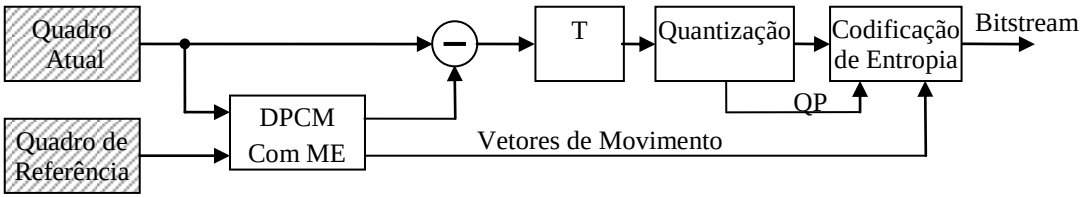

Figura 2.24: Modelo inicial de compressor de vídeo híbrido.

Em um primeiro momento, este modelo parece adequado e funcional. Entretanto, como a quantização irá inserir perdas, a imagem reconstituída pelo decodificador será diferente da imagem original. Esta imagem com perdas será utilizada como quadro de referência para o decodificador DPCM com MC (*Motion Compensation* - compensação de movimento) reconstruir a próxima imagem. Isto leva a propagação cumulativa do erro produzido pela quantização produzindo uma diferença entre o quadro que é usado como referência para o codificador e o que é usado como referência pelo decodificador para reconstruir o quadro. Para resolver este problema, o codificador terá que ser capaz de reconstituir o quadro após a quantização, fazendo o caminho inverso até obter um quadro reconstituído idêntico àquele que será produzido pelo decodificador e usar este quadro reconstituído como referência para a predição DPCM com ME, de forma a eliminar a propagação do erro cumulativo. A Figura 2.25 ilustra este processo.

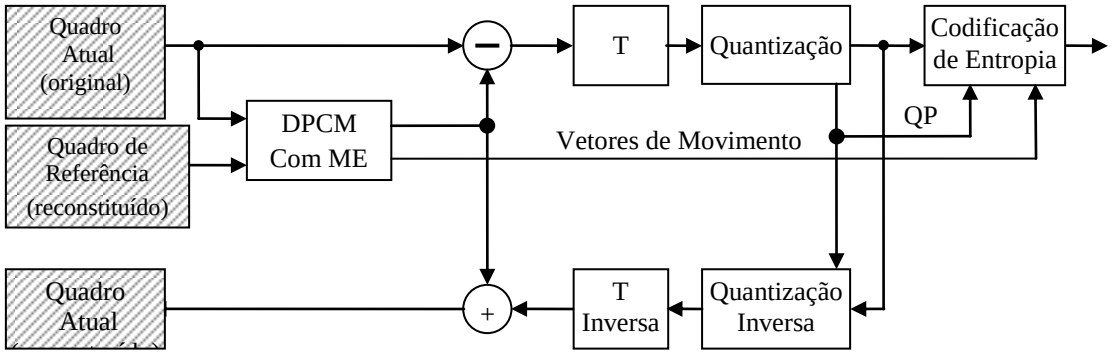

Figura 2.25: Compressor de vídeo híbrido com laço de reconstrução .

Finalmente, outro modo de redundância espacial pode ser explorado através da predição intra-quadros. Neste modo a imagem é divida em blocos e usa o que é denominado **modos predição** onde a informação de vizinhos já codificados é usada para estimar o valor das amostras do boco que está sendo codificado. O preditor irá encontrar um modo pré-estabelecido de preenchimento dos valores do bloco que torne este bloco o mais parecido com o bloco da imagem original. Neste caso, o codificador terá que enviar ao decodificador uma informação de modo de codificação, além da informação residual. A predição intra-quadros pode coexistir com o laço DPCM com ME/MC, sendo possível ao codificador selecionar entre o modo espacial e temporal para bloco da imagem a ser codificada. A Figura 2.26 ilustra o modelo de um codificador completo que explora todas as técnicas de exploração de redundâncias.

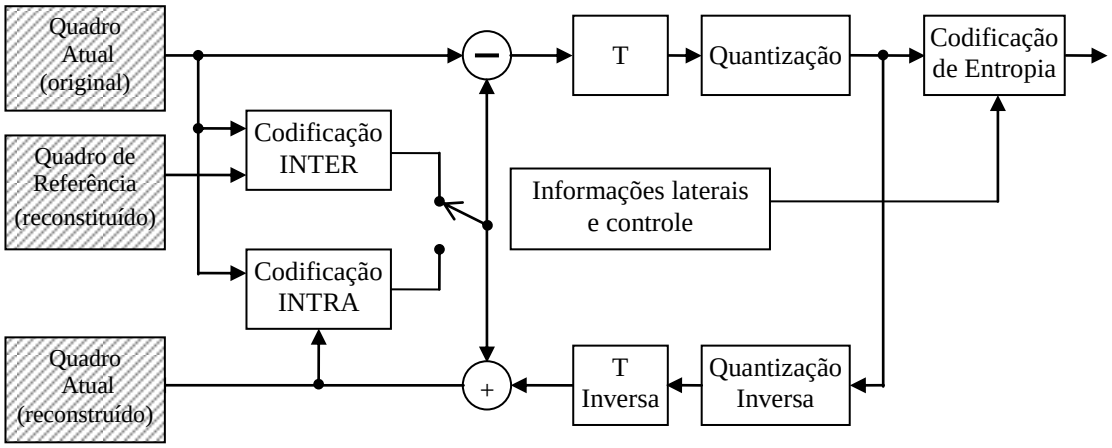

Figura 2.26: Modelo completo de um codificador de vídeo.

Neste modelo, observe a presença de um bloco “informações laterais e controle”. Por simplicidade, todas as informações laterais (vetores de movimento, modos intra, parâmetros de quantização, etc...) e de controle foram concentradas em um único bloco. Estas informações são indispensáveis para que o decodificador possa reconstruir corretamente o quadro codificado e devem ser enviadas de forma eficiente através da codificação de entropia.

A Figura 2.27 apresenta o modelo de um decodificador de vídeo com as mesmas características do codificador apresentado na Figura 2.26.

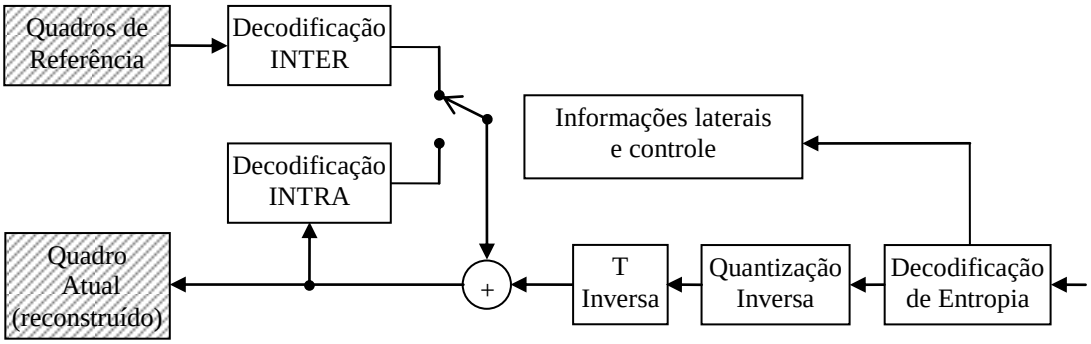

Figura 2.27: Modelo de um decodificador de vídeo.

Para tornar a codificação ainda mais eficiente, este modelo de codificador e decodificador pode incluir diversas técnicas que permitem melhorias na eficiência de codificação. Algumas destas técnicas são listadas abaixo:

- Tamanho de bloco variável: selecionar o tamanho de bloco que seja mais adequado para a região da imagem que está sendo codificada
- Seleção do tipo de predição: seleconar o tipo de predição (temporal ou espacial) a ser utilizada para cada bloco codificado
- Múltiplas transformadas: selecionar o tipo ou tamanho de transformada mais adequado para a região da imagem que está sendo processada
- Multiplos quadros de referência: selecionar o quadro de referência (já codificado) mais adequado para a predição temporal de um bloco ou quadro
- Quadros de referência futuros: capacidade de codificar os quadros fora de ordem de forma a selecionar quadros futuros como referência para a predição temporal
- Outras predições: codificação diferencial dos vetores de movimento a partir dos vetores codificados na vizinhança, codificação diferencial do QP,
- Modo skip: flags que indicam o não envio de informação para parte ou a totalidade do bloco codificado
- Técnicas de entropia: varredura zig-zag dos coeficientes, mapas de significância, uso adaptativo de modos VLC, codificação aritmética adaptativa ao contexto, etc..
- Filtragem: Uso de filtros para a redução de artefatos produzidos pela introdução de perdas. A filtragem pode ser uma etapa de pré ou pós-processamento ou ainda estar no laço de reconstrução.

Quanto mais sofisticado for o codificador mais capacidade de compressão ele terá. Entretanto, é importatante destacar que para cada técnica suportada no codificador, esta tem que ser suportada também no decodificador. De fato, um padrão de codificação de vídeo deve definir apenas o decodificador através das técnicas (também chamadas de ferramentas) empregadas e de suas respectivas sintaxes e semânticas. Um codificador compatível terá que se adequar às restrições impostas pelo decodificador.

3 O PADRÃO H.264

Este capítulo irá apresentar o padrão H.264 (ITU, 2005) de forma resumida, com um breve histórico sobre o padrão, a terminologia utilizada, os perfis e níveis do padrão e uma comparação com padrões anteriores. Será apresentada uma análise de complexidade e os principais desafios envolvidos na implementação de um codec H.264. Será dado um enfoque especial para os blocos do padrão que serão alvo de exploração arquitetural.

3.1 Introdução ao Padrão H.264

O padrão H.264/AVC é uma evolução de padrões anteriores. Nesta seção será apresentado um pequeno histórico do padrão e a terminologia básica. Serão discutidos os perfis e os níveis suportados para tornar o padrão flexível o bastante para uma ampla gama de aplicações e finalmente será apresentada uma comparação do padrão H.264 com outros padrões de compressão de vídeo.

3.1.1 Histórico do Padrão

O padrão H.264 foi iniciado pelo VCEG (*Video Coding Experts Group*) cujas raízes estão no projeto H.26L (sendo L indicando *Long-term*) e foi desenvolvido por um período de aproximadamente quatro anos (ITU, 2005a). A chamada para propostas ocorreu no início de 1998 e o primeiro *draft* deste novo padrão foi completado em agosto de 1999. O objetivo final do projeto H.26L era dobrar a eficiência de codificação atingida pelo padrão H.263 (ITU, 2000).

O padrão H.264 foi desenvolvido como uma evolução natural de diversos outros padrões já estabelecidos e consolidados. O primeiro padrão relevante para a construção do H.264 foi o H.261 da ITU-T (ITU, 1990). A estrutura básica dos codificadores híbridos modernos é a mesma deste padrão: A codificação diferencial (DPCM) com estimação de movimento; a transformada discreta do cosseno aplicada no resíduo; a quantização linear e finalmente a codificação de entropia. Este é conhecido como modelo híbrido de codificação, como descrito na seção anterior. A evolução deste padrão levou ao MPEG-1 da ISO/IEC (ISO, 1993) seguido do padrão MPEG-2 da ISO/IEC, que também se tornou uma recomendação da ITU-T com a denominação H.262 (ITU, 1994). O padrão MPEG-2/H.262 se tornou um padrão popular por ter sido adotado por algumas formas de TV-Digital e pelo formato DVD. Apesar do grande sucesso do padrão MPEG-2, a evolução dos padrões de compressão de vídeo não parou devido à necessidade de maior taxa de compressão e da disponibilidade de maior capacidade de computação. O padrão H.263 (ITU, 2000) foi uma evolução em relação aos padrões descritos anteriormente e incorporou alguns avanços obtidos pelos padrões MPEG-1 e MPEG-2, bem como técnicas novas que vinham sendo pesquisadas pela academia e pela indústria e que possibilitam maior ganho de compressão. Em 2001 foi

finalizado o desenvolvimento do padrão MPEG (*Moving Pictures Experts Group*) da ISO/IEC (ISO/IEC, 2005) mais recente, conhecido como MPEG-4 Parte 2 (ISO, 1999). Ainda em 2001 o MPEG realizou uma nova chamada de propostas, similar à do H.26L da ITU-T, para melhorar ainda mais a eficiência de codificação atingida pelo MPEG-4. Como resposta, o VCEG, da ITU-T, resolveu submeter seu *draft* à chamada de propostas do MPEG e propôs a união de esforços para completar o trabalho.

O MPEG chegou a conclusões que afirmaram as escolhas de desenvolvimento realizadas pelo VCEG no H.26L após analisar as respostas para sua chamada de propostas, (RICHARDSON, 2003): a estrutura de compensação de movimento com a transformada discreta do cosseno (DCT) era melhor do que as outras; algumas ferramentas de codificação de vídeo que foram excluídas no passado (do MPEG-2, do H.263 ou do MPEG-4 Parte 2) por causa da sua complexidade computacional, poderiam ser reexaminadas para inclusão no próximo padrão, devido aos avanços na tecnologia de hardware. Uma nova sintaxe deveria ser criada para maximizar a eficiência de codificação quebrando a compatibilidade com os padrões anteriores.

Para evitar duplicação de esforços e acelerar o desenvolvimento, o ITU-T e o ISO/IEC concordaram em unir esforços para desenvolverem a próxima geração do padrão para codificação de vídeo em conjunto e concordaram em usar o H.26L como ponto de partida. Então, foi criado, em dezembro de 2001, o JVT (*Joint Video Team*) (ITU, 2005), formado por especialistas do VCEG a do MPEG. O JVT tinha o objetivo de completar o desenvolvimento técnico do padrão até o ano de 2003. Posteriormente a ITU-T decidiu adotar H.264 como a denominação para sua nova recomendação e a ISO/IEC decidiu incorporar a especificação como parte do padrão MPEG-4, recebendo a denominação de MPEG-4 parte 10 – AVC (*Advanced Video Coding* - Codificação de Vídeo Avançada). O padrão H.264 teve seu *draft* final (ITU, 2003) aprovado em outubro de 2003 (SULLIVAN, 2005). O esforço conjunto resultou em um documento compartilhado por ambas as entidades e recebeu o título de "*recommendation | international standard*" (recomendação | padrão internacional), pois os documentos da ITU são denominados recomendações, enquanto os da ISO são denominados padrões internacionais. Neste texto será adotada a denominação "padrão H.264", pois esta é a denominação mais comum, apesar do documento produzido pela ITU ser uma recomendação e não um padrão.

Após a finalização do documento original em 2003 o JVT continuou suas atividades no sentido de promover melhorias ao padrão através de extensões. Em julho de 2004, o JVT adicionou algumas novas funcionalidades ao padrão H.264 através de uma extensão do padrão chamada de *Fidelity Range Extensions* (FRExt) (ITU, 2004; ITU, 2005). O próximo esforço foi incluir o suporte a escalabilidade, denomidado *Scalable Video Coder* (SVC) em 2007 (ITU, 2007) e *Multi-view Video Coder* (MVC) em 2009 (INTERNATIOANAL, 2009). Após a criação destas novas extensões, é comum a denominação "H.264/AVC" para a versão de 2005 do padrão (incluindo o trabalho FRExt), "H.624/SVC" para a versão escalável e "H.264/MVC" para a versão multi-visão.

3.1.2 Definições

Uma imagem é composta por pixels (Picture Elements – Elementos de imagem) que podem ser decompostos em amostras, que são elementos de cor da imagem. As imagens monocromáticas possuem uma amostra por pixel. Nas imagens coloridas, cada pixel possui tipicamente 3 amostras (espaço RGB, YCbCr 4:4:4), porém em muitos casos a

imagem pode ser subamostrada, por razões descritas no capítulo anterior, e, por exemplo, no formato YCbCr 4:2:0, uma imagem pode ter apenas uma amostra de cada componente de cor para cada grupo de 4 pixels.

Grupos de 16x16 pixels alinhados com o canto superior direito da imagem em incrementos de 16 unidades em cada direção, em qualquer que seja o espaço de cores são chamados de macroblocos no padrão H.264; estes podem ser divididos em blocos de até 4x4 pixels, chamados de submacrobloco.

Um agrupamento de macroblocos forma um *Slice* (fatia) da imagem.

Um Quadro (*Frame*), em vídeo progressivo, ou Campo (*Field*), em vídeo entrelaçado, é composto por um ou mais *Slices*.

A informação contida em um *bitstream* H.264 é organizada hierarquicamente. Há uma camada mais externa chamada NAL (*Network Abstraction Layer*) que encapsula toda a informação transmitida pelo codificador na forma de blocos de dados de tamanho variável com marcação de início inequívoca através de um algoritmo de prevenção de emulação de cabeçalho (a sequência de bytes que delimita o início de uma NAL jamais ocorre em outro ponto da NAL). A NAL pode conter informação de configuração do decodificador (parâmetros da sequência e da imagem) e informação de vídeo codificado. As NALs que contém informação de vídeo codificado são categorizadas como sendo do tipo VCL (*Video Coding Layer* – camada de codificação de vídeo). As NALs do tipo VCL encapsulam unidades de codificação chamadas *Slices* (fatias), que são partes de um quadro ou campo do vídeo. O padrão prevê alguns tipos de slices que informam ao decodificador como a informação de vídeo codificada está organizada (quadro, campo, MBAFF, Intra, Predito, Bidirecional, etc...). Os *slices* que compõem um quadro ou campo podem inclusive ter ordem arbitrária ou possuir esquemas especiais de posicionamento dos macroblocos que não apenas fatias da imagem. Porém um quadro ou campo é codificado em um único *slice* para a maioria das aplicações, exceto aquelas onde a robustez a erros é privilegiada em detrimento a taxa de compressão.

Em um *slice*, a informação está organizada na forma de macroblocos, sequenciados tipicamente em ordem *raster scan* (da esquerda para a direita para formar linhas de cima para baixo).

Os macroblocos podem ser do tipo I ou SI, quando codificados usando apenas informação já codificada do próprio quadro/campo; P ou SP, quando são codificados usando informação de um quadros/campos anterior ou posterior temporalmente; ou B, quando são codificados usando informação de dois quadros.

Um *slice* do tipo I pode conter somente macroblocos do tipo I. Um *slice* do tipo P pode conter macroblocos do tipo P e I e um *slice* do tipo B pode conter macroblocos do tipo B e I. Além dos *slices* tipo I, P e B, o padrão também permite a existência de outros dois tipos de *slices*: SI e SP. (RICHARDSON, 2003).

Para permitir o chaveamento entre fluxos de bits diferentes, *slices* SP (*Switching* P) e SI (*Switching* I) são transmitidos para permitir o chaveamento sem causar um grande prejuízo na eficiência de codificação (KARCZEWICZ, 2003). Em aplicações de vídeo sob demanda, pode ser necessário o chaveamento entre fluxos de bits diferentes, mas este chaveamento não pode ser acontecer logo antes de um quadro P ou B, pois a codificação destes quadros usa como referência quadros do fluxo de bits atual. É

possível fazer o chaveamento usando um quadro do tipo I, mas o problema é que estes quadros consomem muito mais bits do que os quadros P ou B.

A codificação dos macroblocos tipo I pode acontecer sobre macroblocos completos ou para cada bloco de 4x4 pixels. No FRExt também foi incluída a predição sobre blocos de 8x8 pixels (ITU, 2005). Os macroblocos do tipo I podem, ainda, ser codificados através do modo I_PCM, onde o codificador transmite os valores das amostras diretamente, sem predição ou transformação. Em alguns casos anômalos, o modo I_PCM pode ser mais eficiente do que os demais modos de codificação (RICHARDSON, 2003).

Um macrobloco codificado no modo inter (P/SP) pode ser dividido em partições de macroblocos, isto é, em blocos de 16x16, 16x8, 8x16 ou 8x8. Se o tamanho de partição escolhido for o 8x8, então cada sub-macrobloco 8x8 pode ser dividido novamente em partições de sub-macrobloco de tamanho 8x8, 8x4, 4x8 ou 4x4. Cada partição de macrobloco é codificada utilizando como referência um quadro da lista 0. Se existir uma partição em sub-macrobloco, cada sub-macrobloco é codificado a partir do mesmo quadro de referência utilizado na codificação da partição de macrobloco.

Nos macroblocos tipo B, cada partição de macrobloco pode ser codificada utilizando um ou dois quadros de referência, um na lista 0 e outro na lista 1. Os mesmos quadros de referência são usados para a codificação das partições dos macroblocos, caso existam.

3.1.3 Perfis e Níveis

Para suportar um amplo espectro de aplicações, o padrão H.264 possui 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 (ITU, 2003), define um grupo de três diferentes perfis: *Baseline*, *Main* e *Extended*. O perfil *Baseline* é direcionado a aplicações embarcadas, onde a complexidade e o consumo de energia são parâmetros-chave. O perfil *Baseline* suporta codificação intra e inter (usando somente *slices* I e P) e uma codificação de entropia com códigos de comprimento de palavra variável adaptativos ao contexto (CAVLC). 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 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). A atualização de 2005 da norma (ITU, 2005), que inclui as extensões propostas como FRExt , adiciona um conjunto de perfis *high*, que incluem uma série de melhorias na qualidade do vídeo codificado.

Nos pefis *baseline*, *main* e *extended*, o formato YCbCr 4:2:0 é o único suportado. Isso significa que os elementos de crominância Cb e Cr possuem metade da resolução horizontal e vertical dos elementos de luminância, implicando em uma sub-amostragem dos elementos de crominância já no vídeo original. Esta sub-amostragem contribui na redução da redundância psicovisual, como foi explicado na Seção 2.2 do texto. Os perfis *Baseline*, *Main* e *Extended* também possuem outra característica em comum que é o uso de 8 bits por amostra.

Nos perfis *high*, podem ser suportados outros espaço de cores além do YCbCr, incluindo vídeo monocromático. Esquemas de sub-amostragem 4:2:2 e 4:4:4 também são suportados. No perfil *high10*, as amostras podem ser de 10 bits. Em todos os perfis high, existe suporte a transformada e quantização de blocos de 8x8 amostras, além do suporte a transformada e quantização 4x4 já presente nos perfis anteriores. Também são suportadas matrizes de quantização adaptativas (SULLIVAN, 2004).

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

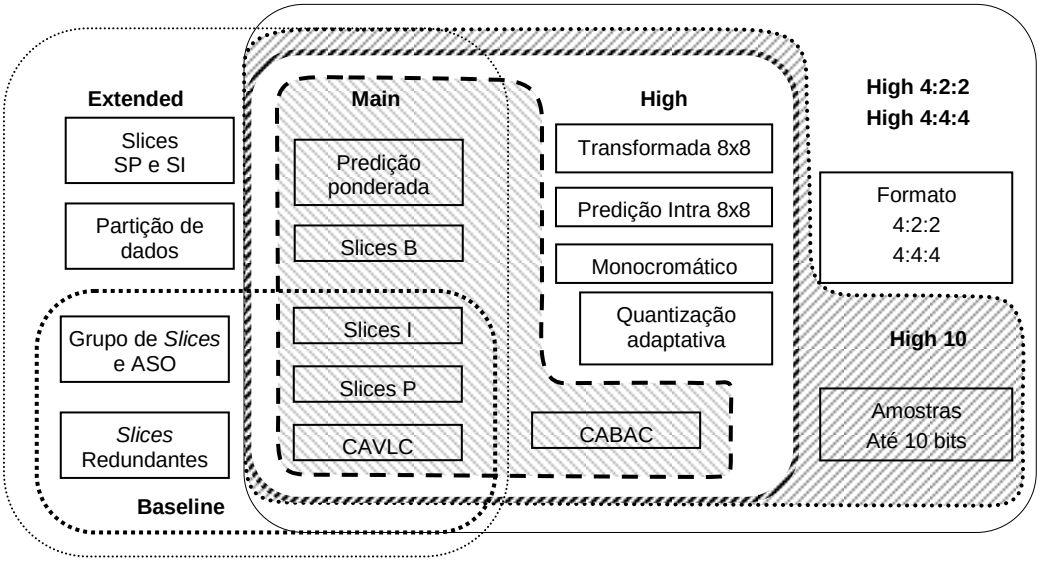

Figura 3.1: Perfis *Baseline*, *Main*, *Extended* e *High* do padrão H.264

Tabela 3.1: Níveis do H.264.

Nível	**Tamanho: Macroblocos por quadro**	**Processamento: Macroblocos/s**	***Bitstream*: kbits/s**
1	99	1485	64
1b	99	1485	128
1.1	396	3000	192
1.2	396	6000	384
1.3	396	11880	768
2	396	11880	2000
2.1	792	19800	4000
2.2	1620	20250	4000
3	1620	40500	10000
3.1	3600	108000	14000
3.2	5120	216000	20000
4	8192	245760	20000
4.1	8192	245760	50000
4.2	8704	522240	50000
5	22080	589824	135000
5.1	36864	983040	240000

Além da divisão em diversos perfis, o padrão H.264 também define 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). A tabela 3.1 ilustra algumas características dos níveis do padrão H.264 comuns a todos os perfis.

Em alguns perfis high e novas extensões do padrão, alguns destes valores são escalonados para refletir o aumento da quantidade de dados a ser processados pelo decodificador (ITU, 2009).

3.1.4 Formato de Dados Codificados

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

O objetivo de especificar separadamente a VCL e a NAL é diferenciar características específicas da codificação (na VCL) daquelas características específicas do transporte (na NAL). Em especial está a capacidade da NAL de fornecer um cabeçalho que jamais é emulado (está presente) no restante de seu conteúdo, facilitando a ressincronização do decodificador em caso de perda de dados (ITU-T, 2005b; RICHARDSON, 2003).

3.1.5 Comparação com Padrões Anteriores

A Tabela 3.2 apresenta uma comparação entre os padrões H.264 original (ITU, 2003), MPEG-4 Parte 2 e MPEG-2 do ponto de vista das diferentes características suportadas por estes padrões (KWON, 2005). As características do H.264 que ainda não foram discutidas nas seções anteriores serão explicadas em mais detalhes nas seções posteriores deste trabalho.

O desenvolvimento do padrão H.264 teve como principal objetivo aumentar a eficiência da codificação. A seguir será apresentada uma comparação entre o padrão H.264 e outros padrões, com foco na eficiência de codificação. Os dados apresentados nesta seção foram colhidos da literatura, e apresentam algumas diferenças, mas todos indicam que o padrão H.264 apresenta ganhos significativos sobre outros padrões em termos de eficiência de codificação.

O trabalho de (KAMACI, 2003) apresenta uma comparação entre o padrão H.264, o padrão MPEG-2 e o padrão H.263, este último nos perfis *Baseline* e *High*. Nesta comparação, é considerada a relação sinal/ruído (PSNR – *Peak signal-to-noise ratio*) sobre vídeos nos formatos CCIR-601, CIF e QCIF. De acordo com os autores, o padrão H.264 atinge cerca 50% de ganho de codificação sobre o padrão MPEG-2, cerca de 47% de ganho sobre o padrão H.263 *Baseline* e cerca de 24% de ganho sobre o padrão H.263 *High*.

Outro trabalho (WIEGAND, 2003) compara a relação sinal ruído entre os padrões MPEG-2, H.263, MPEG-4 e H.264. Foram usadas sequências de vídeos no formato QCIF e CIF, com diferentes taxas de quadros por segundo. A Tabela 3.3 apresenta os

resultados comparativos nos ganhos médios na taxa de bits do padrão H.264 sobre os demais padrões.

Tabela 3.2: Comparação das características dos padrões H.264, MPEG-4 Parte 2 e MPEG-2

	H.264	MEPG-4 parte 2	MPEG-2
Tamanho de Macrobloco	16x16	16x16	16x16 (modo quadro) 16x8 (modo campo)
Tamanho de bloco	16x16, 8x16, 16x8, 8x8, 4x8, 8x4, 4x4	16x16, 16x8, 8x8	8x8
Predição Intra	Domínio espacial	Domínio das frequências	Não
Transformada	DCT inteira 8x8, 4x4 Hadamard 4x4, 2x2	DCT/Wavelet 8x8	DCT 8x8
Quantização	Escalar, em incrementos de 12,5%	Vetorial	Escalar, com incremento de passo constante
Codificação de Entropia	CAVLC, CABAC	VLC	VLC
Precisão de pixels	¼ pixel	¼ pixel	½ pixel
Quadros de referência	Múltiplos quadros	Um quadro	Um quadro
Modo de predição bidirecional	Para frente / para trás Para frente / para frente Para trás / para trás	Para frente / para trás	Para frente / para trás
Predição ponderada	Sim	Não	Não
Filtro de deblocagem no laço de codificação	Sim	Não	Não
Tipos de quadros	I, P, B, SI, SP	I, P, B	I, P, B
Complexidade do codificador	Elevada	Média	Média
Compatibilidade com os padrões anteriores	Não	Sim	Sim

Adaptado de Kwon (2005).

Tabela 3.3: Ganhos na taxa de bits em relação a outros padrões.

Aplicação	MPEG-4 (%)	H.263 (%)	MPEG-2 (%)
Streaming	37,4	47,6	63,6
Vídeo Conferência	29,4	40,6	-
Entretenimento / Qualidade	-	-	45

Fonte: (WIEGAND, 2003).

A Figura 3.2, Figura 3.3 e Figura 3.4 apresentam exemplos de gráficos comparativos montados a partir dos diversos experimentos realizados no trabalho (WIEGAND, 2003). Nestas figuras são apresentados os gráficos do ganho na taxa de bits e os gráficos da economia em relação ao MPEG-2. A Figura 3.2 apresenta a comparação quando a aplicação de *streaming* é considerada. Na Figura 3.3 está apresentada a comparação

para aplicação de vídeo conferência. Finalmente, na Figura 3.4 está a comparação para aplicações de entretenimento/qualidade.

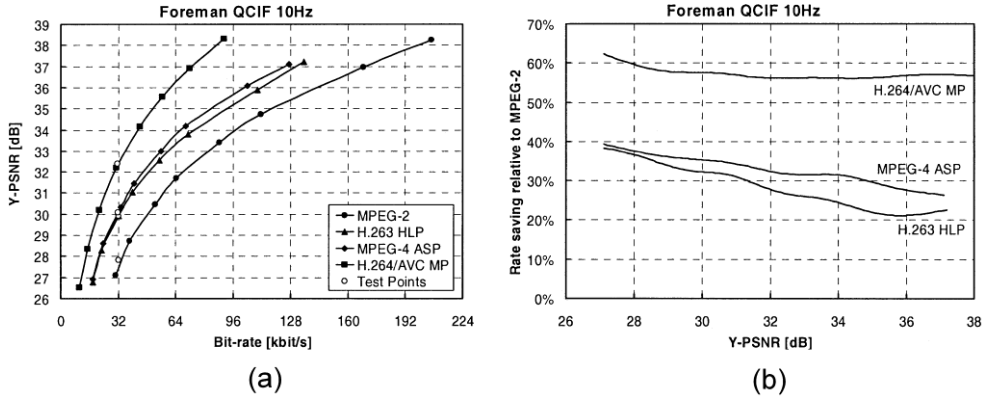

Figura 3.2: Comparação (a) dos ganhos na taxa de bits e (b) na economia na taxa de bits comparada o MPEG-2, para o vídeo "Foreman" considerando resolução de vídeo para *streaming*

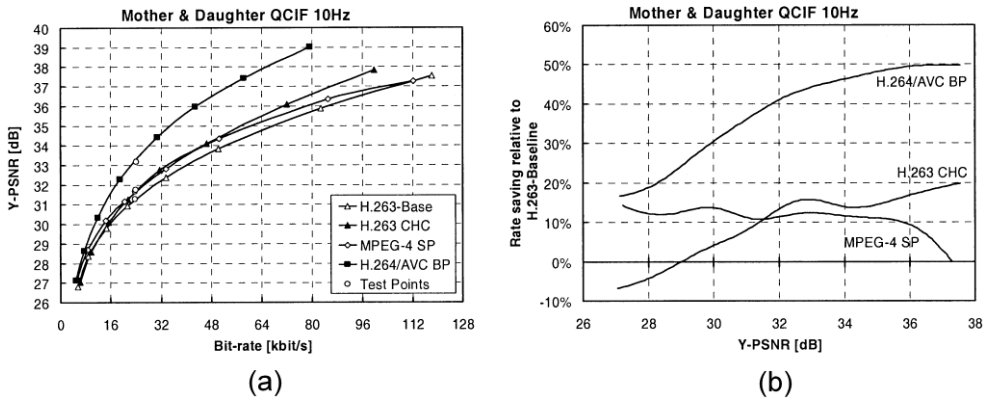

Figura 3.3: Comparação (a) dos ganhos na taxa de bits e (b) na economia na taxa de bits comparada o MPEG-2, para o vídeo "Mother & Daughter" considerando resolução de vídeo para vídeo conferência (QCIF).

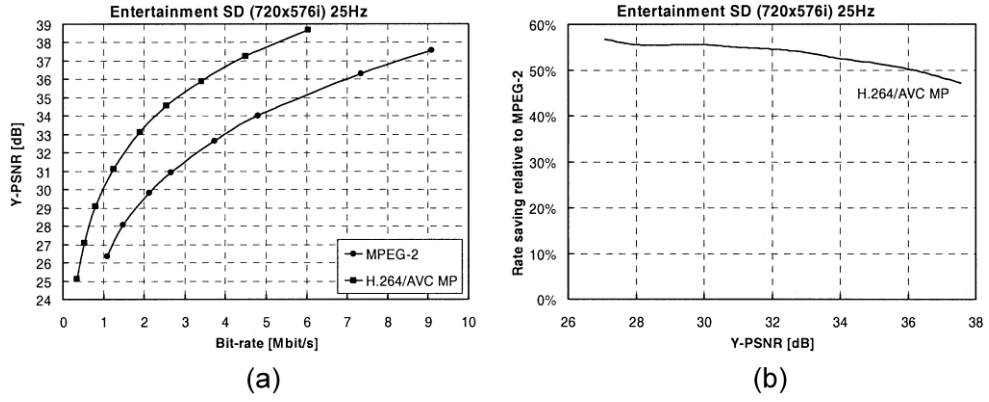

Figura 3.4: Comparação (a) dos ganhos na taxa de bits e (b) na economia na taxa de bits comparada o MPEG-2, para o vídeo "Entertainment" considerando resolução de vídeo padrão (SD).

Em todos os casos apresentados o padrão H.264 apresentou desempenho superior aos demais padrões. Esta capacidade do padrão H.264 de atingir as maiores eficiências de codificação para diversos tipos de aplicação é uma das mais importantes vantagens

deste padrão em relação aos demais padrões de compressão de vídeo que, normalmente, são direcionados para um tipo de aplicação específica (AGOSTINI, 2007).

Os ganhos na relação sinal ruído vêm acompanhados de um considerável aumento da complexidade computacional dos algoritmos utilizados no H.264 em relação aos algoritmos utilizados nos padrões anteriores. Segundo o trabalho (SUNNA, 2005) a relação entre o acréscimo na complexidade no decodificador e os ganhos na eficiência da codificação quando o padrão MPEG-2 é tomado como base, pode ser definido de acordo com os dados apresentados na Tabela 3.3. Na comparação da Tabela 3.3, são considerados os três perfis presentes na primeira versão do padrão H.264 e é considerada a complexidade apenas do decodificador H.264. Os dados apresentados na Tabela 3.3 são aproximados e apresentam apenas uma idéia dos ganhos de eficiência e do acréscimo em complexidade. As métricas de complexidade são mais difíceis de elaborar, especialmente considerando que a complexidade dos algoritmos e a complexidade de hardware são merecedoras de metodologias próprias.

Tabela 3.3: Relação entre o H.264 e o MPEG-2 quanto aos ganhos em eficiência de codificação e o acréscimo de complexidade.

Perfil	Ganho em Eficiência de Codificação (vezes)	Acréscimo de Complexidade (vezes)
Baseline	1,5	2,5
Extended	1,75	3,5
Main	2	4

Fonte: (SUNNA, 2005)

O acréscimo na complexidade computacional de um codificador H.264 é ainda mais elevado do que o acréscimo na complexidade do decodificador quando o padrão MPEG-2 é utilizado como comparação. Considerando o codificador, o H.264 é cerca de oito vezes mais complexo do que o MPEG-2.

3.2 O Codec H.264

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

O diagrama de blocos do codificador H.264 está apresentado na Figura 3.5. É possível perceber que a Figura 3.5 é muito parecida com aquela apresentada na seção 2.4 desta proposta. Aquele modelo apresentado na Figura 2.26 foi justamente construído para servir de base para discussão que será apresentada nesta seção do texto sobre o compressor H.264.

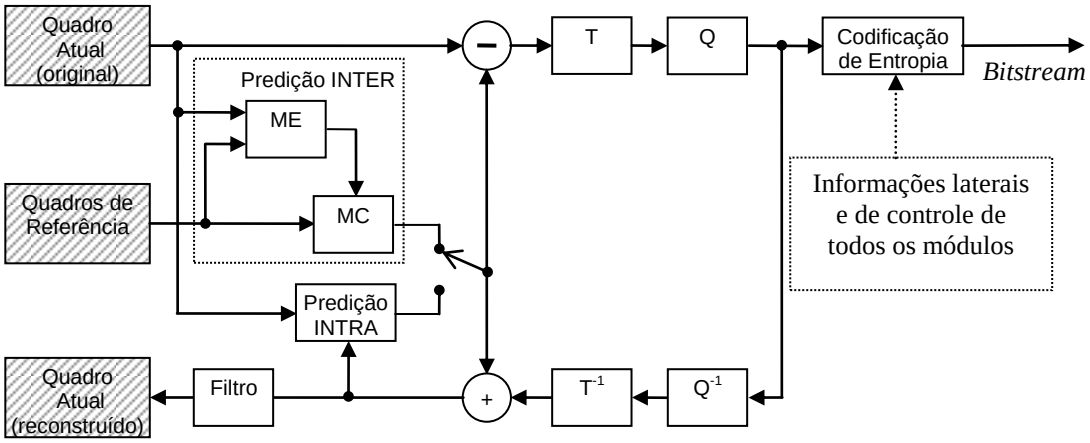

Figura 3.5: Diagrama em blocos de um codificador H.264.

O bloco de Predição Interquadro na Figura 3.5 é formado pela estimação de movimento (ME) e pela compensação de movimento (MC). Este bloco desempenha função similar a do bloco de codificação interquadro da Figura 2.26, mas para desempenhar esta função o padrão H.264 agrega operações de grande complexidade computacional a este bloco.

O bloco de Predição Intraquadro, os blocos das transformadas diretas e inversas (T e T^{-1}), os blocos da quantização direta e inversa (Q e Q^{-1}) e o bloco da Codificação de Entropia na Figura 3.5 possuem a mesma função, respectivamente, que os blocos de codificação intraquadro, das transformadas diretas e inversas (T e T^{-1}), os blocos da quantização direta e inversa e o bloco da codificação de entropia na Figura 3.5.

A única diferença mais significativa entre a Figura 3.5 e a Figura 2.26 está no bloco chamado de Filtro na Figura 3.5, que não está presente na Figura 2.26. Este bloco foi inserido no padrão H.264 e é requisito obrigatório. Nos demais padrões de compressão de vídeo o uso deste filtro não é obrigatório, mas um filtro similar é usado na prática por várias implementações seguindo outros padrões. Este filtro, também chamado de Filtro de Deblocagem ou filtro de *loop*, é usado para minimizar o efeito de bloco.

A Figura 3.6 apresenta o diagrama em blocos de um decodificador H.264. Também no decodificador H.264 está presente o filtro, que não está originalmente apresentado na Figura 2.27.

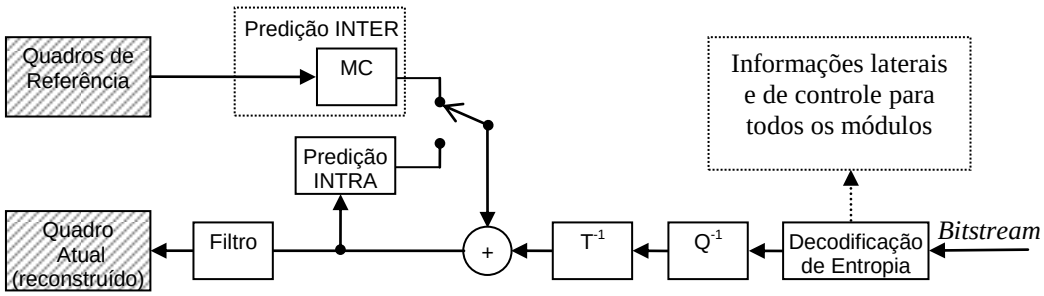

Figura 3.6: Diagrama em blocos de um decodificador H.264.

Cada um dos blocos do codificador e do decodificador será explicado, nas próximas seções, de acordo com sua funcionalidade dentro do padrão H.264.

As etapas de estimação e compensação de movimento (KUHN, 1999) do padrão H.264 (bloco ME na Figura 3.5 e bloco MC na Figura 3.5 e na Figura 3.6) é onde se encontram a maior parte das inovações e dos ganhos obtidos pelo H.264 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 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 das predições bi-preditiva, ponderada e direta para os *slices* do tipo B. Além disso, os vetores de movimento podem apontar para fora da borda do quadro. Por fim, a predição de vetores de movimento com base nos vetores vizinhos também é uma novidade (ITU, 2003; RICHARDSON, 2003; WIEGAND, 2003a).

A etapa de predição INTRA (Figura 3.5 e Figura 3.6) é outra inovação introduzida pelo padrão H.264, pois mesmo nos macroblocos do tipo I é realizada uma predição antes da aplicação da transformada. Esta predição leva em conta as amostras já codificadas do *slice* atual.

No que diz respeito às transformadas direta e inversa (blocos T e T^{-1} na Figura 3.5 e na Figura 3.6), o H.264 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 da transformada DCT direta e inversa, de modo a facilitar a sua implementação em hardware de ponto fixo e evitar erros de casamento entre o codificador e o decodificador (MALVAR, 2003). Além disso, uma segunda transformada é aplicada sobre os elementos DC resultantes da DCT para todos os blocos de crominância (Hadamard 2x2) e para os macroblocos em que é feita a predição INTRA 16x16 (Hadamard 4x4) (RICHARDSON, 2003).

A quantização no padrão H.264 (blocos Q e Q^{-1} na Figura 3.5 e na Figura 3.6) é uma quantização escalar (RICHARDSON, 2002). O fator de quantização é função de um parâmetro QP de entrada, que é usado no codificador para controlar a qualidade da compressão e o *bit-rate* de saída.

O padrão H.264 normatiza a utilização de um filtro redutor do efeito de bloco (bloco Filtro na Figura 3.5 e na Figura 3.6). Este filtro era opcional na maioria dos demais padrões de compressão de vídeo, mas passou a ser obrigatório no H.264. Uma inovação importante do filtro definido no padrão H.264 é que este filtro é adaptativo, conseguindo distinguir entre uma aresta real da imagem de um artefato gerado por um elevado passo de quantização.

Na codificação de entropia (Figura 3.5 e Figura 3.6) o H.264 também introduz ferramentas que aumentam bastante a eficiência de codificação deste bloco. Há, basicamente, dois tipos de codificação: a codificação de comprimento variável adaptativa ao contexto (CAVLC) e a codificação aritmética adaptativa ao 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.

Uma característica interessante do codificador H.264 é que apenas o decodificador é normatizado pelo padrão. Então vários graus de liberdade podem ser explorados na implementação do codificador. O codificador deve gerar um *bitstream* compatível com as exigências do padrão, mas a forma como este *bitstream* é gerado não é normatizada. Então várias opções de implementação podem ser realizadas no codificador que vão deste a inserção de algoritmos mais eficientes ou mais fáceis de serem implementados

em hardware, até a simples eliminação de algumas possibilidades de codificação previstas pelo padrão. É claro que estas modificações poderão gerar impactos na relação sinal/ruído, na taxa de compressão, na velocidade de processamento e/ou na utilização de recursos de hardware pelo codificador e estes impactos devem ser criteriosamente avaliados para tomar qualquer decisão nesta direção.

3.2.1 Os Blocos da Estimação e Compensação de Movimento (ME/MC)

A predição interquadro no codificador H.264, é formada pelos blocos da Estimação de Movimento (ME), usada apenas no codificador e da Compensação de Movimento (MC), usada tanto no codificador (para reconstruir os quadros que serão usados como referência) quanto no decodificador. Esta seção focará o bloco da Estimação de Movimento, mas ambos os blocos estão estreitamente relacionados.

O bloco ME é o que apresenta a maior complexidade computacional dentre todos os blocos de um codificador H.264 (PURI, 2004). Esta complexidade advém do conjunto de ferramentas empregados nesse bloco com o objetivo de atingir elevadas taxas de compressão. Reside nos blocos da ME e MC as principais fontes de ganhos do H.264 em relação aos demais padrões de compressão de vídeo (WIEGAND, 2003, RICHARDSON, 2003). Este bloco usa ferramentas de codificação capazes de localizar qual bloco mais se assemelha ao macrobloco atual nos quadros de referência. 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.

No H.265, a principal inovação está na possibilidade de utilização de tamanhos de blocos variáveis para realizar a estimação de movimento. Ao invés de usar um macrobloco inteiro na estimação de movimento, o padrão 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 3.7 (WIEGAND, 2003, RICHARDSON, 2003).

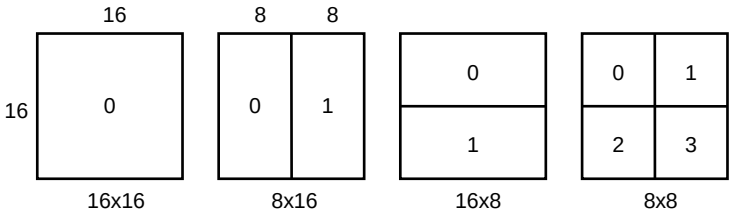

Figura 3.7: Divisão do macrobloco em partições de macroblocos. (RICHARDSON, 2003)

Para as partições 8x8 há ainda a possibilidade da divisão em mais quatro formas, chamadas sub-macroblocos, que podem ter o tamanho 8x8, 4x8, 8x4 ou 4x4, como está apresentado na Figura 3.8.

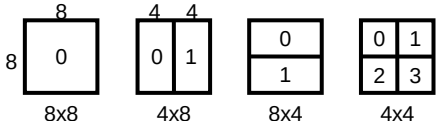

Figura 3.8: Divisão de uma partição de macrobloco em partições de sub-macroblocos. (RICHARDSON, 2003)

Nos formatos sub-amostrados, os blocos de croma terão tamanhos correspondentes ao tamanho dos blocos de luma que pertencem os pixels.

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

O H.264 também 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, os melhores vetores de movimento podem ser fracionários. Por isso, o padrão H.264 prevê a utilização de vetores de movimento com valores fracionários de ½ pixel e de ¼ de pixel. Na Figura 3.9 é apresentado um exemplo da precisão do vetor de movimento com valores fracionários.

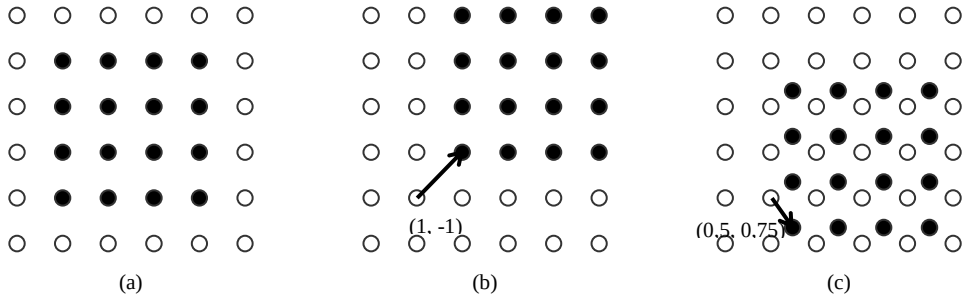

Figura 3.9: Estimação de movimento com precisão de frações de pixel (AGOSTINI, 2007).

Na Figura 3.9, pontos brancos representam posições de luma do quadro de referência e pontos pretos representam pontos do 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 com vetor (0;0) na Figura 3.9(a) e (1;-1) na Figura 3.9(b). Na Figura 3.9(c), um vetor fracionário (0,5, 0,75) é selecionado.

No decodificador, a compensação de movimento usando ¼ de pixel é obrigatória. Quando a compensação de movimento com vetores com resolução de ¼ de pixel é usada em uma imagem no formato 4:2:0, os elementos de crominância possuem resolução de 1/8de pixel. Detalhes sobre a formação dos valores das amostras intermediárias podem ser encontrados em Richardson (2003) ou em Agostini (2007).

Outra característica importante do padrão H.264 é o uso de múltiplos quadros de referência. No H.264 os quadros de referência não precisam ser somente os quadros I ou P imediatamente anteriores ou posteriores ao quadro atual. Há a opção de se usar como referência múltiplos quadros para frente ou para trás do quadro atual (RICHARDSON, 2003), como no exemplo apresentado na Figura 3.10.

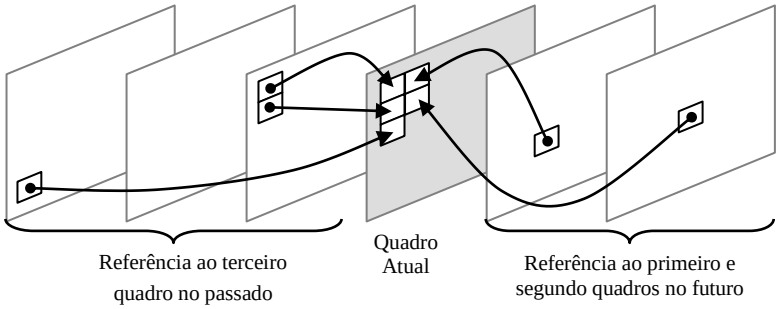

Figura 3.10: Uso de múltiplos quadros de referência (AGOSTINI, 2007).

Para suportar o uso de múltiplos quadros de referência e quadros B, o padrão H.264 permite que o codificador escolha uma ordem dos quadros para codificação completamente diferente da ordem dos quadros para apresentação do vídeo (KWON, 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. O decodificador terá que fazer o reordenamento antes da exibição.

Nos quadros B é possível realizar a estimação de movimento bi-preditiva. Ela 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.

Um outro modo de predição usado em quadros B é a predição direta. Nela 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.

No H.264 também é possível utilizar vetores de movimento que apontam para fora dos limites dos quadros. Neste caso é realizada uma extrapolação dos quadros nas bordas. Além disso, quadros do tipo B podem ser usados como referência na predição, diferente do que acontece no padrão MPEG-2 (ITU, 1994).

Finalmente os vetores de movimento são transmitidos de forma diferencial, baseados em sua vizinhança.

Devido a necessidade da reconstrução dos quadros pelo codificador, após a quantização para obter os quadros de referência, 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. Como a ME foi abordada na seção anterior, todas as características da MC citadas acima já foram explicadas e, deste modo, não serão explicadas novamente nesta seção.

3.2.2 O Bloco de Predição Intra

Nos macroblocos tipo I a predição espacial é utilizada. Para realizar esta tarefa, o bloco de predição intra do padrão H.264 usa a informação de vizinhos já codificados, como apresentado na Figura 3.11. O bloco de predição intra está presente nos codificadores e nos decodificadores H.264 (Figura 3.5 e Figura 3.6).

A predição intra é uma inovação no padrão H.264. Como já foi mencionado na seção 3.1.2, existe um modo adicional de codificação para macroblocos do tipo I, chamado I_PCM (RICHARDSON, 2003). Neste caso, as amostras da imagem são transmitidas diretamente, sem predição, transformada e quantização.

No processo de predição Intra, as amostras marcadas em letras minúsculas (a-p) na Figura 3.11 mostram um bloco 4x4 de luma a ser codificado pela predição intra. As amostras acima e a esquerda, marcadas em letras maiúsculas (A-M) na Figura 3.11 foram codificadas e reconstruídas antes deste bloco ser processado. Estas amostras serão utilizadas como referências para a predição intra.

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 3.11: Identificação das amostras para a predição intra (AGOSTINI, 2007).

O H.264 prevê nove modos de predição Intra com tamanho de 4x4 e quatro com tamanho 16x16. O perfil high suporta um conjunto adicional de mais nove modos para o tamanho 8x8. A Figura 3.12 apresenta os nove tipos diferentes de codificação no modo intra para blocos 4x4 de luminância; para tamanhos de blocos de 8x8 amostras, os modos são os mesmos que os para o tamanho 4x4 (a vizinhança e as funções para cálculo dos valores preditos é diferente, entretanto).

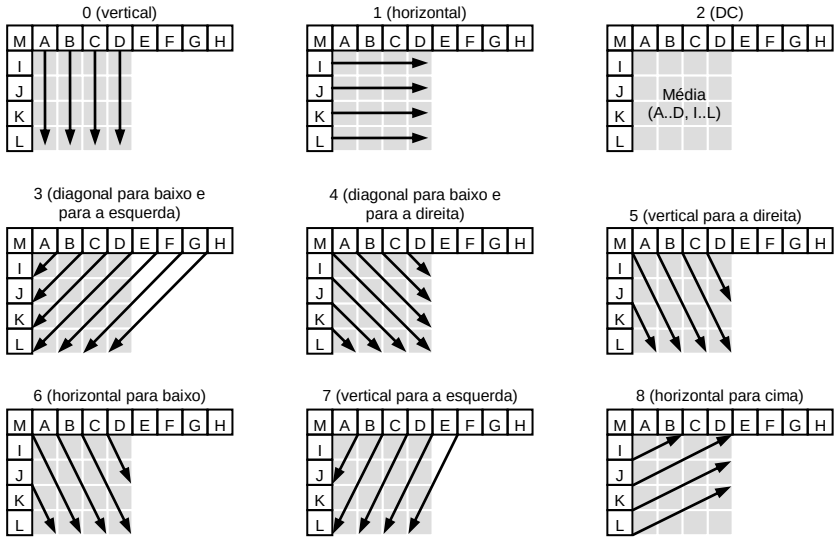

Figura 3.12: Nove modos da predição intra para blocos de luma 4x4 (AGOSTINI, 2007)

Nos modos para os tamanhos de boco iguais a 4x4 e 8x8, 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 das setas na Figura 3.12.

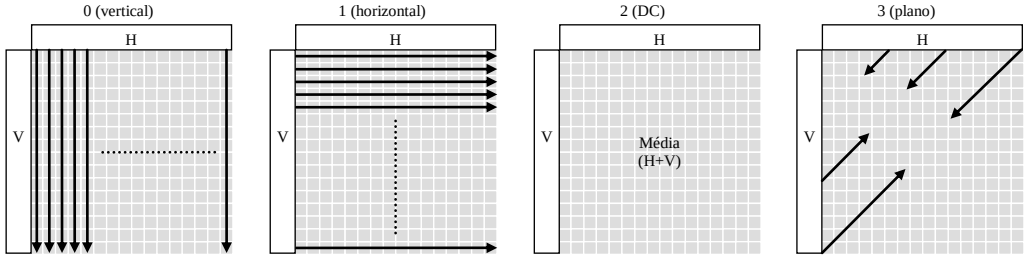

Figura 3.13: Quatro modos da predição intra para blocos de luma 16x16 (AGOSTINI, 2007)

3.2.3 O Bloco das Transformadas Diretas (T) e Transformadas Inversas (T^{-1})

Toda informação residual produzida pelos blocos Intra e ME são transformadas pelo bloco T, apresentado na Figura 3.5. Este bloco é responsável pelas transformadas diretas e está presente apenas nos codificadores H.264. As entradas para o bloco T são blocos 4x4 (ou 8x8 nos perfis *high*) amostras 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.

O H.264 introduziu uma inovação nas transforamdas que as transformadas são representadas por números inteiros exatos, sendo perfeitamente reversíveis.

Durante o desenvolvimento do padrão, foi identificado que os coeficientes poderiam ser ainda simplificados de forma a serem compostos apenas pelos valores (1, -1, 2, -2) se alguns fatores de escala fossem incorporados no cálculo da quantização (Q), que é o passo que segue a aplicação das transformadas diretas na compressão H.264.

Caso de amostras com informação de crominância ou com informação de luminância cuja predição tenha sido do tipo INTRA 16x16, o cálculo da FDCT 2-D é aplicado sobre todos os dados de entrada, mas uma transformada Hadamard 4x4 sob os coeficientes DC dos blocos de luminância, Hadamard 2x2 sob os coeficientes DC dos blocos 4 x 4 de crominância. A transformada Hadamard explora uma correlação residual que ainda permaneça sobre os coeficientes da FDCT 2-D (RICHARDSON, 2003).

As matrizes da transformada Hadamard possuem apenas valores (1, -1). Deste modo, apenas somas e subtrações são necessárias. A divisão por dois realizada em todos os resultados da saída é um simples deslocamento de uma posição binária para a direita.

Com a inclusão dos perfis high, uma transformada 8x8 foi incluída no padrão. Esta transformada é um pouco mais complexa que a 4x4, possuindo 64 coeficientes com valores (-12, -8, -6, -4, -3, 3, 4, 6, 8, 12), porém mais simples que uma IDCT de mesmo tamanho (SULLIVAN, 2004).

O bloco T, sendo formado pelas três transformadas que estão apresentadas acima, deve sincronizar a operação entre elas, de modo a gerar o fluxo correto de dados na sua saída. A ordem de processamento é apresentada na Figura 3.14. O resultado da transformada Hadamard é enviado para a codificação de entropia antes dos coeficientes tranformados pela DCT. Além disso, como os coeficientes DC transformados dos blocos 4x4 ou 8x8 são codificados pela Hadamard, apenas os 15 coeficientes AC de

cada bloco 4x4 ou 255 de cada bloco 8x8 são enviados ao codificador de entropia.

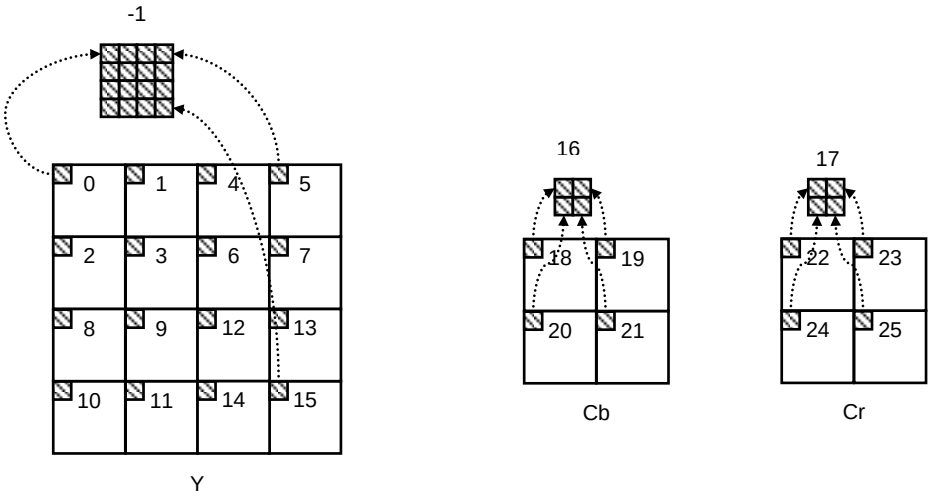

Figura 3.14: Ordem de processamento de amostras pelo bloco T.

O bloco das Transformadas Inversas (T^{-1})do padrão H.264, apresentado na Figura 3.5 e na Figura 3.6, por fazer as operações inversas ao bloco T, possui muitas características idênticas a este bloco. O bloco T^{-1} está presente nos codificadores e nos decodificadores H.264. O padrão H.264 definiu que a operação do bloco T^{-1} 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 (Q^{-1}). Então, os resultados da quantização inversa são entregues novamente para o bloco T^{-1}, que realiza a última etapa de seus cálculos, aplicando a DCT 2-D inversa (IDCT).

3.2.4 O Bloco da Quantização Direta (Q) e Quantização Inversa (Q^{-1})

A quantização é o lugar onde ocorrem as perdas no processo de codificação. No padrão H.264, o bloco Q na Figura 3.14 realiza a quantização direta e a correção da escala do cálculo das transformadas (mencionado na seção anterior). Os cálculos realizados pelo bloco Q dependem do modo de predição utlizado e se o elemento é de crominância ou luminância. As operações realizadas são uma multiplicação da entrada por uma constante, a soma do resultado por uma outra constante e um deslocamento no resultado da soma controlado por uma terceira constante. Estas constantes são associadas ao parâmetro de quantização (QP) que é uma entrada externa que informa ao bloco 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 3.4. 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 3.4: 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

Fonte: (ITU, 2005).

A Quantização Inversa (bloco Q^{-1}, apresentado na Figura 3.5) realiza a reconstrução com perdas dos coeficientes de transformada quantizados pelo bloco Q e na Figura 3.6, está presente nos codificadores e nos decodificadores H.264. Alguns autores defendem que não é correto chamar este bloco de quantização inversa, uma vez que a quantização direta é uma operação irreversível por ser uma divisão inteira (BHASKARAN, 1997). Deste modo, estes autores utilizam o termo "*rescaling*" ao invés do termo quantização inversa. Neste trabalho, o termo quantização inversa é usado, pois é o adotado pelo padrão H.264 (ITU, 2003).

3.2.5 O Bloco do Filtro de Deblocagem

O H.264 trouxe uma inovação em relação a padrões anteriores de codificador de vídeo por incluir um Filtro de Deblocagem no laço de reconstrução da imagem (necessário tanto no codificador quanto no decodificador). O bloco Filtro nas Figura 3.5 e na Figura 3.6 são relativos ao Filtro de Deblocagem que é normatizado pelo padrão H.264 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 que podem ter ocorrido em decorrência de erros da predição intra ou da predição inter, especialmente quando o parâmetro de quantização (QP) utilizado para codificar um dado bloco for elevado.

No padrão H.264, este Filtro de Deblocagem está inserido no laço da predição interquadro. Desta forma, os quadros utilizados como quadros de referência pelo bloco da estimação de movimento devem ter sido filtrados por este filtro. Este filtro é adaptativo, e distingue uma aresta real da imagem, que não deve ser filtrada, e um artefato gerado por um elevado passo de quantização, que deve ser filtrado. A adaptatividade é atingida a partir de um conjunto de decisões que são usadas para definir a força do filtro (BS - *Boundary Strength*) que define o tipo de filtro selecionado e os coeficientes empregados.

O filtro redutor de blocagem é mandatório no H.264 e o processo de adaptação se dá através da seleção de diferentes filtros (com até 7 taps) dependendo de uma força do filtro (*boundary strength* – BS) que deve ser calculada para cada elemento da borda a ser filtrada. O grande desafio do Filtro de Deblocagem está justamente na seleção do BS que depende de uma série de cálculos envolvendo parâmetros que incluem o QP, o tipo de codificação usada (inter ou intra) o tipo de macrobloco (I, P ou B), o tipo de amostra (luma ou croma), os vetores de movimento usados e também os valores dos pixels da borda envolvida. Isso aliado ao fato de que alguns parâmetros são calculados a partir de tabelas de consulta (*lookup tables*) que devem ser implementas como memória. Além disso, o Filtro de Deblocagem precisa realizar filtragem bidimensional (bordas verticais e horizontais), e reordenar e armazenar temporariamente blocos para serem processados na ordem correta, o que torna um grande consumidor de memória. Muitas implementações do Filtro de Deblocagem consideram que o armazenamento temporário dos blocos é feito na memória principal e não em uma memória local específica para ele.

A filtragem é aplicada para bordas verticais e horizontais dos blocos 4x4 de um macrobloco, de acordo com a ordem abaixo. Para os passos apresentados abaixo, considere a Figura 3.15 como referência.

1) Filtrar as quatro bordas verticais do componente de luminância (**a**, **b**, **c** e **d**);

2) Filtrar as quatro bordas horizontais do componente de luminância (**e**, **f**, **g** e **h**);
3) Filtrar as duas bordas verticais de cada componente de crominância (**i** e **j**);
4) Filtrar as duas bordas horizontais de cada componente de crominância (**k** e **l**).

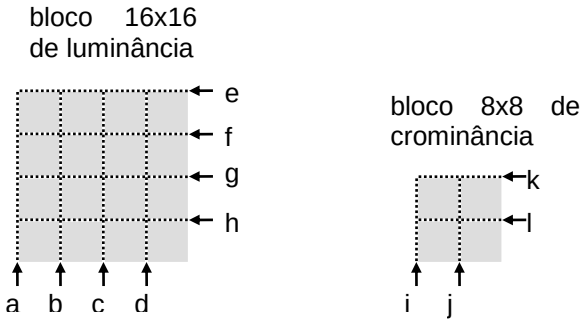

Figura 3.15: Ordem de filtragem de bordas em um macrobloco.

Cada operação de filtragem afeta até três amostras de cada lado da borda, como está apresentado na Figura 3.16. 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 da Figura 3.16.

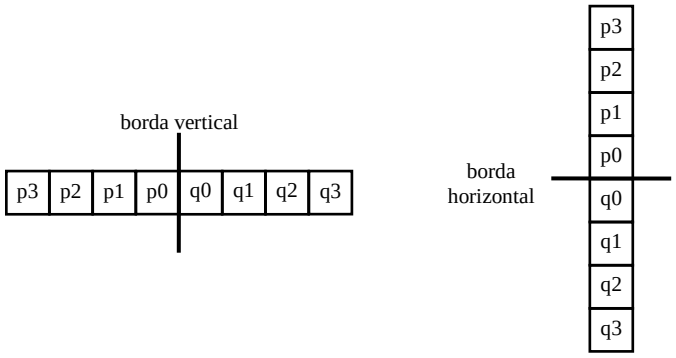

Figura 3.16: Amostras adjacentes para bordas verticais e horizontais.

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.

Existem cinco diferentes forças de filtragem, definidas pelo parâmetro BS (*boundary strength*). O parâmetro BS pode variar de zero a quatro e, portanto, existem cinco diferentes forças de filtragem, onde BS=4 define uma filtragem com força máxima e BS=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 as amostras de LOPs classificadas como Q e P:

- BS=4 é utilizado se os blocos Q e/ou P foram codificados no modo intra e se a borda é uma borda de macrobloco (Q e P pertencem a macroblocos distintos);
- BS=3 é utilizado se os blocos Q e/ou P foram codificados no modo intra e se a borda não é uma borda de macrobloco;
- BS=2 é utilizado se os blocos Q e/ou P não foram codificados no modo intra e se P e/ou Q possuem coeficientes codificados;

- BS=1 é utilizado se os blocos Q e/ou P não foram codificados no modo intra; se P e Q não possuem coeficientes codificados; se Q e P 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;
- BS=0 é utilizado se nenhuma das situações anteriores for verdadeira.

Mas a decisão de realizar a filtragem não depende apenas do parâmetro BS. Considerando um grupo de amostras (p2, p1, p0, q0, q1, q2), a filtragem acontece se BS>0 e se a condição apresentada em (3.1) for verdadeira.

$$|p0-q0|<\alpha \quad \text{e} \quad |p1-p0|<\beta \quad \text{e} \quad |q1-q0|\leq\beta \tag{3.1}$$

Em (3.1), α e β são definidos pelo padrão e crescem de acordo com a média do parâmetro de quantização (QP) dos blocos Q e P. Se o QP tiver 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 ativo mais frequentemente e, assim, mais amostras da borda são filtradas em média, gerando mais suavização nas bordas dos blocos de 4x4 amostras.

3.2.6 O Bloco de Codificação de Entropia

Após finalizado de extração das redundâncias, são produzidos elementos sintáticos que são mensagens destinadas que devem ser enviadas ao decodificador de forma que este possa recompor as imagens e a sequência de vídeo. Cada um destes elementos sintáticos é composto por um tipo, que identifica sua origem e um valor. O tipo pode ser um vetor de movimento, um modo de predição intra ou até mesmo um resíduo de transformada. O valor é o que efetivamente deve ser codificado. A colocação dos elementos sintáticos no fluxo de dados (*bitstream*) é feita de forma hierárquica seguindo os procedimentos estabelecidos na norma.

A simples colocação dos elementos sintáticos no *bitstream* não produziria a compressão desejada para um codificador de video cujo objetivo é a compressão. Todas as outras ferramentas de codificação apresentadas anteriormente, que geram os elementos sintáticos, têm como principal função tornar evidente a redundância do vídeo para que, em um processo de codificação de entropia esta redundância possa ser removida. A Figura 3.17 ilustra o trabalho das ferramentas de codificação na tentativa de tornar a redundância evidente.

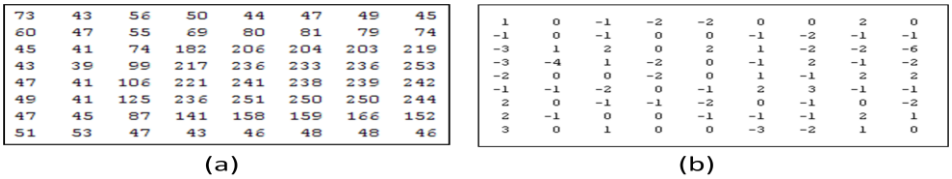

Figura 3.17: Matriz de elementos da imagem (a) amostras (b) após processamento pelo bloco ME.

Observe que na Figura 3.17(a) os valores possuem grande variação enquanto que em (b) os valores se concentram em torno do zero. Isto torna a probabilidade de ocorrência

de alguns valores (símbolos) muito maior que a probabilidade de ocorrência dos demais símbolos, o que permite a melhor exploração de técnicas de codificação de entropia.

A codificação de entropia que está presente nos codificadores e decodificadores (Figura 3.5 e Figura 3.6) do H.264 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 no H.264, tanto no CAVLC quanto no CABAC é o uso de codificação adaptativa baseada em contextos. No CAVLC que é usado apenas para codificar a informação de resíduos de transformada quantizados, o contexto são os valores recentemente codificados e irão influenciar na escolha de tabelas de codificação apropriadas. No CABAC, a codificação é dividida em três etapas: binarização, modelagem e contextos e codificação aritmética. No CABAC, 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 da transformada, após a quantização, são reordenados em forma de ziguezague, como está apresentado na Figura 3.18. Se a predição intra 16x16 foi utilizada, então os coeficientes DC de cada bloco 4x4 de luminância são extraídos do bloco 16x16, formando uma matriz 4x4 de coeficientes DC. Esta matriz é, então, reordenada do mesmo modo que está apresentado na Figura 3.18. Os 15 coeficientes AC que restam em cada bloco 4x4 de luminância são, então, reorganizados do mesmo modo que está apresentado na Figura 3.18, mas iniciando da segunda posição. De forma similar, os coeficientes DC dos blocos 4x4 dos dois componentes de crominância são extraídos das matrizes 8x8 de amostras de cor, formando duas matrizes 2x2 de elementos DC, que também são organizadas em ziguezague. Os 15 coeficientes AC restante nos blocos 4x4 de crominância são, então, reorganizados seguindo a ordem da Figura 3.18, mas iniciando da segunda posição.

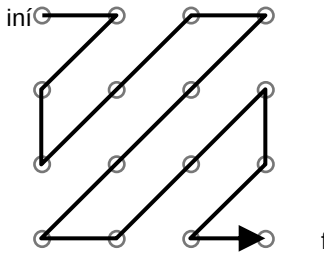

Figura 3.18: Ordem ziguezague de leitura dos blocos 4x4
para a codificação de entropia.

As próximas subseções do texto irão apresentar resumidamente, a funcionalidade dos códigos de Exp-Golomb, a codificação de comprimento variável adaptativa ao contexto e a codificação aritmética binária adaptativa ao contexto, que são ferramentas de codificação utilizadas no codificador de entropia do padrão H.264, como já foi mencionado acima.

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 (3.2):

$$Código = [M\ zeros]\,1\,[INFO] \tag{3.2}$$

Em (3.2), *M* representa o número de zeros que antecedem o primeiro valor 1 no código. O valor de *M* é dado por (3.3)

$$M = \lfloor \log_2 \cdot (num\_cod + 1) \rfloor \tag{3.3}$$

A variável *num_cod* em (3.3) indica qual é o número do código. Códigos com maior probabilidade de ocorrência possuem um número de código mais baixo.

O campo *INFO* em (3.2) possui *M* bits e indica qual é a informação codificada. O valor do campo *INFO* é dado por (3.4).

$$INFO = num\_cod + 1 - 2^M \tag{3.4}$$

O primeiro código de Exp-Golomb não possui os campos *M* **zeros** e *INFO*, sendo sempre representado apenas pelo valor ‘1’. A Tabela 3.5 apresenta os seis primeiros códigos de Exp-Golomb.

Tabela 3.5: Seis primeiros códigos de Exp-Golomb.

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

No padrão H.264, 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.

Codificador de Comprimento Variável Adaptativa ao Contexto – CAVLC

O codificador 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. O CAVLC foi desenvolvida para explorar as características dos blocos quantizados, que são (RICHARDSON, 2003):

- O resultado da quantização produz, tipicamente, matrizes esparsas. Então o CAVLC utiliza RLE (SALOMON, 2000) para representar compactamente as sequências de zeros.
- Os coeficientes não zeros de alta frequência, depois da leitura em ziguezague, são frequentemente sequências de ±1. Então o CAVLC sinaliza o número dos coeficientes ±1 de alta frequência 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 coeficiente não zero nos blocos vizinhos.
- A magnitude dos coeficientes não zero tende a ser maior para os primeiros coeficientes lidos em ziguezague (baixas frequências) e menores para as frequências 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.

O CAVLC está disponível para todos os perfis do padrão H.264 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. Por esta razão, o CAVLC é o único método de codificação de entropia disponível nos perfis *baseline* e *extended*, que é voltado para aplicações portáteis que exigem decodificadores de menor complexidade.

Codificação Aritmética Binária Adaptativa ao Contexto (CABAC)

A codificação aritmética binária adaptativa ao contexto (CABAC) é uma ferramenta de codificação disponível em todos os perfis do H.264, exceto nos perfis *baseline* e *extended*. O CABAC atinge taxas de compressão mais elevadas que o CAVLC 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 a 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 são usados 398 contextos diferentes (ITU, 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:** Para cada *bin* que envolva um contexto, este precisa ser atualizado 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 os 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.

Tabela 3.6: 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

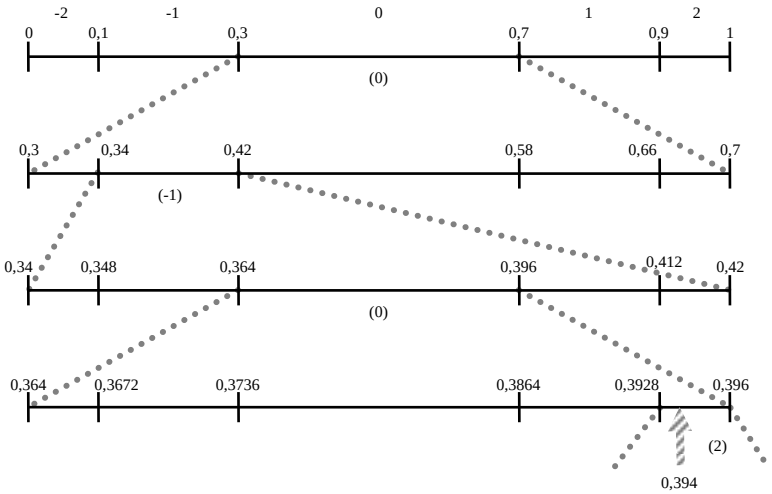

Figura 3.19: Exemplo de codificação aritmética

A Figura 3.19 apresenta um exemplo de codificação aritmética, considerando a sequência de símbolos (0, -1, 0, 2). A tabela de probabilidades para este exemplo está apresentada na Tabela 3.6. Na Figura 3.19, 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 3.19, 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). Na prática, qualquer valor dentro do intervalo final pode ser escolhido para ser transmitido. Com este valor e com o conhecimento dos dados apresentados na Tabela 3.6, o decodificador será capaz de reconstruir os símbolos.

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 contexto da codificação. 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 os intervalos, o torna a codificação aritmética 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 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.

3.3 Escalabilidade no H.264

Após a finalização dos trabalhos de FRext (*Fidelity Range Extensions*) que incluiu os perfis *high* ao padrão H.264 em 2005 (ITU, 2005), o JVT seguiu os trabalhos para dar suporte a escalabilidade no padrão H.264, o que levou ao desenvolvimento da extensão escalável do padrão em 2007 (ITU, 2007), que recebeu o nome de SVC (*Scalable Video Coder*).

O SVC é uma extensão do AVC (H.264), logo todas as ferramentas de codificação usadas no AVC também são utilizadas no H.264. O codificador utiliza técnicas baseadas em camadas para tornar possível a escalabilidade espacial. Durante o processo de codificação, o encoder utiliza um processo de sub-amostragem (*down-sampling)* através de um filtro que gera versões de resolução mais baixa do sinal de vídeo para cada uma das camadas espaciais. O processo de codificação suporta duas formas sub-amostragem: a versão diádica, onde a relação de tamanho é igual a 2 para a camada de menor resolução e a versão estendida, que permite que a camada base seja uma versão recortada com uma taxa de sub-amostragem não inteira. O padrão H.264 também permite a exploração da escalabilidade temporal, permitindo que camadas diferentes possuam taxas de quadros diferentes, e também a escalabilidade em qualidade (SNR), permitindo camadas que apresentem apenas incrementos de qualidade sobre uma camada base.

A informação de movimento de uma camada mais baixa pode ser usada para a predição de uma camada mais alta, podendo esta informação ser utilizada ou não a cada macrobloco. A informação de codificação de blocos tipo intraquadro pode ser empregada em camadas superiores assim como a informação de resíduos das camadas mais baixas pode ser empregada para a construção da camada atual.

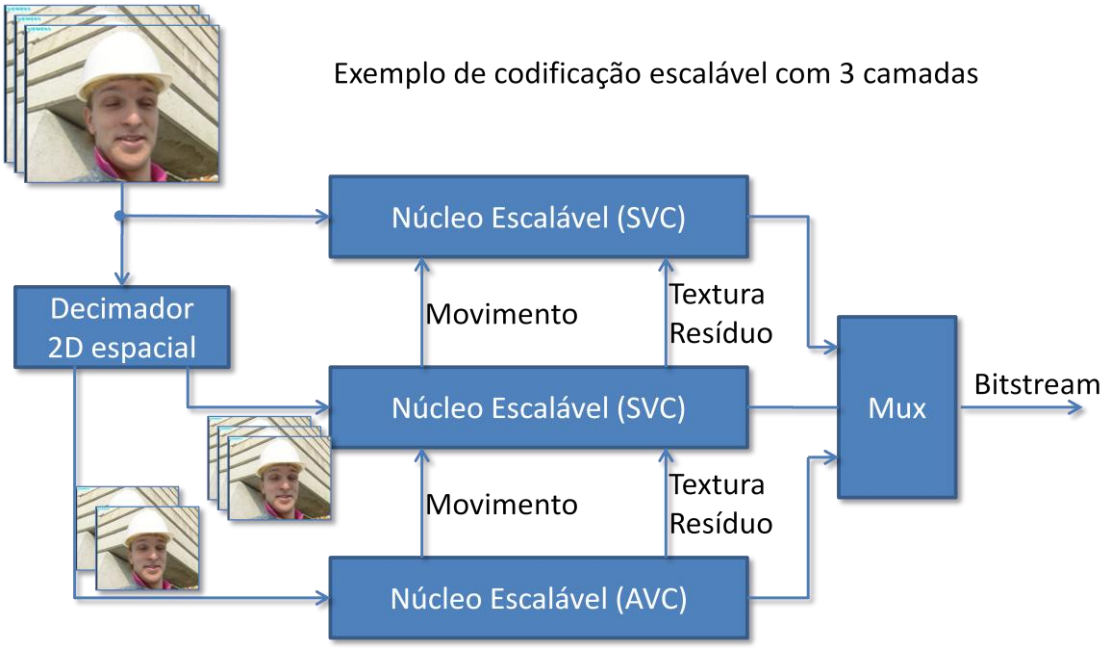

Figura 3.20: Estrutura de um codificador escalável (SVC) com 3 camadas.

A informação de resíduos resultante da predição Inter ou Intra passa pelo processo de codificação normal do H.264. Uma camada base de resíduos pode ser codificada usando um *bitstream* que é compatível com o H.264 não escalável se nenhuma predição entre camadas for empregada, permitindo uma qualidade mínima no momento da decodificação, por um decodificador não escalável. Camadas de enriquecimento podem ser codificadas adicionalmente para prover maior qualidade.

A escalabilidade temporal é alcançada utilizando-se quadros B hierárquicos, que é um conceito já presente no H.264 não escalável. Alternativamente, um filtro temporal com compensação de movimento (MCTF – *Motion Compensated Temporal Filtering*)

pode ser empregado, sendo este uma ferramenta que não faz parte do padrão H.264 escalável. O MCTF pode ser visto como um pré-processamento no processo de codificação, que pode prover melhor qualidade de codificação para algumas sequências. A Figura 3.20 apresenta o diagrama em blocos de um codificador escalável com uma camada base e duas de enriquecimento.

3.3.1 Dimensões da escalabilidade

Há duas formas de introduzir escalabilidade em um codec: pelo uso de uma técnica que é intrínsicamente escalável (como a codificação através de planos de bits aritméticos); ou pelo uso de técnicas baseadas em camadas. Na implementação em software do H.264 escalável (JSVM – *Joint Scalable Vídeo Model*), uma combinação de ambas as técnicas é usada para permitir que um codec escalável capaz de suportar a escalabilidade espaço-temporal em todos os aspectos. A escalabilidade temporal é suportada através de quadros B hierárquicos, enquanto que a escalabilidade espacial é suportada usando camadas.

A Figura 3.21, extraída de (ISSO, 2007) ilustra a estrutura hierárquica de codificação de quadros usada para suportar escalabilidade temporal no H.264, incluindo as relações possíveis de dependências de quadros. Nesta figura, uma estrutura hierárquica diádica é utilizada, com quatro níveis temporais. Cada grupo de quadros (*Group Of Pictures* - GOP) é formado por 8 quadros.

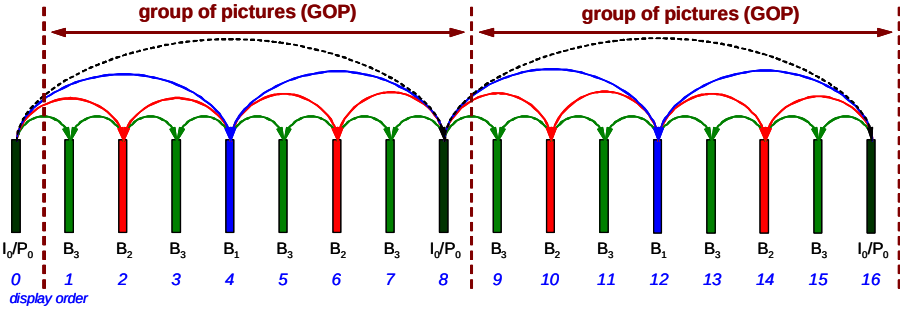

Figura 3.21: Codificação hierárquica diádica com GOP de 8 e 4 níveis temporais.

Para escalabilidade em qualidade (redução progressiva da relação sinal-ruído - SNR), uma técnica baseada em múltiplas resoluções é usada para escalabilidade em grão-grosso (CGS – *Coarse Grain Scalability*) enquanto que uma técnica baseada em codificação aritmética de subplanos de bits é usada para escalabilidade em grão fino (FGS – *Fine Grain Scalability*).

A escalabilidade de grão fino é alcançada através de sucessivos refinamentos na codificação dos coeficientes de transformada, começando com uma qualidade mínima pelo modo intra/resíduo compatível com o H.264. Isto é feito pela redução progressiva do passo de quantização e aplicando a codificação de entropia modificada para permitir a codificação através de subplanos de bits em um processo chamado refinamento progressivo.

A escalabilidade de grão grosso é alcançada através dos mesmos mecanismos que permitem a escalabilidade espacial, porém, na escalabilidade de grão grosso, a informação de textura e movimento é escalonada quando há um aumento de resolução entre as camadas. Os mecanismos são semelhantes aos empregados em padrões anteriores como o MPEG-4 Visual (ISO, 1999), mas muitos mecanismos adicionais para a predição intra são incorporados.

As dimensões temporal, de relação sinal-ruído (SNR) e espacial da escalabilidade podem ser combinadas, gerando uma ampla gama de possibilidades para a codificação escalável. Entretanto, o conceito de camadas só pode ser aplicado se apenas um tipo de escalabilidade for utilizado, pois não pode haver conceito de camadas quando um esquema tridimensional de escalabilidade (temporal, espacial e SNR) é utilizado. Por exemplo, é possível construir um fluxo de bits de vídeo CIF@30fps, de onde podem ser extraídos tanto um fluxo QCIF@30fps quanto um fluxo CIF@15fps. Neste caso, o conceito de camadas não pode ser aplicado.

O H.264/SVC também suporta um modo chamado Escalabilidade Espacial Estendida, que permite uma relação geométrica mais genérica entre camadas espaciais sucessivas. A escalabilidade espacial estendida inclui duas ferramentas: Recortes e *up-sampling* genérico. Essas ferramentas permitem montar um esquema de camadas como o ilustrado na Figura 3.22, extraída de (ISO, 2007).

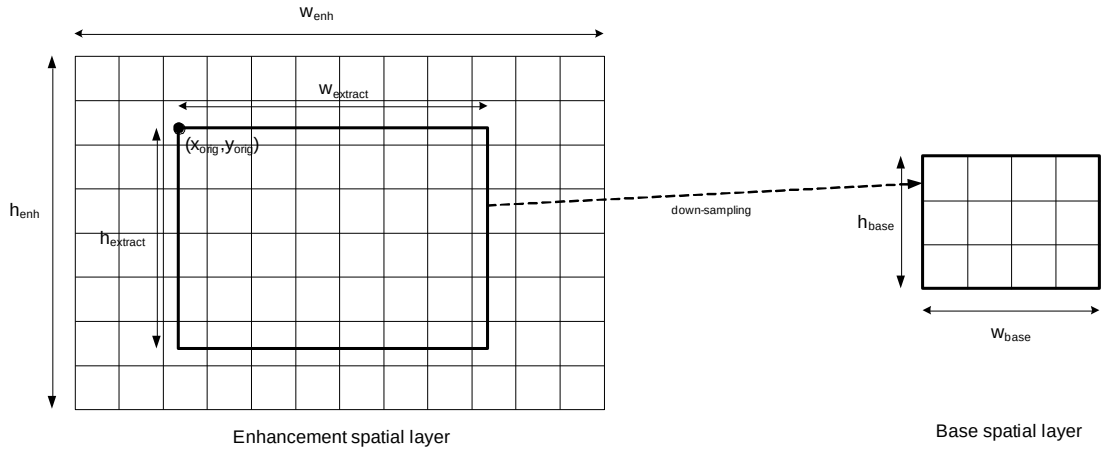

Figura 3.22: Relação genérica entre uma camada base e uma de enriquecimento.

Quando uma janela de recorte alinhada em uma grade de macroblocos com um fator de *up-sampling* de 1 ou 2, o processo de *up-sampling* da informação de movimento e de textura pode ser o processo tradicional (não genérico). Caso contrário, o método genérico tem que ser utilizado, pois um macrobloco da camada base pode fazer parte de zero ou mais macroblocos da camada de enriquecimento e os pixels podem não estar alinhados com a grade, não permitindo o processo de *up-sampling* tradicional.

3.4 Múltiplas visões no H.264

O suporte a Múltiplas Visões (multivisão) no H.264 é chamado de MVC (*Multi-View Coder* – H.264/MVC) (ITU, 2009) e usa um esquema com quadros B hierárquicos para cada uma das visões assim como entre as visões. Esta estrutura de grupos de quadros B (GOP) já estava presente no H.264 AVC e foi extendida para suportar múltiplas fontes de vídeo. A Figura 3.23 ilustra este processo, onde a predição inter-visões é aplicada de forma bidirecional em S1, S3 e S5 na Figura 3.23, extraída de Pandit (2008). Quadros I são colocados a cada início de GOP para permitir a ressincronização.

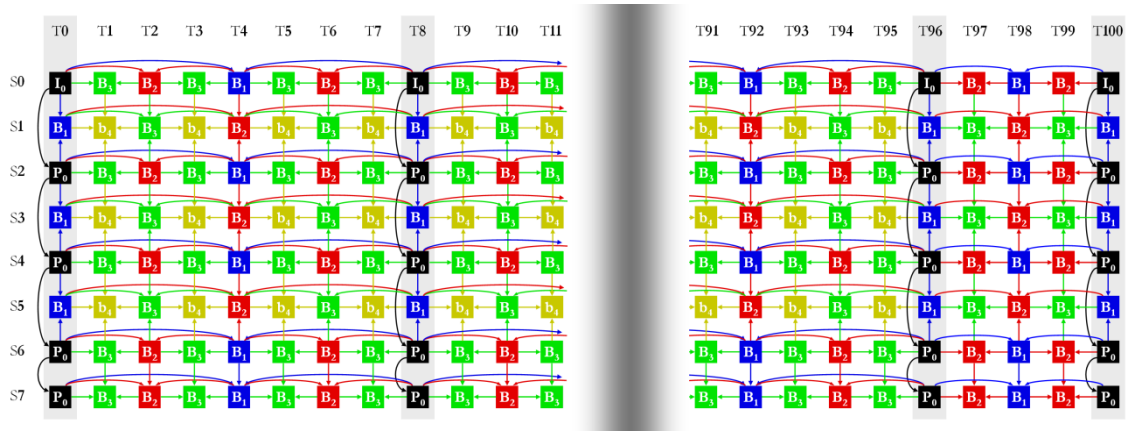

Figura 3.23: Estrutura de predição temporal entre visões usando quadros B hierárquicos.

No JMVM 8.0 (PANDIT, 2008), a ordem de codificação dos quadros prioriza a codificação dos quadros de todas as visões, seguido da codificação dos quadros sucessivos no tempo. Na Figura 3.23, os quadros serão codificados em colunas, primeiro todos para o T0 (S0, S2, S1, S4, S3, S6, S5), seguindo T8, T4, T2, T1, T3, T6, T5, T7. Este esquema de ordenamento também é chamado de Ordem de Codificação Primeiro no Tempo (*Time-First Coding Order*) e pode ser expresso pelo seguinte pseudocódigo da Figura 3.24.

```
for ( i = 0; i < total_num_frms_per_view; i++) {
  for ( j = 0; j < total_num_views; j++) {
    jj = view_id(j);
    ii = temporal_index(i);
    process_one_frame(jj, ii);
  }
}
```

Figura 3.24: Esquema de ordenamento *Time-First Coding Order*.

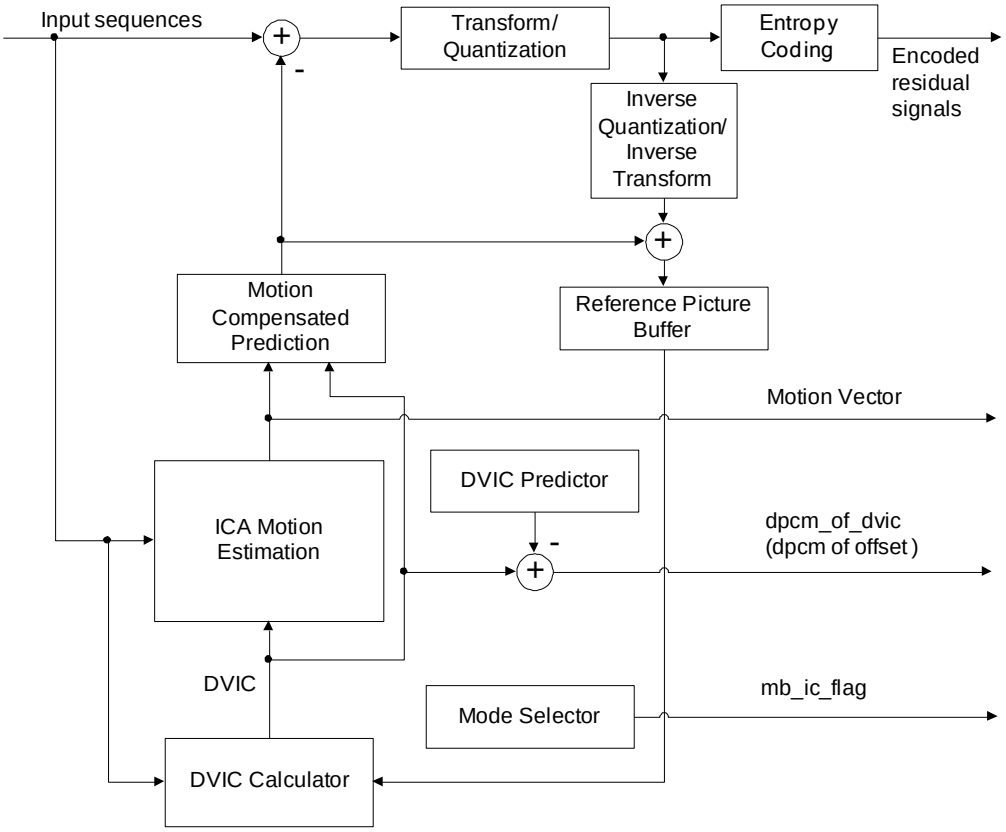

Figura 3.25: Estrutura de codificação um laço ME/MC com ICA baseado em macroblocos (VETRO, 2008).

Uma das inclusões importantes do suporte a múltiplas visões no H.264 é a compensação de iluminação. Para compensar uma variação de iluminação entre quadros, uma técnica chamada ICA MC (*illumination change-adaptive motion compensation*) é habilitada para diversos modos de MB do H.264, que incluem: Inter 16x16, Direct 16x16, (incluindo B_Skip) e o modo P_Skip. O valor diferencial da mudança de iluminação (DVIC – *Difference Value of Illumination Change*) é codificado preditivamente usando DPCM a partir dos valores de DVIC dos macroblocos da vizinhança usando o ICA MC (VETRO, 2008). A Figura 3.25 ilustra a estrutura de codificação usando este método. No encoder, o ICA MC é necessário para o modo Intra 16x16.

Além do esquema de predição, o Filtro de Deblocagem é modificado para filtrar artefatos de bloco gerados pela variação de iluminação.

O movimento de visões vizinhas é altamente correlacionado, por isso o modo Motion Skip é usado para melhorar a eficiência da codificação. Se o modo Motion Skip é usado, nenhuma outra informação de movimento é transmitida para o macrobloco atual. Este modo especifica que a informação de movimento, incluindo o particionamento do macrobloco, o índice de referência e o vetor de movimento do macrobloco atual é obtido a partir do macrobloco correspondente na visão vizinha.

3.5 Principais Desafios

O padrão H.264 é muito extenso e complexo, como foi possível perceber no decorrer deste texto. Deste modo, implementar um codificador ou decodificador H.264 não é uma tarefa trivial, mesmo que esta implementação seja realizada em software. O

problema passa a ser muito mais crítico se a implementação tiver por objetivo atingir tempo real para resoluções mais elevadas, como HDTV ou superior.

Em função do que foi exposto, é possível identificar que o bloco da estimação/compensação de movimento é o gargalo de desempenho do codificador. Por outro lado, os blocos do filtro redutor de blocagem e da compensação de movimento são os gargalos de desempenho do decodificador. Mas todas as análises apresentadas na seção anterior foram desenvolvidas a partir da execução do código de referência sobre um processador de propósito geral. Por isso, esta análise de complexidade não considerou as possíveis adaptações algorítmicas visando aumentar o paralelismo nos blocos do codificador e do decodificador. Este tipo de análise, buscando identificar as dependências de dados dos algoritmos utilizados e, por consequência, definindo qual o grau de paralelismo que estes algoritmos permitem, é muito importante para avaliar a complexidade quando estes algoritmos estão sendo executados em um ambiente multiprocessado ou quando os algoritmos serão implementados em hardware dedicado, com a consequente exploração de paralelismo.

A estimação de movimento, que é o bloco de maior complexidade computacional do codificador, é um exemplo de bloco que pode ser implementado de uma maneira massivamente paralela, pois as dependências de dados são pequenas. Deste modo, tratando o problema da estimação de movimento com múltiplas unidades trabalhando em paralelo, é possível aumentar a taxa de processamento deste bloco e fazer com que os requisitos da aplicação sejam atingidos. É interessante notar que praticamente todos os trabalhos da literatura que visam minimizar a complexidade da estimação de movimento, consideram apenas a redução da complexidade quando um código fonte tipicamente serial é executado em um processador convencional. Algumas destas soluções aumentam a dependência de dados com o objetivo de reduzir o número de instruções executadas sequencialmente. O efeito colateral desta solução é que a exploração do paralelismo fica mais restrita. Assim, é possível que um algoritmo que minimiza a complexidade da estimação de movimento (considerando um processador de propósito geral) acabe por atingir uma taxa de processamento inferior aos algoritmos não otimizados, quando ambos forem implementados em hardware, tendo em vista que a exploração do paralelismo será dificultada com o algoritmo otimizado se a otimização implicar em um aumento na dependência de dados.

Ainda assim, os resultados da seção anterior espelham os maiores desafios para os projetistas de codecs H.264 em software ou em hardware, quais sejam, encontrar uma solução para a questão da complexidade dos blocos da estimação de movimento, da compensação de movimento, da predição intra, do filtro de blocagem, etc.

O foco dos projetistas de hardware reside na exploração do paralelismo com o objetivo de atingir tempo real para qualquer resolução de vídeo. Em função da complexidade e extensão do padrão, não existe nenhum bloco simples de ser implementado em hardware para atingir os requisitos da aplicação alvo, pois em cada bloco do codificador ou do decodificador residem desafios de difícil solução.

O projeto de um codificador ou decodificador de vídeo em hardware envolve uma série de compromissos relacionando requisitos, por vezes, contraditórios. Alguns destes compromissos são comuns para os projetistas de software e hardware, mas outros são específicos para os projetistas de hardware. O projeto de um codec de vídeo em hardware deve estar focado, entre outros, nos seguintes requisitos:

- **Taxa de compressão**: o codificador deve ser capaz de gerar um *bitstream* codificado eficientemente, respeitando as exigências de armazenamento e/ou transmissão e/ou display da aplicação.
- **Qualidade do vídeo**: o codificador, mesmo gerando perdas de informação no processo de codificação, deve ser capaz de gerar um *stream* de bits que, ao ser decodificado, apresente uma qualidade objetiva e subjetiva de imagem que esteja de acordo com as exigências da aplicação alvo.
- **Largura de banda para transmissão**: o codificador deve ser capaz de utilizar apenas a largura de banda exigida pela aplicação quando o vídeo comprimido for transmitido. Este requisito está diretamente relacionado com a taxa de compressão.
- **Taxa de processamento**: o codificador ou decodificador deve ser capaz de processar o vídeo de entrada na taxa de amostras por segundo exigida pela aplicação. A exploração do paralelismo através de implementações em hardware contribui para o atendimento deste requisito, mas aumenta a utilização de recursos de hardware e de uso de memória.
- **Quantidade de elementos de hardware utilizados**: a implementação do codificador ou decodificador deve respeitar as restrições de utilização de elementos de hardware impostas pela aplicação ou pelo custo de fabricação. Deste modo, o projeto deve, respeitando as exigências da taxa de processamento, utilizar a menor quantidade de hardware possível.
- **Largura de banda de memória**: o projeto do codificador e do decodificador deve considerar que um dos maiores gargalos deste tipo de sistema está na comunicação com a memória. Assim, o projeto deve buscar a minimização dos acessos à memória, para respeitar as restrições de largura de banda disponível para a comunicação com a memória. Em projetos completamente dedicados, memórias específicas e até uma hierarquia de memória podem ser implementadas para viabilizar uma comunicação mais efetiva com a memória. Neste caso, a quantidade de recursos de hardware utilizada irá crescer.
- **Energia consumida**: muitas aplicações que utilizam codecs de vídeo são sistemas embarcados, onde a minimização na energia consumida é um requisito fundamental. Neste caso, o projeto do codec pode aplicar algumas técnicas para minimizar o consumo de energia. Contribuem para atender este requisito a minimização dos acessos à memória, a redução na utilização de recursos de hardware, entre outros.

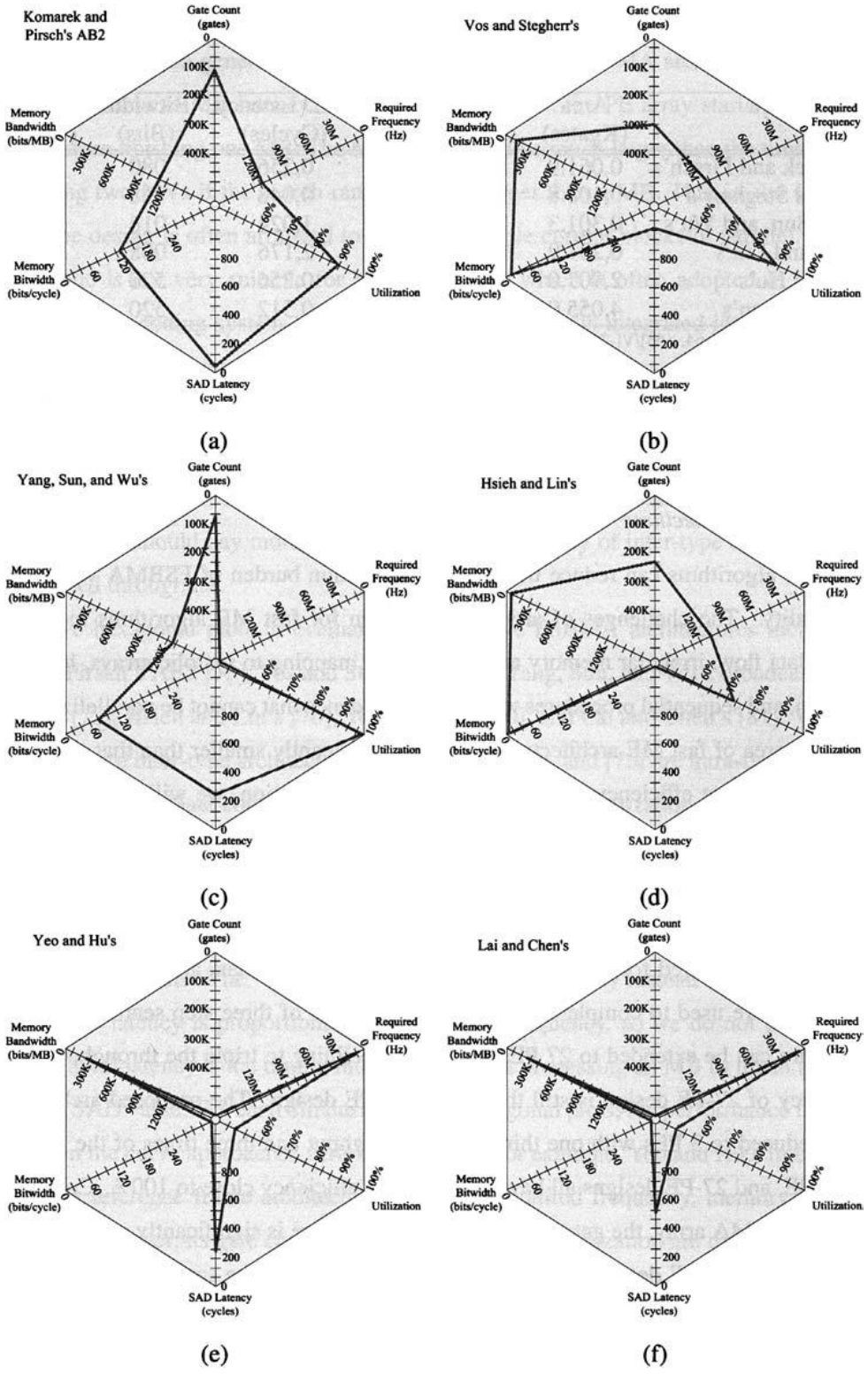

Figura 3.26: Análise para diferentes implementações de estimação de movimento [reproduzido de Huang (2006)].

O projeto de um codec de vídeo deve estar focado nestes requisitos e, dependendo da aplicação alvo, deve ser tomada a decisão de quais requisitos serão priorizados, em detrimento dos demais. Partes específicas de um codec podem exigir métricas mais específicas. A Figura 3.26 ilustra uma análise de diferentes arquiteturas para a estimação de movimento, onde a largura de banda de memória (*Memory Bandwidth*) em

bits por macrobloco e bits por ciclo, o número de portas (*gate count*), a frequência requerida (*required frequency*), o ciclo ativo de utilização (*utilization*) e a latência da operação de soma das diferenças absolutas (*SAD latency*) são utilizados como métricas de comparação.

No projeto de um codec H.264 esta relação contraditória de requisitos é ainda mais problemática, pois este é o padrão mais complexo desenvolvido até então.

Em função do estudo do padrão que conduziu a este texto e em função dos trabalhos relacionados que foram apresentados, é possível concluir que todos os módulos do H.264 são desafiadores devido à complexidade dos algoritmos envolvidos. Porém, cada bloco apresenta desafios específicos:

- **Estimação de movimento (codificador)**: este é o bloco que possui a maior complexidade computacional no codificador, exigindo o maior número de operações aritméticas para concluir seus cálculos, além de exigir o maior número de acessos à memória. Existem vários algoritmos sub-ótimos que podem ser aplicados visando reduzir a complexidade e o número de acessos à memória realizados por este bloco. Além disso, para alguns destes algoritmos a dependência de dados tende a ser reduzida, permitindo uma elevada paralelização das operações, elevando a taxa de processamento, mas também crescendo a utilização de recursos e com impacto na energia consumida.
- **Predição Intra (codificador)**: este bloco possui uma complexidade muito inferior a da estimação de movimento no codificador, mas ainda assim, por possibilitar até treze diferentes tipos de codificação, este bloco possui uma complexidade que é capaz de tornar o seu projeto em hardware uma tarefa desafiadora. Um dos maiores problemas é que as amostras utilizadas para realizar a predição intra são as amostras reconstruídas para o quadro atual. Isso significa que os blocos preditos devem ser processados pelos blocos T, Q, Q^{-1} e T^{-1} antes de serem utilizados como referência para processar os próximos blocos do quadro atual. Isso implica que a predição intra deve ficar "esperando" para que os dados sejam reconstruídos. Uma possível solução para este problema é a utilização das amostras originais, ao invés das amostras reconstruídas, causando uma distorção na imagem no momento da decodificação. Esta distorção deve ser criteriosamente avaliada para que esta solução seja utilizada.
- **Controle (codificador)**: o controle do codificador é responsável por tomar as decisões sobre a codificação. Para tanto, todos os modos possíveis de codificação devem ser testados e seus custos devem ser avaliados em função da relação taxa-distorção-complexidade. As decisões em si não são tarefas complexas, mas elas são tomadas sobre dados que precisam ser gerados para todas as possibilidades de codificação possíveis.
- **Transformadas e quantização (codificador):** estes blocos (T, T^{-1}, Q e Q^{-1}), mesmo tendo uma complexidade computacional reduzida se comparado aos outros blocos de um codec H.264, estão no caminho crítico da predição intra, como já foi explicado. Deste modo, quanto maior for o desempenho destes blocos menor será o tempo em que a predição intra deverá ficar parada. Além disso, os blocos das transformadas e quantização diretas e inversas são muito importantes para o controle do codificador, pois as decisões relativas ao melhor tamanho de bloco utilizado são realizadas a partir de comparações sobre a

imagem reconstruída. A reconstrução da imagem passa pelos quatro blocos das transformadas e quantização e, deste modo, estão no caminho crítico deste bloco e do codificador como um todo.

- **Codificação e decodificação de entropia (codificador e decodificador)**: O padrão H.264 suporta o CABAC e o CAVLC como métodos de codificação de entropia. O perfil *Baseline* suporta apenas o CAVLC, enquanto os demais recomendam o uso CABAC pela sua maior eficiência. A eficiência maior do CABAC para a compressão de entropia está associada a uma complexidade mais elevada que a do CAVLC. Além disso, a presença de contextos (que são probabilidades estatísticas de determinados elementos sintáticos produzidos durante o processo de codificação) aumenta ainda mais a complexidade, pois estes contextos devem ser consultados e atualizados. No CAVLC, a adaptação ao contexto ocorre através do chaveamento de tabelas de probabilidades, enquanto que no CABAC os contextos estão associados à codificação de cada elemento binário (*bin*) que compõe o elemento sintático e devem ser consultados e atualizados a cada passo do codificador aritmético e este opera apenas 1 elemento binário a cada passo. Isto gera dependências de dados no nível de elemento binário, o que torna o CABAC estritamente sequencial e implementações em hardware tendem a gerar um caminho crítico bastante longo devido a dependência de acessos à memória de contextos. Por isso o projeto do CABAC deve atingir altas frequências de relógio devido a dificuldade de paralelismo, porém é penalizado devido ao caminho crítico longo. As propostas arquiteturais devem então tentar extrair paralelismo do CABAC através da execução especulativa do codificador aritmético, e também da redução do caminho crítico através do uso de registradores para os contextos mais frequentes dentre outras técnicas.
- **Filtro de deblocagem (codificador e decodificador)**: O filtro redutor de blocagem é mandatório no H.264 e o processo de adaptação se dá através da seleção de diferentes filtros (com até 7 taps) dependendo de uma força do filtro (*boundary strength* – BS) que deve ser calculada para cada elemento da borda a ser filtrada. O grande desafio do Filtro de Deblocagem está justamente na seleção do BS que depende de uma série de cálculos envolvendo parâmetros que incluem o QP, o tipo de codificação usada (inter ou intra) o tipo de macrobloco (I, P ou B), o tipo de amostra (luma ou croma), os vetores de movimento usados e também os valores dos pixels da borda envolvida. Isso aliado ao fato de que alguns parâmetros são calculados a partir de tabelas de consulta (*lookup tables*) que devem ser implementas como memória. Além disso, o Filtro de Deblocagem precisa realizar filtragem bidimensional (bordas verticais e horizontais), e reordenar e armazenar temporariamente blocos para serem processados na ordem correta, o que torna um grande consumidor de memória. Muitas implementações do Filtro de Deblocagem consideram que o armazenamento temporário dos blocos é feito na memória principal e não em uma memória local específica para ele.
- **Compensação de movimento (decodificador)**: A compensação de movimento é um dos blocos mais complexos do decodificador, envolvendo muitas operações aritméticas e muitos acessos à memória. O maior desafio no projeto deste bloco reside no fato de que ele deve estar apto a tratar qualquer tipo de

informação gerada pela estimação de movimento do codificador. Novamente, a dependência de dados é pequena e, então, o paralelismo pode ser explorado.

- **Predição intra (decodificador)**: A predição intra no decodificador não apresenta os mesmos desafios da predição intra do codificador, uma vez que a dependência de dados relativa à reconstrução do bloco atual existe, mas não é dependente das operações T e Q, que não existem no decodificador. Ainda assim este bloco é um dos blocos mais complexos do decodificador e demanda um projeto dedicado com algum grau de paralelismo, especialmente para aplicações com resoluções elevadas.

O suporte a escalabilidade e múltiplas visões no H.264 torna o projeto do codec ainda mais desafiador, pois aumentam a complexidade de cada um dos blocos individuais, acrescentam blocos e ainda exigem maior taxa de dados para processar múltiplos quadros. No caso específico da escalabilidade, a complexidade do decodificador pode ser reduzida se a aplicação alvo não exigir a decodificação de todas as camadas; o limite inferior de complexidade será semelhante a de um decodificador AVC com as mesmas características da camada base.

4 ARQUITETURA PARA O FILTRO DE DEBLOCAGEM

No padrão H.264, um filtro adaptativo foi incluído para reduzir os artefatos introduzidos pelo processo de transformada/quantização empregado nos codificadores híbridos baseados em bloco (chamados de artefatos ou efeitos de bloco), que se tornam perceptíveis especialmente em vídeos muito comprimidos (onde muita informação no processo de quantização foi desprezada). O principal objetivo da inclusão deste tipo de filtro como parte do padrão foi colocá-lo dentro do laço de reconstrução da imagem, o que resulta em uma melhor desempenho quando comparado com filtros adicionados como uma etapa de pós-processamento (RICHARDSON, 2003). O Filtro de Deblocagem no H.264 não é apenas um filtro passa-baixas simples, mas um conjunto de 5 filtros selecionados de acordo com um algoritmo de decisão. O objetivo é eliminar os artefatos de bloco introduzidos pela quantização do resíduo produzido pela predição intra e pela compensação de movimento que operam em pequenos blocos. Apesar disso, a seleção criteriosa do filtro visa manter os detalhes das imagens.

A complexidade da lógica de decisão e filtragem faz o Filtro de Deblocagem do H.264 ser responsável por aproximadamente um terço do processamento necessário pelo processo de decodificação quando executado em um processador de uso geral.

Uma arquitetura para o Filtro de Deblocagem de alto desempenho para o codificador e decodificador padrão H.264 foi então desenvolvida, que pode ser usada como um módulo de um codec H.264 dedicado ou como um acelerador para o processo de filtragem em conjunto com um processador de uso geral. O objetivo principal foi conceber uma arquitetura capaz de alcançar o desempenho necessário para HDTV 1080p (video progressivo, 1920x1080 pixels a 30 quadros por segundo - 63 milhões de amostras no formato 4:2:0 por segundo) quando sintetizado para um FPGA.

4.1 Algoritmo do Filtro de Deblocagem

No padrão H.264 a imagem é dividida em pequenas unidades chamadas blocos. Cada bloco tem 4x4 pixels. O formato de cores é o YCbCr 4:2:0 (para o perfil *Main*), o que quer dizer que os componentes de crominância são sub-amostrados para a metade da taxa dos componentes de luminância (luma) em ambas as direções. Os blocos são então agrupados em macroblocos que são matrizes de 4x4 blocos para luma e 2x2 blocos para cada componente de croma. Cada borda de bloco deve ser filtrada pelo Filtro de Deblocagem. O filtro então é aplicado a cada uma das bordas de cada bloco de luma ou croma codificado. A Figura 4.1 apresenta as bordas que devem ser filtradas pelo Filtro de Deblocagem.

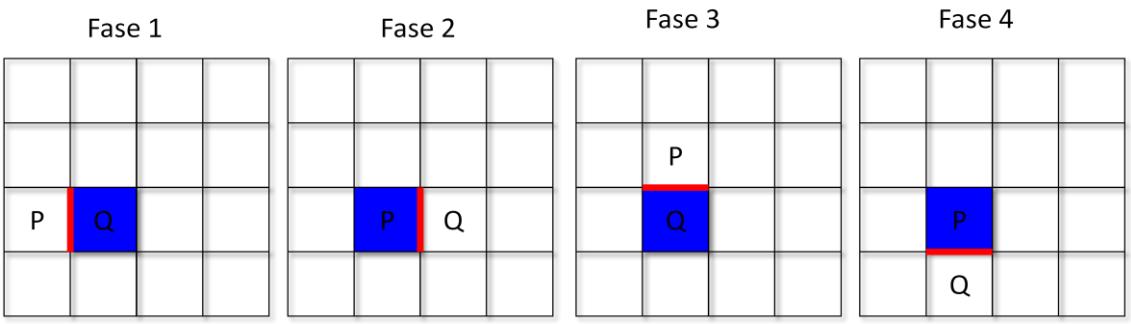

Figura 4.1: Posições das bordas para um dado bloco de 4x4 pixels em um macrobloco de 16x16 pixels.

Para cada borda do bloco, o filtro é aplicado aos componentes dos pixels perpendiculares àquela borda. A convenção de nomes para os pixels em torno da borda é mostrado na Figura 4.2. Os componentes dos pixels tanto no bloco atual (Q) quanto no anterior (P) podem ter seus valores alterados. Os valores dos pixels já modificados durante um estágio da filtragem podem ser modificados novamente em uma subsequente operação de filtragem aplicada a uma borda que envolva o mesmo bloco. O algoritmo de filtragem é adaptativo, o que significa que os valores dos pixels, a posição do bloco dentro de um macrobloco, o tipo de predição aplicado (intra ou inter), os vetores de movimento empregados e os parâmetros de quantização são levados em consideração para o calculo da força do filtro.

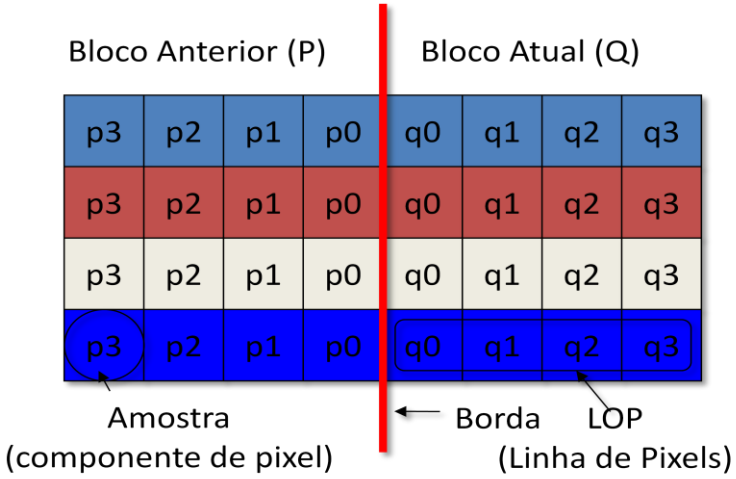

Figura 4.2: Convenções do Filtro de Deblocagem.

A força do filtro (BS – *Boundary Strength*) pode assumer 5 valores diferentes, entre 0 (nenhuma filtragem) até 4 (filtragem mais forte) e é definido como se segue pelo padão H.264 (ITU, 2003):

- Se uma borda não é uma borda do *slice* (o filtro não é aplicado além das bordas do *slice*), então o filtro pode ser aplicado, senão BS=0. Baseado nos valores dos pixels, pode ser verificado que uma borda contém uma descontinuidade natural nos valores dos pixels (e não um artefato de bloco). Neste caso, BS=0.
- Se o bloco é codificado como intra, então BS=3; se é uma borda externa, então BS=4;

- Se nenhum bloco é codificado como intra e pelo menos um contém coeficientes codificados, então o BS=2;
- O BS=1 é usado quando nenhuma das condições acima é satisfeita e os quadros de referência dos dois blocos são diferentes ou quando os quadros de referência são os mesmos, mas pelo menos um dos componentes dos vetores de movimento difere em mais do que 4 (a distância de 1 pixel)

A força do filtro (BS) para a informação de croma é a mesma que a do bloco de luma correspondente, mas os filtros empregados para luma são diferentes. O parâmetro de quantização (QP) e os valores dos pixels são considerados. Os parâmetros α e β são dependentes do QP e definem os limiares para a aplicação do filtro. As funções de saturação (*clip* e *clip3*) precisam ser usadas em alguns passos dos cálculos que levam ao BS. Isto torna o cálculo do BS um conjunto de operações que devem ser feitas em sequência para cada borda da linha de pixels (*Line of Pixels* - LOP) a ser filtrada. Maiores detalhes podem ser encontrados em ITU (2003) e um fluxograma completo em Puri (2004).

4.2 Arquitetura proposta

A arquitetura proposta foi desenvolvida para suportar a taxa de pixels necessária para HDTV (1920x1080x30) em um FPGA. O filtro foi desenvolvido para suportar o perfil *Main* do padrão H.264.

O conceito arquitetural inicial foi baseado no trabalho desenvolvido por Sima (2004) que implementa um acelerador de hardware conectado a um barramento para ser usado em conjunto com um microprocessador. Este trabalho apresenta uma implementação em do filtro de borda através de um *pipeline* de 4 estágios, que aumenta significativamente a frequência de *clock*.

A arquitetura proposta neste trabalho é auto-contida, implementando todas as memórias necessárias para o armazenamento dos dados intermediários e também incorpora no filtro de borda a decisão do modo de filtragem.

A decisão de focar o desenvolvimento para FPGA levou a algumas decisões arquiteturais importantes. Uma delas foi o balanceamento entre lógica e registradores, uma vez que os elementos lógicos dos FPGA contêm ambos. Outra decisão importante foi o uso intenso dos blocos de memória distribuída. Em uma implementação ASIC faz sentido tentar minimizar a quantidade total de lógica, pois isso tende a minimizar a área em silício; em FPGA o uso balanceado dos recursos disponíveis pode levar a um desempenho maior sem uso adicional de células lógicas.

Neste cenário, o desenvolvimento de uma arquitetura com um *pipeline* profundo foi direto: Como cada elemento lógico no FPGA (Virtex II) contém uma *lookup-table* (LUT) de quarto entradas e um flip-flop (registrador) e mais lógica de propagação de *carry*, a utilização de registrador após cada operação lógica ou aritmética básica não acarreta no uso de recuros adicionais; a não utilização deste registrador na maioria dos casos faz com que o mesmo seja colocado em modo *bypass*, não sendo utilizado por questões de roteamento.

Para um melhor entendimento da arquitetura proposta, algumas convenções sobre o posicionamento dos blocos são apresentadas na Figura 4.3. Os 16 blocos de luma que compõe um macrobloco são numerados primeiro e então cada um dos 4 blocos de crominância em azul e vermelho. Num dado ponto do processamento os blocos

pertencentes ao macrobloco atual em processamento são coloridos em branco, os do macrobloco à esquerda são coloridos em cinza e os blocos do macrobloco superior são coloridos em preto.

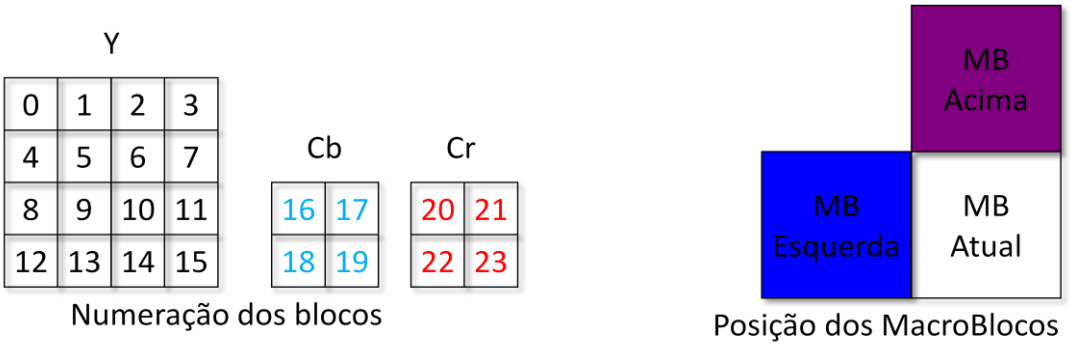

Figura 4.3: Enumeração dos blocos de luma/croma e diagrama de cores de macroblocos.

Na arquitetura proposta para o Filtro de Deblocagem a entrada é composta de valores de pixels reconstituídos e informações relativas ao processamento efetuado sobre estes pixels (QP, intra/inter, etc.) e produzirá valores de pixels filtrados. A sequência de dados de entrada para um dado macrobloco é apresentada na Figura 4.4. Cada bloco numerado corresponde a um bloco 4x4 de luma ou croma, de acordo com a numeração da Figura 4.3.

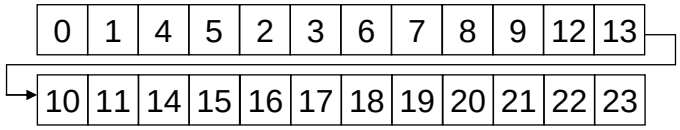

Figura 4.4: Sequência de blocos esperada na entrada do Filtro de Deblocagem.

No Filtro de Deblocagem, o filtro de borda é o coração do processo. Ele é responsável por toda a funcionalidade do filtro, incluindo o cálculo dos limiares de filtragem, do BS e a própria aplicação do filtro selecionado pelo BS. O restante do processo é apenas controle e reordenamento de valores.

A arquitetura do filtro de borda desenvolvida aceita duas Linha de Pixels (LOP) por ciclo para os blocos Q e P, de acordo com a convenção apresentada na Figura 4.3 e produz duas LOP filtradas Q' e P'. Este processo é ilustrado na Figura 4.5. Usando este método, uma borda inteira entra no filtro a cada 4 ciclos.

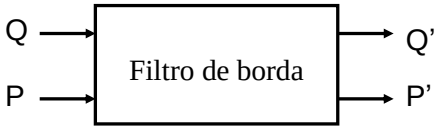

Figura 4.5: Filtro de borda.

Baseado no diagrama apresentado na Figura 4.5, uma arquitetura em *pipeline* para o filtro de borda foi projetada. Como mencionado anteriormente, o processo para o cálculo de todos os parâmetros necessários para decidir o BS a ser empregado requer uma quantidade significativa de cálculos em sequência devido a dependências de dados. Esta arquitetura foi desenvolvida de forma que apenas uma operação lógica ou aritmética por estágio do *pipeline* fosse necessária. Isto levou ao desenvolvimento de um *pipeline* de 11 estágios apenas para calcular o BS. Todos os filtros são aplicados em paralelo e os resultados da filtragem para cada um dos filtros já está disponível quando

o cálculo do BS finaliza no *pipeline*. O BS então é usado para selecionar um MUX que leva o resultado correto para a saída. Um total de 12 estágios são empregados para todo o processamento. Analisando-se a lógica de processamento do Filtro de Deblocagem, é possível perceber que após uma coluna inteira de blocos de luma ser processada os blocos que foram processados como blocos atuais (Q) para a filtragem da borda esquerda (ou superior) devem entrar novamente no filtro para que a operação de filtragem possa ocorrer na borda direita (ou inferior). Como cada coluna de blocos contém 16 LOPs, 16 ciclos após a entrada de um LOP Q no *pipeline*, este terá que entrar novamente, mas desta vez como P. Por esta razão, o *pipeline* pode ser entendido através de um FIFO para conter exatamente 16 estágios, de forma que a saída Q' possa ser conectada diretamente à entrada P'.

Os macroblocos de croma, entretanto possuem apenas 8 LOPs empilhadas. Para resolver este problema, os macroblocos Cb e Cr são empilhados de forma a conterem 16 LOPs, tornando o processamento análogo ao processamento dos blocos de luma. A Figura 4.6 ilustra a arquitetura do *pipeline* de 16 estágios proposta para o filtro de borda.

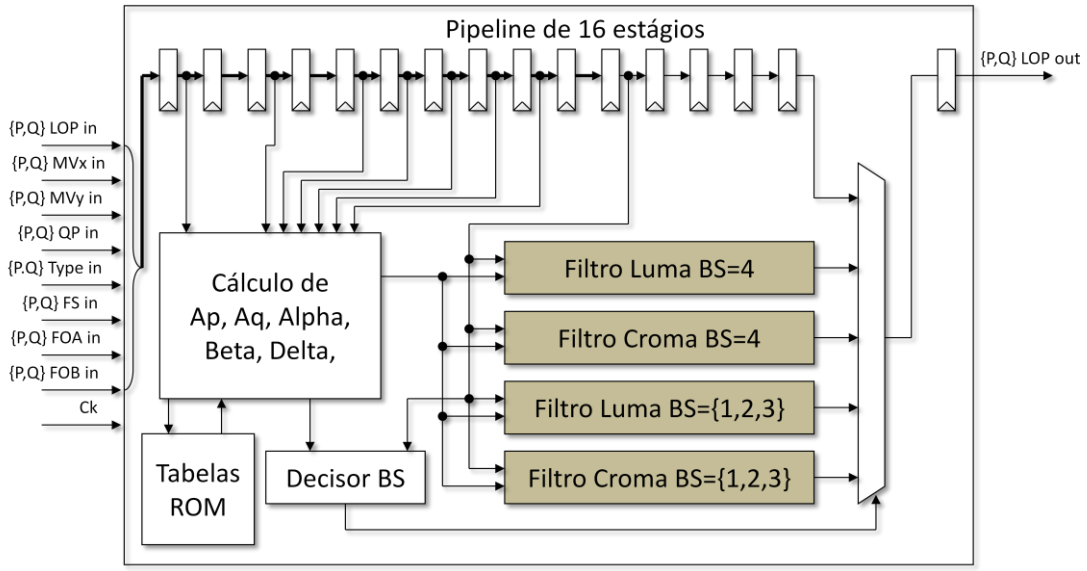

Figura 4.6: Arquitetura do filtro de borda.

A Figura 4.7 apresenta o encapsulamento do filtro de borda, de forma que a saída Q' seja conectada a entrada P.

Figura 4.7: Encapsulamento do filtro de borda.

O controle do Filtro de Deblocagem utilizando o *pipeline* apresentado na Figura 4.6 e o encapsulamento proposto na Figura 4.7 se torna significativamente mais simples, entretanto há um pequeno *overhead*: Os dados (valores de pixel e informações de processamento dos blocos) destinados à entrada P só podem entrar através da saída Q'. Isto faz com que os dados tenham que percorrer os 16 estágios do *pipeline* antes que o processamento efetivo se inicie. Além disso, após o processamento de um macrobloco, o *pipeline* tem que ser esvaziado. Felizmente, o esvaziamento do *pipeline* para um

macrobloco pode ser sobreposto ao carregamento do *pipeline* para o próximo macrobloco.

A arquitetura proposta para o Filtro de Deblocagem é apresentada na Figura 4.8. O filtro tem que ser aplicado tanto nas bordas verticais quanto nas horizontais. Na arquitetura proposta, um único filtro de borda filtra tanto as bordas verticais quanto as horizontais. Um módulo de transposição é empregado para converter as amostras alinhadas verticalmente em amostras alinhadas horizontalmente, de forma que as bordas horizontais possam ser filtradas pelo mesmo filtro de borda.

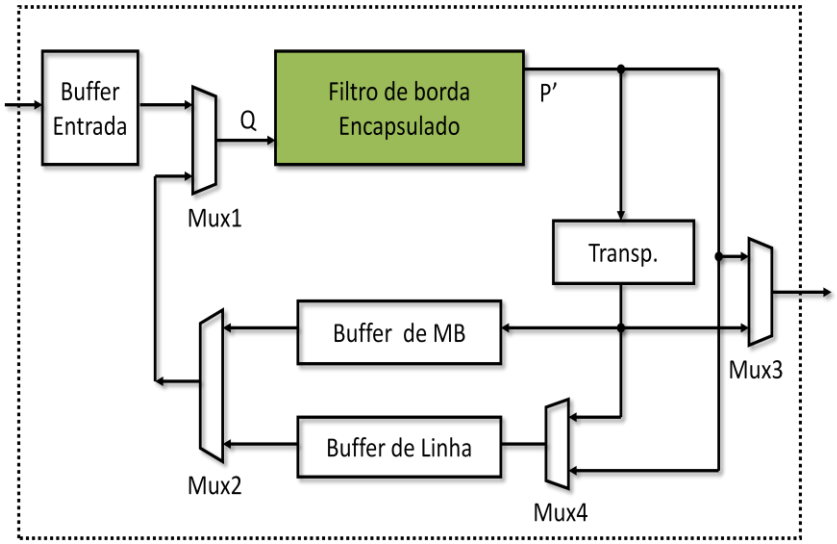

Figura 4.8: Arquitetura proposta para o Filtro de Deblocagem.

A arquitetura proposta contém alguns buffers necessários ao processamento do filtro. O buffer de entrada é necessário apenas para o reordenamento dos dados, uma vez que os dados chegam ao Filtro de Deblocagem em uma ordem diferente da ordem de processamento. O buffer de macroblocos e o buffer de linha de blocos são usados para armazenar os dados (valores de amostra e informações sobre o bloco) que ainda não foram totalmente filtrados. O Buffer de MB guarda informações relativas a 32 blocos enquanto o Line Buffer guarda o número de blocos necessário para formar uma linha de blocos 4x4 na resolução escolhida (480 blocos para 1080p).

4.3 Operação do filtro

O fluxo de dados do filtro é descrito a seguir. Inicialmente, os valores das amostras e as informações necessárias para a decisão do BS entram pelo Input Buffer. Este *buffer* é necessário pois os blocos de entrada estão em uma ordem diferente dos da ordem de processamento do filtro. Uma vez que um macrobloco completo é carregado no Input Buffer, o processo de filtragem pode começar.

O filtro de borda encapsulado contém o filtro de borda implementado como um *pipeline* de 16 estágios. O MUX1 e o MUX2 são selecionados de forma que os dados do Buffer de MB sejam postos na entrada Q do filtro de borda encapsulado. Os dados são lidos do Buffer de MB correspondem aos blocos 3, 7, 11 e 15 dos macroblocos da esquerda. Para facilitar o controle, esta operação é realizada mesmo para o macrobloco mais a esquerda do *slice* (neste caso, o filtro é desligado, pois não há macrobloco à esquerda do atual e os dados lidos não fazem sentido para a imagem). Como os dados são lidos uma LOP por ciclo, um bloco demora 4 ciclos para entrar no filtro de borda

encapsulado; para ler os 4 blocos que compõe uma coluna de blocos, exatamente 16 ciclos são necessários. Durante esta fase, o filtro é permanece no modo *bypass* (BS=0), para que os valores não sejam filtrados durante esta fase de preenchimento do *pipeline*. Após 16 ciclos, os dados da entrada P estão disponíveis e a filtragem pode começar com a alimentação da entrada Q com os LOPs do bloco atual. Os blocos 0, 4, 8 e 12 são lidos do Buffer de entrada, uma LOP por ciclo.

Neste ponto o filtro de borda está sendo alimentado com dados para a filtragem de uma borda externa. Imediatamente após o bloco 12, os blocos 1, 5, 9 e 13 são lidos do Buffer de entrada e iniciam seu processamento, usando os blocos 0, 4, 8 e 12 recentemente filtrados como blocos P. O processo se repete para os blocos 2, 6, 10 e 14 e para os blocos 3, 7, 11 e 15. 32 ciclos após o bloco 3 do macrobloco à esquerda começar a ser lido do MB Buffer este aparece filtrado na saída do filtro de borda encapsulado.

Na saída P' do filtro de borda, os dados podem ter 3 destinos distintos. O Buffer de MB, o Buffer de Linha ou a saída do filtro. No caso dos blocos 3, 7 e 11 que foram lidos no começo da sequência apresentada acima, não há mais processamento a ser feito pois todas as bordas foram filtradas e estes blocos são encaminhados à saída do filtro através do MUX3. O bloco 15 do macrobloco a esquerda é posto no Line Buffer, através do processo Transpose para ser usado posteriormente na filtragem horizontal, visto que ainda resta a borda inferior deste bloco a ser filtrada. Este bloco se tornará parte do macrobloco superior quando ocorrer a próxima linha de macroblocos. Os blocos de 0 a 15 são enviados para o MB Bufer através do bloco de transposição para que a filtragem das bordas horizontais possa ocorrer. O processo Transpose faz a transposição de blocos 4x4 pixels.

Após a filtragem das bordas verticais de todos os blocos de luma, o trabalho do filtro para as bordas horizontais pode ser feito. Para obter o máximo desempenho, a filtragem das bordas verticais de croma é realizada imediatamente após a filtragem das bordas verticais de luma, na seguinte ordem: os blocos 17, 19, 21 e 23 dos macroblocos Cb e Cr à esquerda são lidos do MB buffer. Da mesma forma que para luma, estes blocos são lidos do MB Buffer mesmo que não haja informação de bloco à esquerda de forma a simplificar o controle. Os blocos de Cb e Cr ficam empilhados para que os 16 LOPs necessários para encher o filtro de borda sejam atingidos. O processamento então ocorre da mesma forma que para os blocos de luma, exceto que os filtros e os valores de BS são distintos para luma e croma: os blocos 16, 18, 20 e 22 são lidos do Input Buffer e processados e logo a seguir os blocos 17, 19, 21 e 23. Neste ponto, os blocos 17 e 21 do macrobloco à esquerda estão prontos e podem ser enviados para a saída. Os blocos 19 e 23 do macrobloco à esquerda são enviados para o Line Buffer através do processo Transpose para que seu processamento seja finalizado quando a próxima linha de macroblocos for processada (pois está faltando a filtragem da borda horizontal inferior).

Neste ponto, a filtragem das bordas horizontais para luma e croma pode ser realizada. Os blocos 12, 13, 14 e 15 do macrobloco superior são carregados do Line Buffer para o filtro de borda encapsulado e logo após os blocos 0, 1, 2 e 3 do bloco atual transposto, armazenado no MB Buffer. O processamento segue pela leitura dos blocos 4, 5, 6 e 7, seguido do 8, 9 10 e 11 e finalmente o 12, 13 14 e 15. Após a filtragem, os blocos 12, 13 14 e 15 do macrobloco superior estão prontos e podem sair do Filtro de Deblocagem. Assim como os blocos 0, 1, 2, 4, 5, 6, 8, 9 e 10. Os demais blocos podem ser processados através de dois caminhos distintos. Os blocos 3, 7, 11 e 15 vão para o MB Buffer através do processo Transpose para a próxima filtragem vertical (eles farão

parte dos blocos de luma do macrobloco à esquerda durante o processamento do próximo macrobloco); Os blocos 12, 13 e 15 vão para o Line Buffer sem ser transpostos (eles já estão transpostos e serão usados como blocos do macrobloco superior durante a filtragem da próxima linha de macroblocos).

A última fase da filtragem é a aplicação da filtragem das bordas horizontais nos blocos de croma. Assim como na filtragem das bordas verticais, os macroblocos de croma precisam ser alinhados e filtrados juntos para preencher os 16 estágios do filtro de borda. A filtragem das bordas horizontais para as amostras de croma começam pela leitura dos blocos 18, 19, 22 e 23 do macrobloco superior do Line Buffer, seguido pela leitura dos blocos 16, 17, 20 e 21 do macrobloco atual armazenado no MB Buffer e finalmente dos blocos 18, 19, 22 e 23. Após o processamento, os blocos 16 e 20 estão completamente filtrados e podem sair do filtro. Os blocos 17, 19, 21 e 23 são armazenados no MB Buffer através do processo Transpose e os blocos 18 e 22 são armazenados diretamente no Line Buffer.

Ao total, 256 ciclos de relógio são necessários para processar um macrobloco no perfil *Main* (24 blocos de 4x4 amostras). Se não há dados disponíveis para o início de um ciclo de operação (que comprende o ciclo de processamento de 1 macrobloco), uma bolha de 256 ciclos é inserida. Todos os *pipelines* são esvaziados durante estes 256 ciclos e a operação do filtro é interrompida até que um novo ciclo de processamento com dados disponíveis seja iniciado.

A Figura 4.9 ilustra a sequência de saída dos blocos para a arquitetura do filtro implementada. Para os 24 blocos correspondentes a um macrobloco no perfil *Main*, a ordem de saída é misturada com blocos dos macroblocos superior e à esquerda.

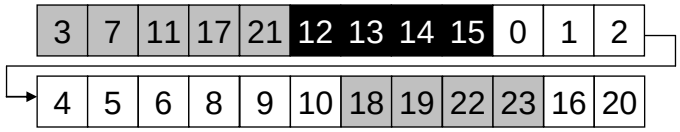

Figura 4.9: Sequência de blocos de saída do filtro.

4.4 Saída RGB

Para aplicações que necessitem de saída de vídeo RGB a partir do Filtro de Deblocagem, um módulo adicional de reordenamento e conversão do espaço de cores YCbCr para o formato RGB foi desenvolvido. A Figura 4.10 apresenta a arquitetura desenvolvida, que utiliza o Filtro de Deblocagem descrito anteriormente.

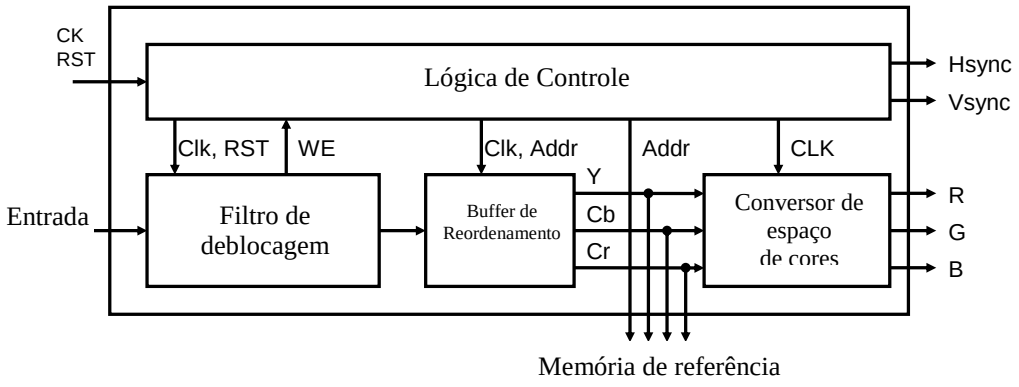

Figura 4.10: Saída RGB e para a memória de referência a partir do Filtro de Deblocagem desenvolvido.

Nesta arquitetura, o Filtro de Deblocagem é instanciado como um módulo interno. O Buffer de reordenamento é um buffer capaz de armazenar 1080x32 amostras de luma e 540x16 amostras para cada componente de croma, totalizando 51840 bytes. A implementação usa memórias RAM de porta dupla para possibilitar que enquanto os dados estejam chegando para serem reordenados, os dados reordenados possam estar saindo na taxa necessária para a exibição. A lógica de controle é capaz de regular o relógio do Filtro de Deblocagem de forma que a saída RGB possa estar na taxa correta de pixels, incluindo os períodos necessários para a sinalização do sincronismo horizontal (Hsync) e sincronismo vertical (Vsync) onde nenhum pixel é enviado para exibição. O conversor de espaço de cores é composto por um *pipeline* de 4 estágios que implementa a multiplicação de matrizes necessária para a conversão do espaço YCbCr para o RGB e ainda possui uma função de saturação na saída, de forma que valores maiores que 255 sejam saturados em 255 e valores menores que 0 sejam forçados para 0. Esta saturação é necessária, pois pequenas imprecisões aritméticas poderiam levar a condições de overflow ou underflow aritmético, o que leva ao aparecimento de artefatos claramente visíveis na imagem exibida.

Este conversor também possui uma saída YCbCr de forma que a memória de quadros de referência possa ser alimentada com pixels YCbCr já reordenados. Para aplicações onde se deseja usar quadros do tipo B (bidirecional) no perfil *Main*, modificações deverão ser feitas de forma a permitir a leitura dos pixels YCbCr da memória de referência, pois o processo de decodificação usando quadros do tipo B ocorre fora da ordem de exibição, exigindo a presença de um Frame Buffer para reordenamento.

4.5 Suporte ao perfís HIGH, SVC e MVC

A arquitetura desenvolvida originalmente é capaz de trabalhar apenas nos três perfis iniciais do H.264: *baseline*, *extended* e *main*. Os perfis HIGH, a escalabilidade (SVC) e multivisão (MVC) foram incluídos no padrão após o início da concepção arquitetural. Estes adendos ao padrão, entretanto, não significaram em mudanças importantes ao Filtro de Deblocagem e o suporte a eles foi possível com pequenas modificações arquiteturais e a parametrização da profundidade de cores das amostras (perfil HIGH10). O perfil HIGH 4:2:2 pode ser obtido a partir de pequenas modificações na máquina de controle, porém estas modificações serão realizadas em trabalhos futuros. O suporte ao SVC/MVC é possível através da codificação da entrada MB_TYPE do Filtro de Deblocagem, sem modificações na estrutura interna do Filtro. A arquitetura do filtro não permite a multiplexação da filtragem de diferentes slices simultaneamente, o que pode limitar seu uso em CODECs de vídeo que codificam ou decodificam parcialmente os slices. Esta limitação só se aplica no caso de uso de um filtro para múltiplas instâncias de codificação ou decodificação.

4.6 Implementação

A arquitetura proposta para o Filtro de Deblocagem foi descrita em VHDL. Aproximadamente 4100 linhas de código foram escritas para o Filtro de Deblocagem, incluindo o módulo de saída RGB, apresentado na Figura 4.10. O projeto foi validado comportamentalmente por simulação usando dados extraídos do software de referência do H.264. Após o projeto sintetizado, a *netlist post-place&route* incluindo os atrasos da implementação foram novamente validados por simulação, usado dados extraídos do

software de referência do padrão, e finalmente o resultado foi sintetizado para um FPGA Virtex2-pro.

Figura 4.11: Saída do Filtro de Deblocagem para um quadro da sequência FOREMAN_QCIF, codificada como Intra.

A Figura 4.11 apresenta os resultados de prototipação realizada através da metodologia apresentada do Capítulo 6, usando dados extraídos do software de referência antes da filtragem e após ser filtrado usando a arquitetura proposta.

4.7 Resultados em FPGA

Como passo inicial de validação da implementação da arquitetura proposta foi utilizada uma placa FPGA e a estratégia de prototipação descrita no Capítulo 6. A Tabela 4.1 apresenta os resultados de síntese da arquitetura em FPGA para dois dispositivos da Xilinx pertencentes a famílias diferentes: o Virtex II pro XC2VP30 (VIRTEX, 2008) e o Virtex 5 XC5VLX30 (VIRTEX, 2008a).

Observe a similaridade entre a quantidade de lógica e de memória utilizada neste projeto. Com uma frequência de operação de 148 MHz no dispositivo Virtex II pro, esta implementação é 2,36 vezes mais rápida do que o necessário para HDTV. Para o dispositivo Virtex-5, a velocidade de processamento é 3,14 vezes maior que o requerido para HDTV. Com esta taxa é possível atingir resoluções tão altas quanto 4000x2000 pixels (nível 5.2 do perfil *Main*), em ultra-alta definição ou ainda para processar múltiplos fluxos de vídeo HDTV simultaneamente, para aplicações de multivisão.

Tabela 4.1: Resultados de síntese em FPGA para o Filtro de Deblocagem.

FPGA	XC2VP30	XC5VLX30
LUTs	4008/27392 (14%)	4275/19200 (21%)
BRAMs	20/136 (14%)	7/32 (21%)
Fmax (MHz)	148	197
FPS@1080p	71	95

Ferramenta de síntese: ISE 9.1i

Utilizando-se o Filtro de Deblocagem com a saída RGB os resultados são apresentados na tabela 4.2. Devido à necessidade de reordenamento, a arquitetura proposta para a saída RGB consumiu significativamente mais blocos de memória no FPGA e também promoveu uma redução na frequência máxima atingível. Ainda assim

a frequência máxima alcançada é bastante elevada, atingindo 65 quadros por segundo, mais que o dobro do necessário para suportar HDTV no padrão 1080p.

Tabela 4.2: Resultados de síntese para o Filtro de Deblocagem incluindo o módulo de saída RGB.

FPGA	Xilinx XC2VP30
LUTs	4331/27392 (17%)
BRAMs	65/136 (47%)
Fmax (MHz)	135
FPS@1080p	65

Ferramenta de síntese: ISE 9.1i

A Tabela 4.3 apresenta os resultados de síntese FPGA para o Filtro de Deblocagem para o perfil HIGH10/SVC/MVC.

FPGA	XC2VP30-7		XC5VLX30-3	
	Main	High10	Main	High10
LUTs	4117 (15%)	4900 (17%)	2586	3067
BRAMs	7(5%)	7 (5%)	4(12%)	5(15%)
Fmax (MHz)	195	194	342	341
FPS@1080p	93	92	162	162

Ferramenta de síntese: ISE 10.1i

Em função da mudança da ferramenta ISE da versão 9.1 para a versão 10.1, os resultados de síntese para o perfil high foram significativamente diferentes. Para melhor comparação, a síntese para o perfil Main (amostras de 8 bits) foi refeita nesta ferramenta. Os resultados comparativos mostram que o suporte a amostras de 10 bits gerou um aumento no uso de LUTs de 19% tanto para o dispositivo XC2VP30 quanto para o XC5VLX30. A frequência de operação sofreu uma redução de menos de 1% em ambos os dispositivos da versão High10 em relação à Main. Em todos os casos, a ferramenta de síntese está configurada para obter a maior frequência de *clock* possível, o que explica a pequena variação de frequência e grande variação de área.

4.8 Resultados em ASIC

Após a validação FPGA foi realizada a implementação em ASIC usando o fluxo de projeto em Standard Cells da Cadence. A Tabela 4.4 apresenta um sumário das especificações e resultados obtidos. O fluxo de projeto utilizado envolve a definição de uma frequência alvo de operação para o circuito. As memórias do Filtro de Deblocagem não foram sintetizadas neste resultado, pois precisavam ser configuradas individualmente e esta tarefa ficou para um momento posterior. Com esta restrição de projeto, as ferramentas do fluxo produziram um circuito em Standard Cells com 18,7 mil portas lógicas equivalentes (NAND de menor área). Utilizando vetores aleatórios como entrada, a potência consumida pelo circuito, quando operando a 200MHz foi de

17,5mW. Nesta frequência, o filtro é capaz de processar aproximadamente 3,2 vezes mais que o requisto para a resolução 1080p (1920x1080x30).

Tabela 4.4: Espeficiação da tecnologia ASIC utilizada e resultados de implementação.

Tecnologia	TSMC 0,18µm CMOS 6ML
Tensão	1,8V
Area	0,27mm^2
Portas lógicas	18,7 mil
Frequência de Operação (especificada)	200MHz
Potência em 200MHz	17,5mW

A Figura 4.12 ilustra o circuito roteado por Silva (2010) usando a descrição RTL desenvolvida pelo autor deste trabalho. Observe que as memórias não estão presentes neste circuito, apenas a lógica.

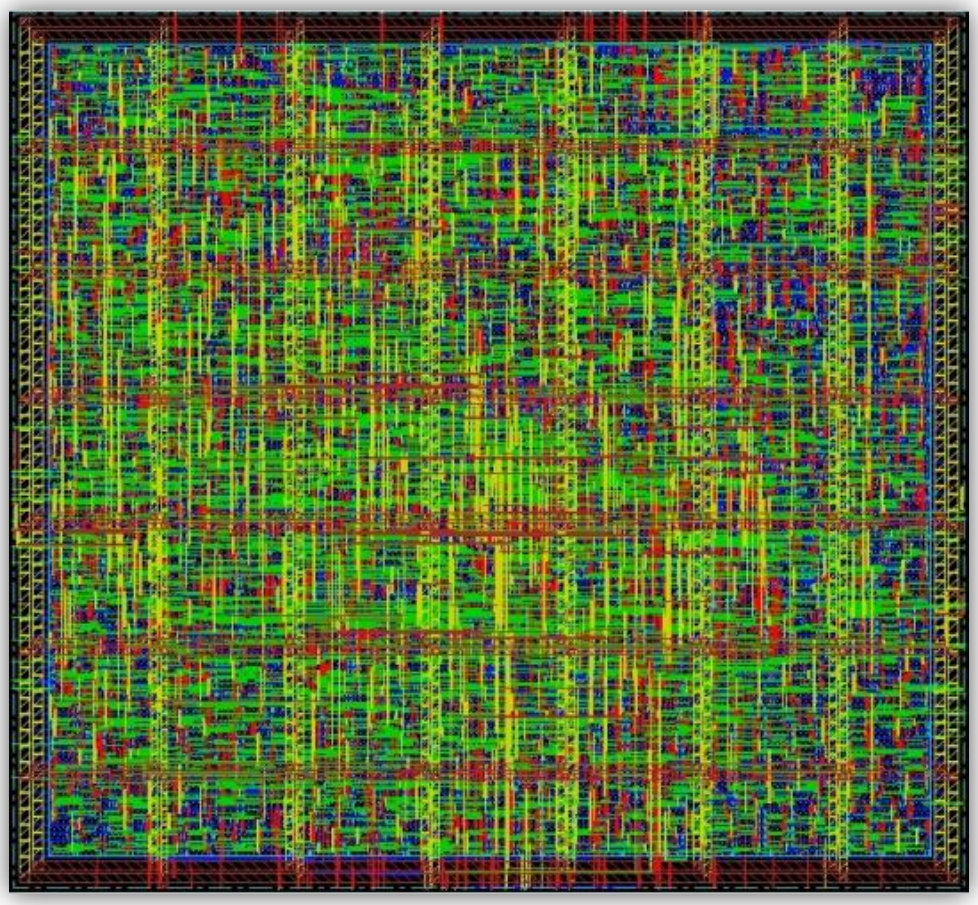

Figura 4.12: Layout do Filtro de Deblocagem, sem memórias (SILVA, 2010).

Após a verificação pós-síntese do Filtro de Deblocagem sem as memórias, estas foram parametrizadas e incluídas no circuito. A Figura 4.13 ilustra o layout do circuito após a inclusão das memórias.

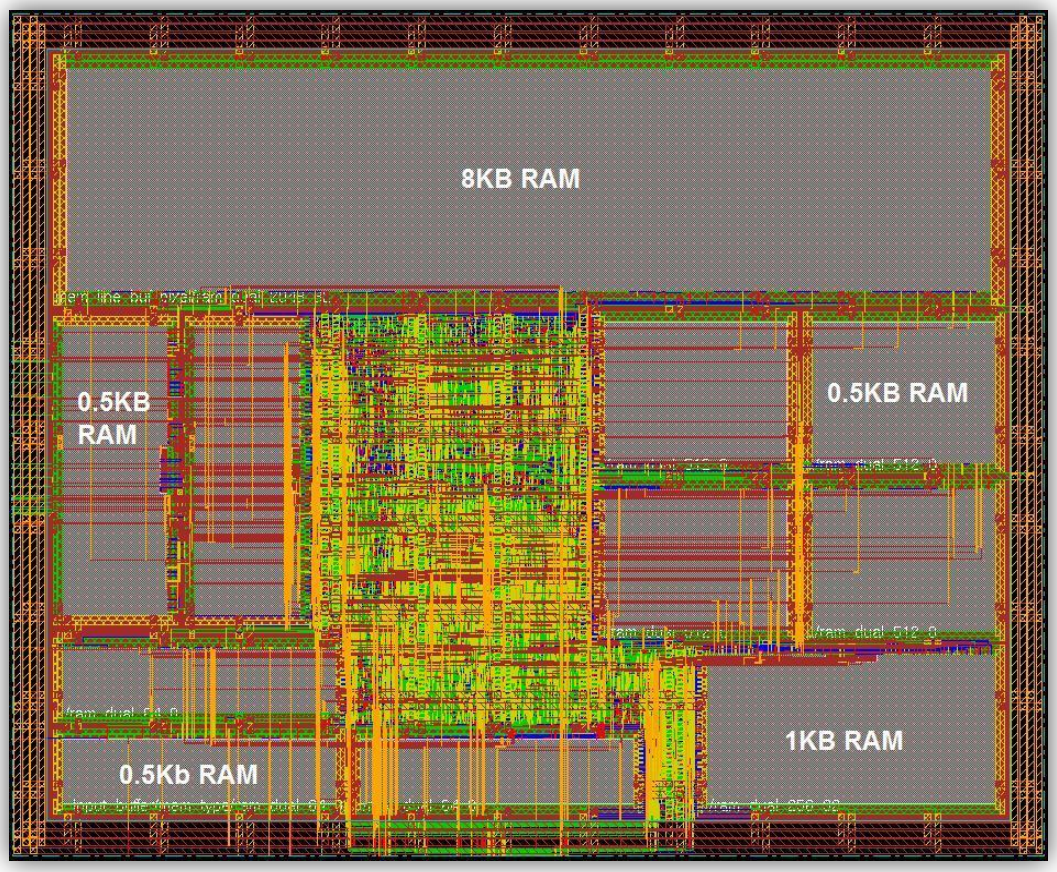

Figura 4.13: Layout final do Filtro de Deblocagem (SILVA, 2010).

Com a inclusão das memórias, totalizando 12,4KB de SRAM, frequência de operação teve que ser reduzida para 100MHz e a área ocupada pelo circuito subiu para 4,6mm2 (2,4x1,9 mm). A potência subiu para 42mW. A maior memória (8KB) é usada para módulo "*Line Buffer*" e é capaz de suportar vídeo com largura de até 2048 pixels (um pouco superior ao exigido pelo formato 1080p, que é 1920). O tamanho dessa memória é proporcional à largura do quadro e não pode ser reduzida sem a redução do tamanho máximo do quadro admitido. As demais memórias, entretanto, podem ser reduzidas através de um trabalho de otimização arquitetura, como apresentam alguns trabalhos publicados posteriormente ao desenvolvimento arquitetural deste.

4.9 Comparação com a literatura

A comparação de trabalhos que apresentam propostas arquiteturais para o Filtro de Deblocagem é difícil, pois muitos trabalhos não apresentam dados completos sobre seus resultados. Em Sima (2004) é citado que a validação foi realizada em FPGA, mas não é fornecido maiores informações sobre o processo de validação. Além disso, nenhum trabalho apresenta dados sobre as ferramentas de síntese empregadas e, como já foi visto na geração dos resultados em FPGA, mesmo a mudança de versão da ferramenta de síntese afeta significativamente os resultados produzidos. A Tabela 4.5 apresenta alguns trabalhos selecionados para comparação.

Tabela 4.5: Resultados ASIC sem memórias e comparação com a literatura.

Parametro	Este trabalho	Huang (2003)	Sheng (2004)	Khurana (2006)
Tecnologia (µm)	0,18	0,25	0,25	0,13
Portas Lógicas Equivalentes (mil)	18,7	20,7	24	7,5
Frequência (especificada - MHz)	200	100	100	200
Ciclos/MB	256	614	446	192

A arquitetura desenvolvida para este trabalho foi projetada para gerar o maior desempenho quando sintetizada em FPGA, por isso a proposta arquitetural engloba um *pipeline* de 16 estágios. Em FPGA, onde cada LUT contém o seu flip-flop, o *pipeline* profundo traz o benefício de aumentar a frequência de operação sem aumentar significativamente a área. O mesmo não acontece na implementação ASIC, onde cada elemento de lógica existente na descrição do hardware é implementado. Apesar do número de portas estar abaixo de duas publicações apresentadas (HUANG, 2003, SHANG, 2004), ela é mais de duas vezes maior que o resultado de Khurana (2006). Entretanto, este último foi sintetizado para a mesma frequência deste trabalho, porém em uma tecnologia menor (0,13µm contra 0,18µm), o que justifica parte da redução em área. Apesar disso, a arquitetura de Khurana (2006) consegue processar um macrobloco no menor número de ciclos (192) dentre os trabalhos comparados. As versões deste trabalho que suportam os perfis HIGH/SVC/MVC e a saída RGB não foram sintetizados para ASIC e não serão comparadas com outros trabalhos.

4.10 Conclusões

A arquitetura proposta para o Filtro de Deblocagem é distinta da maioria dos trabalhos da literatura por considerar a integração das memórias necessárias para o armazenamento dos dados intermediários da filtragem. Este trabalho foi focado no desenvolvimento de uma arquitetura que pudesse ser rapidamente integrada no protótipo de um decodificador ou codificador de vídeo em FPGA. Por isso, o foco inicial do desenvolvimento deste trabalho foi a otimização para síntese em FPGA.

A inclusão das memórias *Line Buffer* e *MB Buffer* na arquitetura resultou em uma utilização harmônica dos recursos do FPGA, sem acréscimo na lógica. A síntese ASIC mostrou que o desempenho é compatível com outras implementações da literatura, entretanto as memórias são o principal gargalo em área para a o Filtro de Deblocagem. Apesar disso, é possível superar o requisito de processamento para vídeos HDTV (1080p) para ambas as implementações (FPGA e ASIC).

A arquitetura proposta foi concebida para trabalhar apenas com vídeo no formato progressivo até a resolução horizontal de 2048 pixels no formato 4:2:0 com amostras a partir de 8 bits. Trabalhos futuros irão considerar o suporte a outros formatos de vídeo suportados pelos perfis *high* do padrão H.264, assim como vídeo entrelaçado codificado com suporte a MBAFF. Otimizações arquiteturais para a otimização do uso das memórias e avaliação do desempenho do uso de memórias externas para substituir as maiores memórias poderão ser avaliados.

5 ARQUITETURAS PARA O CODIFICADOR E DECODIFICADOR ARITMÉTICOS DO CABAC

O codificador de vídeo H.264 suporta dois tipos de codificação de entropia, o CAVLC (*Context Adaptive Variable-lenght Coder*) e o CABAC (*Context Adaptive Binary Arithmetic Coder*), como descritos no Capítulo 3 deste trabalho. Neste capítulo, serão apresentados detalhes do projeto de arquiteturas para o codificador e decodificador CABAC (neste trabalho denominado apenas CABAC quando se tratar do codificador e CABAD do decodificador) do padrão H.264 e o desenvolvimento da implementação lógica dos codificadores aritméticos para o CABAC e o CABAD.

O CABAC é um codificador de entropia baseado em um codificador aritmético binário adaptativo ao contexto e representa uma das inovações do H.264. Ele alcança uma taxa de compressão cerca de 15% superior ao CAVLC, que também é uma inovação do H.264, sendo um método mais eficiente de codificaçãoe que o empregado em padrões anteriores (MARPE, 2003). O CABAC é opcional nos codificadores, mas obrigatório nos decodificadores aderentes aos perfis *Main* e *High* (ITU, 2005) do padrão H.264 e deve ser implementado em um decodificador que seja aderente ao padrão nestes perfis.

Uma das principais dificuldades do algoritmo do CABAC é que ele é essencialmente sequencial, e cada passo de iteração depende estritamente do passo anterior, sendo apenas 1 bit produzido e/ou consumido por passo do algoritmo (WIEGAND, 2003). Esta natureza sequencial de operação do CABAC está presente tanto na codificação quanto na decodificação (CABAD) e muitos trabalhos na literatura tentam superar esta limitação do algoritmo tentando encontrar formas de extrair paralelismo tanto no codificador quanto no decodificador.

Um estudo no sentido de identificar as arquiteturas de maior desempenho para o CABAC/CABAD foi realizado, com o objetivo de encontrar uma arquitetura capaz de suportar as taxas de processamento necessárias para HDTV para uso em um decodificador H.264. Além disso, uma análise estatística foi realizada para se identificar os perfis de uso dos motores de decodificação do CABAD, o que levou a proposta de uma nova arquitetura.

Nas seções seguintes serão apresentados os detalhes do funcionamento do CABAC, propostas arquiteturais para os codificadores aritméticos para o CABAD e CABAC, incluindo uma arquitetura para o codificador CABAC completo, assim como resultados de síntese e comparação com trabalhos relacionados.

5.1 Visão geral do CABAC

O Codificador Binário Aritmético Adaptativo ao Contexto definido pelo padrão H.264/AVC é uma ferramenta para a codificação de entropia que transforma os valores

de um símbolo em uma palavra de código com tamanho variável próximo ao limite teórico de entropia (MARPE, 2003). O codificador aritmético binário funciona pela divisão recursiva de intervalo, combinada por modelos de contexto que permitem alcançar uma melhor eficiência de codificação. Cada sub-intervalo representa um único símbolo da fonte e o tamanho do intervalo é proporcional a probabilidade de ocorrência do símbolo. A modelagem das probabilidades de ocorrência de cada símbolo leva a uma maior complexidade computacional, que no CABAC é reduzida através do uso de um alfabeto binário (MARPE, 2003).

Desta forma, apenas símbolos binários poderão ser codificados. Todo e qualquer elemento sintático (SE – *Syntax Elements*) que possuir informação não binária deve inicialmente passar por um processo de binarização para produzir um conjunto de elementos binários chamados *bins*. Estes *bins* passarão por um processo de modelagem de contextos para que sua probabilidade de ocorrência seja estabelecida e atualizada, de forma que o codificador aritmético consiga atingir os níveis de entropia necessários. A Figura 5.14 ilustra o fluxo básico de dados do CABAC.

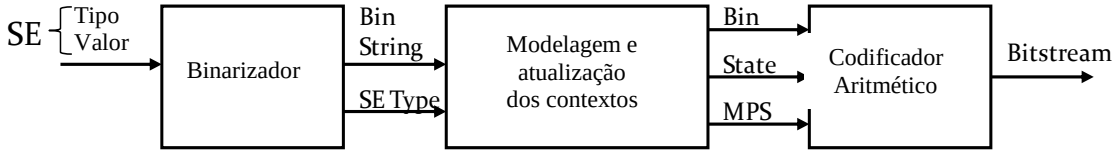

Figura 5.14: Fluxo de dados no CABAC.

A entrada do codificador de entropia no padrão H.264 são os elementos sintáticos produzidos pelas demais ferramentas de compressão, o que inclui informações de bloco, vetores de movimento e resíduos quantizados. Cada elemento sintático é identificado por um tipo e possui um valor. A Figura 5.1 ilustra a origem dos elementos sintáticos em um codificador H.264. O processo de binarização é dependente do tipo de elemento sintático e a saída é uma string de tamanho variável de valores binários, denominados *bins*. Então, contextos associados ao tipo de elemento sintático e também a posição do *bin* no string que compõe o elemento sintático binarizado é utilizado para selecionar o contexto que será empregado para a codificação daquele *bin*. Por esta razão, diferentes *bins* pertencentes a um mesmo elemento sintático podem possuir informações de contexto diferentes. O *bin* e a informação de contexto são então usados como entrada para o codificador binário aritmético e a saída é um contexto atualizado e zero ou mais bits, produzidos pelo processo de renormalização característico do codificador aritmético binário empregado no CABAC.

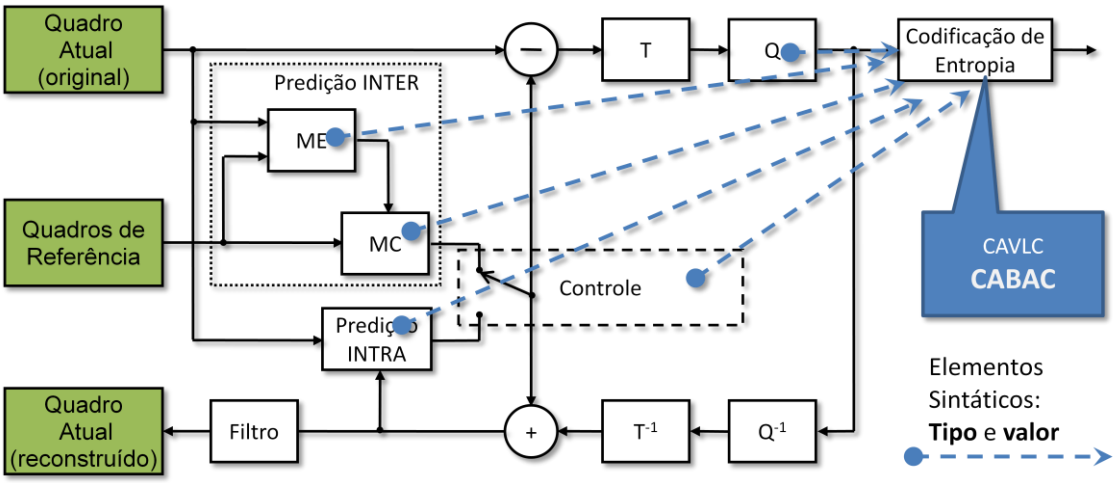

Figura 5.1: Origem dos elementos sintáticos.

A complexidade do CABAC é similar no codificador e no decodificador. O processo de decodificação é comumente chamado de CABAD (*Context-Adaptive Binary Arithmetic Decoder* – decodificador aritmético binário adaptativo ao contexto), que faz o caminho inverso do CABAC, produzindo como saída elementos sintáticos (SE – *Syntax Elements*).

5.1.1 O processo de binarização e debinarização

O processo de binarização consiste em mapear os valores dos elementos sintáticos (SE) em uma única sequência de bits que representa o valor original. Este mapeamento é realizado para reduzir os símbolos no alfabeto a ser codificado, simplificando a quantidade de elementos a serem modelados, minimizando os custos desta modelagem e facilitando a tarefa do codificador aritmético.

Tabela 5.1: Exemplos do processo de binarização.

Valor	***Binstring* para alguns métodos de binarização**			
	Unário	**Unário Truncado (cMax=4)**	**Tamanho Fixo (cMax=4)**	**Exp-Golomb**
0	0	0	0000	0
1	10	10	0001	100
2	110	110	0010	101
3	1110	1110	0011	11000
4	11110	1111	0100	11001

Cada bit neste processo é denominado *bin* e o conjunto de todos os *bins* é denominado "*string* de *bins*" ou *binstring*. O tamanho da *string* de *bins* gerado por cada valor de entrada e variável e depende do tipo de SE que é processado e valor. O padrão define um total de 7 métodos de binarização, sendo 4 fundamentais (unário, unário truncado, tamanho fixo e Exp-Golomb) e outros 3 que são formados por combinações dos fundamentais (ITU, 2003). A Tabela 5.1 ilustra 5 dos métodos utilizados para esta finalidade.

Os métodos apresentados na Tabela 5.1 são métodos algoritmos para a conversão de valores não-binários em strings de *bins*. Existem também métodos baseados na consulta de tabelas, como o método utilizado para decidir o tamanho da partição de um macrobloco.

Vale destacar que para alguns elementos sintáticos, como, por exemplo, o indicador do modo *Skip* de codificação, o valor do elemento sintático já é binário (é ou não é skip). Neste caso, o processo de binarização não se aplica.

5.1.2 Modelagem de probabilidade

Um contexto é um modelo probabilístico que representa a distribuição estatística de um símbolo em particular com base na ocorrência dos símbolos processados previamente. Ele tenta identificar a probabilidade de ocorrência do símbolo que será codificado, que leva a uma codificação mais eficiente.

Uma característica especial dos codificadores aritméticos, como o CABAC do H.264, é que a modelagem de probabilidade de ocorrência de um determinado símbolo é completamente distinta do processo de codificação. Para modelar adequadamente a probabilidade de ocorrência de cada símbolo associado a cada elemento sintático, o padrão H.264 inicialmente definiu 460 contextos (para o perfil *Main*) (ITU, 2003), chegando a um total de 1031 contextos quando o suporte a escalabilidade e multi-visão são implementados (ITU, 2009). Cada *bin* em um determinado elemento sintático pode estar associado a um ou mais contexto. Em todo o processo de codificação, as estimativas probabilísticas precisam ser mantidas atualizadas para garantir a eficácia do processo.

Cada contexto é uma palavra de 7 bits que armazena dois valores distintos: um valor de estado de 6 bits que é usado como índice da probabilidade (há uma tabela de probabilidades) e um valor binário (1 bit) que define o símbolo mais provável (MPS), que pode ser 0 ou 1.

Por razões de otimização de velocidade de processamento, o padrão H.264/AVC não prevê a modelagem de contextos para alguns *bins* em que a probabilidade de ocorrência foi estabelecida como próxima de 0,5; a modelagem de contexto neste caso não traria ganho adicional de codificação. Entretanto, estes *bins* não podem ser considerados bits e enviados diretamente ao *bitstream*, pois o decodificador não teria como identificar se a informação presente no *bitstream* passou ou não pelo processo de codificação aritmética. Para resolver este problema, um modo especial de codificação aritmética que não envolve a utilização de contextos foi desenvolvido. Este modo é significativamente mais simples do que o modo regular de codificação, gerando uma redução de complexidade no processo de codificação.

Para aumentar o ganho de codificação, o CABAC prevê a incialização dinâmica do valor dos contextos (ITU, 2009). Esta inicialização é realizada antes de codificar o primeiro *bin* de cada novo *slice* e leva em consideração o tipo de *slice* (I ou outro), um parâmetro de codificação (Init_IDC) que permite a seleção de três valores distintos (apenas para quadros não-I) além do QPY inicial, que é o valor do parâmetro de quantização que será utilizado para codificar as amostras de luminância do primeiro macrobloco do *slice*.

A inicialização dos contextos ocorre consultando um conjunto de tabelas que fornecem dois parâmetros “m” e “n” de uma reta que será utilizada para o cálculo do valor inicial do contexto com base no QPY (o QPY é o domínio e o valor do contexto é

a imagem da função). Para garantir que esta função não extrapole os limites permitidos para o valor do contexto, algumas operações de saturação são executadas durante o processo de cálculo. O algoritmo para o cálculo do valor de um contexto com base nos parâmetros "m" e "n" é apresentado no quadro da Figura 5.2, conforme extraído de ITU (2009).

```
preCtxState = Clip3( 1, 126, ( ( m  * Clip3( 0, 51, QPY ) ) >> 4 ) + n )
if( preCtxState  <=  63 )  {
    pStateIdx = 63 - preCtxState
    valMPS = 0
} else {
pStateIdx = preCtxState - 64
valMPS = 1
```

Figura 5.2: Algoritmo para cálculo dos valores iniciais do contexto

A função Clip3(***Min***, ***Max***, ***Val***) retorna o valor de ***Val*** saturado com mínimo em ***Min*** e máximo em ***Max***.

5.1.3 O codificador aritmético binário (BAC)

O BAC (*Binary Arithmetic Coder* – Codificador Aritmetico Binário), no codificador, e o BAD (*Binary Arithmetic Decoder* – Decodificador Binário Aritmético), no decodificador) trabalham com base no princípio da subdivisão recursiva de um intervalo com tamanho R, como proposto por (WITTEN, 1987). A partir da estimativa de probabilidade para o LPS (*least probable symbol* – símbolo menos provável) em uma determinada faixa, dois sub-intervalos são obtidos. O primeiro é dado por rLPS = R*pLPS, o qual é associado com LPS enquanto o segundo é dado por rMPS = R-rLPS, o qual é associado com o MPS. De acordo com o *bin* codificado, o rMPS ou o rLPS é escolhido como o novo intervalo *R*. Durante o processo de codificação aritmética binária dois registradores, o de faixa (*R* - *Range*) e o de deslocamento ou base (O *Offset* ou *L Low*) precisam ser atualizados. O *R* armazena o valor do intervalo atual enquanto o segundo marca o limite inferior dentro deste intervalo.

No CABAC do H.264, o intervalo unitário é composto por 1024 valores discretos possíveis. Este intervalo contém a informação codificada. A Figura 5.3 apresenta um exemplo da condição inicial do intervalo e após a execução de alguns passos da codificação aritmética.

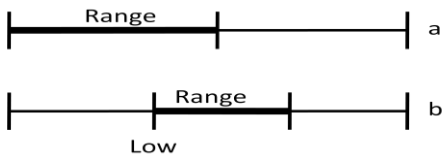

Figura 5.3: Intervalo inicial (a) e após vários passos de codificação (b).

Diferentemente do que ilustra a Figura 3.19, que foi concebida apenas para o entendimento do processo de codificação aritmética, no H.264 o intervalo unitário está dividido em 1024 posições (de 0 a 1023) por razões práticas de implementação. O trabalho de Marpe (2003) apresenta detalhes que justificam a decisão desta faixa de valores discreta. Além disso, o valor do intervalo *R* deve sempre permanecer entre 256 e 511 e o valor da base do intervalo (*L*) entre 0 e 511. Por esta razão, a ilustração do intervalo apresentada na Figura 5.3 apresenta o mesmo tamanho de intervalo após

vários passos de codificação. Para que isto ocorra, o codificador aritmético deve passar por um processo chamado renormalização a cada vez que um *bin* for codificado.

O CABAC é capaz de codificar de forma eficiente símbolos com probabilidade superior a 0,5. No CAVLC, um símbolo com probabilidade maior que 0.5 sempre ocupará pelo menos 1 bit no *bitstream*. No caso da codificação aritmética vários símbolos com probabilidade maior que 0,5 poderão ser codificados sem que um bit seja produzido no *bitstream*.

O processo de codificação aritmética envolve a divisão do intervalo representado por *L* e *R*. O processo de divisão do intervalo exige a multiplicação do *R* pela probabilidade de ocorrência do LPS. Para reduzir a complexidade computacional, o *R*, originalmente representado por 9 bits (que possui valores no intervalo [256,511]), é re-quantizado para apenas 2 bits, a partir dos dois bits mais significativos do *R*. Desta forma, o rLPS, que é o resultado da multiplicação, pôde ser pré-armazenada em uma tabela fixa de 64x4, indexada pelos 6 bits de estado obtidos do modelo de contexto e *R* re-quantizado. O estudo realizado por Marpe (2003) mostrou que perda de capacidade de compressão com esta modificação é desprezível.

O processo de divisão do intervalo irá reduzir o tamanho do intervalo toda vez que um *bin* for codificado, não importando sua probabilidade. Para prevenir a redução do intervalo além do limite de resolução permitido pela faixa dinâmica limitada utilizada para a representação do intervalo, toda vez que um *bin* é codificado, um processo de renormalização é realizado para garantir que o tamanho do intervalo sempre esteja dentro de um limite entre 256 e 511. Esta redução de intervalo sempre ocorre, pois a probabilidade de ocorrência de um *bin* sempre será menor do que a unidade e a multiplicação do intervalo por um valor menor que a unidade irá produzir uma redução no seu tamanho. Entretanto, quanto maior for a probabilidade, menor será a redução do tamanho do intervalo e menor será o número de vezes que o processo de renormalização terá que atuar durante o processo de codificação de todos os *bins* que compõe o quadro de vídeo. Toda vez que o processo de renormalização necessita modificar o tamanho do intervalo, um bit é enviado ao *bitstream*. Por isso, é vantajoso renormalizar o menor número de vezes possível (através de uma modelagem adequada da probabilidade de ocorrência dos *bins*) quando o objetivo é a compressão.

Para que o processo de codificação possa ser finalizado corretamente, um modo especial de codificação aritmética também é utilizado para encerrar o processo de codificação. Este modo é chamado de modo final (*Terminate*) e ocorre sempre que o processo de codificação aritmético precisa ser finalizado.

O Fluxograma da Figura 5.4 ilustra o processo de codificação aritmética para um *bin*.

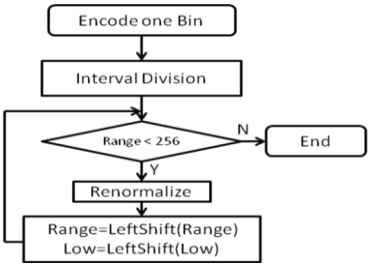

Figura 5.4: Processo de codificação aritmética para um *bin*.

A codificação do *bin* inicia com a divisão de intervalo que reduz o valor do *R* (Range) dependendo da probabilidade de ocorrência do valor do *bin*. O processo de divisão de intervalo é apresentado no fluxograma da Figura 5.5. Após a divisão do intervalo pelo processo de codificação do *bin*, um loop da renormalização irá ocorrer até que o valor do R se tornar maior ou igual a 256.

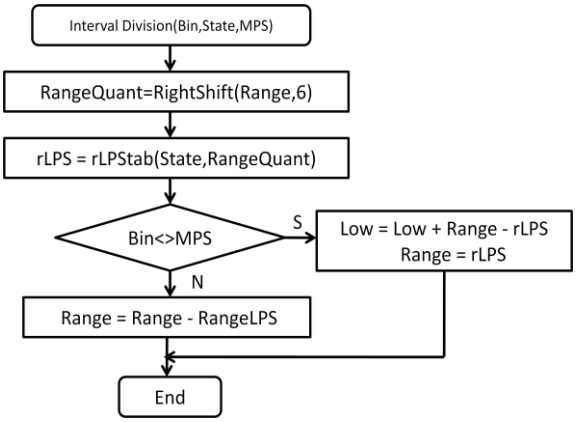

Figura 5.5: Processo de divisão de intervalo.

Toda vez que a renormalização é realizada, um bit é enviado ao *bitstream*. O valor deste bit depende da consulta ao registrador de base do intervalo (*L* – Low). Se este registrador (marcador de início do intervalo) se torma maior que 511, um bit de valor 0 é enviado ao *bitstream*. Se o valor do *L* for menor do que 256, então um bit de valor 1 é enviado ao *bitstream*. Toda vez que a renormalização ocorre com a produção de um bit, o intervalo é multiplicado por dois de forma a superar o problema da produção de um valor infinitesimal para representar o intervalo, como fica evidente na Figura 3.19. Os bits emitos para o *bitstream* são a parte mais significativa de um número que representa um ponto contido no intervalo *R* que será usado no processo de decodificação. Existe um caso especial que ocorre sempre que ocorre um valor do *L* entre 256 e 511: neste caso é impossível decidir o valor do bit a ser enviado ao *bitstream*. Isto ocorre porque a operação aritmética de geração do *bitstream* ocorre através de sucessivas adições de valores de 9 bits deslocados N-passos-de-renormalização à direita de forma a compor um número com um grande número de dígitos chamado *bitstream*. Em algumas situações irá ocorrer o "vai-um" que irá modificar o valor de dígitos já calculados, à esquerda do bit mais significativo da operação atualmente sendo realizada (9 bits), necessitando uma atualização no valor destes. No codificador aritmético do CABAC estes bits são chamados de *outstanding bits*. No algoritmo da Figura 5.6, um contador é usado para marcar a quantidade de bits *outstanding* que são produzidos em sequência durante o processo de renormalização. Uma vez que um bit sabidamente 0 ou 1 é produzido, a condição de "vai-um" é resolvida e todos os bits que dependem desta decisão, marcados como *outstanding*, são decididos e enviados ao *bitstream*. No algoritmo da Figura 5.6 isso é resolvido pela chamada do processo "PutBit" duas vezes de forma subsequente. Na primeira para enviar os bits *outstanding* já resolvidos e na segunda para enviar o bit codificado que gerou a decisão.

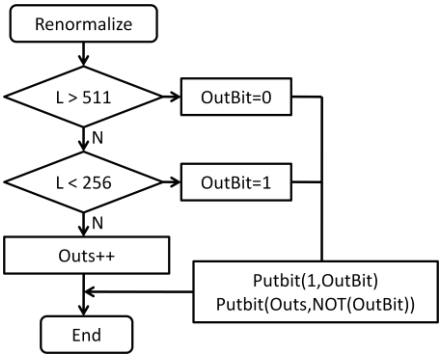

Figura 5.6: Processo de renormalização.

Como dito anteriormente, um método especial de codificação de probabilidade para os *bins* determinados pelo padrão como tendo probabilidade próximo a 0,5 é usado. Esse método é a codificação *Bypass*. O algoritmo para este processo de codificação é ilustrado pelo fluxograma da Figura 5.7.

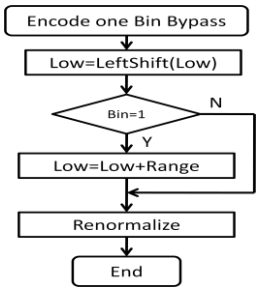

Figura 5.7: Processo de codificação Bypass.

Finalmente, na Figura 5.8, é ilustrado o algorimo de inserção de bits no *bitstream*.

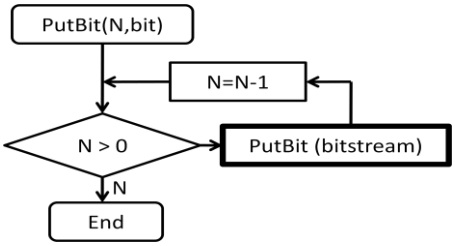

Figura 5.8: Processo de inserção de bits no *bitstream*.

No codificador de vídeo H.264/AVC, a taxa de compressão típica alcançada pelo codificador aritmético é de 1,3 *bins* (DEPRÁ, 2008) processados para cada bit produzido (podendo chegar a 2,8 *bins* por bit em algumas sequências). Desta forma, a maior parte da compressão irá ocorrer durante o processo de binarização e a capacidade de compressão do codificador aritmético depende fortemente do processo de modelagem de contextos empregada (definida no padrão H.264).

5.1.4 O decodificador aritmético binário (BAD)

O processo do BAD, da mesma forma que o BAC, envolve uma série de ações que ocorrem abaixo da camada de *slice*. O padrão H.264 estabelece que o CABAD é reiniciado a cada novo *slice*. No inicio de um novo *slice*, a tabela de contexto é construída a partir de um algoritmo de cálculo das probabilidades baseado em tabelas

iniciais que dependem do tipo de *slice*. Após esta etapa, o CABAD inicializa as variáveis *CodlOffset* (os nomes derivam do software de referência do padrão - JM (SUHRING, 2008)), lendo os primeiros 9 bits a partir do *bitstream* codificado e a variável *CodlRange* assume o valor padrão. A decodificação pelo CABAD de SEs da camada de macroblocos é realizada até que um elemento sintático denominado fim de *slice* (EOS – *end of slice*) seja encontrado. O primeiro passo para a decodificação dos SEs é a decisão de seu tipo e, baseado nesta informação, o método de debinarização é escolhido. Após isso, se o tipo de SE é um EOS, o processo de decodificação terminal é selecionado. Caso contrário, para cada *bin* de um SE, um ou dois outros processos são selecionados: o regular e o *bypass*. Para os *bins* sendo decodificados pelo processo regular, um endereço na tabela de contextos tem que ser realizado. A informação obtida da tabela de contextos inclui o MPS e o índice na tabela de probabilidade estimada, denominada variável *pState*. O CABAD usa um *offset* no índice da tabela de contextos fixo para cada tipo de SE para gerar o índice para um determinado *bin*. Para alguns SEs, obter o índice de incremento envolve referenciar o SE pertencente ao macrobloco à esquerda e acima do macrobloco atualmente sendo processado e para outros tipos de SE o índice do *bin* na binstring é usado para este propósito.

Para *bins* que usam o processo de decodificação regular, o CABAD obtém um novo *rLPS* de uma tabela indexada por *pState* e então quatro possíveis valores são selecionados pelo valor de *CodlRange* (bits 7 e 6). Então um novo valor de *CodlRange* é calculado e a comparação entre *CodlRange* e *CodlOffset* define se a ocorrência é do MPS ou do LPS. Após isso, a tabela de contextos deve ser atualizada com o novo valor de MPS e *pState*, que são obtidos de uma tabela de constantes com as transições de estados com valores diferentes para a ocorrência de MPS ou LPS, aumentando ou reduzindo a probabilidade de ocorrência baseado na ocorrência atual. A seguir, os registradores *CodlRange* e o *CodlOffset* passam pelo processo de normalização. Neste caso, um ou mais bits do *bitstream* podem ser consumidos. Finalmente, um passo do processo de decodificação regular é finalizado com as variáveis do ambiente de decodificação atualizadas.

Para outros *bins* (sufixos de MVD e nível de coeficiente de transformada), o processo de decodificação *bypass* é aplicado. O modo *bypass* é significativamente mais simples que o regular. Os *bins* no modo *bypass* não usam contextos, eliminando acessos e atualizações na memória de contextos e reduzindo a profundidade lógica.

Finalmente o processo de debinarização é realizado e os SEs estão prontos para serem usados pelos demais módulos do decodificador H.264.

5.1.5 Paralelização acima do nível de *Slice* para o CABAC e CABAD

Há uma forma de paralelização do CABAC e CABAD que raramente é mencionada na literatura relacionada ao codificador e decodificador de entropia do padrão H.264 que consiste em colocar diversos codificadores e decodificadores operando em paralelo no nível de *slice*. Isto é possível pois todo o estado do codificador e, da mesma forma, do decodificador CABAC é reiniciado a cada novo *slice*. Isso significa que não há dependência de dados no processo de codificação de entropia acima da camada de slice e, se múltiplos quadros estiverem aguardando a codificação de entropia, o uso de vários codificadores em paralelo pode ser uma alternativa rápida e viável de obter maior desempenho do processo de codificação de entropia.

O uso de múltiplas instâncias do CABAC/CABAD pode levar a economia de energia, se estes forem ligados sob demanda, favorecendo implementações de baixa potência.

5.2 Arquitetura para o decodificador aritmético binário adaptativo ao contexto (CABAD)

Os motores do BAD (*Regular*, *Bypass* e *Terminate*) fazem parte do núcleo do CABAD. Eles são responsáveis por regenerar o *binstring* a partir do *bitstream* e dos contextos associados aos *bins*. Considerando que o processo de decodificação aritmética é realizado um *bin* por vez, os motores de decodificação devem possuir alto desempenho para dar conta da produção de *bins* na taxa de chegada do *bitstream*. Para uma implementação H.264 para HDTV no nível 4 do perfil *main*, uma taxa da ordem de 20 Mbps deve ser suportada para que seja atingido tempo real (PURI, 2004). A Figura 5.9 apresenta uma arquitetura, destacando os motores de decodificação. Todos os três motores recebem dados de entrada do *bitstream* a partir de um *Buffer Signal* (BS). De acordo com o fluxo de decodificação, um dos motores de decodificação opera por vez para reconstruir o valor do SE enquanto bits do *bitstream* (através do BS) são consumidos. Os registradores RANGE e LOW, representam as variáveis *CodlRange* e *CodlOffset* definidas anteriormente. Após cada passo de execução, a saída de apenas um dos motores (aquele que estava operante) é selecionada.

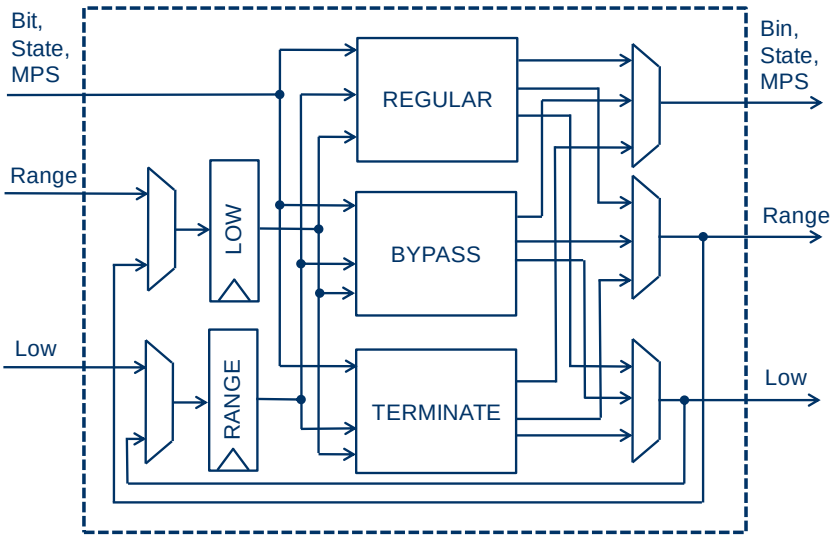

Figura 5.9: Organização do BAD com os três tipos de motores produzindo 1 *bin* por ciclo.

Técnicas para reduzir a latência e a dependência de dados no CABAD têm sido amplamente propostas na literatura e elas seguem basicamente cinco correntes distintas:

- *Pipeline*
- Pre-busca de contextos e uso de cache para contextos
- Eliminação do laço de renormalização
- Paralelização dos motores de decodificação
- Organização da memória

A estratégia de *pipeline* é empregada por Yang (2006) para aumentar a taxa de *bins* por ciclo. Uma alternativa para resolver o problema da latência na renormalização é apresentada em Eeckhaut (2006). O processamento especulativo através do uso de motores de decodificação em paralelo é explorado inicialmente em Yu (2005), e então em Kim (2006) e Bingbo (2007). Processos de decodificação de alta eficiência empregando pré-busca e cache para os contextos são apresentados em Yu (2005) e Zhang (2007) respectivamente. A otimização e reorganização da memória é abordada em Yang (2006).

O trabalho desenvolvido em Bingbo (2007) apresenta otimizações no motor de codificação aritmética através da execução paralela de modo especulativo e a antecipação da contagem de zeros a frente de *CodlRange*. Estas duas abordagens trazem reduções no atraso do caminho crítico.

A arquitetura de hardware proposta em Yu (2005) é baseada na análise da relação entre a contagem de *bins* para cada tipo de SE e a ocorrência de cada tipo de SE em um macrobloco. A taxa de uso imposta por cada SE em cada um dos três motores de decodificação é um aspecto relevante e é usada para otimizar o processo de decodificação como um todo.

Uma avaliação das dependências de dados no modo regular do decodificador aritmético binário é apresentada em Kim (2006). Neste estudo, a frequência de mudanças nos registradores RANGE e LOW é considerada para casos onde o processo de renormalização ocorre combinado com a observação da decisão pelo MPS ou LPS.

Considerando as várias características propostas por diferentes autores para o CABAD e a falta de alguns dados relativos ao uso dos motores de decodificação, uma caracterização estatística mais profunda se mostra necessária para a perfeita compreensão das dependências de dados que tipicamente ocorrem em vídeos comprimidos usando todas as ferramentas do H.264.

5.2.1 Análise estatística do fluxo de dados no CABAD

A partir do software de referência do padrão H.264, o JM 10.2 (SUHRING, 2008), algumas modificações foram realizadas para inserir pontos de coleta de informações para montar tabelas de dados de uso e relações de acesso de cada um dos módulos que compõem o CABAD. Para ter um conjunto de dados representativos, foram usados vídeos (REISSLEIN, 2008) em três diferentes resoluções: 16 QCIF, 19 CIF e 7 HD 1080p. Além disso, cada sequência de vídeo foi codificada com vários valores do parâmetro de quantização (QP). Apenas os 200 primeiros quadros de cada sequência foram considerados.

Os parâmetros de codificação empregados no codificador foram os seguintes:

- Profile_IDC = 77 (main)
- Level_IDC = 40 (4.0)
- SymbolMode = CABAC
- GOP=IPBB
- RDO=ON

Os parâmetros de quantização dos *slices* I e P empregados para codificar cada sequência de vídeo foram os seguintes: (0,0); (6,0); (18,12), (24,18), (36,26). Usando estas seis definições de QP para cada vídeo codificado, o conjunto total de sequências

codificadas foi de 252. Os dados foram então coletados durante a decodificação de cada uma destas 252 sequências de vídeo e os dados coletados foram estudados e sintetizados para conduzir a tomada de decisão para a concepção de uma nova arquitetura para o CABAD.

Um dos problemas do CABAD é determinar a taxa de dados (*throughput*) para o processo de decodificação ocorrer em tempo real na taxa de dados, resolução e taxa de quadros escolhida. Este problema é pertinente porque o tamanho da palavra de código gerada pelo CABAC pode variar significativamente entre iterações e a taxa de dados do *bitstream* pode mascarar a necessidade de desempenho do CABAD. Além disso, para alguns tipos de SE, pode ser difícil determinar o tamanho da *binstring* e a sequência de SEs, à medida que estas variam de acordo com o tipo de *slice* e com o tipo de macrobloco. O padrão H.264 no nível 4.0, define a taxa limite de dados do *bitstream* em 20 Mbps. Durante a análise dos dados, percebeu-se que a taxa média de *bins* produzidos para cada bit do *bitstream* ficava entre 1,3 e 2,1, dependendo da sequência de vídeo e dos parâmetros de quantização escolhidos, por isso a taxa de saída do CABAD para um *bitstream* chegando a 20 MBps pode atingir 42 M*bins*/s em média. A tabela 5.2 apresenta um sumário das análises realizadas.

Tabela 5.2: Distribuição dos *bins* produzidos (DEPRA, 2009)

		Coded Block Flag (%)	*Sig & Last Flags (%)*	*Coefficient Levels (%)*	Outros SEs *(%)*
I MB	Ocorrências	2,32	60,38	34,30	3,00
	# bins	0,67	17,62	80,00	2,31
	Motor Regular	0,86	22,31	74,68	2,15
	Motor *Bypass*	0	0	100	0
P MB	Ocorrências	2,51	58,50	37,07	1,92
	# bins	1,47	34,21	62,05	2,27
	Motor Regular	1,76	40,95	54,90	2,39
	Motor *Bypass*	0	0	98,33	1,67
B MB	Ocorrências	2,61	57,58	38,56	1,25
	# bins	1,80	39,75	46,29	12,16
	Motor Regular	2,17	47,98	37,46	12,39
	Motor *Bypass*	0	0	98,97	1,03
Média	Ocorrências	2,55	58,07	37,65	1,73
	# bins	1,54	35,14	61,10	2,22
	Motor Regular	1,86	42,41	53,30	2,43
	Motor *Bypass*	0	0	98,78	1,22

No processo de decodificação parte de um *binstring* que compõe um elemento sintático pode ser produzida por um motor de decodificação (*Regular*, *Bypass* ou *Terminate*) enquanto outra parte é produzida por outro motor. Considerando que o motor regular é o bloco do BAD mais complexo e que o *bypass* é usado somente pelo sufixo dos SE de tipo MVD (*motion vector differential*), pelos coeficientes de transformadas e pelo bit de sinal dos coeficientes de transformadas, foi decidido observar o comportamento dos motores de decodificação para cada tipo de SE para determinar a melhor estratégia para o projeto de uma arquitetura.

5.2.2 Proposta arquitetural para os motores de decodificação do CABAD

Analisando a contagem da ocorrência de *bins* nos vídeos codificados, apenas quatro tipos de SE (coded_block_flag, coeff_level, sig_coeff_flag e last_sig_flag, de um total de 17 possíveis) correspondem a mais de 93% de todos os *bins* na média para todos os tipos de macroblocos (I, B e P). Por isso, uma análise profunda no comportamento destes tipos de SE foi realizada para melhorar o ganho no BAD. Em um primeiro estudo, foi investigada a distribuição dos *bins* para diferentes tipos de SE em cada tipo de macrobloco. Os resultados são apresentados na Tabela 5.2, em que a percentagem de ocorrência é informada. O numero de vezes que um tipo de SE ocorre é relacionado com o total de ocorrências e a contagem de *bins* mostra o número total de *bins* para cada tipo de SE, normalizado para o a contagem total de *bins*. Além disso, há uma média de 7 sig_coef_flag e 5 last_sig_flag para cada cada bloco de resíduos de 4x4 amostras. As estatísticas de uso dos motores de decodificação mostram que o motor regular produz 80,8% do total de *bins* enquanto o bypass produz 19,2%.

De acordo com a análise descrita na seção anterior, uma nova arquitetura foi proposta com o objetivo de explorar o comportamento estatístico observado dos motores de decodificação por Deprá (2009). O objetivo foi aumentar a taxa de processamento através do uso de paralelismo e execução especulativa e maximizando o uso dos recursos de hardware. A proposta inicial foi baseada no trabalho desenvolvido por Yu (2005), que apresenta a melhor proposta encontrada na literatura. A Figura 5.10 apresenta a proposta de Yu (2005). Observe que apenas a implementação dos motores regular e bypass são descritas.

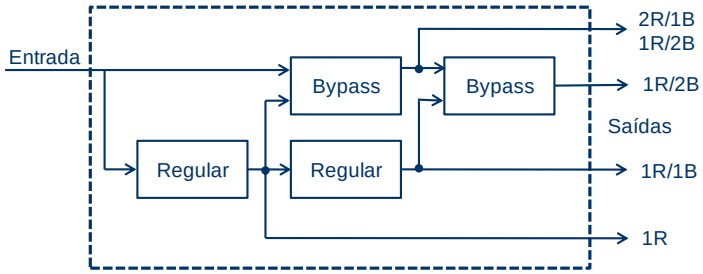

Figura 5.10: Arquitetura para os motores de decodificação conforme concepção de Yu (2005).

Na arquitetura proposta, a estratégia *first one detect* (FOD) para resolver o problema da renormalização foi aplicada, de uma forma similar à apresentada por Mei-hua (2007). Após isso, as operações de reordenamento feitas pelo motor regular foram empregadas em uma abordagem similar à apresentada em Kim (2006). Isto resultou em dois caminhos paralelos dentro do motor regular, um para tratar a ocorrência do MPS e outro para tratar o LPS. Outro aspecto importante é o acesso às tabelas de constantes para obter informações sobre o próximo estado (MPS_TABLE e LPS_TABLE) e a tabela de estimativa de probabilidade do rLPS (RLPS_TABLE). Estas memórias são endereçadas pelo *pState*, que é gerado pelo modelo de contexto armazenado na memória de contexto. Estas memórias são combinacionais e estão dentro do motor regular afeta o caminho crítico. Além disso, quando os dois motores regulares são concatenados, dois acessos a essas memórias no mesmo ciclo é necessário. Para resolver este problema, uma abordagem similar a de Mei-hua (2007) foi adotada, em que as memórias são concatenadas e combinam as informações a respeito do atual e do próximo estado do MPS, o próximo LPS e a probabilidade estimada para o rLPS. Com isso é possível obter

toda a informação necessária para decodificar dois *bins* que referenciam o mesmo contexto com apenas um acesso à memória de constantes.

Cada linha na memória de constantes contém informação sobre todas as possíveis decisões dentro dos motores regulares. Esta memória é dividida em dois blocos: o primeiro mantém as informações necessárias para atualizar as estimativas de probabilidade quando o símbolo MPS é decodificado enquanto o segundo mantém as informações necessárias para quando o símbolo LPS é decodificado. A diferença entre o primeiro e o segundo bloco é que um é usado quando o primeiro motor regular decide pelo MPS enquanto o outro é usando quando este decide pelo LPS. O segundo motor de decodificação estará então fazendo uma execução especulativa. O estado para a decisão do MPS não é armazenado na memória de constantes porque ele pode ser gerado por uma simples operação de adição: State_MPS= pState+1.

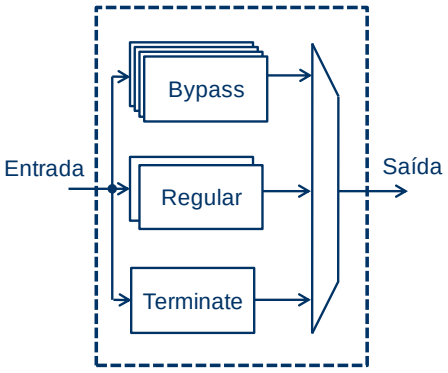

Figura 5.11:Arquitetura proposta para o BAD.

A alternativa apresentada para a memória de constantes resolve o problema de conflito de acesso quando ambos os motores estão tentando acessar o mesmo contexto, mas não quando eles estão usando contextos distintos. Para resolver esta contenção, a memória é duplicada, uma para cada motor regular. Para reduzir o atraso no caminho crítico, uma cópia das memórias de constantes foi colocada em cada motor regular. Assim, a segunda memória de constantes pode ser acessada em paralelo com a decodificação do primeiro motor regular, minimizando o caminho crítico dos motores regulares.

Finalmente, baseado na proposta apresentada por Yu (2005) e combinando as abordagens descritas acima, uma primeira solução foi proposta. Entretanto, baseado na análise estatística realizada, observou-se que a adição de motores *bypass* poderia aumentar significativamente a desempenho de todo o BAD, sem um aumento significativo na área devido a simplicidade do motor *bypass*. Os resultados de simulação mostram que com 4 motores *bypass*, o desempenho do caminho *bypass* poderia ser aumentado em 21,4% quando comparado com 2 *bypass*. Por isso, duas versões da arquitetura foram implementadas, uma com 2 motores *bypass* e outra com 4 motores *bypass*. Além disso, podem ser decodificados dois *bins* pelo caminho regular, um pelo regular e um pelo *bypass* ou ainda dois pelo regular e um pelo *bypass* por ciclo. Considerando que no caminho regular os dois motores estão concatenados e que este é o caminho crítico, algumas explorações no projeto do hardware foram realizadas para tentar minimizar a profundidade lógica deste caminho como será descrito a seguir. A Figura 5.11 apresenta o arranjo arquitetural desenvolvido com os três tipos de motores do BAD.

Considerando que o motor regular é responsável pela maioria dos *bins* produzidos pelo CABAD, parece uma boa alternativa aumentar o nível de paralelismo dentro deste motor instanciando mais motores regulares. Entretanto, o motor regular está no caminho crítico devido à grande profundidade lógica. O acréscimo de mais motores regulares aumenta a profundidade lógica e a execução especulativa teria mais probabilidade de falhar, levando a um aumento significativo na área para um aumento marginal na taxa de *bins* produzida. Uma análise detalhada a este respeito não foi realizada, entretanto. A Figura 5.12 mostra a arquitetura básica para o motor regular do BAD.

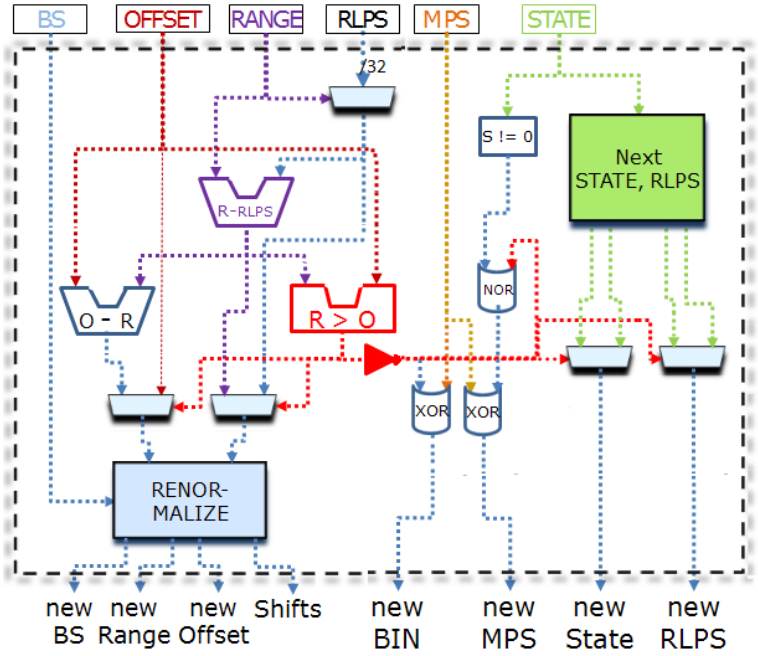

Figura 5.12: Arquitetura do motor regular (DEPRÁ, 2009).

5.2.3 Resultados experimentais

A arquitetura concebida foi descrita em VHDL e sintetizada para um processo CMOS *standard cells* da IBM de 0.18um usando o Cadence RTL Compiler. A ferramenta ModelSim versão 6.01a foi usada durante a simulação e o processo de validação. A arquitetura desenvolvida é capaz de decodificar até 4 *bins* no modo *bypass* por ciclo no melhor caso. A tabela 5.3 apresenta os resultados de síntese para a arquitetura proposta. A arquitetura baseada em 2 motores *bypass* com as otimizações descritas na seção anterior também foi implementada para ser usada como referência.

Os resultados apresentados na Tabela 5.3 indicam que o aumento no hardware para o BAD com 4 motores *bypass* é em torno de 7%, enquanto a frequência de operação permaneceu praticamente a mesma. O aumento de desempenho do caminho *bypass* foi de 21% em média.

Nos 252 casos de teste empregados, a máxima taxa de *bins* no modo *bypass* (4 *bins*/ciclo) nunca foi alcançada. Nestes testes, foi observado que uma taxa máxima de 4,8 M*bins*/s pode ser produzida pelo BAD com 2 motores *bypass*, enquanto 5,8 M*bins*/s podem ser produzidos pelo BAD com 4 motores de *bypass*.

Tabela 5.3: Resultados de síntese ASIC.

Parâmetro	Arquiteturas para motores do BAD de múltiplos *bins*		
	2 Regular, 2 Bypass	*2 Regular, 4* Bypass	*Ganho (%)*
Portas equivalentes	3671	3928	-7
Frequência máxima	192	190	-0,84
Bins/Ciclo max.	3(2R1B)	4(4B)	33,4
Taxa Max. *Bypass* (M*bins*/s)	576	762	32,4
Taxa média *Bypass* (M*Bins*/s)	4,78	5,80	21,3

5.2.4 Conclusões

No CABAD, os resultados apresentados mostram que há um ganho de desempenho na codificação dos *bins* do modo *bypass* comparado ao trabalho considerado o estado-da-arte da literatura no momento de sua concepção (YU, 2005). No entanto este ganho é menor considerando-se a taxa média de *bins* produzidos incluindo os do motor regular e o acréscimo de complexidade envolvido nessa solução. Em especial, considerando que o debinarizador não foi alvo deste desenvolvimento e que o cálculo do índice do próximo contexto pode levar a erros de predição e necessidade de recomputação, o desempenho da arquitetura completa poderá ser um pouco penalizado. Isso indica que um aumento futuro no paralelismo do decodificador pode não trazer benefícios.

Os próximos passos no desenvolvimento do CABAD são a integração do BAD com o debinarizador e demais módulos do decodificador de entropia, de forma a obter um decodificador de entropia completo para o H.264. Alguns cenários mais específicos de uso do decodificador também poderiam ser analisados (apenas HDTV, apenas CIF, muito ou pouco movimento nas cenas, apenas intra, etc..) para melhor avaliar a arquitetura desenvolvida.

5.3 Proposta arquitetural para o CABAC

Estudos realizados durante a investigação da literatura relativa a implementação em harware do CABAC (YANG, 2006; PASTUSZAK, 2008; OSORIO 2006; TIAN, 2009) assim como o trabalho na implentação do CABAD (DEPRÁ, 2009) demonstram que há um acréscimo considerável de complexidade no codificador CABAC se o codificador aritmético for capaz de processar mais do que um *bin* por ciclo de *clock*. Isso ocorre porque para fornecer os dados de entrada para o codificador aritmético (*bin* e estado de probabilidade) é necessário que o processamento do *bin* anterior (acesso, atualização do contexto e codificação aritmética) tenha que ser concluído. Há casos na codificação em que um mesmo contexto é usado para codificar vários *bins* em sequência; e também há casos em que um contexto é usado de forma intercalada. Isso torna o controle do acesso aos contextos significativamente mais complexo, pois há a necessidade de ter o valor atualizado do contexto no próximo ciclo de *clock* ou em poucos ciclos. Este fato é muitas vezes omitido nos trabalhos disponíveis na literatura (ZHENG, 2008), enquanto outros utilizam esquemas sofisticados de cache, forward e memórias combinacionais para lidar com este problema (TIAN, 2009; DEPRÁ, 2009), aumentando

significativamente a complexidade da arquitetura. Há ainda trabalhos que modificam o algoritmo padrão para tornar mais rápido o processo de atualização dos contextos (BIN, 2007), entretanto tornam a implementação incompatível com o H.264.

Baseado nas análises e no desenvolvimento realizado para o CABAD (DEPRÁ, 2009), as seguintes decisões arquiteturais foram tomadas:

- BAC capaz de processar até 1 *bin* por ciclo
- *Pipeline* para o acesso e atualização dos contextos
- Equilíbrio entre os módulos (frequencia de operação e desempenho)
- Maximização da frequência de *clock*
- Desempenho suficiente para alta-definição em tempo real na tecnologia presente

Uma arquitetura para os módulos que compõe o codificador de entropia foi então proposta de forma alcançar frequências de *clock* e taxas de processamento que estejam em harmonia entre si, gerando assim um codificador CABAC capaz de atingir taxas de processamento compatíveis com alta-definição através de caminhos críticos mais curtos. Com esta proposta, uma ampla gama de aplicações pode ser coberta, desde aplicações embarcadas, até alta definição. Como o uso de paralelismo acima do nível de slice, através do uso de múltiplas instâncias do codificador proposto, é possível suportar virtualmente qualquer taxa de processamento exigida para o CABAC. A Figura 5.13 apresenta a proposta arquitetural para o codificador CABAC.

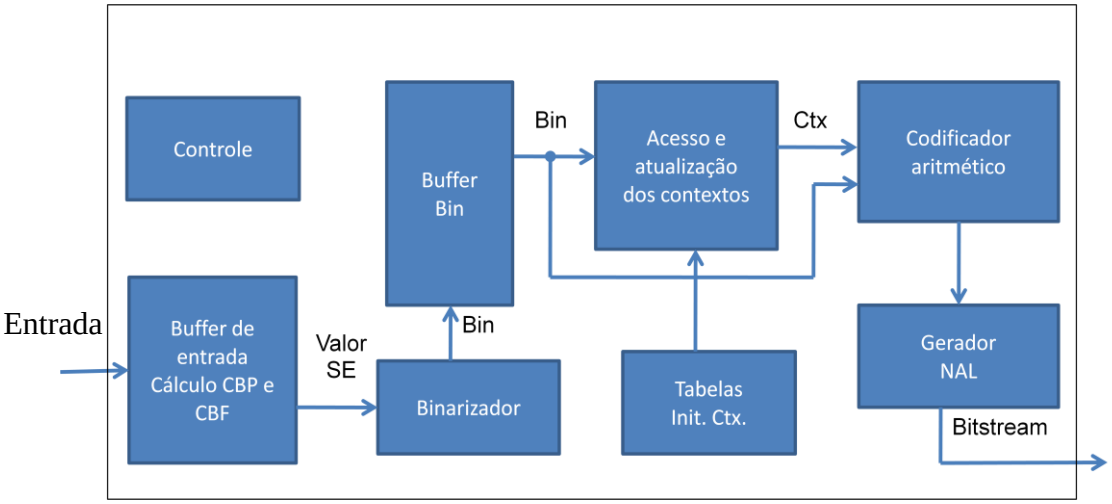

Figura 5.13: Proposta arquitetural para o codificador CABAC.

Na Figura 5.13 os elementos sintáticos produzidos pelos demais módulos do codificador H.264 alimentam o buffer de entrada. Quando os elementos sintáticos forem da camada de macrobloco, as informações de padrão de bloco codificadas (CBP) e flags de bloco codificadas (CBF) podem ser computadas. Os valores e os tipos dos SEs são enviados para o binarizador para a sua conversão em strings de bits de acordo com o modo de binarização apropriado (especificado pela norma H.264). O bloco de controle se encarrega de fazer o sequenciamento das operações do buffer de entrada e do binarizador.

O Binarizador irá produzir uma string de *bins* a uma taxa variável que depende do elemento sintático que está sendo processado. Da mesma forma, o codificador aritmético consumirá *bins* em uma taxa variável de até 1 *bin* por ciclo de *clock* de acordo com a arquitetura selecionada para o processo de renormalização. Por isso, um buffer de *bins* (FIFO) foi posto entre o binarizador e o codificador aritmético.

O acesso aos contextos é gerenciado pelo controle e pode ser realizado em um estágio anterior a chegada do *bin* correspondente no bloco de acesso e atualização dos contextos. Entretanto, a atualização do estado de probabilidade depende do conhecimento do valor do *bin* que deverá ser codificado, por isso este passo só poderá zer efetuado quando o valor do *bin* estiver disponível.

5.3.1 Proposta arquitetural para o codificador aritmético do CABAC

Com base nas decisões arquiteturais tomadas para o CABAC, ilustradas pela proposta arquitetural global apresentada na Figura 5.13, uma arquitetura para o núcleo do CABAC, englobando o acesso, atualização e inicialização dos contextos além do próprio codificador aritmético foi desenvolvida. Esta arquitetura é ilustrada na Figura 5.14 ilustra a arquitetura desenvolvida.

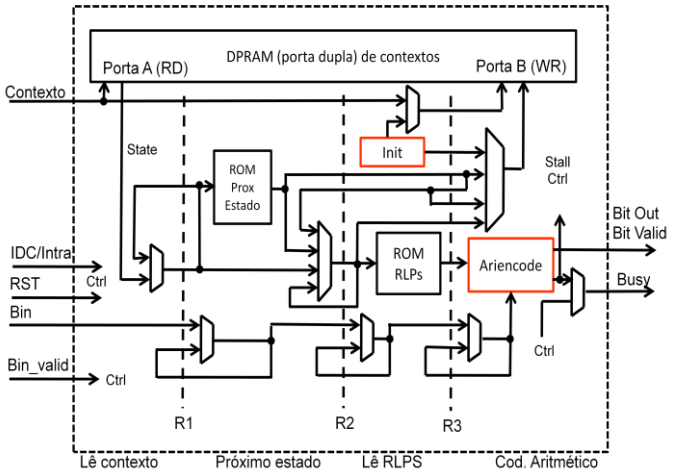

Figura 5.14: Proposta arquitetural o núcleo do CABAC.

Esta arquitetura é composta por um *pipeline* de 4 estágios. A memória de contextos contém 1024 palavras de 7 bits. O MPS é o bit mais significativo e o estado de probabilidade são os 6 bits menos significativos. Os 1024 bytes garantem que esta arquitetura é capaz de suportar todas as extensões do padrão até os perfis HIGH (ITU, 2009). Esta memória é de porta dupla, uma para acesso aos contextos e outra para a gravação dos contextos atualizados. Desta forma um contexto pode ser lido ou escrito no mesmo ciclo de *clock*. Na arquitetura proposta, o valor atualizado do contexto é efetivamente escrito na memória e está disponível para leitura 4 ciclos depois de ser lido. O controle desenvolvido utiliza forward do *pipeline* para todas as situações em que o valor de contexto está desatualizado na memória de contextos, o que ocorre sempre que o valor atualizado de um mesmo contexto tem que ser usado entre dois e quatro ciclos de *clock* após a sua primeira leitura da memória de contextos. Além disso o controle também insere bolhas no pipeline para os casos onde não há disponibilidade de *bins* na entrada. Um flag "*bin valid*" irá indicar quando o codificador aritmético deve considerar as informações de entrada como válidas. O módulo de acesso e atualização de contextos deverá monitorar um flag do codificador aritmético que indica se este está disponível para o processamento de novos *bins*, para o caso da utilização da versão onde a renormalização pode utilizar mais de um ciclo de *clock*. Quando uma operação de renormalização multi-ciclo tem que ser executada no codificador aritmético, um sinal de ocupado é gerado e o controle deve parar o pipeline.

O estado de probabilidade de cada um dos contextos deve ser iniciado com um valor pré-estabelecido pela norma ao início de cada slice. O valor deve ser computado a partir

de algumas operações envolvendo 4 tabelas de inicialização de contextos (selecionadas através do valor do parâmetro InitIDC do codificador e do tipo de slice a ser codificado: inter/intra) e também do valor do parâmetro de quantização (QP) inicial do *slice*. Este processo de inicialização deve definir o estado de probabilidade de cada um dos contextos antes que estes possam ser utilizados. Na arquitetura desenvolvida, optou-se por iniciar todos os contextos possíveis para um slice antes do início da codificação do mesmo. Isto significa uma latência inicial que depende do perfil utilizado no codificador (299 para o perfil main até 1024 para o MVC). Uma arquitetura pipeline para a inicialização dos contextos foi desenvolvida de forma a maximizar a frequência de *clock* e facilitar a integração com os demais módulos, evitando que este se torne um gargalo para a frequência máxima de operação do sistema. A Figura 5.15 ilustra a arquitetura proposta para a sequencialização da inicialização dos contextos.

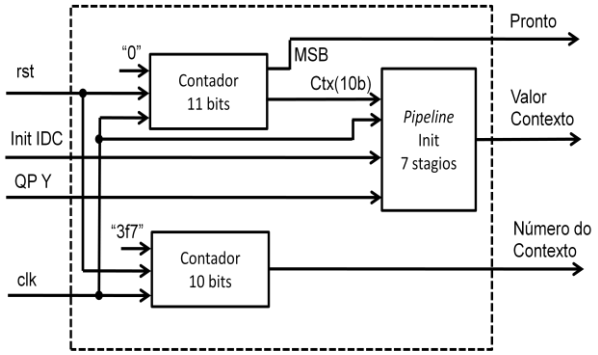

Figura 5.15: Sequenciamento da inicialização dos contextos.

Neste módulo o sinal *rst* deve ser ativado para carregar nos dois contadores seus valores iniciais. Um dos contadores iniciará em zero e o outro em 3F7h, 8 estados antes do overflow deste contador de 10 bits. O pipeline de 7 estágios usado pra o cálculo do valor inicial dos contextos irá fornecer o primeiro contexto válido (numero zero) 7 ciclos de *clock* após receber o valor do contexto na entrada. No oitavo ciclo de *clock* o contador de 10 bits terá sofrido overflow e seu valor irá para zero, e este será o índice do contexto que estará pronto na saída do pipeline de inicialização dos contextos. A lógica de controle não irá impedir a escrita dos 7 estados inválidos no final da memória de contextos pois eles serão novamente escritos ao final do processo. Quando o contador de 11 bits chegar no valor 1024, o bit mais significativo (MSB) será ligado e este é o sinal de término da contagem de inicialização dos contextos. O controle da lógica de acesso e atualização dos contextos deverá esperar até uma nova passagem por zero do contador de 10 bits para que o último contexto inicial seja produzido (índice 1023). A partir do término da inicialização dos contextos os elementos sintáticos do slice que precisam ser codificados pelo codificador aritmético podem começar a ser processados. A inicialização de 1024 contextos garante ao núcleo do CABAC o suporte aos contextos do pefil HIGH do padrão, incluindo MVC. Um total de 1032 ciclos são consumidos com a inicialização dos contextos, que é realizada ao início de cada slice. Uma discussão sobre o impacto na performance do codificador durante a inicialização dos contextos será discutido mais adiante neste capítulo.

O cálculo da inicialização dos contextos envolve duas operações de saturação, uma multiplicação, uma soma, uma subtração em cascata. Para otimizar a performance, um pipeline de 7 estágios foi desenvolvido e é apresentado na Figura 5.16.

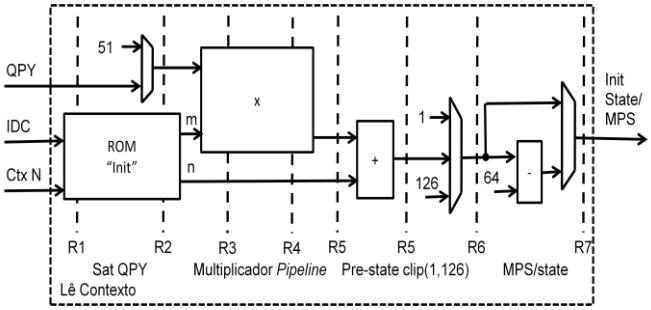

Figura 5.16: Pipeline para o cálculo do valor inicial dos contextos.

O Padrão H.264 fornece uma série de tabelas para a inicialização dos contextos que produzem como saída um valor denominado "m" e outro denominado "n", que conjuntamente com o valor do QP utilizado para as amostras de luminância (Y) são usados para computar o valor inicial de cada contexto. Alguns valores iniciais de contextos ocorrem apenas em quadros do tipo I/SI e independem do valor do IDC (Init IDC na norma). Outros ocorrem apenas para quadros que não são I/SI e dependem dos três valores possíveis para o IDC. Os demais contextos possuem um par de valores "m" e "n" para os slices I/SI e três pares (dependendo do valor do parâmetro Init IDC), totalizando quatro pares de valores. Para otimizar a organização da memória e o controle, uma memória ROM de 1024x64 bits foi utilizada. A palavra desta memória contém 4 pares "m" e "n" para as quatro possibilidades de inicialização: Slices I/SI e os três valores possíveis de Init IDC (número da tabela de inicialização de contextos utilizada). Nos casos de contextos de quadros I/SI que não possuem valores de "m" e "n" dependentes do Init IDC, os valores para o slice I/SI são usados para preenchimento da memória. Desta forma, o controle não necessita de sinais específicos para desativar a entrada Init IDC quando esta não for usada. Para os contextos que não possuem valores de "m" e "n" para os slices tipo I/SI, o valor é irrelevante. Desta forma, uma única memória pode ser utilizada para conter os valores de "m" e "n", simplificando o controle sob pena de uma pequena quantidade de dados redundantes armazenados. Os valores redundantes da tabela são apresentados na Tabela 5.4. Os valores redundantes estão destacados. Um total de 224 valores redundantes são representados em uma tabela de 8192 valores, ou 2,7% do total.

Tabela 5.4: Inicialização dos contextos

CtxIdx	***Slices* I e SI**		**Valor do cabac_init_idc**						**ctxIdx**	***Slices* I e SI**		**Valor do cabac_init_idc**					
			0		**1**		**2**					**0**		**1**		**2**	
	m	**n**	**m**	**n**	**m**	**n**	**m**	**n**		**m**	**n**	**m**	**n**	**m**	**n**	**m**	**n**
0	20	-15	20	-15	20	-15	20	-15	35	1	62	1	62	5	52	5	57
1	2	54	2	54	2	54	2	54	36	-6	86	-6	86	6	69	-6	93
2	3	74	3	74	3	74	3	74	37	-17	95	-17	95	-13	90	-14	88
3	20	-15	20	-15	20	-15	20	-15	38	-6	61	-6	61	0	52	-6	44
4	2	54	2	54	2	54	2	54	39	9	45	9	45	8	43	4	55
5	3	74	3	74	3	74	3	74	40	-3	69	-3	69	-2	69	-11	89
6	-28	127	-28	127	-28	127	-28	127	41	-6	81	-6	81	-5	82	-15	103
7	-23	104	-23	104	-23	104	-23	104	42	-11	96	-11	96	-10	96	-21	116
8	-6	53	-6	53	-6	53	-6	53	43	6	55	6	55	2	59	19	57
9	-1	54	-1	54	-1	54	-1	54	44	7	67	7	67	2	75	20	58
10	7	51	7	51	7	51	7	51	45	-5	86	-5	86	-3	87	4	84
11	23	33	23	33	22	25	29	16	46	2	88	2	88	-3	100	6	96
12	23	2	23	2	34	0	25	0	47	0	58	0	58	1	56	1	63

Tabela 5.4: Inicialização dos contextos

CtxIdx	*Slices* I e SI		Valor do cabac_init_idc						ctxIdx	*Slices* I e SI		Valor do cabac_init_idc					
			0		1		2					0		1		2	
	m	n	m	n	m	n	m	n		m	n	m	n	m	n	m	n
13	21	0	21	0	16	0	14	0	48	-3	76	-3	76	-3	74	-5	85
14	1	9	1	9	-2	9	-10	51	49	-10	94	-10	94	-6	85	-13	106
15	0	49	0	49	4	41	-3	62	50	5	54	5	54	0	59	5	63
16	-37	118	-37	118	-29	118	-27	99	51	4	69	4	69	-3	81	6	75
17	5	57	5	57	2	65	26	16	52	-3	81	-3	81	-7	86	-3	90
18	-13	78	-13	78	-6	71	-4	85	53	0	88	0	88	-5	95	-1	101
19	-11	65	-11	65	-13	79	-24	102	54	-7	67	-7	67	-1	66	3	55
20	1	62	1	62	5	52	5	57	55	-5	74	-5	74	-1	77	-4	79
21	12	49	12	49	9	50	6	57	56	-4	74	-4	74	1	70	-2	75
22	-4	73	-4	73	-3	70	-17	73	57	-5	80	-5	80	-2	86	-12	97
23	17	50	17	50	10	54	14	57	58	-7	72	-7	72	-5	72	-7	50
24	18	64	18	64	26	34	20	40	59	1	58	1	58	0	61	1	60
25	9	43	9	43	19	22	20	10	60	0	41	0	41	0	41	0	41
26	29	0	29	0	40	0	29	0	61	0	63	0	63	0	63	0	63
27	26	67	26	67	57	2	54	0	62	0	63	0	63	0	63	0	63
28	16	90	16	90	41	36	37	42	63	0	63	0	63	0	63	0	63
29	9	104	9	104	26	69	12	97	64	-9	83	-9	83	-9	83	-9	83
30	-46	127	-46	127	-45	127	-32	127	65	4	86	4	86	4	86	4	86
31	-20	104	-20	104	20	101	20	-15	66	0	97	0	97	0	97	0	97
32	1	67	1	67	2	76	2	54	67	-7	72	-7	72	-7	72	-7	72
33	-13	78	-13	78	3	71	3	74	68	13	41	13	41	13	41	13	41
34	-11	65	-11	65	20	79	20	-15	69	3	62	3	62	3	62	3	62

No processo de operação normal do CABAC a inicialização de todos os contextos não é necessária, visto que muitos contextos são exclusivos para certos perfis e níveis que são mutuamente exclusivos. Isto certamente gera uma carga de processamento desnecessária. Entretanto, o tempo total da incicialização dos contextos é muito menor que o tempo em que o CABAC opera entre inicializações sucessivas de contextos. Para vídeo de alta-definição, a relação entre o tempo de inicialização e o tempo de compressão do codificador aritmético é tipicamente menor que 0,1% para resulução 1080p nas arquiteturas propostas. Nestas condições, uma seleção criteriosa dos contextos a serem inicializados traria muito pouco benefício em desempenho ao custo de complexidade arquitetural adicional, com aumento de área e possível redução da frequência máxima de operação.

Os estágios de pipeline estão distribuídos de forma a maximizar a frequência de operação. Em especial, três estágios de pipeline são destinados à operação de multiplicação, através de um multiplicador pipeline. Esta é uma otimização voltada pra a síntese em FPGAs da Xilinx (VIRTEX, 2008; VIRTEX, 2008a), e pode necessitar de uma reavaliação, em caso de síntese ASIC. Os demais estágios de pipeline são utilizados para manter uma operação aritmética por estágio, maximizando a frequência de operação e evitando que este módulo represente um gargalo na frequência de operação de todo o CABAC.

O *bitstream* produzido pelo codificador aritmético deve passar por um processo de formatação para formar pacotes da camada NAL (Network Abstraction Layer). Este processo envolve a criação de pacotes NAL que não contém informação de vídeo

codificado (VCL) que servem para informar ao decodificador alguns parâmetros da codificação (tamanho do quadro em pixels, etc..). Estas informações não-VLC não são produzidas pelo processo de codificação aritmética e um módulo específico na arquitetura chamado Gerador NAL foi desenvolvido para lidar com esta tarefa. Além de criar estes pacotes que não contém informação de vídeo codificado, o gerador de NAL também fará a inserção de informações de cabeçalho de NAL e o controle para evitar que o marcador de cabeçalho da NAL apareça ocasionalmente no *bitstream* codificado. Caso isso ocorra, uma sequência especial chamada pela norma H.264 de byte de prevenção de emulação de cabeçalho (emulation prevention byte) é inserido, fazendo com que o marcador de cabeçalho da NAL possa ser reconhecido inequivocamente pelo decodificador. Este processo é útil em situações onde interferências no meio de transmissão ou armazenamento de dados gerem a perda de sincronismo no decodicador. O marcador de cabeçalho de NAL tornará possível ao decodificador idenficar a próxima NAL. Uma NAL que contém informação de vídeo codificado contém exatamente a informação de um Slice do vídeo.

5.3.2 Proposta arquitetural para o codificador aritmético do CABAC

Uma das principais limitações do processo de codificação aritmética empregado no CABAC é a natureza sequencial das suas operações, que gera dependências de dados entre elas. Embora a quantidade de *bins* a serem processados seja significativamente menor que a quantidade de dados que chegam ao codificador de entropia através dos elementos sintáticos devido ao processo de binarização, a necessidade de codificar um *bin* por vez devido a dependências de dados no processo de divisão de intervalo e de renormalização limita a performance que pode ser obtida por uma implementação em hardware. Ainda assim, o projeto de um codificador aritmético para o CABAC pode ser realizado usando diversas estratégias para alcançar um bom compromisso entre área e velocidade de processamento. A Figura 5.17 apresenta uma das possíveis alterativas para o projeto do codificador CABAC.

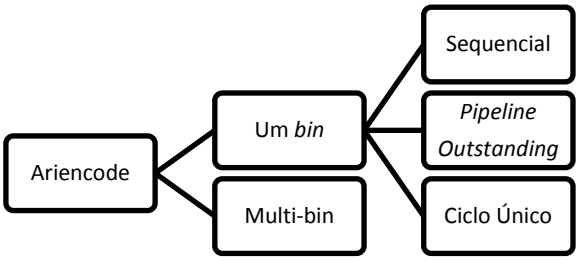

Figura 5.17: Alternativas de implementação do CABAC, destacando alternativas para "Um *Bin*".

Na Figura 5.17, as arquiteturas classificadas como "Um *bin*" são capazes de processar no máximo um bin por ciclo de *clock* enquanto a implementação "multi-*bin*" pode processar mais de um *bin* por ciclo de *clock*. As implementações que utilizam processamento multi-*bin* tornam-se complexas porque o codificador aritmético necessita realizar o processo de renormalização e atualização dos contextos toda vez que um *bin* é processado; a alternativa típica é concatenar o processo de codificação de vários *bins*, adicionando um controle complexo para gerenciar o acesso e atualização dos contextos. Neste caso há um ganho em desempenho, mas uma redução na frequência máxima alcançável e aumento na área utilizada. Além disso, as implementações multi-*bin* exigem que o binarizador seja capaz de fornecer os pares *bin*/contexto de acordo com com o nível de paralelismo do codificador aritmético, tornando este módulo mais complexo ou criando bolhas que penalizam o desempenho global do codificador. Estas

arquiteturas classificadas como "multi-*bin*" na Figura 5.17 podem ser subclassificadas de diversas formas, porém não serão alvo de maior investigação neste trabalho.

Nas implementações que processam até um *bin* por ciclo de *clock* (classificadas como "Um *bin*"), existem diversas possibilidades que vão desde a implementação totalmente sequencial dos algoritmos apresentados anteriormente, caracterizada como "Sequencial" na Figura 5.17; possibilidades que aceleram o processamento dos bits *outstanding* colocando o processamento destes em um outro estágio de pipeline e utilizando uma FIFO entre os estágios, ou ainda a possibilidade de realizar todo o processo de divisão do intervalo e renormalização em um único ciclo, criando um mecanismo especial para lidar com o fluxo variável de bits regulares e *outstanding* produzidos neste processo.

Em resumo, as alternativas que processam até um *bin* por ciclo de *clock* podem ser classificadas da seguinte forma:

Sequencial: um *bin* e, no máximo, um passo de renormalização é realizado a cada ciclo de *clock*; apenas um bit pode ser emitido para o bistream por ciclo de *clock*. Caso seja necessário mais de um passo de renormalização ou a decisão de um bit exija o processamento de bits "outstanding" pendentes, ciclos extras de processamento são executados pela máquina de controle.

Pipeline Outstanding: O processamento ocorre de forma similar ao sequencial. O tratamento dos bits "outstanding" é realizado em um estagio separado de pipelining através de um processo de bufferização com uma FIFO.

Ciclo Único: Um *bin* é processado e todo o processamento necessário para a renormalização é realizado em um único ciclo de *clock*. As arquiteturas para o codificador aritmético do CABAC que utilizam ciclo único produzem um número variável de bits por ciclo de *clock* como resultado. Alguns destes bits podem ser "outstanding". Isto é tratado através de um estágio adicional de processamento, com a bufferização dos bits resultantes do processo de renormalização através de uma FIFO.

As arquiteturas classificadas anteriormente como "Um *bin*" serão o foco deste trabalho. O objetivo é otimizar a arquitetura para chegar em frequências de operação em hardware capazes de alcançar o nível 5 especificado pela norma H.264/AVC (Nível 5), sem necessitar do processamento paralelo de *bin*, tornando a arquitetura simples, com menor área e menor consumo de energia. Uma exploração arquitetural para as arquiteturas "Um *bin*" como descritas anterioremente foi realizada.

Uma das grandes dificuldades em construir um codificador aritmético para o CABAC que possa procesar exatamente um *bin* por ciclo de *clock* (arquitetura "Ciclo Único") é o processo de renormalização. Toda vez que um *bin* é codificado, o processo de renormalização deve ser realizado. Se o valor do registrador Range for maior que 255, nenhum processamento adicional é realizado; caso contrário, a renormalização precisa operar até que o valor da variável *Range* (tamanho do intervalo) seja maior que 255. Quando o processo de renormalização precisa modificar o *Range*, bits são enviados ao *bitstream*. Dependendo do valor do registrador *Low* (base do intervalo) bits *outstanding* podem ser gerados. Por isso mais processamento precisa ser realizado quando o estado dos bits *outstanding* é decidido. A codificação aritmética não pode continuar até que o processo de renormalização termine e isso limita a habilidade das arquiteturas que implementam o codificador aritmético de alcançarem um *bin* processado por ciclo de *clock*.

Para alcançar uma frequência de operação mais elevada, a consulta em tabela do valor do rLPS e divisão do intervalo (codificação aritmética) pode ser dividida em dois estágios de pipeline. O rLPS é uma tabela bidimensional de 64x4 elementos em que um dos índices é o estado de probabilidade e o outro é o valor do registrador *Range* quantizado para dois bits, como ilustrado no algoritmo da Figura 5.5. O Cálculo e a atualização do estado pode ser feita de forma independente do processo de codificação e os quatro possíveis valores de rLPS indexados pelo valor do estado atual de probabilidade podem ser obtidos em um estágio de pipeline anterior ao processo de codificação, como iustrado na Figura 5.18.

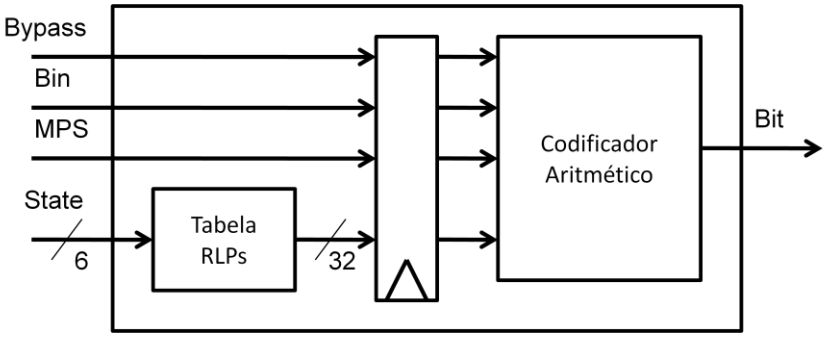

Figura 5.18: Pipeline do rLPS.

Outra decisão arquitetural é a união do processo de codificação regular e bypass para simplificar o controle. Um sinal adicional para controlar o modo de operação do codificador aritmético é necessário como mostrado na Figura 5.19.

Devido a taxa de processamento variáel do codificador aritmético, sinais de *handshaking* tanto para sinalizar um *bin* de entrada válido quanto para sinalizar que o processamento de um *bin* tomará vários ciclos de *clock* são necessários. A Figura 5.19 ilustra a interface desenvolvida para o codificador aritmético.

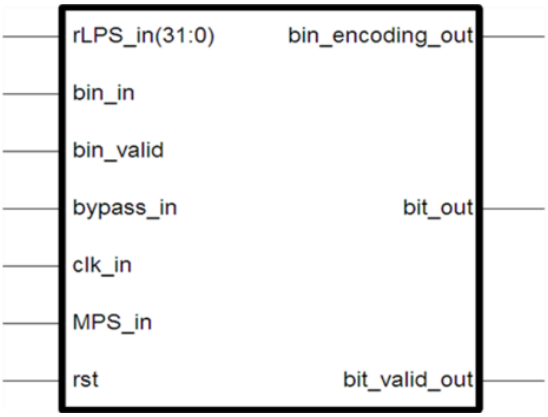

Figura 5.19: Interface do codificador aritmético proposta.

Todas as três arquiteturas desenvolvidas usam a mesma interface apresentada na Figura 5.19.

A Figura 5.20 apresenta a arquitetura básica do codificador aritmético. Esta arquitetura inclui os três motores de codificação do CABAC: *regular*, *bypass* e *terminate*. Para codificar um *bin* no modo regular, a entrada *bypass* precisa ser mantida em nível lógico baixo enquanto a entrada *bin_valid* estiver em nível alto. Esta arquitetura suporta a renormalização multi-ciclo, que é sinalizada através do sinal bin_encoding. Este sinal vai a nível alto ao final do ciclo de processamento se uma

operação multi-ciclo for necessária e se mantém neste nível enquanto o processo de renormalização estiver operando. Durante uma renormalização multi-ciclo, os sinais de entrada devem permanecer estáveis com os valores adequados para a codificação do *bin* atual.

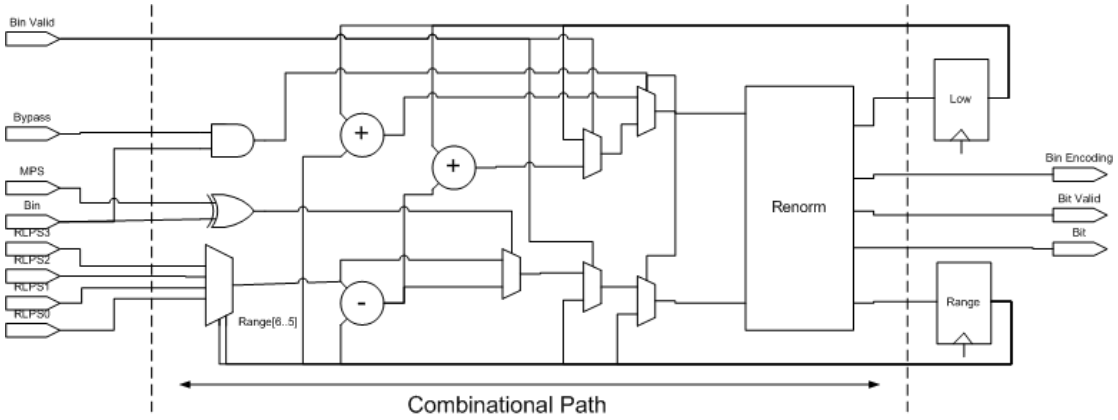

Figura 5.20: Arquitetura para a divisão do intervalo.

Três arquiteteturas alternativas foram propostas para a fase de renormalização do codificador aritmético. A arquitetura "Sequencial" para o processo de renormalização é ilustrada na Figura 5.21. Esta arquitetura é capaz de realizar no máximo um passo de renormalização por ciclo de *clock*. Dependendo do valor do registrador *Range* resultante da divisão de intervalo, até seis passos de renormalização podem ocorrer para tornar o valor do registrador Range novamente maior do que 255. A renormalização também deve ser capaz de lidar com os bits "outstanding" produzidos. A arquitetura básica usa uma ULA e um registrador para contar o número de bits outstanding que são produzidos e não envia nenhum bit valido para o *bitstream* até que um bit 0 ou 1 seja produzido. Uma vez que um bit com estado decidido (zero ou um) é produzido, os bits "outstanding" também são resolvidos (são o inverso do bit decidido) e são enviados um a um para a saída do codificador aritmético. A quantidade de bits outstanding que podem ser produzidos durante o processo de codificação depende dos dados sendo codificados. Tipicamente apenas alguns poucos bits "outstanding" são produzidos até que um bit decidido seja produzido, mas não é possível estabelecer um limite superior. A arquitetura proposta prevê um contador de 8 bits para até 256 bits outstanding em sequência.

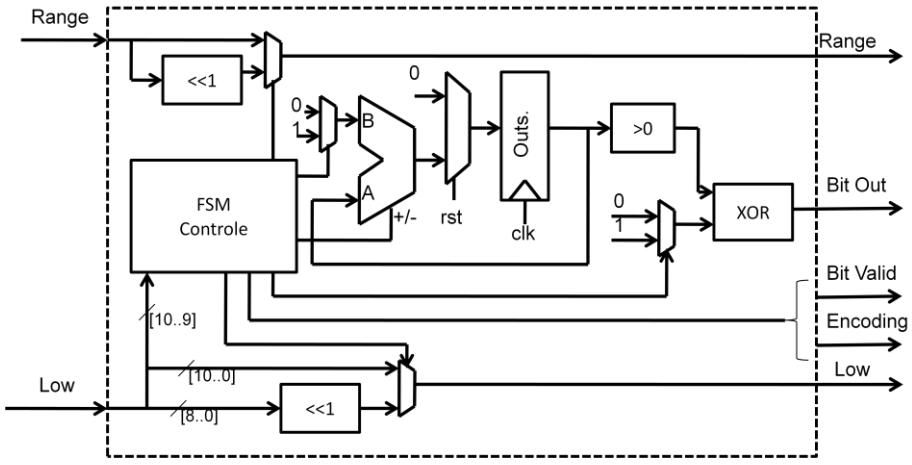

Figura 5.21: Arquitetura sequencial para a renormalização.

Para melhorar o desempenho do codificador aritmético, uma modificação no processo de renormalização foi proposta. Nesta arquitetura, um FIFO e lógica de controle adicional foram adicionados à arquitetura básica. Esta FIFO recebe na entrada 2 bits, um para registrar a produção de um bit outstanding e outro para registrar o bit produzido, caso um bit decidido seja produzido. Um bloco aritmético semelhante à parte de tratamento dos bits "outstanding" na arquitetura anterior é usado. A vantagem aqui é que quando os bits outstanding são resolvidos, o processo de renormalização não é interrompido para enviar a sequência de bits resolvidos para a saída. Ao invés disso, o processo de renormalização continua operando normalmente, parando apenas caso o FIFO fique cheio. Dependendo do tamanho deste FIFO, o tempo necessário para processar os bits "outstanding" pode ser negligenciado, pois este processamento ocorre em paralelo sempre que há espaço no FIFO. Figura 5.22 ilustra esta proposta arquitetural.

Finalmente, uma proposta arquitetural de ciclo único foi desenvolvida. Para tanto, um contador de zeros a frente (LZC – *Leading Zero Count*) é usado no registrador Range para verificar a quantidade de operações de deslocamento que devem ser realizadas tanto nos registradores Range e Low. Nesta arquitetura, um número varável de bits entre zero e seis são produzidos a cada ciclo de *clock*.

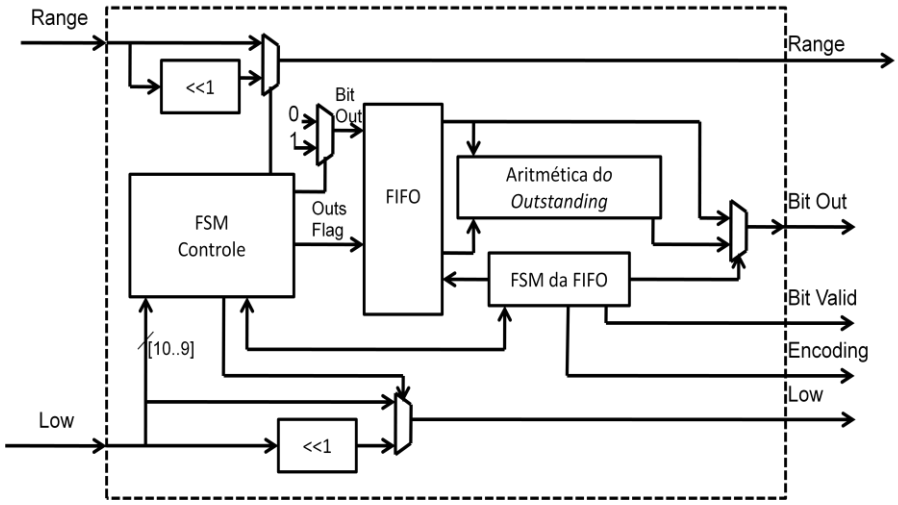

Figura 5.22: Arquitetura "*Pipeline Outstanding*" para aceleração da renormalização.

Neste processo, alguns ou até mesmo todos os bits produzidos podem ser do tipo *outstanding* (OSORIO, 2006) e precisam ser adequadamente resolvidos antes de serem enviados ao *bitstream*. Para manter a compatibilidade com a interface das arquiteturas anteriores a saída de um único bit por ciclo de *clock* foi mantida. Isto é possível pois, na média, o codificador aritmético produz 30% menos bits do que o número de *bins* de entrada. Isso significa que, apesar do processo de renormalização puder produzir até 6 bits em um único ciclo, na maioria dos ciclos de operação do codificador aritmético nenhum bit é produzido. Para exemplificar, foi realizado um experimento usando o software de referência JM11 (SUHRING, 2008), onde os passos de renormalização foram contados para cada *bin* processado em uma amostra de 259736 *bins*. O codificador foi configurado para processar apenas quadros do tipo I com QP=0, produzindo desta forma uma grande quantidade de *bins* oriundos de resíduos de transformadas. Para cada *bin* processado foi contado o total de renormalizações efetuadas chegando-se ao histograma apresentado na Tabela 5.5. Os dados apresentados nesta tabela estão classificados em todos os motores e sem bypass. Nesta tabela observe

que para a maioria dos *bins*, nenhum passo de renormalização foi realizado, mesmo quando os dados do motor bypass estão incluídos, que sempre produz um bit.

Tabela 5.5: Bits produzidos para cada *bin* processado.

Bits produzidos	Todos os motores	% do total	Sem *bypass*
0	134083	51,6	134083
1	102729	39,6	50368
2	11453	4,4	11453
3	7119	2,7	7119
4	2823	1,1	2823
5	1113	0,4	1113
6	416	0,2	416

A Figura 5.23 ilustra graficamente o histograma da Tabela 6.2, com o eixo vertical em escala logarítmica para facilitar a visualização, enquanto a Figura 5.24 apresenta os mesmos dados, descontando os bits produzidos pelo motor *bypass*.

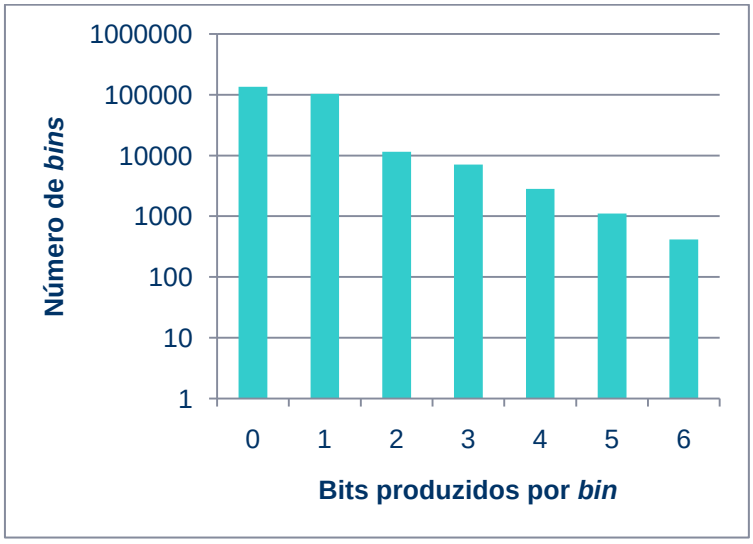

Figura 5.23: Histograma da produção de bits para um conjunto de *bins* de uma amostra de video real.

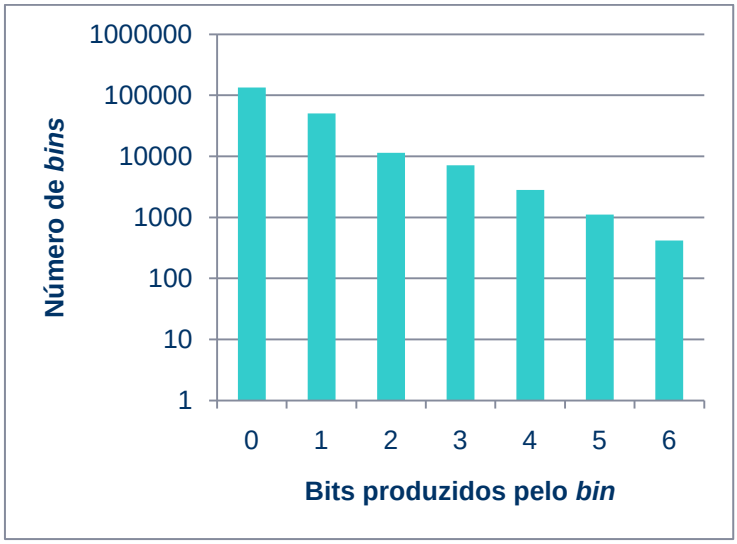

Figura 5.24: Histograma da produção de bits para um conjunto de *bins* de uma amostra de video real – desprezando o motor *bypass*.

Nesta arquitetura, o desafio é escolher uma arquitetura de buffer capaz de lidar com esta taxa variável de produção de bits. A proposta arquitetural apresentada consiste em um buffer FIFO com 3 entradas: duas entradas de 6 bits cada armazenam o estado dos bits produzidos e os marcadores de bits *outstanding*, da mesma forma que na arquitetura com aceleração do tratamento dos bits *outstanding* proposta anteriormente; uma terceira entrada de 3 bits armazena o número de bits válidos presentes na entrada anterior. A largura total da palavra de entrada deste FIFO é de 15 bits.

No lado da saída desta FIFO, uma máquina de estados processa uma palavra da FIFO por vez, inicialmente recebendo o número de bits e marcadores de "outstanding" válidos e começa a produzir bits da mesma forma que na arquitetura sequencial. O tamanho deste FIFO deve ser obtido experimentalmente usando um conjunto variado de sequências de vídeo e modos de codificação para obter o melhor compromisso entre a operação contínua do codificador aritmético e a quantidade de memória usada para implementar este buffer FIFO. Experimentalmente foi identificado que com 128 palavras, uma taxa de processamento de 1 *bin* por ciclo de *clock* pode ser sustentada, exceto em alguns poucos casos.

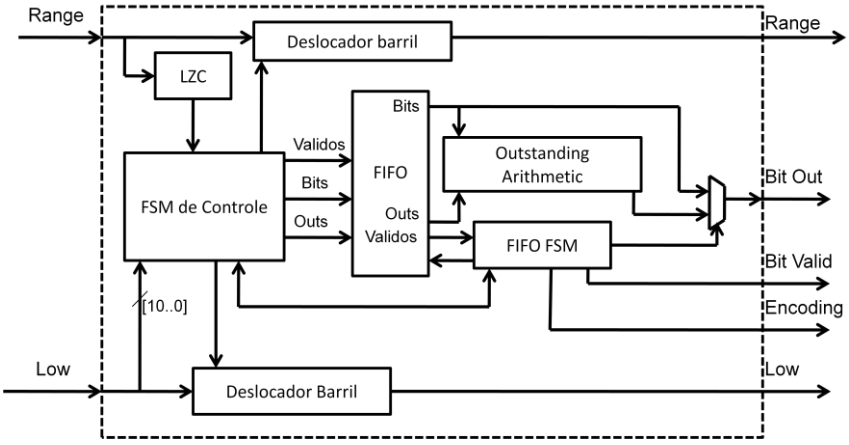

Figura 5.25: Arquitetura de **Ciclo Único** para a aceleração da renormalização.

5.3.3 Resultados

Uma análise do *bitstream* baseada em dados reais do software de referência do codificador (ITU, 2009) foi usado para avaliar a taxa de processamento do codificador aritmético usando cada uma das três arquiteturas propostas para o codificador aritmético. Os dados gerados para uma grande sequência de vídeos com resoluções, qualidades e tipos de cena variados foram empregados. A Tabela 5.6 sumariza estes resultados para um dispositivo FPGA Virtex 5 da Xilix (VIRTEX, 2008). Uma taxa de compressão típica de 1,3 *bins* para cada bit, como observada durante a coleta de dados para este trabalho e também por Deprá (2009), foi empregada para calcular a taxa de bits máxima de cada uma das arquiteturas propostas.

Tabela 5.6: Resultados de síntese em FPGA dos codificadores aritméticos implementados.

Arquitetura	Sequencial	*Pipeline Outstanding*			Ciclo Único
Palavras de FIFO	n/a	4	8	16	128
***Bins*/ciclo**	0,68	0,84	0,86	0,86	1
Frequência Max.	239	234	239	234	189
Registradores	78	95	101	95	91
LUT	161	225	226	234	436
M*Bins*/s	163	197	206	201	189
MBits/s	125	151	158	155	145

Os projetos também foram sintetizados para ASIC usando uma tecnologia 0.18 standard-cells da TSMC (ARTISAN, 2003) e a ferramenta Cadence RTL Compiler (CADENCE, 2010). Os resultados de síntese ASIC são mais facilmente comparáveis com os resultados da literatura e mostram que as arquiteturas desenvolvidas tem capacidade de processaemto para o nível 5 do padrão H.264/AVC, com uma boa eficiência tanto em termos de área quanto de consumo. As extensões SVC e MVC aumentam o limite de bits permitidos em cada um dos níveis do H.264. Ainda assim, é possível utilizar a implementação da versão Sequencial proposta e atingir requisitos para alta definição - nível 5 do padrão H.264/MVC (ITU, 2009).

Além dos dados de síntese para as alternativas do codificador aritmético implementadas,também foi gerado resultado de síntese para o núcleo do CABAC utilizando a implementação Sequencial do codificador aritmético. Os resultatos de síntese são apresentados na Tabela 5.8 e 5.9.

Os resultados de síntese do núcleo apontam para uma elevada frequência de operação (193MHz) apesar da maior complexidade desta arquitetura. Devido ao foco em FPGA o desempenho desta arquitetura é favorecido pela utilização de blocos de memória RAM (BRAM) e pela utilização de um multiplicador embarcado para a geração dos valores iniciais dos contextos. A taxa de processamento alcançada é suficiente para atingir o nível 5 da norma H.264, sendo suficiente para processar com

folga vídeos em alta-definição no formato 1080p e, com isso, reduzindo os requisitos de *buffers* devido a produção variável de *bins* e bits pelo CABAC.

Tabela 5.7: Resultados de síntese em ASIC dos codificadores aritméticos implementados.

Arquitetura	Sequencial	*Pipeline Outstanding*	Ciclo Único	Wu (2009)	Tian (2009)
Palavras FIFO	None	4	128	n/d	n/d
***Bins*/ciclo**	0.68	0.84	1	2.37	n/d
Frequência Max.	294	313	250	222	200
Gates (k)	1.2	2.8	31.4	14.7	n/d
Potência (mW)	1.9	3.4	9.6	n/d	5.5
M*Bins*/s	200	263	250	526	328
MBits/s	154	202	192	404	252
Eficiêcia em área (K*Bins*/s)/Gate	167	94	8	36	n/d
Eficiência energética M*Bins*/J	105	77	26	n/d	60

Tabela 5.8: Resultados de síntese em FPGA do núcleo do CABAC.

Arquitetura	Núcleo Sequencial
Dispositivo	XC5VLX30-3
***Bins*/ciclo**	0.68
Frequência Max.	193
LUTs	593
Registradores	263
BRAMs	4
DSP48E	1
MBins/s	131
MBits/s	101

Na arquitetura do núcleo não há ciclos de espera além do período de inicialização dos contextos, por isso foi considerado que a taxa média de processamentos em *bins*/ciclo de *clock* permanece idêntica a da arquitetura "Sequencial" do codificador aritmético. Neste processo de síntese ASIC, a máxima frequência de *clock* foi penalizada pela implementação do multiplicador utilizado na incialização dos contextos, que no FPGA utiliza um elemento DSP48E do FPGA para a síntese do multiplicador. Por esta razão, a frequência de *clock* possível acabou sendo penalizada na síntese ASIC, de forma diferente do que aconteceu na síntese para FPGA.

Tabela 5.9: Resultados de síntese em ASIC do núcleo do CABAC.

Arquitetura	Núcleo Sequencial
Tecnologia	TSMC 0,18µm
Palavras FIFO	None
***Bins*/ciclo**	0,68
Frequência Max.	166
Memória SRAM (kbits)	7
Gates (k)	14,7
Potência (mW)	23
M*Bins*/s	113
MBits/s	87

5.3.4 Codificador CABAC completo

O prosseguimento deste trabalho prevê o desenvolvimento de uma arquitetura completa para o CABAC, baseado na concepção apresentada neste trabalho e deverá ser resultado de uma dissertação de mestrado a ser concluída em breve. A seguir são apresentados dados preliminares da implementação RTL que está em desenvolvimento em co-autoria com o mestrando André Martins.

Baseado nas decisões arquiteturais globais para o CABAC apresentadas no diagrama da Figura 5.13, e da interface do núcleo do CABAC, apresentada na Figura 5.14, uma proposta arquitetural para os demais módulos do CABAC foi desenvolvida, conforme apresentada na Figura 2.24. No momento da conclusão deste texto, a implementação de todo o CABAC, com suporte ao perfil *main* estava concluída.

Comparando a proposta original realizada neste trabalho (Figura 5.13) com a proposta apresentada na Figura 5.26, observa-se que o "Buffer de entrada" da primeira encontra-se no primeiro macro-estágio de pipeline, delimitado pela primeira FIFO. O "Buffer Bin" está no macro-estágio de pipeline intermediário, juntamente com o acesso a informações de vizinhança e controle. Finalmente, o "Acesso e atualização dos contextos" foi denominado contextualizador, entregando como saída o *bin* a ser codificado, o contexto associado e o tipo de codificador a ser empregado (Regular, Bypass ou Terminate). O controle de fluxo de dados não estão representados na Figura 5.26, mas estão previstos para evitar perda de informação no processo, visto que todos os módulos operam com taxas de dados variáveis, estando limitado um elemento sintático por ciclo de clock, um **bin** por ciclo de clock, e um **bit** por ciclo de clock.

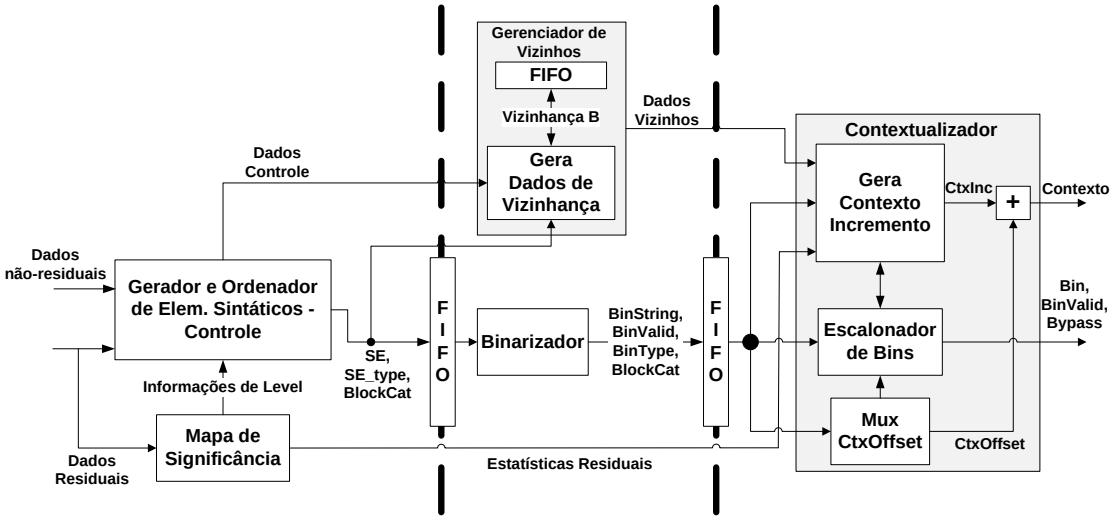

Figura 5.26: Arquitetura do Binarizador-Contextualizador.

A hierarquia RTL da implementação completa do CABAC é apresentada na Figura 5.27, e os resultados de síntese na Tabela 5.10. Este trabalho ainda encontra-se em desenvolvimento e a proposta arquitetural está posta neste texto para fins apenas ilustrativos, pois a parte denominada Binarizador-Contextualizador não foi amplamente validada e esta validação deu-se no âmbito do grupo de pesquisa da UFRGS através do trabalho de mestrado em desenvolvimento no PPGC-UFRGS.

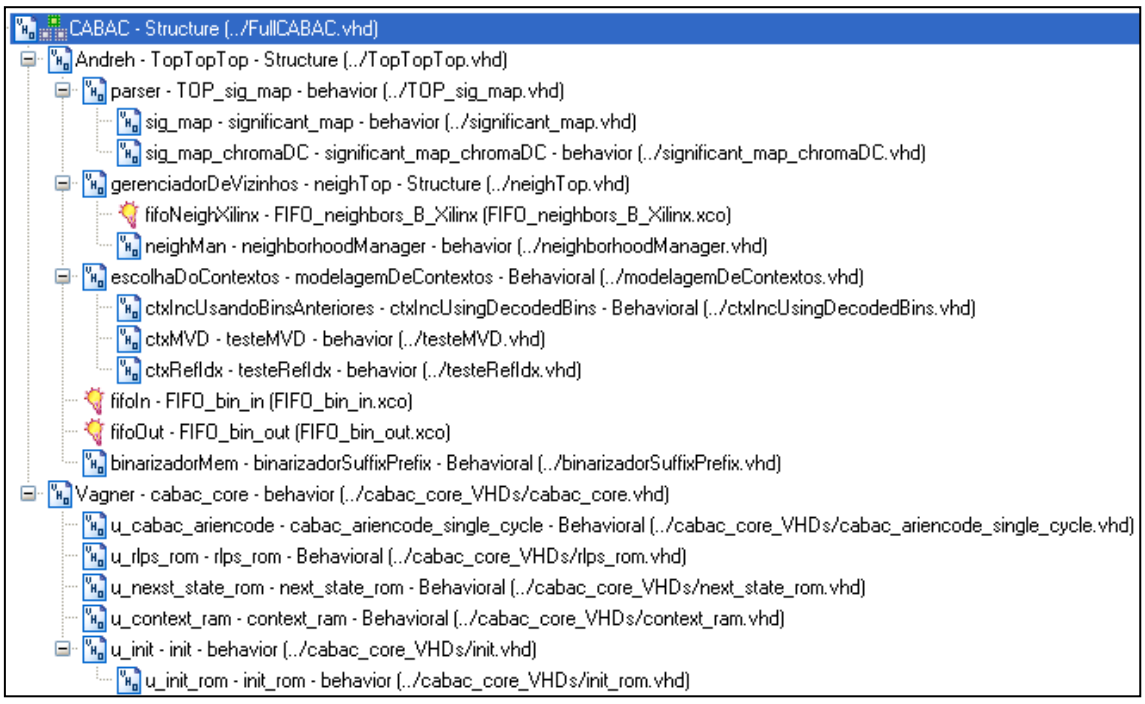

Figura 5.27: Estrutura hierárquica do RTL do CABAC, usando o codificador aritmético **Ciclo Único**.

O suporte a todos os perfis e extensões do padrão H.264 para o CABAC está em desenvolvimento. No núcleo, todos os perfis e extensões podem ser suportados, com a utilização dos 1024 contextos. Para suportar os 7 contextos adicionais do SVC, uma

realocação na tabela de inicialização de contextos pode ser realizada para remapear contextos não utilizados por esta extensão. No MVC não há novos elementos sintáticos a serem suportados. Em todos os casos (perfis *High*, SVC e MVC) o suporte depende do reconhecimento dos elementos sintáticos e associação a números de contextos apropriados.

5.3.5 Conclusões do CABAC

O projeto do CABAC foi guiado pelos resultados obtidos no CABAD, levando a ao projeto de uma arquitetura em pipeline do codificador aritmético capaz de processar até um *bin* por ciclo. Uma avaliação de alternativas de aceleração do processo de renormalização foi realizada, mostrando que a arquitetura mais simples é também a mais eficiente em área e potência. O desempenho alcançado é suficiente para codificar as taxas de bits necessárias para o nível 5 do padrão H.264/AVC. O núcleo do CABAC, construído após a concepção do codificador aritmético é capaz de sustentar o desempenho alcançado por este através de um pipeline eficiente para o acesso e atualização dos contextos e possui suporte a todos os modelos de contexto necessários para todas as extensões do padrão até o MVC.

A possibilidade de trabalhar em uma frequência mais elevada e sem paralelismo no nível de *bins* também traz vantagens para o núcleo do codificador, eliminando a maior parte da complexidade necessária para o acesso e atualização dos contextos sem tornar esta parte o gargalo de processamento do CABAC. Com o uso de paralelismo acima do nível de slice, através do uso de múltiplas instâncias do codificador proposto, é possível suportar virtualmente qualquer taxa de processamento exigida para o CABAC.

O núcleo do CABAC pode ainda ser otimizado, através das seguintes modificações, que incluem algumas restrições ao funcionamento:

- Eliminação dos contextos não necessários para o cojunto de perfis suportados (caso a aplicação não suporte todos os perfis – ex. High 4:4:4);
- Uso de QPY fixo para a inicialização dos contextos, removendo a necessidade do módulo “Init” no núcleo do codificador (incluindo o multiplicador);
- Exploração de alternativas de implementação do multiplicador usado no módulo “init”, com a avaliação do compromisso entre área e desempenho do multiplicador.

O CABAC completo incui o projeto do módulo denominado Binarizador-Contextualizador, descrito na Seção 5.3.4 e um módulo de geração NAL que basicamente coloca um cabeçalho no *bitstream*, faz com que o último bit do *bitstream* esteja no alinhamento de byte (múltiplo de 8 bits) e insere um elemento sintático “emulation_prevention_three_byte” para evitar a aparição acidental do marcador de cabeçalho no *bitstream*, facilitando o sincronismo para o decodificador.

6 PLATAFORMA DE VALIDAÇÃO

No desenvolvimento dos módulos do decodificador H.264, os dados para a validação foram obtidos através do software de referência do padrão (SUHRING, 2008). Isso exigiu um estudo extensivo do software de referência para incorporar as modificações necessárias para a obtenção dos dados intermediários do processo de decodificação necessários para validar cada módulo individualmente. Para gerar os dados, uma quantidade expressiva de sequências de vídeo codificadas foram processadas pelo software de referência modificado para gerar uma expressiva quantidade de dados para a validação com vídeos reais.

A ferramenta ModelSim da Mentor Graphics (MODELSIM, 2008) foi usada para rodar as simulações necessárias para a validação dos módulos do decodificador. Arquivos de *testbench* capazes de ler os estímulos de entrada a partir de arquivos de dados e gerar arquivos de saída para posterior comparação com os dados de referência gerados pelo software de referência foram gerados para cada um dos módulos validados.

Em um primeiro passo de validação, uma simulação comportamental da descrição VHDL dos módulos desenvolvidos é realizada. Durante o desenvolvimento, dados sintéticos são usados para estimular comportamentos específicos do módulo em desenvolvimento. Então o módulo é instanciado no *testbench* descrito acima e o módulo é testado por simulação comportamental com dados das sequências de vídeo de teste. Após a conclusão da simulação comportamental, um modelo *post-place&route* do módulo é gerado e este inserido no mesmo conjunto de *testbenchs* usados para a simulação comportamental. Esta simulação é significativamente mais lenta que a primeira e por isso deve ser feita apenas quando todos os problemas identificados na simulação comportamental já estiverem resolvidos.

Após a finalização das simulações, a validação em um sistema com dispostivo FPGA físico pode começar. Para a prototipação dos módulos do decodificador H.264 foi escolhida a placa XUP-V2pro da Xilinx, com o FPGA XC2VP30 (VIRTEX, 2008). O modelo *post-place&route*, que inclui dados de atraso devido a alocação de células e ao roteamento final dos sinais, é obtido através do uso da ferramenta ISE (ISE, 2008). Esse modelo contém o mapeamento da lógica do módulo desenvolvido para as primitivas do FPGA, incluindo o seu posicionamento, roteamento e os atrasos envolvidos. Por essa razão a simulação do modelo *post-place&route* é muito mais lenta que a do modelo comportamental. Este modelo contém estimativas dos atrasos baseada na caracterização do dispositivo alvo e há razoável confidência que um projeto validado nesta fase irá rodar no dispositivo alvo. Entretanto o elevado tempo de simulação (pode levar vários dias para arquiteturas que usam muita lógica e muitos dados de entrada) a possibilidade de avaliação extensiva é limitada. Essa limitação levou ao

desenvolvimento de mais um passo no processo de validação que é a validação dos módulos no hardware, de forma a acelerar o processo de validação para módulos grandes com uma extensiva quantidade de dados de entrada.

6.1.1 Prototipação dos módulos do H.264

Para superar a limitação imposta pelo demorado tempo de simulação do modelo *post-place&route*, uma estratégia de prototipação foi desenvolvida para validar individualmente todos os módulos de um decodificador H.264. Ao invés de usar técnicas comerciais complexas de validação em hardware, a abordagem de prototipação foi baseada nos recursos disponíveis na família Virtex II pro de FPGAs e também na placa de desenvolvimento XUP-V2pro da Digilent (DIGILENT, 2008). A placa XUP-V2pro contém um FPGA Xilinx XC2VP30, com 27 mil LUTs e dois processadores Power PC (VIRTEX, 2008). Essa placa também contém uma série de interfaces físicas, incluindo porta serial, saída VGA, conector para módulos de memória DDR DIMM, dentre outros recursos. Para a estratégia de prototipação desenvolvida, um módulo de memória DDR de 512MB foi instalado.

A estratégia adotada para a prototipação dos módulos foi o uso de um processador PowerPC disponível no FPGA da placa de desenvolvimento XUP-V2pro para comunicação entre um computador chamado *Host_PC* e uma interface do módulo a ser validado (chamado MUV – *Module Under Validation*). Um sistema baseado em processador foi construído e usado para obter os estímulos de entradas do módulo em validação a partir de um computador, estimular o módulo em validação (MV) e enviar os dados colhidos da saída do MV de volta ao computador para comparação com os dados de saída obtidos a partir do software de referência.

A ferramenta EDK da Xilinx (PLATFORM, 2008) foi usada para criar uma plataforma de prototipação completa. Neste processo, cada módulo do decodificador H.264 desenvolvido foi conectado a um barramento do processador e prototipado individualmente. O processador *PowerPC* embarcado foi usado como um controlador para o módulo prototipado, emulando os outros módulos do decoder. Os estímulos de entrada são enviados para a plataforma de prototipação através da porta RS232, usando um programa de terminal no PC. O processador da plataforma de prototipação interpreta os dados recebidos e envia os estímulos para o MUV. Ao mesmo tampo também coleta os dados de saída do módulo e armazena temporariamente em memória para posterior envio para o computador para que a comparação com o modelo de referência seja realizada.

Desta forma, a validação *post-place&route* pode ser reduzida, uma vez que a validação em placa FPGA pode ser realizada. Em caso de falha do processo de validação em placa, uma validação subsequente por simulação do modelo *post-place&route* pode ser realizada para delimitar o problema no RTL.

6.1.2 Validação funcional

O processo de prototipação é dividido em dois estágios. No primeiro estágio, a validação começa com uma verificação de funcionalidade. A plataforma de prototipação baseado no processador PowerPC é sintetizada em conjunto com o MUV e todos os estímulos de entrada do MUV, incluindo o sinal de relógio (*clock*) são enviados através do barramento do sistema. A saída do MUV é amostrada diretamente pelo processador através do barramento do sistema. Como o relógio é gerado por software, o processador da plataforma pode funcionar de forma arbitrariamente lenta. A Figura 6.1 ilustra esta abordagem para a prototipação.

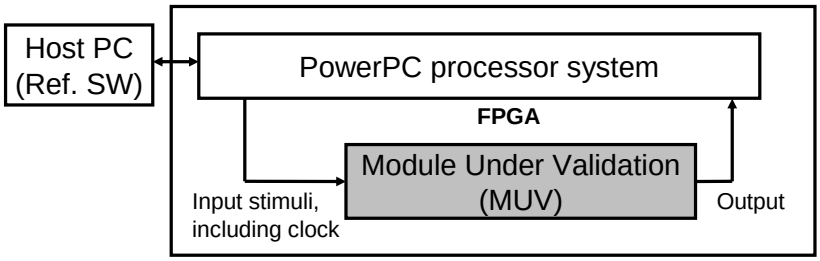

Figura 6.1: Abordagem para a validação funcional do protótipo (ROSA, 2007).

Como nesta abordagem o sinal de *clock* do MV é gerado por software, os caminhos críticos do MUV não são testados, mas a conectividade e a funcionalidade do módulo instanciado são verificadas, assim como o software embarcado que prepara os estímulos de entrada e colhe os resultados, pois cada sinal pode ser analisado a cada ciclo de *clock* sem restrições de tempo de processamento pelo processador embarcado. Isto se mostrou importante durante a prototipação dos módulos do decodificador H.264, pois erros no MUV por causa de timing durante a síntese (que levariam a um não funcionamento na frequência esperada) podem ser separados de erro de lógica, conectividade e no software embarcado que prepara os estímulos.

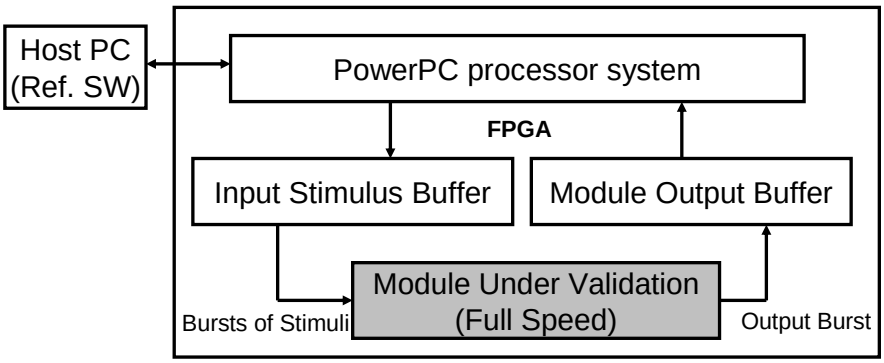

Figura 6.2: Validação em velocidade plena (ROSA, 2007).

6.1.3 Validação na frequência nominal de operação

O segundo estágio da prototipação é a validação do MUV na sua frequência nominal. A Figura 6.2 ilustra a arquitetura para este processo de validação. Os estímulos de entrada são enviados do processador para um buffer de entrada a uma velocidade arbitrariamente baixa. Este buffer armazena estímulos suficientes para um surto de operação na frequência nominal. Quando este buffer está cheio, os estímulos armazenados são enviados para o MUV na velocidade nominal de operação do módulo. Ao mesmo tempo, um buffer de saída irá armazenar os resultados produzidos pelo MV, que poderão ser recolhidos posteriormente pelo processador embarcado. O tamanho deste buffer deve ser compatível com a quantidade de dados produzida durante um surto de operação e está diretamente ligado ao tamanho do buffer de entrada. Como estes buffers devem operar na velocidade nominal do MUV, estes devem ser instanciados em blocos de memória interna ao FPGA. O número de ciclos do surto de validação é limitado pelo tamanho dos buffers de entrada e saída e estes são limitados pela quantidade de memória distribuída disponível no FPGA.

6.1.4 Detalhamento da arquitetura da plataforma de validação

Para facilitar o desenvolvimento da plataforma de validação, foi estabelecido desenvolvimento da menor quantidade possível de módulos, valendo-se de todos os recursos disponíveis no pacote de desenvolvimento de sistemas embarcados EDK (PLATFORM, 2008). A Figura 6.3 apresenta a arquitetura de hardware desenvolvida. Nesta arquitetura, todos os módulos, exceto o módulo "IP interface" e o módulo "BRAM Buffer Interface" foram obtidos a partir do pacote EDK.

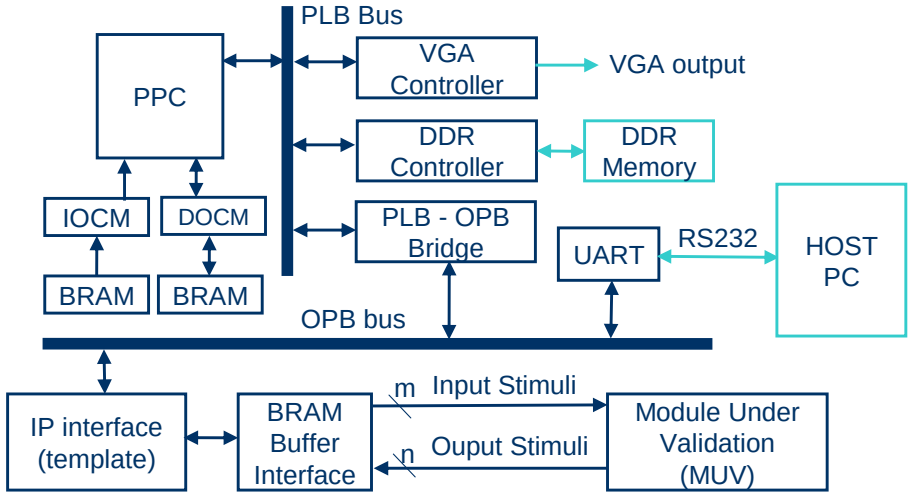

Figura 6.3: Detalhes da arquitetura da plataforma de prototipação (ROSA, 2007).

Nesta arquitetura, um dos dois processadores PowerPC (PPC) hard-core do XC2VP30 (VIRTEX (2008)) foi utilizado. Este processador está conectado a três dispositivos de transferência de dados distintos: PLB Bus (*Processor Local Bus* – Barramento Local do Processador), IOCM (*Instruction On-chip Memory* – Memória de instruções interna ao dispositivo) e DOCM (*Data On-chip Memory* – Memória de dados interna ao dispositivo).

O IOCM e o DOCM são canais dedicados a instruções e a dados do programa do PPC, respectivamente. Desta forma o processador pode executar mais rapidamente sem ter que concorrer pelo acesso ao barramento principal (PLB), onde estão ligados os demais periféricos. Os módulos IOCM e DOCM são na verdade controladores de interface entre a memória BRAM e o processador. A quantidade de memória de cada um destes módulos BRAM depende da disponibilidade de entidades físicas de BRAM disponíveis no FPGA e de disponibilidade de canais de roteamento. Como as entidades físicas de BRAM estão distribuídas no FPGA, a utilização de módulos de memória BRAM muito grandes pode gerar problemas de congestionamento de roteamento no FPGA, prejudicando o desempenho global do sistema.

O barramento PLB é um barramento de 64 bits com suporte a múltiplos mestres e arbitragem e na plataforma de prototipação desenvolvida está conectado ao processador e a outros três módulos: DDR Controller, que é o gerenciador de memória externa DDR (*Double Data Rate* – Taxa de Dados Dobrada) DRAM (*Dynamic Random Access Memory* – Memória de Acesso Aleatório Dinâmica), o VGA Controller, que é o adaptador para a saída de vídeo no formato VGA 640x480x60Hz e o PLB-OPB Bridge, que fará a ponte entre o barramento PLB e o OPB (*On-chip Periferal Bus* – Barramento de periféricos internos ao dispositivo).

O controlador de memória DDR irá conectar o barramento PLB a uma memória DDR DRAM externa ao dispositovo FPGA (instalada na placa de prototipação), permitindo maior capacidade de armazenamento. Na plataforma desenvolvida, esta memória é composta por um módulo DIMM (*Dual In-line Memory Module* – Módulo de memória em linha dupla) com 256MB ou 512MB, com 64 bits de largura operando em taxa DDR a uma frequência de 100MHz. Esta memória é utilizada principalmente para o armazenamento temporário de dados de entrada e saída específicos para os módulos a serem validados, além de um buffer de quadros para a interface VGA.

O Controlador VGA irá gerar os sinais necessários para a produção de imagem em um dispositivo FPGA. O Sinal de vídeo é produzido a partir de informações lidas de uma memória de quadros (*Frame Buffer*) obtido a partir do barramento PLB. O Controlador VGA é um dos mestres do barramento PLB (o outro é o próprio processador) e deve ter prioridade de acesso ao barramento, sob pena de haver artefatos na exibição do sinal de vídeo por indisponibilidade de dados. A memória de quadros é composta por um segmento da memória externa, iniciando em uma posição configurável.

A ponte PLB-OPB ira fazer a conexão entre o barramento PLB, de 64 bits, com o PLB de 32 bits. O PLB é um barramento de alta velocidade; para maximizar a frequência de operação, apenas uns poucos dispositivos de mais alta velocidade devem estar conectados a ele. O OPB é um barramento mais lento, que pode operar de forma concorrente ao barramento PLB e permitir que dispositivos lentos sejam conectados sem afetar o desempenho dos dispositivos conectados ao barramento PLB. Além disso, alguns módulos do EDK são fornecidos apenas com a opção de interface OPB (UART, por exemplo) impedindo a eliminação deste barramento do sistema quando este módulo é usado.

No barramento OPB estão conectados os módulos PLB-OPB Bridge, descrito acima, o UART (*Universal Assincronous Receive Transmitter* – Receptor/Transmissor Assíncrono Universal) e o IP Interface (*Intellectual Property Interface* – Interface de Propriedade Intelectual). Este barramento opera na mesma frequência do PLB, porém o acesso pode demorar vários ciclos de *clock*.

O UART está configurada para ser conectada a um computador PC e pode operar a uma taxa máxima de 115200bps. Na plataforma desenvolvida, este módulo é utilizado para a transferência de dados entre o PC e a plataforma. Apesar de lento, os estímulos para validação podem vir comprimidos e pré-processados por um software rodando no PPC embarcado específico para esta finalidade.

O IP Interface é, na realidade, um modelo de módulo de usuário que implementa todo o mecanismo de interface de barramento, disponibilizando uma interface parametrizável mais simples que abstrai a presença da maioria dos sinais de controle do barramento: As posições endereçáveis são acessíveis por meio de registradores. Na plataforma desenvolvida, uma pequena quantidade de código VHDL foi escrito para implementar modelos de interface para a validação funcional e para a validação na velocidade nominal (através do uso de *buffers* de estímulos). Este módulo deve ser modificado, juntamente com o software do PowerPC para refletir as necessidades de sinais de cada módulo prototipado.

Finalmente o módulo prototipado (MUV) recebe o seu sinal de *clock* através de estímulos de software (para a validação funcional) ou através do barramento OPB (para

a validação em velocidade nominal), além dos “m” sinais de entrada. Os “n” sinais de saída são coletados para comparação com a referência de simulação ou software.

6.1.5 Aplicação da metodologia

Durante a fase de validação, o processador PowerPC pode realizar outras tarefas, como a composição de uma imagem a partir dos estímulos de entrada ou de saída quando dados de pixels estão envolvidos nos estímulos. Este é o caso de módulos como a predição intra, a compensação de movimento e o Filtro de Deblocagem. Na plataforma de prototipação, um frame buffer VGA foi sintetizado e alguns resultados visuais puderam ser enviados diretamente da plataforma de prototipação para um monitor de vídeo. A Figura 6.4 apresenta o exemplo de uma saída de vídeo da saída do módulo de compensação de movimento sendo prototipado na plataforma desenvolvida. O vídeo de saída é na verdade exibido como uma sequência de figuras porque não é possível atingir tempo real durante a exibição devido à necessidade de constante intervenção do processador para processar estímulos de entrada e saída do MV.

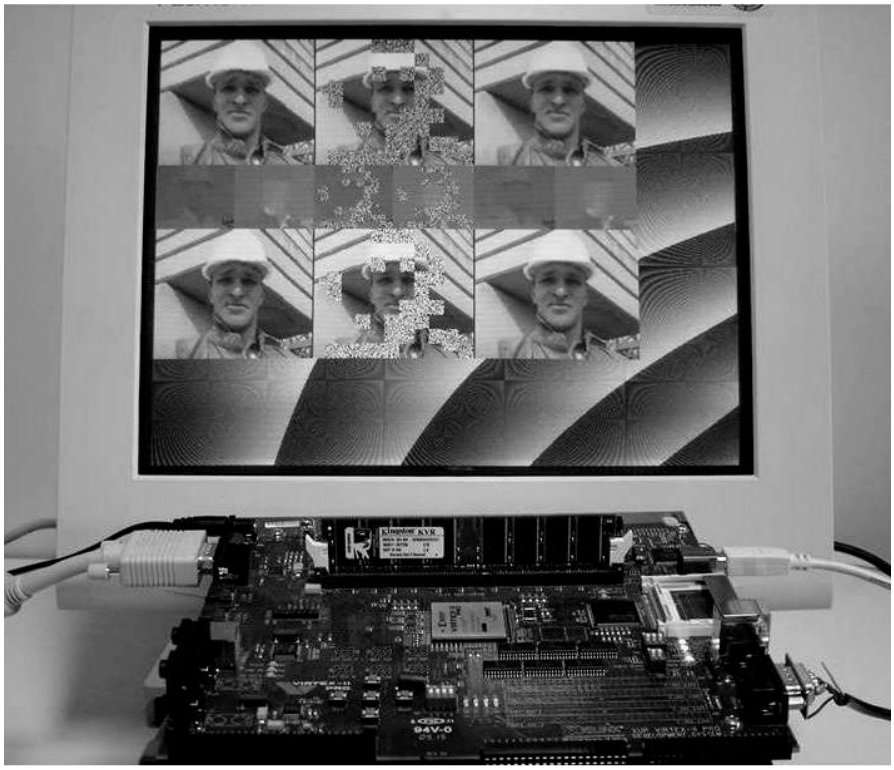

Figura 6.4: Prototipação do módulo da compensação de movimento.

O Filtro de Deblocagem, uma das arquiteturas apresentadas nesta tese foi prototipado usando esta estratégia,

O próximo passo do desenvolvimento de um protótipo de um decodificador H.264 é a integração dos módulos. Os três caminhos principais do decodificador, como apresentados na Figura 3.6 são a compensação de movimento, a predição intra e o grupo formado pela decodificação de entropia, a transformada e a quantização inversa. Estes caminhos de dados principais rodam na mesma frequência de relógio (ou velocidade de pixels, quando aplicável) na integração final do decodificador, sendo a velocidade de processamento limitada pelo módulo mais lento. A mesma abordagem aplicada para a verificação por prototipação dos módulos individuais pode então ser empregada para

verificar os módulos integrados. A Figura 6.5 mostra a integração parcial dos do H.264 composta pelos módulos da Predição Intra, T^{-1}, Q^{-1} e o Filtro de Deblocagem.

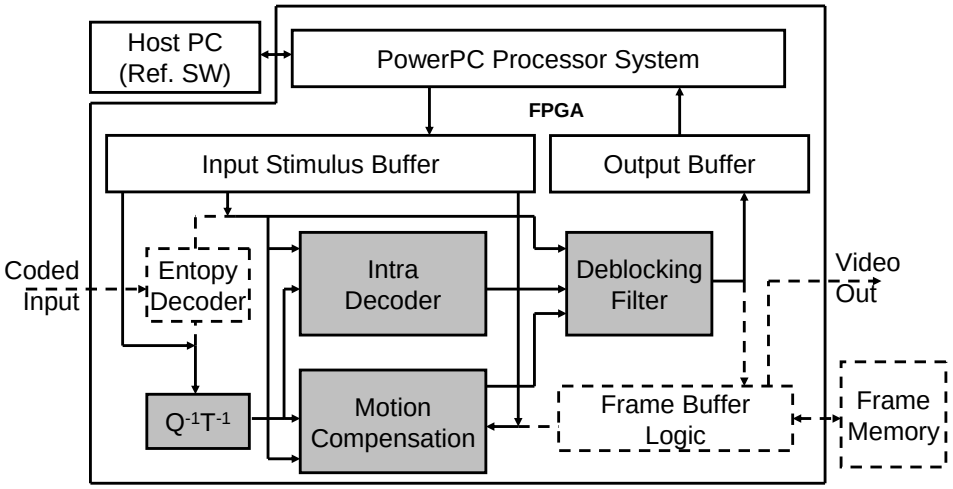

Figura 6.5: Exemplo de prototipação da integração de alguns módulos do codificador H.264 (ROSA, 2007).

Da mesma forma que na validação dos módulos individuais, na validação de integração entre módulos, os módulos ainda não integrados (exemplificado através de linhas pontilhadas na Figura 6.5) são simulados pela estrutura de prototipação desenvolvida. Com todos os módulos integrados, a plataforma de prototipação não é mais necessária e pode ser retirada do sistema. Neste caso, um FPGA sem processador pode ser usado para outro propósito no sistema de decodificação de vídeo padrão H.264.

O Filtro de Deblocagem, uma das arquiteturas apresentadas nesta tese foi prototipado usando esta estratégia, usando a saída VGA para visualização direta do resultado. Uma fotografia do sistema é apresentada na Figura 6.6.

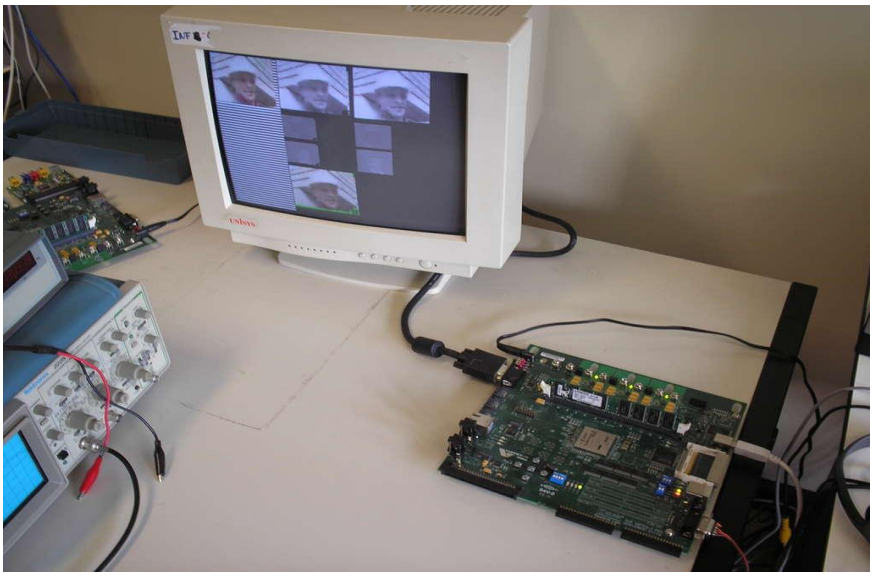

Figura 6.6: Prototipação do Filtro de Deblocagem.

A arquitetura completa do CABAC também será prototipada neste sistema quando concluída. Neste caso, o resultado não poderá ser exibido na tela através da saída VGA pois não há formação de imagem. Entretanto, códigos de verificação intermediários

poderão ser enviados para a saída VGA e visualizados imediatamente, através de codificação numérica ou por forma de onda.

6.1.6 Conclusões

Esta estratégia se mostrou simples e eficaz para a validação dos módulos FPGA sem a necessidade de aquisição de hardware/software adicional para o processo de validação. A proposta apresentada pode ser utilizada para a validação de módulos de outro sistema complexo qualquer.

7 CONCLUSÕES

A presente tese apresentou o projeto de arquiteturas para os móulos CABAC e Filtro de Deblocagem do H.264, além de uma estratégia para a prototipação em tempo real em FPGA para ser usada de módulos do H.264. A escolha destes módulos está de acordo com os trabalhos já realizados pelo autor desta tese e pelo grupo e pelo grupo de pesquisa da UFRGS, descritos na Seção 1.1, assim como com os trabalhos atualmente em andamento.

Como apresentado no Capítulo 3, o padrão H.264 possui elevada complexidade e apenas o estudo das ferramentas suportadas envolveu muito tempo de estudo. O trabalho desenvolvido apresenta resultados concretos que extendem o estado da arte. Além disso, uma sólida proposta de validação em FPGA e um estudo avançado do hardware de prototipação e a experiência adquirida com o desenvolvimento de trabalhos anteriores e pelo grupo, permitiram um melhor direcionamento no desenvolvimento das propostas arquiteturais, sua implementação e validação.

A arquitetura proposta no Capítulo 4 para o Filtro de Deblocagem é distinta da maioria dos trabalhos da literatura por considerar a integração das memórias necessárias para o armazenamento dos dados intermediários da filtragem. Este trabalho foi focado no desenvolvimento de uma arquitetura que pudesse ser rapidamente integrada no protótipo de um decodificador ou codificador de vídeo em FPGA. Por isso, o foco inicial do desenvolvimento deste trabalho foi a otimização para síntese em FPGA. A inclusão das memórias *Line Buffer* e *MB Buffer* na arquitetura resultou em uma utilização harmônica dos recursos do FPGA, sem acréscimo na lógica. A síntese ASIC mostrou que o desempenho é compatível com outras implementações da literatura, entretanto as memórias são o principal gargalo em área para a o Filtro de Deblocagem. Apesar disso, é possível superar o requisito de processamento para vídeos HDTV (1080p) para ambas as implementações (FPGA e ASIC). A arquitetura proposta foi concebida para trabalhar apenas com vídeo no formato progressivo até a resolução horizontal de 2048 pixels no formato 4:2:0 com amostras a partir de 8 bits. Trabalhos futuros irão considerar o suporte a outros formatos de vídeo suportados pelos perfis *high* do padrão H.264, assim como vídeo entrelaçado codificado com suporte a MBAFF. Otimizações arquiteturais para a otimização do uso das memórias e avaliação do desempenho do uso de memórias externas para substituir as maiores memórias poderão ser avaliados.

O Capítulo 5 apresentou o desenvolvimento do CABAC, iniciando pelo decodificador (CABAD) e seguido pelo codificador (CABAC). No CABAD, os resultados apresentados mostram que há um ganho de desempenho na codificação dos *bins* do modo *bypass* comparado ao trabalho considerado o estado-da-arte da literatura no momento de sua concepção (YU, 2005). No entanto este ganho é menor considerando-se a taxa média de *bins* produzidos incluindo os do motor regular e o

acréscimo de complexidade envolvido nessa solução. Em especial, considerando que o debinarizador não foi alvo deste desenvolvimento e que o cálculo do índice do próximo contexto pode levar a erros de predição e necessidade de recomputação, o desempenho da arquitetura completa poderá ser um pouco penalizado. Isso indica que um aumento futuro no paralelismo do decodificador pode não trazer benefícios. Os próximos passos no desenvolvimento do CABAD são a integração do BAD com o debinarizador e demais módulos do decodificador de entropia, de forma a obter um decodificador de entropia completo para o H.264. Alguns cenários mais específicos de uso do decodificador também poderiam ser analisados (apenas HDTV, apenas CIF, muito ou pouco movimento nas cenas, apenas intra, etc..) para melhor avaliar a arquitetura desenvolvida. O projeto do CABAC foi guiado pelos resultados obtidos no CABAD, levando a ao projeto de uma arquitetura em pipeline do codificador aritmético capaz de processar até um *bin* por ciclo. Uma avaliação de alternativas de aceleração do processo de renormalização foi realizada, mostrando que a arquitetura mais simples é também a mais eficiente em área e potência. O desempenho alcançado é suficiente para codificar as taxas de bits necessárias para o nível 5 do padrão H.264/AVC. O núcleo do CABAC, construído após a concepção do codificador aritmético é capaz de sustentar o desempenho alcançado por este através de um pipeline eficiente para o acesso e atualização dos contextos e possui suporte a todos os modelos de contexto necessários para todas as extensões do padrão até o MVC. A possibilidade de trabalhar em uma frequência mais elevada e sem paralelismo no nível de *bins* também traz vantagens para o núcleo do codificador, eliminando a maior parte da complexidade necessária para o acesso e atualização dos contextos sem tornar esta parte o gargalo de processamento do CABAC. Com o uso de paralelismo acima do nível de slice, através do uso de múltiplas instâncias do codificador proposto, é possível suportar virtualmente qualquer taxa de processamento exigida para o CABAC. O CABAC completo incui o projeto do módulo denominado Binarizador-Contextualizador, descrito na Seção 5.3.4 e um módulo de geração NAL que basicamente coloca um cabeçalho no *bitstream*, faz com que o último bit do *bitstream* esteja no alinhamento de byte (múltiplo de 8 bits) e insere um elemento sintático "emulation_prevention_three_byte" para evitar a aparição acidental do marcador de cabeçalho no *bitstream*, facilitando o sincronismo para o decodificador.

Uma estratégia de prototipação foi apresentada no Capítulo 6. Esta estratégia se mostrou simples e eficaz para a validação dos módulos FPGA sem a necessidade de aquisição de hardware/software adicional para o processo de validação. A proposta apresentada pode ser utilizada para a validação de módulos de outro sistema complexo qualquer.

Como resultado desta tese, 9 publicações como autor ou co-autor relativas ao trabalho de doutorado, sendo 8 em conferências (ROSA, 2007), (ROSA, 2007a), (ROSA, 2009), (ROSA, 2010), (DEPRA, 2008), (DEPRA, 2008a), (DEPRA, 2008b), (AGOSTINI, 2006), e um artigo para periódico internacional (AGOSTINI, 2007a). No momento da finalização deste texto, há dois artigos aprovados para o ICECS 2010, um tratando da exploração de espaço de projeto do codificador aritmético (Seção 5.3.2) e outro em co-autoria tratando de uma arquitetura para o binarizador. Também foi submetido ao ISCAS 2011 um artigo com a proposta do núcleo do CABAC (Seção 5.3.1). Alguns dos resultados mais recentes ainda serão enviados a eventos conceituados e a continuidade deste trabalho deverá resultar em um número ainda maior de contribuições significativas para a área.

No decorrer deste trabalho verificaram-se diversos desafios para seu desenvolvimento. **O** padrão H.264 é mesmo tempo o mais avançado e o mais complexo padrão de codificação de vídeo. Durante o processo de desenvolvimento de novas arquiteturas de hardware para os módulos desta tese, um dos maiores desafios foi lidar com esta complexidade e implementar as funcionalidades e casos de canto das ferramentas do padrão que serão alvo de proposta arquitetural. Uma vez que a proposta arquitetural foi concebida, a descrição em linguagem de hardware (VHDL), síntese, validação e prototipação para a coleta de resultados certamente representam a maior parte do trabalho a ser desenvolvido, Problemas que estão fora do alcance ou do escopo da tese geraram atrasos adicionais durante a fase de desenvolvimento.

Por outro lado, o padrão H.264 também é um padrão alvo de intensas pesquisas tanto pela academia quanto pela indústria, visto que é um padrão de codificação de vídeo para aplicações de eletrônica de consumo. Uma das consequências é a grande quantidade de publicações relacionadas, muitas de baixa qualidade e com resultados que não podem ser reproduzidos ou avaliados apenas pela descrição informada, dificultando a análise e comparação dos resultados obtidos.

REFERÊNCIAS

AGOSTINI, L. V. **Desenvolvimento de arquiteturas de alto desempenho dedicadas a compressão de vídeo segundo o padão H.264/AVC**. 2007. 172f. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

AGOSTINI, L. V.; Azevedo Filho, A.P.; Rosa, V.S.; Berriel, E.A.; Santos, T.G.S.; Bampi, S.; Susin, A.A.; FPGA Design of A H.264/AVC Main Profile Decoder for HDTV. In: International Conference on Field Programmable Logic and Applications, 2006. FPL '06. Madrid, Spain. **Proceedings...** Ago. 28-30 2006 pp:1 – 6.

AGOSTINI, L. V. ; AZEVEDO, Arnaldo ; STAEHLER, W. ; ROSA, Vagner ; ZATT, Bruno ; PINTO, Ana Cristina ; PORTO, Roger Endrigo Carvalho ; BAMPI, Sergio ; SUSIN, Altamiro . Design and FPGA Prototyping of a H.264/AVC Main Profile Decoder for HDTV. **Journal of the Brazilian Computer Society**, v. 12, p. 25-36, 2007ª

ARTISAN Components, **TSMC 0.18mm Process 1.8-Volt SAGE-XTM Standard Cell Library**, Release 4.1, Set. 2003.

AZEVEDO, A. P. **MOCHA: Arquitetura Dedicada para a Compensação de Movimento em Decodificadores de Vídeo de Alta Definição, Seguindo o Padrão H.264**. 2006. 120f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

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

BINGBO, L.; et al. A high-performance VLSI architecture for CABAC decoding in H.264/AVC. In: ASICON '07 - International Conference on ASIC, 2007 Guilin, china. **Proceedings…** October 2007 pp. 790-793.

BIN, G.; WEI-DONG, W.; YU-LI, S.; HONG, Z. A High Speed CABAC Algorithm Based on Probability Estimation Update. Fourth International Conference on Image and Graphics, ICIG 2007. Proceedings… [s.n.; s.l.] pp 195 – 199, 22-24 de agosto de 2007.

BOTTREAU, V. Draft text conformance testing for SVC profiles – **Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG** - JVT-AB202 - 28th Meeting: Hannover, Germany, 20-25 Jul. 2008.

CADENCE, **RTL Compiler.** Disponível em <www.cadence.com>, acesso em março de 2010.

CHU, C.; et all. Hierarchical Global Motion Estimation/Compensation in Low Bitrate Video Coding. ISCAS 1997 – IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS. **Proceedings…** Hong Kong: IEEE, 1997, p. 1149-1152.

DEPRÁ, D. A.; Rosa, V. S.; Bampi, S. A Novel Hardware Architecture Design for Binary Arithmetic Decoder Engines Based on *Bitstream* Flow Analysis. In: SBCCI2008 – Brasiliam Symposium on Integrated Circuits and Systems Design. 21. 2008. Gramado, Brasil. **Proceedings…** 1-4 de setembro de 2008.

DEPRÁ, D. A.; Rosa, V. S.; Bampi, S. A Novel Hardware Architecture Dedicated to Binary Arithmetic Decoder Engines for the H.264/AVC CABAD. IFIP International Conference on Very Large Scale Integration. VLSI-SOC. 16th. 2008. Rhodes, Greece. **Proceedings…** 13 a 15 de outubro de 2008a.

DEPRÁ, D. A.; Rosa, V. S.; Bampi, S. Design and Implementation of a High-performance Architecture for Binarization Methods Defined by H.264/AVC Standard. Iberchip Workshop. XIV. 2008. Puebla, México. **Proceedings…** 20-22 de fevereiro de 2008b.

DEPRÁ, D. A. Algoritmos e desenvolvimento de uma arquitetura para o CABAC do decodificador H.264/AVCAlgorithms and Architecture Development for CABAC. Dissertação de mestrado. **Instituto de Informatica: Universidade Federal do Rio Grande do Sul**. 152p. 2009

DINIZ, C. M. Arquitetura de hardware dedicada para a predição intra-quadro em codificadores do padrão H.264/AVC de compressão de vídeo. Dissertação de mestrado. **Instituto de Informatica: Universidade Federal do Rio Grande do Sul**. 96p. 2009

DIGILENT Inc. XUP-V2Pro board. Disponível em <http://www.digilentinc.com/>. Acesso em: março de 2008.

EECKHAUT, Hendrik et. al. Optimizing the critical loop in the H.264/AVC CABAC decoder. In: IEEE International Conference on Field Programmable Technology, 2006. FPT 2006. Bangkok, India. **Proceedings...** December 2006.

EYE, **Wikipedia**, Disponível em: <http://en.wikipedia.org/w/index.php?title= Human_eye&oldid=251386620>. Link permanente. Acesso em: Novembro de 2008.

GAMUT, **Wikipedia**. Disponível em: <http://en.wikipedia.org/w/index.php? title=Gamut&oldid=251017396>. Link permanente. Acesso em: Novembro de 2008.

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

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

HOROWITZ, M. et al, H.264/AVC Baseline Profile Decoder Complexity Analysis. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 13, n. 7, p. 704-716, Jul. 2003.

HUANG, Y.; CHEN, T. Architecture Design for Deblocking Filter in H.264/AVC. International Conference on Multimedia Electronics. **Proceedings…**Baltimore, Maryland, USA, pp. 693-696, Jul 6-9, 2003

HUANG, Y-W et al. Survey on Block Matching Motion Estimation Algorithms and Architectures with New Results. **Journal of VLSI Signal Processing,** Springer Science, 42, 297–320, DOI: 10.1007/s11265-006-4190-4, 2006

ISE Foundation. **XILINX**. Disponível em: <http://www.xilinx.com/ise/logic_design_prod/foundation.htm>, acesso em dezembro de 2008b.

ISO - 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.

ISO - International Organization for Standardization. **ISO/IEC 14496-2 - MPEG-4 Part 2 (01/1999)**: coding of audio visual objects – part 2: visual. [S.l.;S.n.], 1999.

ITU - International Telecommunications Union. **ITU-T Recommendation H.261 v1 (11/90)**: video codec for audiovisual services at 64 kbit/s. [S.l.], 1990.

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

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

ITU - International Telecommunications Union. **ITU-T Rec. H.264/ISO/IEC 14496-10 AVC**. [S.l.;S.n.]. 2003.

ITU - International Telecommunications Union. **H.264 AVC Fidelity Range Extensions**. JVT-L050. [S.l.], 2004.

ITU - International Telecommunications Union. **ITU-T Rec. & ISO/IEC 14496-10 AVC**, "Advanced Video Coding for Generic Audiovisual Services," version 3, 2005.

ITU - International Telecommunications Union. ITU-T Home. Disponível em: <www.itu.int/ITU-T/>. Acesso em: ago. 2005a.

ITU - International Telecommunications Union. **ITU-T Rec. & ISO/IEC 14496-10 AVC**, "Advanced Video Coding for Generic Audiovisual Services," Amendment 3 – Scalable Video Coding, 2007.

ITU - International Telecommunications Union. **ITU-T Rec. & ISO/IEC 14496-10 AVC**, "Advanced Video Coding for Generic Audiovisual Services," Version 03/2009 - MVC –Multiview Video Coding, 2009.

KAMACI, N.; ALTUNBASAK, Y. Performance Comparison of the Emerging H.264 Video Coding Standard with the Existing Standards. In: IEEE INTERNATIONAL CONFERENCE ON MULTIMEDIA AND EXPO. **Proceedings...** Baltimore: IEEE, 2003. p. I345-I348

KALVA, H. The H.264 Video Coding Standard. **IEEE Computer Society Press**. Vol. 13, pp. 86-90, No. 4. Los Alamitos, CA, USA. October 2006.

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

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

KIM, C-H.; PARK, I-C. High speed decoding of context-based adaptive binary arithmetic codes using most probable symbol prediction. In: IEEE International Symposium on Circuits and Systems, 2006. ISCAS 2006. **Proceedings...** 2006. May 2006.

KHURANA, G. et. al. Pipelined hardware implementation of in-loop deblocking filter in H.264/AVC **IEEE Transactions on Consumer Electronics.** V.52, I. 2, May 2006. pp.536-540.

KROUPIS, N.; et all. A Modified Spiral Search Motion Estimation Algorithm and its Embedded System Implementation. ISCAS 2005 - IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS. **Proceedings…** Kobe: IEEE, 2005, p. 3347-3350

KUANG, S. R. et al. Dynamic Pipeline Design of an Adaptive Binary Arithmetic Coder. **IEEE Transactions on Circuits and Systems—II: Analog and Digital Signal Processing**. Vol. 48, No. 9. September 2001.

KUOEL, C-C. et al. Design of a Low Power Architecture for CABAC Encoder in H.264, **Proceedings…** IEEE APCCAS, pp. 243-246, Dez. 2006.

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

KWON, S. et all. Overview of H.264 / MPEG-4 Part 10. In: 5th WSEAS INTERNATIONAL CONFERENCE ON MULTIMEDIA, INTERNET AND VIDEO TECHNOLOGIES. **Proceedings…** Corfu Island, Grecia: [S.n.], 2005.

Li, L.; et al., A CABAC Encoding Core with Dynamic Pipeline for H.264/AVC Main Profile, **Proceedings…** IEEE APCCAS, pp. 761 – 764, dez, 2006

LI, R.; et all. 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, Ago. 1994.

LIN, H-Y et. al. Efficient deblocking filter architecture for H.264 video coders. IEEE International Symposium on Circuits and Systems. ISCAS. 2006, **Proceedings…** 21-24 May 2006.

LIN, Y-K. et. al. A 242mW, 10mm2 1080p H.264/AVC High Profile Encoder Chip. Design Automation Conference – DAC. 2008., Anahen, California, EUA. **Proceedings…** Jun 8-13, 2008 pp 78-83.

LIST, P. et. al., Adaptative deblocking filter, **IEE trans. Circuits Syst. Video Technol.** Vol. 13, pp. 614-619, July 2003.

MACULA, **Wikipedia**. Disponível em: < http://en.wikipedia.org/w/index.php?title= Macula&oldid=247905727>. Link permanente. Acesso em: Novembro de 2008.

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, Jul. 2003.

MARPE, Detlev; SCHWARZ, Heiko; WIEGAND, Thomas. Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard. **IEEE Transactions on Circuits and Systems for Video Technology**, Vol. 13, Nº 7, July 2003.

MEI-HUA, Xu; YU-LAN, Cheng. FENG, Ran. ZHANG-JIN, Chen. Optimizing Design and FPGA Implementation for CABAC Decoder. In: International Symposium on High Density packaging and Microsystem Integration, 2007. HDP '07. **Proceedings...** pp. 1-5. June 2007.

MIANO, John; Compressed Image File Formats: JPEG, PNG, GIF, XBM, BMP, **Addison-Wesley Professional** ISBN: 978-0201604436, aug. 1999.

MODELSIM, **Mentor Graphics**. Disponível em <www.mentor.com>. Acesso em: dezembro de 2008.

OSORIO, R. R.; Bruguera, J. D. Arithmetic coding architecture for H.264/AVC CABAC compression system. DSD, 2004. **Proceedings...** Euromicro Symposium. pp 62-69. Sept. 2004.

OSORIO, R. R.; Bruguera, J. D., High-Throughput architecture for H.264/AVC CABAC compression system, **IEEE Transactions on Circuits and Systems for Video Technology**, v. 16, n. 11, pp. 1376–1384, Nov, 2006.

PANDIT, P. Vetro, A. JVT-AA208: JMVM 8.0 Software, **Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG**, 27th Meeting: Geneva, CH, 23-29 April, 2008.

PASTUSZAK, G. A High-Performance Architecture of the Double-Mode Binary Coder for H.264.AVC. **IEEE Tansactions on Circuits and Systems for Video Technology**, v. 18, n. 7, p. 949-960, jul. 2008.

PORTO, M. S. **Arquiteturas de Hardware de Baixo Custo e Alto Desempenho para a Estimação de Movimento em Vídeos HDTV**. 2007. 100f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática. UFRGS. Porto Alegre.

PLATFORM Studio and EDK. **XILINX**. Disponível em: <http://www.xilinx.com /support/mysupport.htm#Virtex-II%20Pro>. Acesso em novembro de 2008a.

PURI, A.; Chen, X.; Luthra, A. "Video coding using the H.264/MPEG-4 AVC compression standard", **Signal Processing: Image Communication**, Elsevier, n. 19, pp. 793-849, 2004.

REISSLEIN, Martin. YUV Video Sequences. **Video Traces Research Group**. Disponível em: <http://trace.eas.asu.edu/yuv/index.html>. Acesso em março de 2008.

RETINA, **Wikipedia**. Disponível em: < http://en.wikipedia.org/w/index.php?title= Retina&oldid=249530431>. Link permanente. Acesso em: Novembro de 2008a.

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

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

ROSA, V. S.; Staehler, W. T.; Azevedo, A.; Zatt, B.; Porto, R. E.; Agostini, L. V.; Bampi, S.; Susin, A.A.; FPGA Prototyping Strategy for a H.264/AVC Video Decoder, In: IEEE/IFIP International Workshop on Rapid System Prototyping, 18th. 2007. RSP 2007. Porto Alegre, **Proceedings…** 28-30 May 2007 Page(s):174 - 180

ROSA, V.; Susin, A. A.; Bampi, S.; An HDTV H.264 deblocking filter in FPGA with RGB video output. In: IFIP International Conference on Very Large Scale Integration. 15th. 2007a. VLSI - SoC 2007. Atlanta, EUA. **Proceedings…** 15-17 Oct. 2007a Page(s):308 – 311

ROSA, V. S.; Susin, A. A.; Bampi, S. A high performance H.264 Deblocking Filter. The 3rd Pacific-Rim Symposium on Image and Video Technology (PSIVT2009), Springer: LCNS, Tokyo, Japan, **Proceedings…** January 13th—16th, 2009.

ROSA, V. S.; Silva, L. M.; Bampi, S.; Na H.264 Deblocking Filter in FPGA with RGB Video Output. XVI Iberchip, Workshop, 2010, Foz do Iguaçu, **Proceedings…**: 2010.

SAHAFI, L. **Context-Based Complexity Reduction Applied to H.264 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**. 2. ed. New York: Springer, 2000.

SHENG, B.; GAO, W. WU, D. An implemented architecture of deblocking filter for H.264/AVC. International Conference on Image Processing. **Proceedings…** [s.n.; s.l.], V.1. pp. 665-66824-27 de outubro de 2004

SCHWARZ, Heiko; MARPE, D.; WIEGAND, T. Overview of the scalable H.264/MPEG4-AVC extension, IEEE ICIP. 2006. **Proceedings…** 2006.

SHI, Y.; SUN, H. **Image and Video Compression for Multimedia Engineering: Fundamentals, Algorithms and Standards.** Boca Raton: CRC Press, 1999.

SHOJAINA, H.; Sudharsanan, S. A high performance CABAC encoder, **Proceedings…** IEEE NEWCAS Conference, Quebec City, Canada, pp. 104–107, Jun. 2005.

SILVA, L. M. Implementação Física de Arquiteturas de Hardware para a Decodificação de Vídeo Digital Segundo o Padrão H.264/AVC. **Instituto de Informatica: Universidade Federal do Rio Grande do Sul**. 152p. 2

SIMA, M; Zhou, Y.; Zhang, W. An Efficient Architecture for Adaptative Deblocking Filter of H.264/AVC Video Coding. **IEEE Trans. On Consumer Electronics** Vol. 50, n. 1, pp. 292-296. Feb. 2004.

SMT310Q User Manual. **SUNDANCE Inc**. Disponível em: <www.sundance.com/docs/SMT310Q%20User%20Manual.pdf>. Acesso em: Julho de 2006.

SMT395E User Manual. **SUNDANCE Inc**. Disponível em: <www.sundance.com/docs/SMT395EUserManual.pdf>. Acesso em: Julho 2006.

SMT6001: Sundance Flash Programming Utility (SFPU) User Manual. **SUNDANCE Inc.** Disponível em: < www.sundance.com/docs/SMT6001.chm>. Acesso em: Julho de 2006c.

SMT6012: EPK Driver for Code Composer Studio User Manual. **SUNDANCE Inc**. Disponível em <http://www.sundance.com/docs/SMT6012.chm>. Acesso em: Julho 2006.

SMT6025: Host-side software interface to Sundance hardware User Manual. **SUNDANCE Inc**. Disponível em: <www.sundance.com/docs/SMT6025.chm>. Acesso em: Julho de 2006.

SMT6400: DSP Module Package User Manual. **SUNDANCE Inc**. Disponível em: <www.sundance.com/docs/SMT6400.chm>. Acesso em: Julho 2006.

STAEHLER, Wagston Tassoni. **Projeto de sistemas digitais complexos: uma aplicação ao decodificador H.264**. 2006. 108f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática. UFRGS. Porto Alegre.

SUHRING, Karsten. H.264/AVC Reference Software. In: Fraunhofer Heinrich-Hertz-Institute. Disponível em: <http://iphome.hhi.de/suehring/tml/download/>. Acessado em dezembro de 2008.

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

SULLIVAN, G.; et all. The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions. In: SPIE 2004 XXVII CONFERENCE ON APPLICATIONS OF DIGITAL IMAGE PROCESSING. **Proceedings…** Denver: SPIE, 2004.

SUNDANCE Home Page. **SUNDANCE Inc.** Disponível em <www.sundance.com/ default.asp>. Acesso em: Julho 2006.

SUNNA, P. AVC / H.264 – An Advanced Video Coding System for SD and HD Broadcasting. **European Broadcasting Union Technical Review.** [S.l.], n. 302, Abr. 2005. Disponível em: <http://www.ebu.ch/departments/ technical/trev/trev_302-sunna.pdf>. Acesso em: setembro de 2005.

TIAN, X. et al. Full RDO-Support Power-Aware CABAC Encoder With Efficient Context Access. **IEEE Transactions on Circuits and Systems for Video Technology**, V.19 , n. 9. Pp. 1262-1273, 2009.

TOURAPIS, M.; et all. Fast Motion Estimation Using Circular Zonal Search. In: VISUAL COMMUNICATIONS AND IMAGE PROCESSING. **Proceedings…** San Jose: [s.n.], 1999, v. 2, p. 1496-1504.

VETRO, A. et. al. JVT-AB204: Joint Draft 8.0 on Multiview Video Coding, **Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG**, 28th Meeting: Hannover, DE, 20-25 July, 2008.

VETRO, A. et. al. JVT-AA207: Joint Multiview Video Model (JMVM), **Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG**, 27th Meeting: Geneva, CH, 23-29 April, 2008.

VETRO, A. et al. Joint Draft 8.0 on Multiview Video Coding, Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG – JVT-AB204, 28th Meeting: Hannover, DE, 20-25 Jul. 2008.

VIERON, J. et al. Draft reference software for SVC. **Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG**- JVT-AB203, 28th Meeting: Hannover, DE, 20-25 Jul. 2008.

VIRTEX II pro support. **XILINX**. Disponível em: <http://www.xilinx.com/support /mysupport.htm#Virtex-II%20Pro>. Acesso em: Novembro de 2008.

VIRTEX 5 Multi-platform FPGA. **XILINX**. Disponível em: < http://www.xilinx.com/ products/virtex5/>. Acesso em: Dezembro de 2008a.

WIEGAND, T.; SULLIVAN, G., J.; BJØNTEGAARD, G.; LUTHRA, A.. Overview of the H.264/AVC Video Coding Standard. **IEEE Transactions on Circuits and Systems for Video Technology**, Vol. 13, pp. 560-576, Nº 7, July 2003.

WITTEN, I. H.; NEAL, R. M.; and CLEARY, J. G. Arithmetic coding for data compression. **Communications of ACM**, vol. 30, no. 6, pp. 520–540, June 1987.

WU, D.; et all. A Single Scalar DSP based Programmable H.264 Decoder. In: SSoCC´05 – SWEDISH SYSTEM-ON-CHIP CONFERENCE 2005. **Proceedings…** Tammsvik: [S.n.], 2005.

WU, L-C.; Lin, Y-L., A high throughput CABAC encoder for ultra high resolution video. **Proceedings…** IEEE International Symposium on Circtuits and Systems-ISCAS2009. pp. 1048-1051. 2009.

YANG, Y-C.; LIN, C-C.; CHANG, H-C.; SU, C-L.; and GUO, J-I. A High Throughput VLSI Architecture Design for H.264 Context-Based Adaptive Binary Arithmetic Decoding With Look Ahead Parsing. In: IEEE International Conference on Multimedia and Expo - ICME, 2006 **Proceedings…** pp. 357-360, July 2006.

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

YU, Wei; HE, Yun. A High Performance CABAC Decoding Architecture. **IEEE Transactions on Consumer Electronics**, Vol. 51, pp. 1352-1359, No. 4, November 2005.

ZATT, B. **Modelagem de Hardware para Codificação de Vídeo e Arquitetura de Compensação de Movimento Segundo o Padrão H.264/AVC**. 2008. 121f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática. UFRGS. Porto Alegre.

ZHANG, J.; et all. Performance and Complexity Joint Optimization for H.264 Video Coding. In: ISCAS 2003 – IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS. **Proceedings…** Bangkok: IEEE, 2003, v. 2, p. II888-II891.

ZHANG, P.; GAO, W.; XIE, D.; WU, D. High-Performance CABAC Engine for H.264/AVC High Definition Real-Time Decoding. In: International Conference on Consumer Electronics, 2007. ICCE 2007. **Digest of Technical Papers**.. pp. 1-2. Las Vegas, NV, USA. Janeiro, 2007.

ZHENG, W. et al.: Efficient Pipelined CABAC Encoding Architecture, **IEEE Transactions on Consumer Electronics**, V. 54, N. 2, Maio 2008.

ITRS **International Technology Roadmap for Semiconductors**, Disponível em <http://www.itrs.net/>, acessado em fevereiro de 2010.

APÊNDICE A ESTUDO DO HARDWARE DE PROTOTIPAÇÃO

Para tornar possível a protipação das arquiteturas desenvolvidas para o codificador e o decodificador H.264, foram adquidas placas de desenvolvimento das empresas Digilent Inc. (DIGILENT, 2008) e Sundance DSP inc. (SUNDANCE, 2006). Da empresa Digilent Inc, foram adquiridos kits baseados no FPGA Virtex II pro da Xilinx (VIRTEX, 2008), enquanto da empresa Sundance DSP inc. foram adquiridos kits baseados em DSP da série C6000 da Texas Instruments e FPGA Virtex II pro (SMT395E, 2006a). A razão da escolha destas empresas e dos kits em questão foge do escopo deste trabalho e não será descrita. A seção A.1 irá apresentar uma descrição detalhada do hardware da placa XUP-V2P assim como os acessórios que compõe o kit adquiridos da empresa Digilent Inc e a seção A.2 tratará do kit obtido da Sundance DSP Inc.

A.1 Hardware do kit da Digilent.

O hardware do conjunto Digilent Inc é composto pelos seguintes itens:

- Placa XUP-V2Pro
- Placa Video Decoder
- Placa DIO5
- Módulo de memória DDR de 512MB
- Fonte de alimentação
- Cabo de programação USB
- Cabo de comunicação serial RS232

A placa XUP-V2Pro é uma placa baseada no FPGA Virtex II pro e possui os seguintes recursos (DIGILENT, 2008):

- FPGA XC2VP30, um Virtex II pro com 30.816 células lógicas, 136 multiplicadores de 18 bits, 2.448Kb de memória “block RAM” e dois processadores PowerPC integrados;
- Capacidade para módulos de memória DIMM DDR SDRAM de até 2GB
- Porta ethernet 10/100
- Porta USB 2.0 para programação
- Slot para cartão do tipo Compact Flash

- Saída de vídeo no padrão XSGA
- Codec de áudio
- Interface SATA
- Interface PS/2
- Interface RS-232
- Conectores de expansão de alta e baixa velocidade

Esta placa possui uma riqueza de recursos de I/O como apresentado acima. Ela constitui o centro do kit de desenvolvimento, contendo o FPGA. A Figura A.1 apresenta uma foto da placa XUP-V2P.

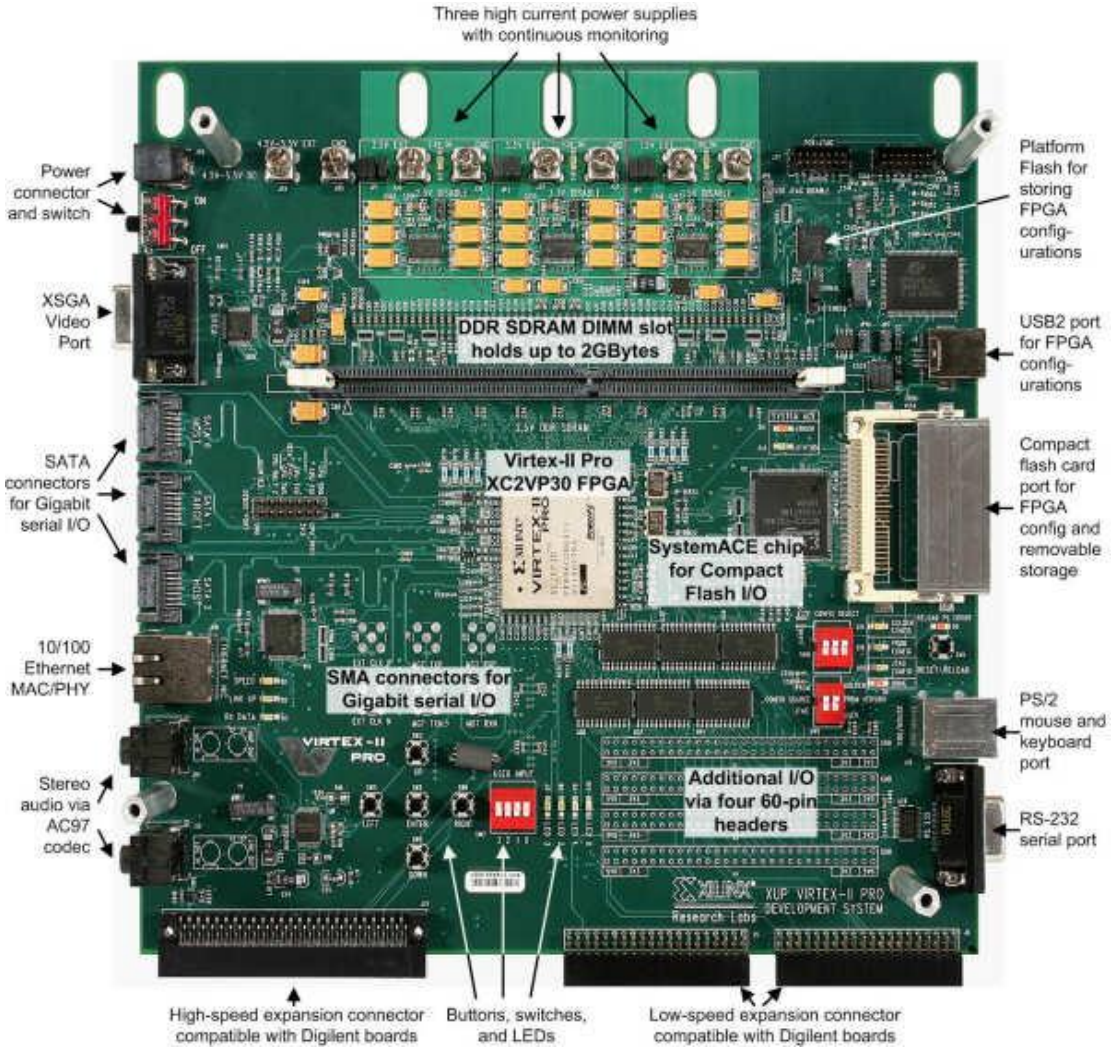

Figura A.1: Foto da placa XUP-V2P.

A programação do FPGA contido nesta placa é feita através de um cabo USB conectado a um PC. Para programação, é necessário o uso do programa iMPACT, que faz parte do pacote ISE da Xilinx (ISE, 2008). A interface USB da placa servirá apenas para fins de programação e depuração, não sendo possível a sua utilização para aplicações de usuário.

Para a comunicação com o usuário, há diversas possibilidades. Em aplicações que utilizam o processador *PowerPC*, a interface padrão de comunicação é o console, através de funções da biblioteca C como *printf()* e *scanf()*. Este tipo de comunicação é realizado através da interface serial RS232 da placa. Um cabo serial do tipo DTE-DCE pode ser utilizado para conectar a placa a um computador que possua interface RS232 e que rode um emulador de terminal. Chamadas a funções de escrita no console como a *printf()* irão exibir dados na tela de terminal, enquanto que chamadas a funções de leitura do console, como *scanf()*, irão obter dados digitados no terminal. A velocidade de comunicação da interface serial RS232 pode ser livremente configurável, uma vez que os pinos desta interface estão conectados diretamente a pinos do FPGA, sendo a UART (ou USART) implementada em lógica programável. A velocidade padrão desta comunicação será de 9600bps. A velocidade máxima será tipicamente 115200bps. Outras opções de comunicação são os LEDs, os *push-buttons*, os *dip-switches*, as entradas PS2 para teclado e mouse, a saída VGA. Todas estas interfaces estão conectadas diretamente a pinos do FPGA, sendo suas funcionalidades implementadas através de lógica programável. Nesta placa também está disponível um *codec* de áudio, compatível com o padrão AC97, que utiliza um chip específico para esta aplicação. Esta placa também pode ser conectada a uma rede *Ethernet* através de uma interface *ethernet* 10/100Mbps.

Para aplicações que usarem a UART RS232 disponível na placa como console (STDIN e STDOUT), um cabo serial será necessário para ligar a placa a um terminal serial (um PC rodando um emulador de terminal, por exemplo). Este cabo deve ser um cabo do tipo DTE-DCE, sendo opcional a implementação dos sinais de *handshake* (cabo de 3 ou 7 fios).

A.1.1 Configurando o software para a placa Digilent

A placa Digilent necessita do pacote ISE para o desenvolvimento de projetos em HDL e do EDK para o desenvolvimento de projetos baseados em plataforma (utilizando tanto processador quanto HDL). O pacote EDK depende do ISE e, por isso, este deve ser instalado inicialmente. Os *drivers* para a comunicação com a interface de programação USB da placa Digilent são instalados automaticamente durante a instalação do ISE, por isso não é necessário instalar qualquer pacote adicional para o desenvolvimento de aplicações que possuam apenas código HDL.

Ao conectar o cabo USB no computador é possível que o dispositivo seja detectado várias vezes seguidas. Em todas, o procedimento automático de instalação deve ser selecionado afim de que todos os *drivers* sejam instalados corretamente.

A.2 Sistema Sundance

O sistema Sundance (SUNDANCE, 2006) é baseado em um computador PC hospedeiro (*host*) ao qual ficam conectadas uma ou mais placas *carrier* e em cada uma destas carriers pode-se conectar uma ou mais placas de processamento ou de entrada e saída, também chamadas de TIMs (*Texas Instruments Modules*). A Figura A.2 mostra uma visão geral do sistema Sundance.

Como pode ser visto na Figura A.2, a placa *carrier* (*carrier board*) é conectada a um computador PC através do barramento PCI. Esta placa especificamente é a SMT310Q (SMT310Q, 2006), que possui lugares (*sites*) para conectar até quatro módulos TIM. Nas placas *carrier* que possuem mais de um lugar para TIMs, apenas um dos sites possui acesso ao barramento global (*global bus*) e apenas este site possui uma ComPort

que pode ser acessada pelo PC sem a necessidade de cabeamento especial. Os elementos do hardware Sundance que são comuns a todas as placas serão apresentados na seção A.2.1. Na seção A.2.2 a placa *carrier* será abordada em maiores detalhes. A seção A.2.4 irá mostrar detalhes da placa SMT395E.

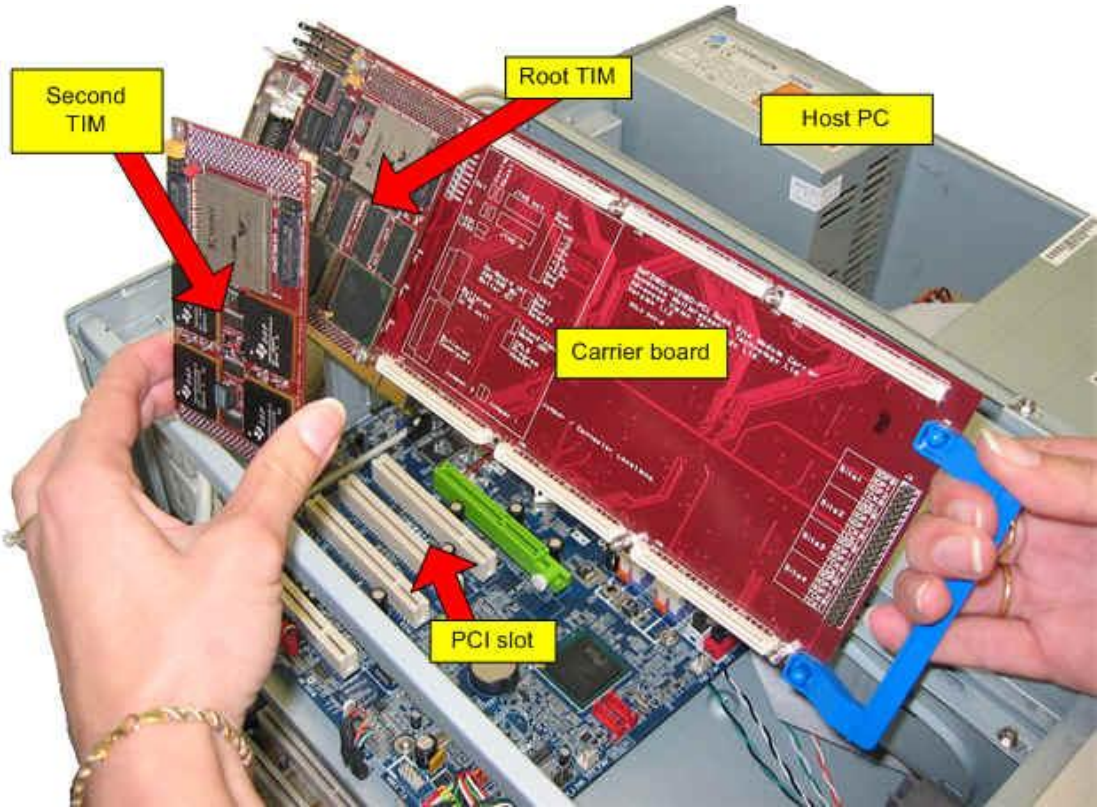

Figura A.2: Um sistema típico usando placas Sundance.

Os sistemas sundance são compostos por placas *carrier*, placas TIM e por um PC convencional com interface PCI que serve como *host*, além de fornecer energia para todo o hardware sundance (SUNDANCE, 2006). A seguir os aspectos que são comuns a todos os sistemas sundance serão apresentados.

A.2.1 Módulos

Os módulos do sistema sundance são placas que obedecem à especificações de conectividade estabelecidas pela Texas Instruments e, por isso, são chamados de TIMs (*Texas Instruments Module*). São placas que podem possuir diversas opções de interfaceamento e podem ser de três tipos básicos: placas de processador, placas de FPGA ou ainda placas de interface analógica. Neste texto será abordado apenas as placas de processamento. A Figura A.3 apresenta um diagrama em blocos comum às placas de processador Sundance.

Todas as TIMs desenvolvidas pela Sundance possuem um FPGA que é usado como controlador e implementa os protocolos de comunicação em hardware para a maioria das interfaces. As placas de processador possuem um ou mais processadores DSP da Texas Instruments. A interface JTAG, que fica ligada diretamente aos processadores é o único caminho de comunicação que não passa pela lógica do FPGA. A memória Flash que contém o código que será executado no processador DSP assim que ele for ligado pode ser escrita através da interface JTAG. Desta forma, o FPGA não precisa estar configurado para que um programa ou dados sejam descarregados na memória flash.

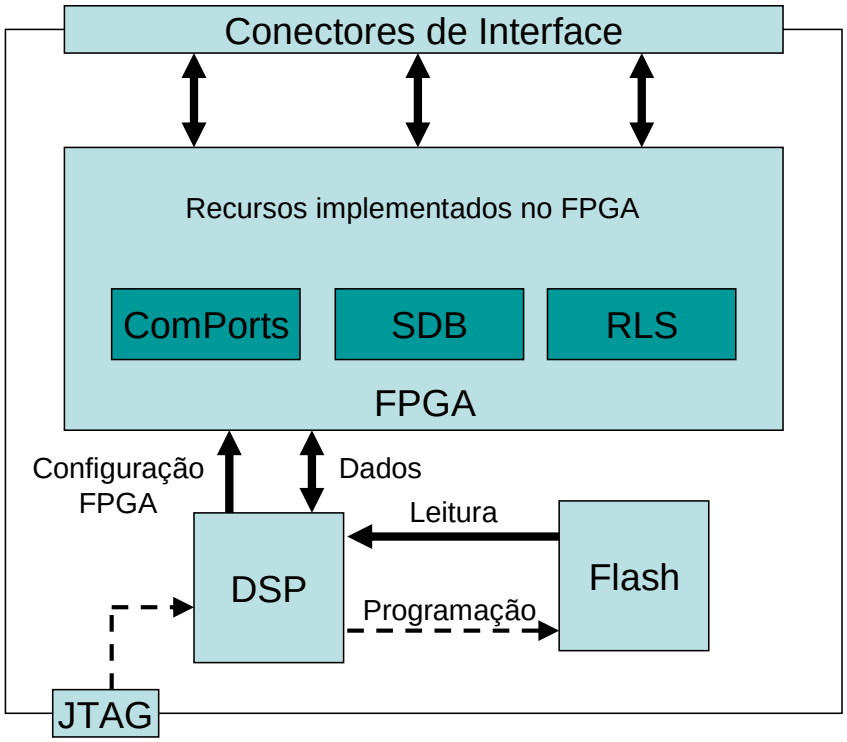

Figura A.3: Diagrama em blocos de uma placa TIM de processador.

Nas placas de processador da Sundance, o DSP é responsável pela programação do FPGA e, por isso, os pinos de programação do FPGA ficam ligados diretamente a pinos de entrada e saída do DSP, de forma que este gera o *bitstream* para a programação do FPGA a partir de dados que podem estar armazenados na memória Flash do DSP. De fato, nas placas de processador da Sundance, a seguinte sequência de eventos ocorre a partir do momento da liberação do sinal de *Reset* da placa.

1. O processador carrega um *bootstrap* da memória flash
2. Sob controle do *bootstrap*, o restante do *bootloader* é carregado
3. O DSP é configurado para refletir os periféricos a ele conectados
4. O FPGA é configurado usando um *bitstream* armazenado na memória Flash do DSP
5. O *bootloader* carrega um programa de usuário armazenado na memória flash OU aguarda que o programa de usuário chegue através de uma *ComPort*

Para esta sequência de eventos acontecer e o sistema operar corretamente, duas partes fundamentais precisam estar armazenadas na memória flash no momento da partida do sistema: O *bootloader* e o *bitstream* para o FPGA. Estas duas partes juntas colocam o sistema em um estado que tanto o DSP quanto o FPGA possuam o comportamento esperado em virtude da configuração específica de conexões da placa em que estão inseridos. Por isso, diferentes placas de processador poderão possuir *bootloaders* e *bitstreams* do FPGA diferentes.

A.2.2 Placas Carrier

As placas *carrier* oferecem diversos recursos aos TIMs, de forma a torná-los operacionais. Suas principais finalidades são:

- Fornecer sustentação mecânica e energia aos TIMs

- Estabelecer um enlace de comunicação rápida com um PC
- Estabelecer um enlace de comunicação entre outras TIMs
- Encadear módulos TIM para formar uma cadeia JTAG

Embora algumas TIMs possam operar autonomamente, usando conectores adicionais para alimentação, todas precisarão ser conectadas a uma placa *carrier* ao menos uma vez para programação de sua memória flash com o programa de usuário específico para a aplicação ou até mesmo com os dados de configuração do FPGA (*bitstream*). O uso das placas *carrier* tornam a utilização dos TIMs mais simples.

As placas *carrier* possuem diversos recursos que complementam o hardware contido nos TIMs. A Figura A.4 apresenta um diagrama simplificado de uma placa *carrier*.

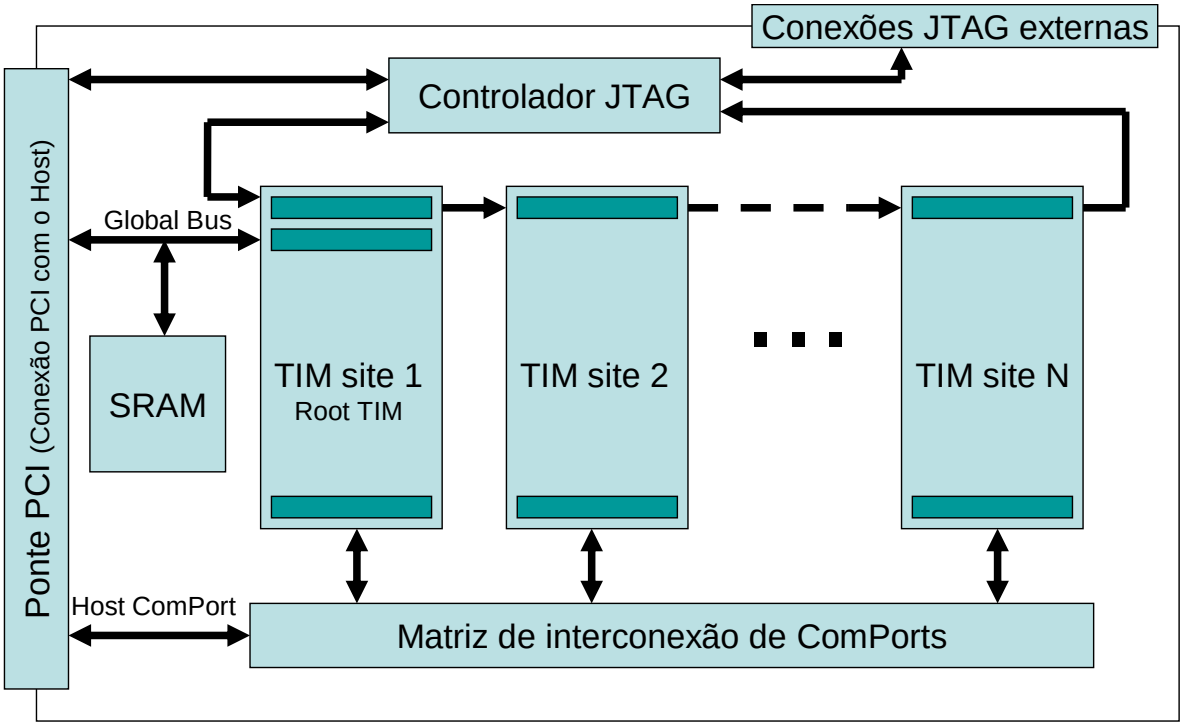

Figura A.4: Diagrama simplificado de uma placa Carrier.

Observe pelo diagrama da Figura A.4 que os módulos TIM podem ser conectados aos *TIM sites* disponíveis na placa. A Sundance oferece placas que possuem de um a quatro *TIM sites*.

Nas placas *carrier* há três formas básicas de comunicação entre o PC e os módulos TIM. A primeira forma e mais elementar é através da interface JTAG. Um controlador JTAG, comandado pelo PC, encadeia todos os sites disponíveis na placa e ainda oferece conexões JTAG externas através de conectores específicos. Esta interface possui as funções de configuração das TIMs e depuração de código em tempo de execução, sendo que não é possível utilizá-la para o trafego de dados entre o PC e os TIMs em tempo de execução. Outra opção de comunicação é através das *ComPorts*. Uma matriz de interconexão de *ComPorts* formada por chaves eletrônicas rápidas, conectores e cabeamento específico tornam possível a comunicação tanto entre TIMs quanto entre TIMs e o PC. Há uma *ComPort* ligada à ponte PCI na placa *carrier* e seis *ComPorts* em cada "TIM site". Esta estrutura possibilita uma ampla gama de possibilidades de interconexão. Uma importante limitação deste sistema de interconexão é que o

chaveamento na matriz de interconexão é realizado parcialmente através de chaves eletrônicas rápidas e parcialmente através de conectores que precisam ser interligados através de cabos específicos para esta finalidade. Isto reduz a flexibilidade. Sem o uso de cabos, a única possibilidade é a ligação direta entre TIMs que estão em sites vizinhos. Uma limitação derivada desta é que a *Host ComPort* possui um caminho eletrônico apenas para o site 1, necessitando de cabeamento para atender aos demais sites. Isto impede que ocorra a comunicação entre o PC e os sites [2..N] sem o uso de cabeamento adequado. A terceira forma de comunicação com o PC é através do barramento global (global bus). Esta forma de comunicação é a mais rápida disponível, permitindo usar toda a largura de banda disponível no barramento PCI, assim como permitindo o acesso direto à memória do DSP pelo PC e vice-versa. O acesso a este barramento global só é possível através do site número 1. Uma pequena quantidade de memória SRAM (tipicamente 1MB) também está disponível na placa *carrier* e é acessível através de uma janela específica na faixa de endereçamento disponível.

A.2.3 Mecanismos de interconexão

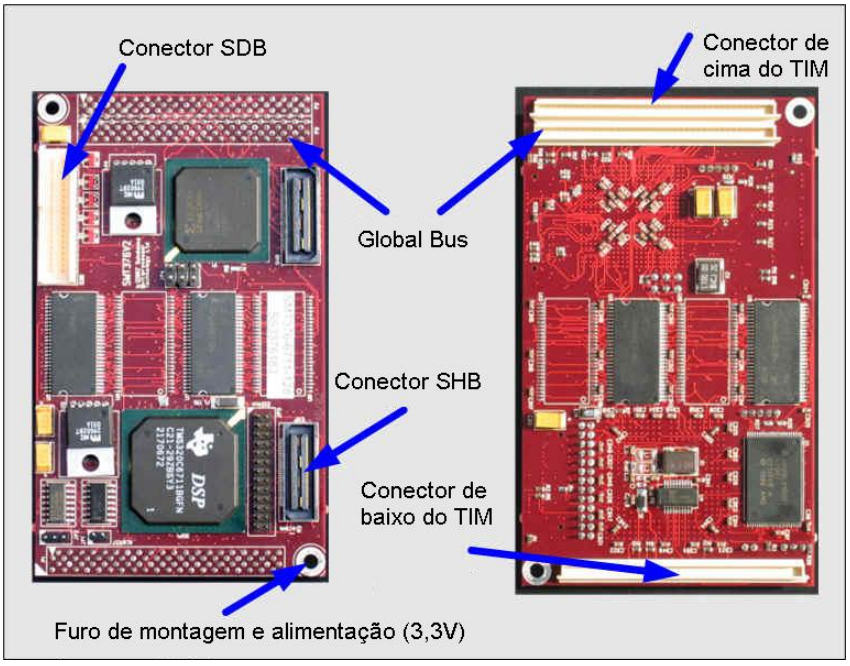

Figura A.5: Foto de um TIM ilustrando os conectores de interfaceamento.

O hardware Sundance prevê diversos mecanismos de interconexão que serão descritos nesta seção. Estes mecanismos são:

- *ComPorts*
- SDB - *Sundance Digital Bus*
- SHB – *Sundance High-speed Bus*
- RSL – *Rocket Serial Link*
- *Global Bus*

Estes mecanismos serão descritos em detalhes a seguir. A Figura A.5 ilustra alguns destes mecanismos presentes na maioria dos TIMs fabricados pela Sundance.

ComPorts

As *ComPorts* (*Communication Ports*) são ligações bi-direcionais que podem ser usadas para transferir fluxos de valores de 32 bits no nível no usuário. No nível do hardware, esta interface é implementada através de um canal paralelo de 8 bits *half-duplex*. O controlador da *ComPort* torna esta comunicação de 8 bits *half-duplex* transparente ao usuário, de forma que o usuário não precisa selecionar a direção do fluxo de dados e nem quebrar uma palavra de 32 bits em transferências de 8 bits. A implementação deste protocolo é feita através de lógica programável no FPGA disponível em todos os TIMs da Sundance. Como apresentado na Figura A.4, há uma matriz de interconexão na placa *carrier* que é parcialmente implementada através de chaves eletrônicas rápidas e parcialmente implementada através de cabos específicos. A Figura A.6 ilustra a conexão destes cabos que irão completar a matriz de interconexão de ComPorts.

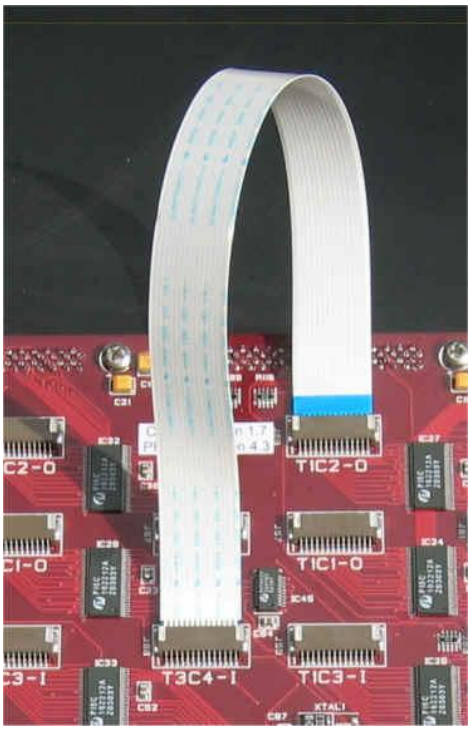

Figura A.6: Completando a matriz de interconexão de *ComPorts* através de cabos e conectores FMS.

Estes conectores estão localizados no lado traseiro (lado oposto ao lado dos TIMs) da placa *carrier*. Uma vez que não há qualquer marcador de orientação da posição do cabo, um pouco de cuidado deve ser tomado para realizar a interconexão de maneira correta, caso contrário haverá risco de danos à placa *carrier*. Um outro aspecto a ser observado sobre estes conectores é que algumas das portas estão configuradas como entrada e outras como saída logo após o *Reset*. Por esta razão as conexões devem ser realizadas entre *ComPorts* que estarão em estados opostos logo após o *Reset* sob pena de danificar a placa *carrier*. Como pode ser visto naFigura A.6, há uma identificação logo abaixo de cada conector FMS. Esta identificação indica o número do site, o número da *ComPort* naquele site e a direção da *ComPort* após o *Reset*. Em T3C4-I, por exemplo, o T3 indica que é o site 3, o C4 indica que é a *ComPort* 4 enquanto que o I indica que o estado após o *reset* é de entrada. T1C2-O indica o site 3, *ComPort* 2, configurada como saída após o *Reset*.

SDB - *Sundance Digital Bus*

O SDB (*Sundance Digital Bus*) é um mecanismo de comunicação de 16 bits empregado para a comunicação entre TIMs. Cabos e conectores específicos interligam diretamente os TIMs sem passar pela placa *carrier*. A Figura A.7 mostra a foto de um conector SDB.

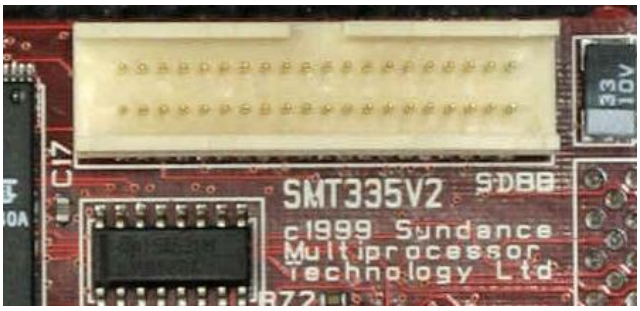

Figura A.7: Conector SDB.

Este barramento está disponível em algumas placas da Sundance, porém está sendo substituído pelo SHB, descrito a seguir. O SDB é apresentado aqui apenas por ser parcialmente compatível com o SHB: é possível fazer duas conexões SDB distintas a partir de uma conexão SHB.

SHB – *Sundance High-speed Bus*

O barramento SHB é uma extensão do barramento SDB e possui 32 bits de largura. O protocolo de baixo nível é semelhante a dois barramentos SDB em paralelo. Desta forma uma interface (conector) SHB pode efetivamente conectar dois dispositivos que possuam interface SDB. Como o conector é substancialmente diferente, um adaptador se torna necessário. A Figura A.8 mostra a foto de uma conexão entre dois TIMs através de um cabo SHB.

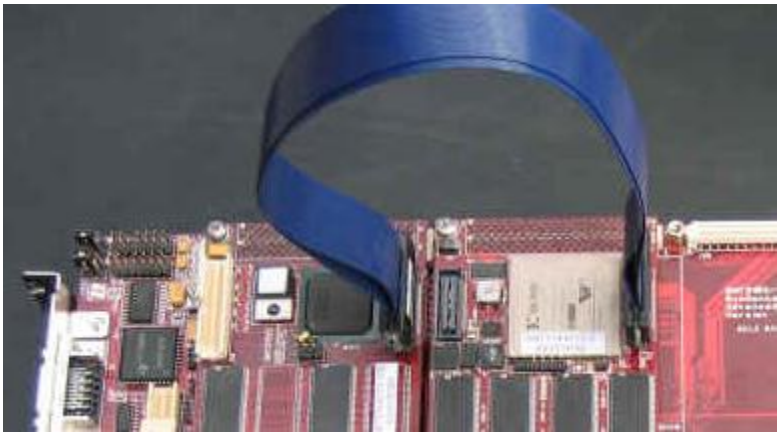

Figura A.8: Conexão através de um cabo SHB.

RSL – Rocket-IO Serial Link

O RSL é uma interface para a comunicação entre TIMs que se vale da interface de comunicação *Rocket-IO* presente nos FPGAs da família Virtex-II pro e superiores. O RSL especifica quatro linhas de transmissão de dados e quatro de recepção, formando um canal de comunicação *Full-Duplex* de até 10Gbps. Por exigir recursos de hardware disponíveis apenas nos FPGAs da família Virtex-II pro e superiores, a interface RSL

está disponível apenas em TIMs que possuam um destes FPGAs. A Figura A.10 mostra uma foto dos conectores RSL.

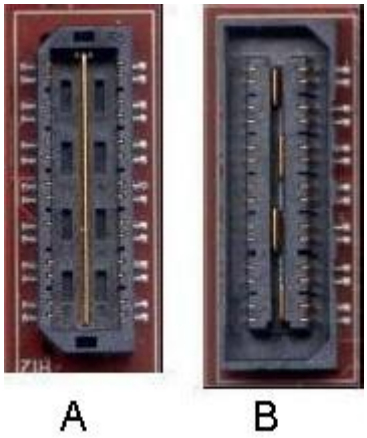

Figura A.9: Conectores RSL.

Observe pela Figura A.9 que estes conectores são classificados como “A” ou “B”. Isto porque as linhas de transmissão de um devem ser conectadas às linhas de recepção de outro e vice-versa. As TIMs que implementam o RSL possuem pelo menos dois conectores RSL, um do tipo A e outro do tipo B.

Outra observação importante é que estes conectores são idênticos aos utilizados para a interface SHB. Desta forma, Algum cuidado deve ser tomado para evitar conectar uma interface SHB em uma RSL. Entretanto uma regra simples pode ser utilizada: estas interfaces são projetadas de forma que uma pequena placa com dois conectores possa ser usada para conectar TIMs vizinhas usando ou SHB ou RSL e por esta razão, os conectores SHB e RSL ficam em posições distintas ao longo da borda do TIM.

A.2.4 Placa SMT395E-VP70

A placa SMT395E (SMT395E, 2006) da Sundance é uma placa de desenvolvimento para aplicações que necessitam de DSP de alta performance assim como grande área em FPGA. Esta placa possui as seguintes características

- DSP TMS320C6416, rodando a 1GHz
- FPGA XC2VP70-5
- 128MB de memória SDRAM conectada ao DSP e ao FPGA
- 256MB de memória DDR SDRAM conectada ao FPGA
- 8MB de memória FLASH para armazenamento do bootloader, configuração do FPGA e programas ou dados de usuário.
- 4 *ComPorts*
- Conector para o barramento global da placa *carrier* (*global bus*)
- 2 interface SHB (*Sundance High-speed Bus*) para a comunicação entre placas
- 2 interfaces SDB (*Sundance Digital Bus*) para a comunicação entre placas
- Gerenciamento de energia e temperatura

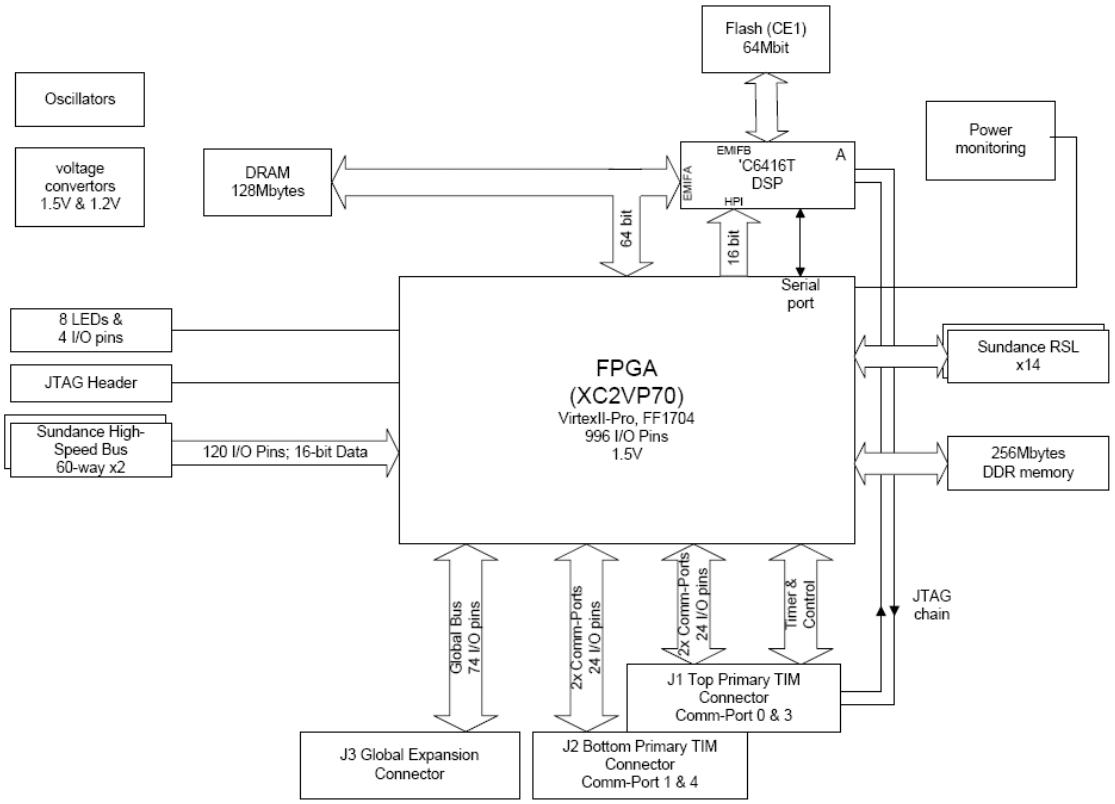

Figura A.10: Diagrama em blocos da SMT395E.

A placa SMT395E é uma versão reduzida da SMT395Q. A diferença é que enquanto a placa SMT395Q possui quatro processadores DSP, a SMT395E possui apenas um. Cada um dos três processadores adicionais da SMT395Q possui 64MB de memória SDRAM de uso exclusivo. A Figura A.10 apresenta um diagrama em blocos detalhado da SMT395E. Observe que o barramento de memória do DSP está ligado também a pinos do FPGA. Desta forma, o FPGA pode se comportar como uma memória, possibilitando a implementação de recursos de hardware com acesso mapeado em memória do DSP. Toda a infra-estrutura necessária para o funcionamento do FPGA como um controlador de todas as interfaces disponíveis é fornecida pela configuração padrão do FPGA. Esta configuração padrão, cujo código fonte é disponibilizado através de um NDA (*Non Disclosure Agreement*), deve ser modificada para que se acrescente a funcionalidade de hardware necessária ao funcionamento do sistema. A substituição da configuração padrão do FPGA por uma outra que não implemente os recursos de interface padrão da placa iria comprometer os recursos de interface da placa, uma vez que todas as interfaces utilizam o FPGA para a implementação dos protocolos de hardware. Uma observação importante é que na configuração padrão não está previsto o acesso à memória DDR SDRAM. O código HDL para esta interface deve ser gerado por uma ferramenta específica para este fim, porém também está disponível e já foi testado na placa. A Figura A.11 apresenta uma foto ilustrativa da placa SMT395E. Note o alto grau de integração desta placa e o lugar para 3 outros processadores DSPs que são colocados na versão SMT395Q. Muitos componentes são montados na face oposta da placa, não mostrada aqui. Um conector externo para alimentação externa está disponível, caso se deseje utilizar esta placa como uma solução standalone.

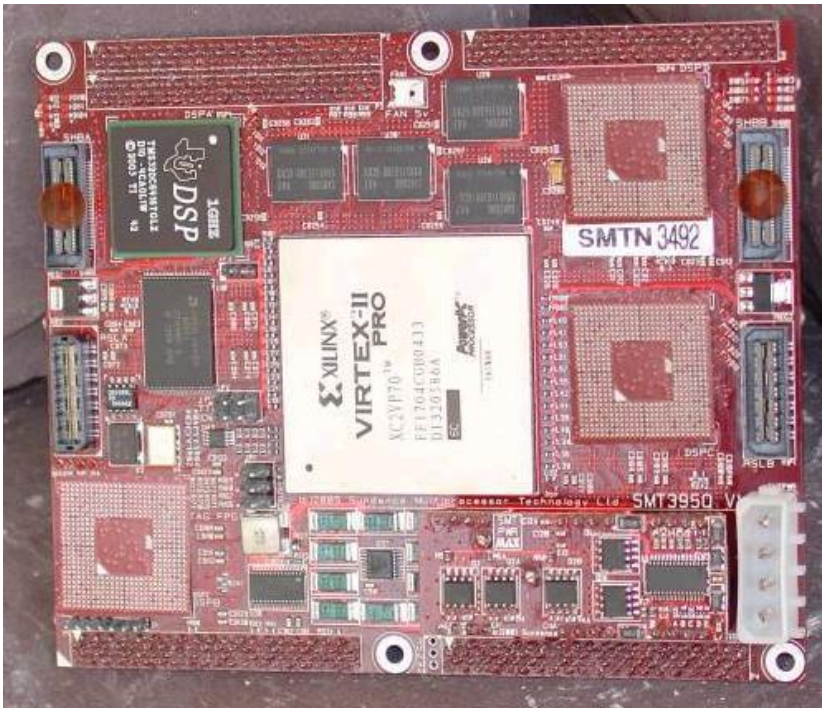

Figura A.11: Foto da placa SMT395E.

Funcionamento

A SMT395E é configurada de forma que seu processador DSP execute o "*boot*" a partir da memória Flash. O *bootloader* tem a função de configurar o DSP para sua operação no contexto da placa, configurar o FPGA com um *bitstream* de configuração armazenado na memória flash e por fim carregar um programa de usuário na memória principal do DSP e transferir o controle para este programa (SMT6001, 2006).

O procedimento para dar boot no processador, usando o *bootloader* padrão, a partir da memória flash é o seguinte (SUNDANCE, 2006d) (SUNDANCE, 2006e):

1. O processador copia o um programa *bootstrap* da primeira da memória flash para uma memória RAM de programa interna, começando no endereço 0 e a execução começa neste ponto.

2. Todos os registradores relevantes do DSP são colocados em seus valores padrão.

3. O FPGA é configurado a partir de dados armazenados na memória flash. No *bitstream* padrão, as *ComPorts*, o Barramento Global demais interfaces de comunicação são ativados. Este procedimento precisa ser completado para que a comunicação possa ocorrer entre o DSP e qualquer interface de comunicação disponível, uma vez que todas elas ficam conectadas ao FPGA.

4. Um *bootloader* é executado. Se houver um programa armazenado na flash, este programa será carregado. Caso contrário, este *bootloader* irá continuamente examinar as *ComPorts* até que dados apareçam em alguma delas. Uma vez que inicie a transmissão de dados a partir de uma das *ComPorts*, as outras serão ignoradas e todo o programa será carregado a partir desta porta.

5. O controle é passado ao programa carregado

Tanto o *bootloader* (e seu respectivo *bootstrap*) quanto a configuração do FPGA podem ser modificados livremente pelo usuário. Estas configurações de usuário, entretanto, devem ser planejadas com cuidado, pois o tanto o DSP quanto o FPGA estão conectados a um sistema complexo, que precisa ser corretamente ativado para um funcionamento adequado.

Para programas que irão rodar no DSP, a solução mais simples é configurá-los como um programa de usuário a ser carregado pelo *bootloader*. Este procedimento garante que o DSP vai estar em um estado consistente e que o FPGA vai estar programado logo no início do programa.

Para as aplicações que necessitam a execução no FPGA, é necessário ter o código fonte (VHDL) da configuração padrão e conectar a este código o HDL do projeto que deverá rodar no FPGA. Com isso, se mantém as funcionalidades da configuração padrão e se cria um caminho de acesso à "lógica de usuário" compatível com todos os módulos fornecidos pela Sundance.

O atraso típico entre a liberação do sinal de *Reset* e a configuração do FPGA é em torno de 2s.

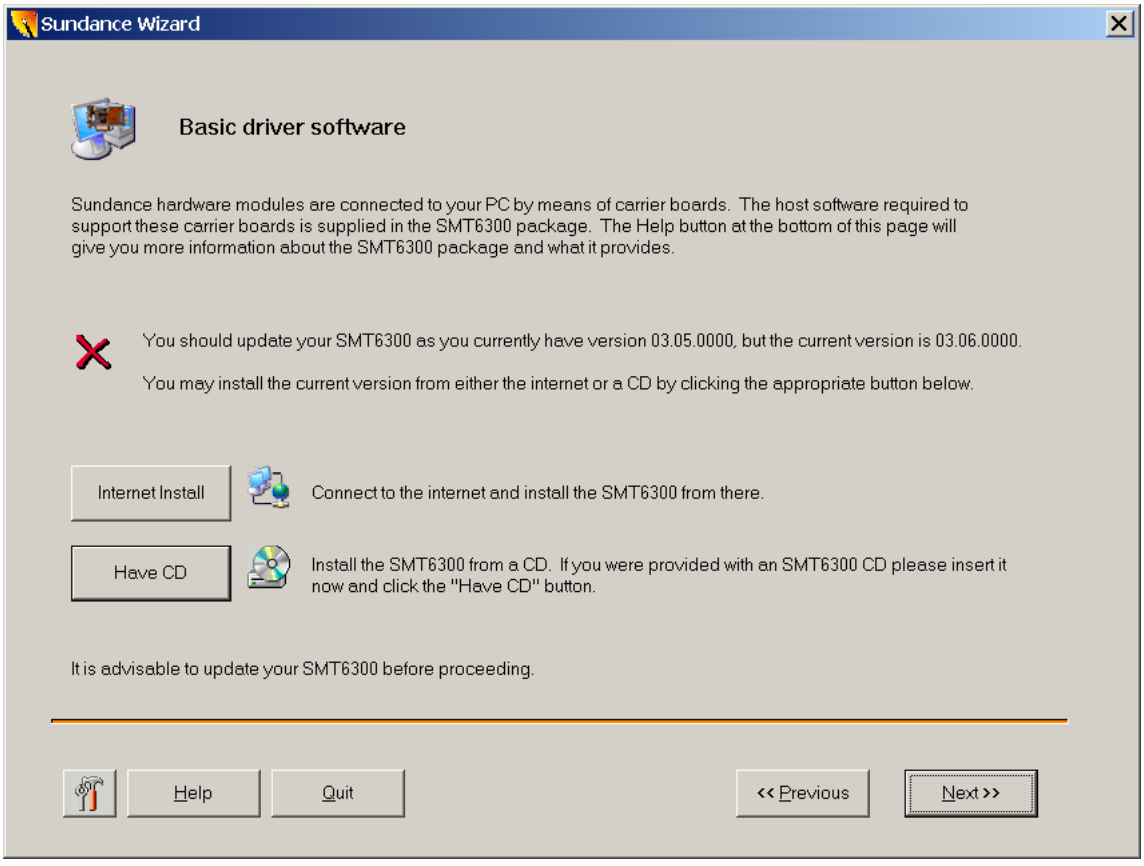

Figura A.12: Sundance Wizard.

A.2.5 Instalação do software Sundance

O software Sundance pode ser instalado a partir de CDs fornecidos juntamente com as placas ou baixado diretamente do *website* da empresa (SUNDANCE, 2006). Para que o hardware sundance possa operar corretamente, o Code Composer Studio, da Texas Instruments precisa estar instalado no computador hospedeiro antes da instalação do software Sundance.

Uma vez com o Code Composer Studio instalado, deve-se instalar o Sundance Wizard. Este programa é um assistente que irá instalar todos os demais módulos necessários para o funcionamento do sistema Sundance. A Figura A.12 mostra uma foto da tela do Sundance Wizard, exibindo os procedimentos para a atualização de um dos pacotes que compõe o sistema Sundance.

Na Figura A.12 observe que há duas opções para a instalação do software. Deve ser escolhida a que for mais apropriada. Em geral a opção "*Internet Install*" deve ser preferida, por conter as versões mais atualizadas possíveis para cada um dos pacotes. A instalação dos pacotes será executada na sequência correta pelo Sundance Wizard, então basta seguir adiante ao final da instalação de cada pacote. Ao final do procedimento, um aviso de configuração do Code Composer Studio (CCS) será emitido. Dependendo da versão do CCS instalada no computador hospedeiro, este procedimento automatizado não irá funcionar.

Configurando o Code Composer Studio

O sistema Sundance necessita que o CCS (Code Composer Studio) esteja instalado e configurado corretamente para funcionar (SMT6400, 2006). A seguir será apresentado os procedimentos para a correta configuração do CCS para operar com o Sistema Sundance. Este trabalho assume a que a versão 3.1 do CCS será instalada e os procedimentos apresentados são válidos apenas para esta versão.

Uma vez com o CCS instalado, o utilitário "*Code Composer Studio Setup*" deve ser executado. Uma janela semelhante a da Figura A.13 será aberta. Esta figura será usada como referência para o processo de configuração do CCS. Inicialmente o sistema (representado na Figura A.13 por "*My System*") deve conter nenhum item. O primeiro passo da configuração será a definição da placa *carrier*. Selecione o nodo raiz (*My System*) na janela da esquerda e selecione "*Create Board*" na janela do meio. Será exibida a lista de opções apresentada na Figura A.13.

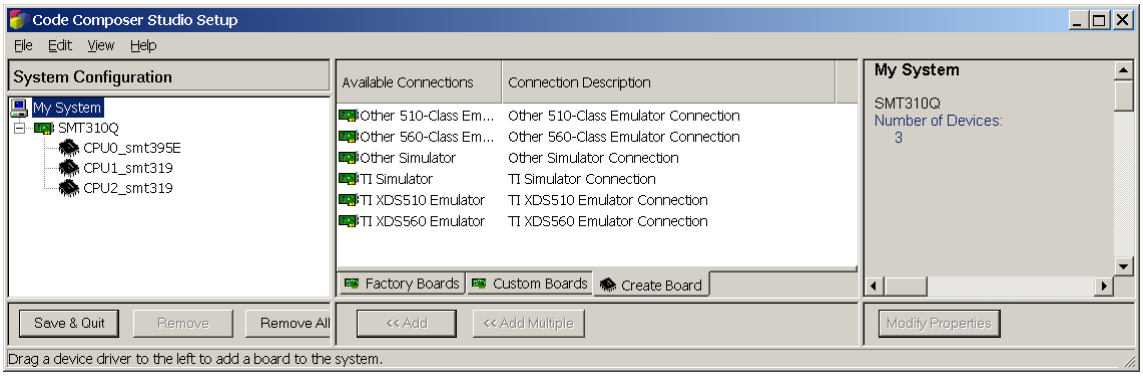

Figura A.13: Configuração do Code Composer Studio.

Na Figura A.13, a opção "TI XDS510 Emulator" deve ser selecionada e adicionada ao sistema. Ao fazer isso uma janela pop-up irá aparecer e os parâmetros mostrados na Figura A.14 e Figura A.15 deverão ser inseridos.

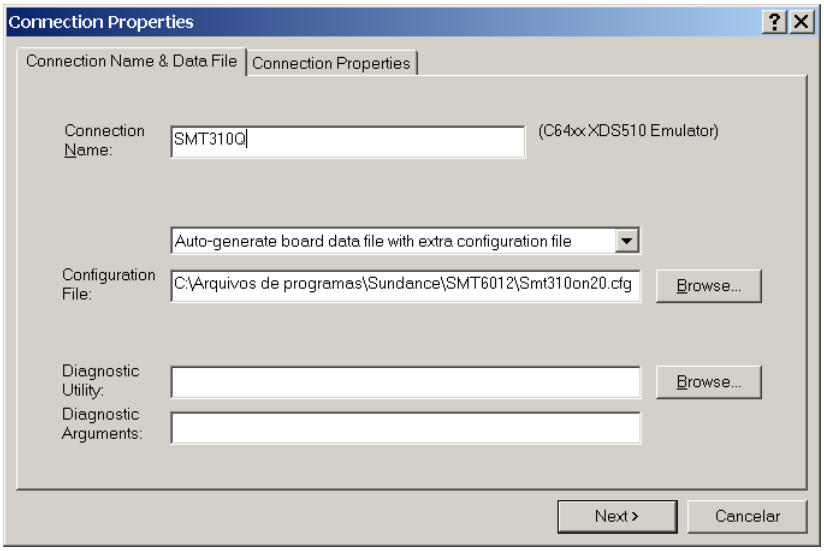

Figura A.14: Configurando a placa *carrier*, passo 1.

Eventualmente o número da porta de I/O atribuído à placa poderá ser diferente. O número desta porta pode ser obtido executando-se o programa "smtboardinfo" que está no pacote SMT6300 da Sundance.

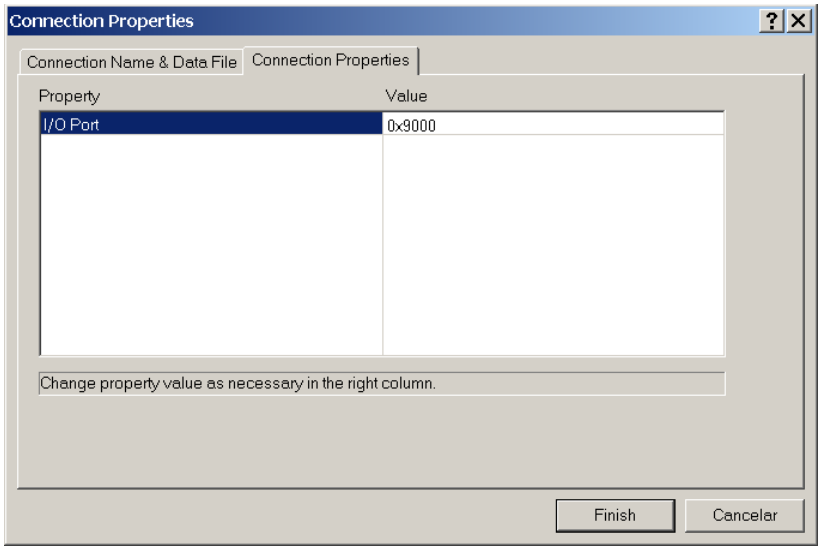

Figura A.15: Configurando a placa *carrier*, passo 2.

Após adicionar a placa *carrier* ao sistema, os DSPs que fazem parte da cadeia JTAG desta devem ser adicionados. No exemplo da Figura A.16, uma SMT395E e duas SMT319 estão ocupando todos os sites da placa SMT310Q. Para adicionar um novo processador, selecione placa *carrier*, como mostrado na Figura A.16 na janela da esquerda e selecione a opção "*Create Board*" na janela central. Selecione a opção TMS320X64xx e na janela pop-up apenas coloque um nome para o processador. Após adicionar todos os processadores DSP ligados na cadeia JTAG da placa *carrier*, a configuração do sistema estará completa. Neste momento, basta salvar e sair.

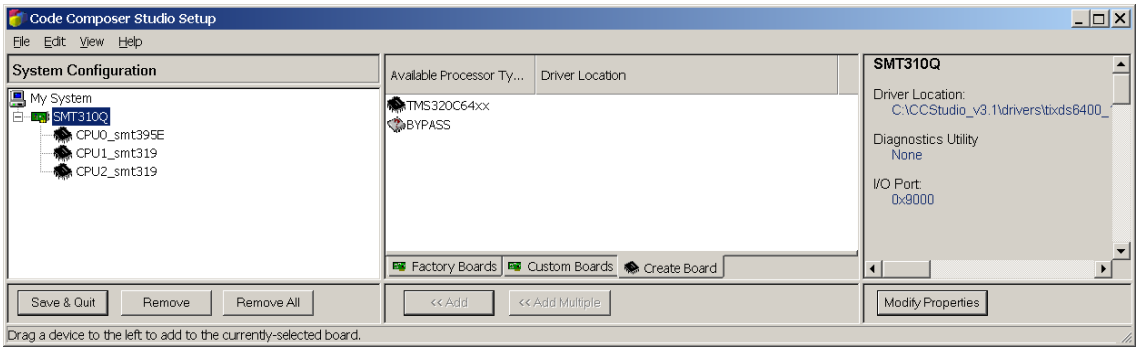

Figura A.16: Configuração do Code Composer Studio

Para testar se as configurações realizadas estão corretas, inicie o CCS. A IDE (*Integrated Developement Enviroment*) do CCS deverá aparecer imediatamente, caso haja apenas um processador no sistema. Caso haja mais de um, como no exemplo da Figura A.16, uma janela chamada "Parallel Debug Manager", como ilustrado na Figura A.17 será apresentada. Em qualquer dos casos, o CCS deve ser conectado ao processador através da placa *carrier* usando a cadeia JTAG. Pra isso, deve ser selecionado o menu Debug->Connect. Se a configuração estiver correta, uma mensagem na barra de status do CCS irá indicar o sucesso. Caso contrário, uma janela *popup* irá aparecer, relatando o problema.

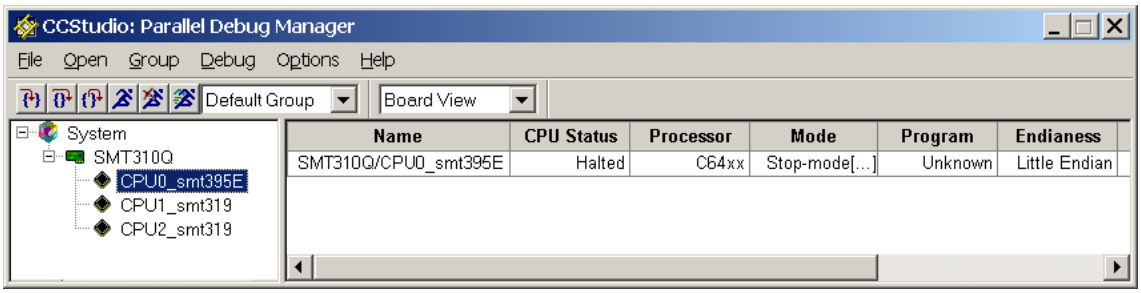

Figura A.17: Tela inicial do Code Composer Studio quando há mais de uma placa de processador conectada à placa *carrier*.

Uma característica importante dos TIMs da Sundance é que há a necessidade de um *bootloader* armazenado em Flash para carregar a configuração do FPGA. Este mesmo *bootloader* pré-configura o DSP e carrega o programa de usuário. Um utilitário, o Flash Programming Utility, é deve ser usado para programar a memória Flash do TIM. Com este utilitário é possível reprogramar o *bootloader*, a configuração do FPGA assim como colocar um ou mais programas de usuário na memória Flash para serem carregados pelo *bootloader*.

A Figura A.18 mostra a tela principal do "Flash Programming Utility". Nesta figura, observe que é possível selecionar o modelo do TIM, um executável para o *bootloader* e um *bitstream* de configuração para o FPGA. O *bootloader* deverá ser alterado sempre que uma nova versão for disponibilizada pela Sundance apenas para refletir melhoramentos nas ferramentas. O *bitstream* de configuração do FPGA deverá ser modificado para conter a lógica de usuário. A Figura A.19 mostra a tela de seleção de blocos de usuário (*User Blocks*). Estes blocos de usuário podem ser dados ou programas que serão armazenados na memória Flash e carregados automaticamente para a memória RAM do DSP pelo *bootloader*. Quando programas forem carregados, um marcador especial, acessível através do menu de contexto, poderá ser ligado para

indicar que o programa dará boot no sistema, ou seja, o controle do processador será transferido do *bootloader* para este programa quando o *bootloader* terminar suas tarefas.

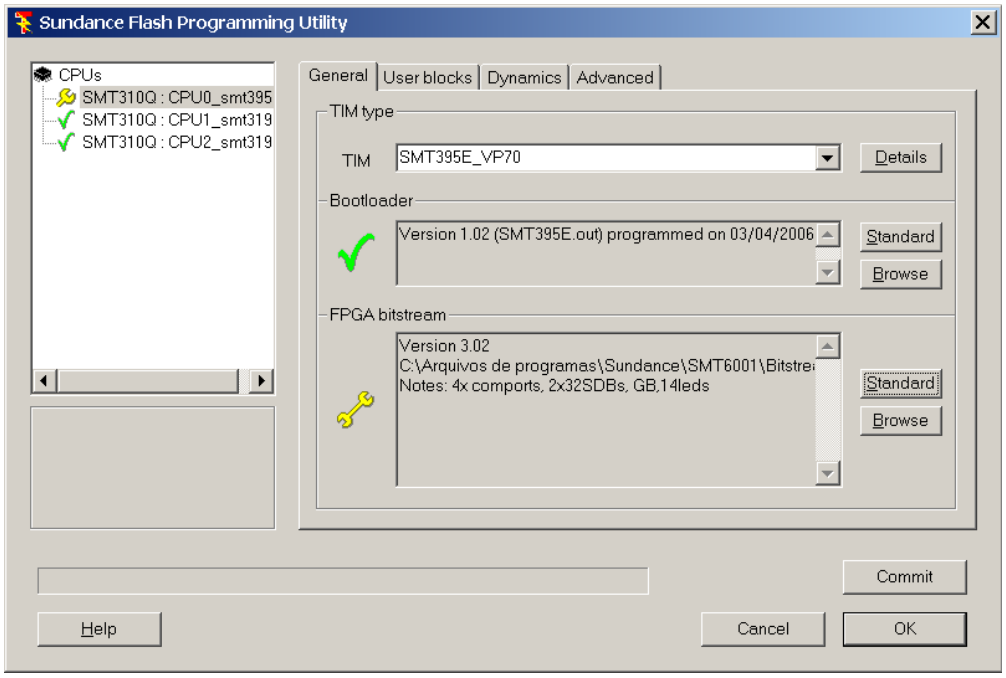

Figura A.18: Tela principal do Flash Programming Utility.

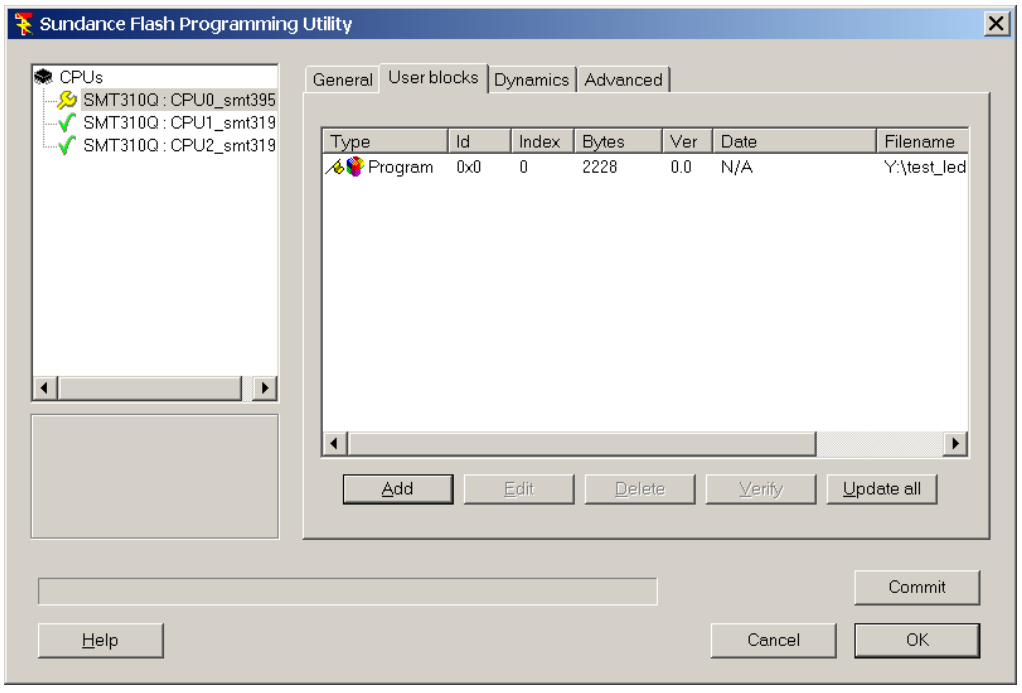

Figura A.19: Flash Programming Utility – Blocos de usuário.

O Flash Programming Utility utiliza a interface JTAG do DSP para fazer a gravação de dados na memória flash. Por esta razão, para que ele possa operar é preciso que o CCS esteja aberto e conectado aos processadores.