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

ANDRÉ LUÍS DEL MESTRE MARTINS

Projeto da Arquitetura de Hardware para Binarização e Modelagem de Contextos para o CABAC do Padrão de Compressão de Vídeo H.264/AVC

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

Prof. Dr. Sergio Bampi
Orientador

Porto Alegre, fevereiro de 2011.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Martins, André Luís del Mestre

Projeto da Arquitetura de Hardware para Binarização e Modelagem de Contextos para o CABAC do Padrão de Compressão de Vídeo H.264/AVC

95 f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2011. Orientador: Sergio Bampi;

1. Binarização e Modelagem de Contextos, 2. Codificação de Entropia, 3. Arquitetura VLSI Dedicada, 4. Codificador Aritmético Binário Adaptativo ao Contexto, 5. Compressão de Vídeo H.264/AVC I. Bampi, Sergio. II.Título.

AGRADECIMENTOS

O primeiro agradecimento e o mais especial é para minha Vó, Alba Silva Del Mestre. Quando soube que eu iria estudar em Porto Alegre, a Vó prontamente ofereceu sua casa para minha moradia, pois sabia que eu precisava dessa ajuda e, com sua atitude, eu me senti em casa. A Vó me cuidou, me deu carinho, me passou tranquilidade para trabalhar. A Vó dividiu seu próprio quarto comigo, abrindo mão de sua privacidade para me abrigar. Isso foi incrível, Vó! Te agradeço demais e se hoje eu sou candidato ao título de Mestre, é porque eu tive a senhora me ajudando no primeiro ano. Muito obrigado por tudo. Te amo.

Gostaria de agradecer aos meus pais, Nika e Mário, e à minha irmã, Gabih. Agradeço o feijãozinho novo com couve, o molho de cachorrão, o carro aspirado e sempre disponível, os desodorantes comprados nos tickets, as cantorias com violão, todos amontoados na cama assistindo qualquer coisa na televisão. Minha família, tenho orgulho de vocês, amo vocês. Quando me vejo pressionado, com preguiça ou desmotivado, é em vocês que penso para seguir batalhando.

Agradeço à família que adotei em Porto Alegre, a família Del Mestre da Rocha. Obrigado Tia Cris e Pedro pelas "free trips" na praia e na serra e pela hospitalidade, sou um admirador do caráter e estilo de vida de vocês. Valeu Pablo Verme, pelas curtidas que fizemos e pelos duetos de Jack Johnson. Valeu Mateus e Ângelo, por serem meus fregueses no Pró-Evoluttion.

Sou muito grato ao Dr. Vagner Rosa. Posso dizer que o Vagner é meu padrinho nesse mestrado. O Vagner me recebeu na UFRGS, foi muito atencioso na minha chegada e, juntamente com meu orientador, me propôs o desafio de trabalhar com o CABAC. Mais do que me cobrar, me dar conselhos e revisar meus trabalhos, agradeço ao Vagner por ter acreditado no meu potencial.

Quero agradecer ao pessoal do Lab215. No Lab215 eu aprendi o que é Grupo de Pesquisa. O ambiente de trabalho é excelente e a capacidade técnica do pessoal é absurda. As confraternizações que fizemos (e continuaremos fazendo) transformaram o coleguismo em amizade. Obrigado Fábio Ramos, Zatt, Marcelo, Dieison, Thaísa, Franco, Miklécio, Guilherme, Débora, Gustavo, Max, Fábio Walter, Claudio, Cristiano, Vizzotto, Duda, Felipe e Vinícius. Espero não ter esquecido de ninguém.

Quero agradecer também os colegas da UFRGS que fizeram cadeira comigo, especialmente aos que estudaram para as provas ou trabalharam comigo: Victor, Jair, Gracieli e Bóris.

Agradeço aos amigos Purtuga, Leitoso e demais amigos de Porto Alegre e Rio Grande, pelos momentos de descontração.

Agradeço ao professor Sergio Bampi pela minha aceitação no mestrado, críticas, sugestões, orientação e compreensão. Espero continuar colaborando com o senhor e com o grupo.

Agradeço à Capes e ao CNPq por terem financiado meus estudos nesses dois anos.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

ASIC	*Application Specific Integrated Circuit*
AVC	*Advanced Video Coding*
BAE	*Binary Arithmetic Encoder*
BCM	*Binarization and Context Modeling*
BIN	Bit de um *binstring*
BINSTRING	*String* de bits binarizados
BITSTREAM	Fluxo de *bits*
CABAC	*Context-Based Adaptive Binary Arithmetic Coding*
CABAD	*Context-Based Adaptive Binary Arithmetic Decoding*
CAVLC	*Context-Based Adaptive Variable Length Coding*
CBF	*Coded Block Flag*
CBP	*Coded Block Pattern*
CIF	*Common Intermediate Format*
CODEC	Codificador e Decodificador
DCT	*Discrete Cosine Transform*
EG	*Exp-Golomb*
EGk	*Kth Order Exp-Golomb* (Exp-Golomb Parametrizável)
FIFO	*First In First Out*
FL	*Fixed Length*
FPGA	*Field Programmable Gate Array*
HDTV	*High Definition Digital Television*
IEC	*International Electrotechnical Commission*
IEEE	*Institute of Electric and Electronics Engineers*
INTER	*Inter Prediction*
INTRA	*Intra Prediction*
ISO	*International Organization for Standardization*
ITU-T	*International Telecommunication Union - Telecommunication*

JVT	*Joint Video Team*
LPS	*Least Probable Symbol*
LUT	*Look-Up Table*
MC	*Motion Compensation*
ME	*Motion Estimation*
MPEG	*Moving Picture Experts Group*
MPS	*Most Probable Symbol*
MVC	*Multiview Video Coding*
MVD	*Motion Vector Difference*
NAL	*Network Adaptation Layer*
PPS	*Picture Parameter Sets*
Q	*Quantization*
IQ	*Inverse Quantization*
QCIF	*Quarter Common Intermediate Format*
QP	*Quantization Parameter*
QPd	*Quantization Parameter* (referente ao processo de binarização)
RAM	*Random Access Memory*
RDO	*Rate-Distortion Optimization*
RGB	*Red, Green, Blue*
rLPS	*Range Least Probable Symbol*
rMPS	*Range Most Probable Symbol*
ROM	*Read Only Memory*
SBTVD	Sistema Brasileiro de Televisão Digital
SE	*Syntactic Element*
SPS	*Sequence Parameter Sets*
SUB&MB	Submacrobloco & Macrobloco (relativo ao processo de binarização)
SVC	*Scalable Video Coding*
T	*Transform*
TB	Terabyte
IT	*Inverse Transform*
TU	*Truncated Unary*
U	*Unary*
UEGk	*Unary/Kth Order Exp-Golomb* (Unário/Exp-Golomb Parametrizável)
UFRGS	Universidade Federal do Rio Grande do Sul
VCL	*Video Coding Layer*

VHDL	*VHSIC Hardware Description Language*
VLC	*Variable Length Coding*
YCbCr	*Luminance, Chrominance Blue, Chrominance Red*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE EQUAÇÕES

RESUMO

O codificador aritmético binário adaptativo ao contexto adotado (CABAC – *Context-based Adaptive Binary Arithmetic Coding*) pelo padrão H.264/AVC a partir de perfil *Main* é o estado-da-arte em termos de eficiência de taxa de bits. Entretanto, o CABAC ocupa 9.6% do tempo total de processamento e seu *throughput* é limitado pelas dependências de dados no nível de bit (LIN, 2010). Logo, atingir os requisitos de desempenho em tempo real nos níveis mais altos do padrão H.264/AVC se torna uma tarefa árdua em *software*, sendo necesário então, a aceleração do CABAC através de implementações em *hardware*.

As arquiteturas de *hardware* encontradas na literatura para o CABAC focam no Codificador Aritmético Binário (BAE - *Binary Arithmetic Encoder*) enquanto que a Binarização e Modelagem de Contextos (BCM – *Binarization and Context Modeling*) fica em segundo plano ou nem é apresentada. O BCM e o BAE juntos constituem o CABAC.

Esta dissertação descreve detalhadamente o conjunto de algoritmos que compõem o BCM do padrão H.264/AVC. Em seguida, o projeto de uma arquitetura de *hardware* específica para o BCM é apresentada. A solução proposta é descrita em VHDL e os resultados de síntese mostram que a arquitetura alcança desempenho suficiente, em FPGA e ASIC, para processar vídeos no nível 5 do padrão H.264/AVC. A arquitetura proposta é 13,3% mais rápida e igualmente eficiente em área que os melhores trabalhos relacionados nestes quesitos.

Palavras-Chave: Binarização e Modelagem de Contextos, Codificação de Entropia, Arquitetura VLSI Dedicada, Codificador Aritmético Binário Adaptativo ao Contexto, Compressão de Vídeo H.264/AVC.

Hardware Architecture Design for Binarization and Context Modeling for CABAC of H.264/AVC Video Compression

ABSTRACT

Context-based Adaptive Binary Arithmetic Coding (CABAC) adopted in the H.264/AVC main profile is the state-of-art in terms of bit-rate efficiency. However, CABAC takes 9.6% of the total encoding time and its throughput is limited by bit-level data dependency (LIN, 2010). Moreover, meeting real-time requirement for a pure software CABAC encoder is difficult at the highest levels of the H.264/AVC standard. Hence, speeding up the CABAC by hardware implementation is required.

The CABAC hardware architectures found in the literature focus on the Binary Arithmetic Encoder (BAE), while the Binarization and Context Modeling (BCM) is a secondary issue or even absent in the literature. Integrated, the BCM and the BAE constitute the CABAC.

This dissertation presents the set of algorithms that describe the BCM of the H.264/AVC standard. Then, a novel hardware architecture design for the BCM is presented. The proposed design is described in VHDL and the synthesis results show that the proposed architecture reaches sufficiently high performance in FPGA and ASIC to process videos in real-time at the level 5 of H.264/AVC standard. The proposed design is 13.3% faster than the best works in these items, while being equally efficient in area.

Keywords: Binarization and Context Modeling, Entropy Encoding, VLSI Dedicated Architecture, Context-Based Adaptive Binary Arithmetic Coding, H.264/AVC Video Compression.

1 INTRODUÇÃO

O crescente interesse por vídeo digital é reflexo das diversas possibilidades de aplicações para este tipo de recurso. Muitos fatores têm contribuído para a popularização do vídeo digital: fatores comerciais, legislação, mudanças sociais e avanços tecnológicos. Do ponto de vista da tecnologia, estes fatores incluem a melhor infraestrutura de comunicações, com acesso generalizado e relativamente barato para redes de banda larga, redes móveis 3G, eficazes redes locais sem fio e maior capacidade de transmissão. Além disso, dispositivos cada vez mais sofisticados, com grande conjunto de funcionalidades condicionados em dispositivos embarcados e o desenvolvimento de aplicações fáceis de usar para gravação, edição, compartilhamento e visualização de material de vídeo compõem um cenário de grande relevância para a difusão de vídeo digital (RICHARDSON, 2010). Esta dissertação se concentra em um aspecto técnico fundamental para a ampla adoção da tecnologia de vídeo digital: a compressão de vídeo.

Diversas técnicas de codificação de vídeo têm sido propostas e pesquisadas. Centenas de trabalhos de pesquisa são publicados a cada ano descrevendo inovadoras técnicas de compressão de vídeo digital (RICHARDSON, 2003). Apesar dessa ampla gama de inovações, aplicações de codificação de vídeo tendem a utilizar um número limitado de técnicas padronizadas para a compressão de vídeo (RICHARDSON, 2010). Atualmente, o padrão H.264/AVC é o estado-da-arte em compressão de vídeo digital, atingindo ganhos de 50% nas taxas de compressão com a mesma qualidade de vídeo comparado com outros padrões (WIEGAND, 2003). Entretanto, a baixa taxa de *bits* e a alta qualidade de vídeo é obtida através de sofisticados algorítmos que aumentam significativamente a complexidade computacional nos processos de codificação e decodificação de vídeo (RICHARDSON, 2003).

Para algumas aplicações, a capacidade de processamento em tempo real é a chave para o desenvolvimento de soluções de mercado como: *broadcasting*, videoconferência, videofone, TV digital, filmadoras e câmeras digitais. Esses dispositivos necessitam de aceleradores de *hardware* que permitam a codificação de vídeos em tempo real enquanto executam algoritmos de elevado custo computacional. A codificação de entropia é a última etapa de codificação de vídeo, envolvendo um número significativo de operações a nível de bit e de controle (MARPE, 2003). Entre os codificadores de entropia presentes no padrão H.264/AVC, o codificador aritmético binário adaptativo ao contexto (CABAC – *Context Adaptive Binary Arithmetic Coder*) é o codificador de entropia mais eficiente em termos de taxas de compressão (MARPE, 2003).

O CABAC pode ser divido em duas partes: Binarização e Modelagem de Contextos (BCM – *Binarization and Context Modeling*) e Codificador Aritmético Binário (BAE – *Binary Arithmetic Encoder*). As dependências de dados que constituem o gargalo do CABAC residem no bloco BAE (LIN, 2010), por isso alguns trabalhos relacionados

apresentam arquiteturas de *hardware* apenas para o BAE (KUO, 2006; CHEN, 2005; ZHENG, 2008; ROSA, 2010) ou parte do BAE (CHEN, 2007b; LI, 2006). Os trabalhos que propõem arquiteturas completas para o CABAC (PASTUZSAK, 2008; WU, 2009), não detalham satisfatoriamente os algoritmos e as soluções arquiteturais encontradas para o BCM. Além disso, algumas dessas arquiteturas de CABAC desenvolvidas implementam apenas um conjunto limitado de funcionalidades do BCM (OSORIO, 2006; LIU, 2007). Em geral, os trabalhos relacionados são capazes de processar vídeos em alta resolução, mas (ROSA, 2010) prova em seu trabalho que paralelizar o CABAC é uma solução demasiadamente complexa e ineficiente em área e energia.

Nesse cenário, o principal objetivo desta dissertação é desenvolver arquiteturas de *hardware* compatíveis com o padrão H.264/AVC para o BCM. As arquiteturas de *hardware* para codificação de vídeo devem ser eficientes em área e energia e devem processar vídeos de alta resolução em tempo real.

A metodologia de desenvolvimento adotada foi a de descrição dos módulos de *hardware* em nível RTL, incluindo a implementação e síntese em FPGA e ASIC. A metodologia de validação do *hardware* foi a de realizar a simulação funcional e a validação da arquitetura pela extração dos vetores de teste de um modelo referencial (*golden model*) em software disponível na literatura, o x264 (X264, 2010).

As principais contribuições deste trabalho são: (1) uma descrição detalhada dos algoritmos que compõe o BCM, apresentada no Capítulo 3, em conjunto de uma visão geral do CABAC e (2) o desenvolvimento de arquiteturas de *harware* eficientes para o BCM do padrão H.264/AVC, apresentado no Capítulo 4. O Capítulo 2 introduz conceitos básicos de compressão de vídeo digital e do padrão H.264/AVC. O Capítulo 4 contém ainda, uma revisão bibliográfica com as principais arquiteturas de *hardware* desenvolvidas para o CABAC, seguido de uma comparação com os resultados obtidos. Finalmente, o trabalho é concluído no Capítulo 5.

2 CONCEITOS BÁSICOS DE COMPRESSÃO DE VÍDEO DIGITAL E PANORAMA DO PADRÃO H.264/AVC

Este capítulo tem por objetivo apresentar conceitos básicos relacionados ao vídeo digital. Além disso, um panorama do padrão de compressão de vídeo H.264/AVC, ilustrando seus aspectos mais importantes. Os tópicos abordados neste capítulo são essenciais para a compreensão dos demais capítulos desta dissertação.

2.1 Conceitos Básicos de Compressão de Vídeos

Uma imagem digital é composta por vários pontos discretos, chamados de *pixel* (*Picture Element* – Elemento da imagem), dispostos em uma matriz bidimensional. A dimensão dessa matriz é chamada de resolução. Cada *pixel* contém informações referentes à cor e/ou ao brilho. Um vídeo digital é formado por uma série de imagens exibidas em sequência sob uma determinada taxa temporal. Cada imagem que compõe o vídeo é chamada de quadro (*frame*). A frequência em que os quadros aparecem na tela tem que ser suficiente para fornecer a percepção de movimento ao espectador. No codificador H.264/AVC, por exemplo, a frequência típica de quadros é 30 *fps* (*frames per second* – quadros por segundo).

Vídeos digitais sem compressão necessitariam de uma enorme quantidade de bits de memória para serem armazenados e de elevadas taxas de transmissão para que possam ser transmitidos. Por exemplo, para processar um vídeo na resolução 1080p, seria necessário enviar ou armazenar 30 *frames* de 1920x1080 *pixels* em um segundo (assumindo que cada pixel é uma informação de 24 bits), ou seja, seria como enviar 30 imagens de 2 *Mpixel* por segundo, o que demandaria uma memória de 1 TB em 10 minutos de vídeo e uma largura de banda de 1,5Gbps. Assim, fica clara a necessidade de comprimir os vídeos para permitir seu envio ou armazenamento em aplicações multimídia.

Para realizar a compressão de vídeo é necessário explorar o elevado grau de redundância de seus dados (GONZALEZ, 2003). O objetivo da compressão é justamente o desenvolvimento de técnicas para eliminação dessa informação redundante a fim de reduzir a quantidade de informação para representar um vídeo. Isso significa que grande parte da enorme quantidade de dados existentes em uma sequência de vídeo pode ser descartada, possibilitando sua representação com uma quantidade de bits muito menor do que a original.

Existem muitas formas de representar as cores dos *pixels* por meio digital variando de acordo com a característica do dispositivo digital e da percepção da cor e da luz pelo sistema visual humano. Cada sistema para a representação da informação de cor e luz é chamado de "espaço de cores" e a definição do espaço de cores a ser utilizado pelo compressor de vídeo é o primeiro passo em seu desenvolvimento. No espaço de cores RGB, cada *pixel* de uma imagem é representada por três informações que correspondem à intensidade das cores vermelha, verde e azul (***R**ed, **G**reen and **B**lue*).

Entretanto, o padrão H.264/AVC utiliza o espaço de cores YCbCr, onde também três componentes são utilizadas na representação e correspondem à luminância (Y), que define a intensidade luminosa ou o brilho, à crominância azul (Cb) e à crominância vermelha (Cr) (MIANO, 1999). O YCbCr pode ser obtido através de uma transformação linear da informação de cores no espaço RGB. O YCbCr é utilizado no H.264/AVC porque o sistema visual humano é mais sensível às informações de brilho que das informações de cores. Essa característica do espaço de cores YCbCr pode ser utilizada para reduzir a quantidade de informação necessária a ser processada através de uma operação conhecida como subamostragem de cores. Além disso, os componentes R, G e B tem elevado grau de correlação, ao contrário do YCbCr, dificultando o processamento dos componentes de forma independente (GONZALEZ, 2003).

A subamostragem de cores é obtida através da redução da resolução espacial da informação de crominância no espaço YCbCr. A relação entre a resolução de crominância e de luminância é uma relação proporcional de quantidade de cada componente do espaço de cores. As relações mais comuns são 4:4:4, 4:2:2 e 4:2:0 (ou 4:1:1). No formato 4:2:0, para cada 4 amostras de luminância organizada em uma matriz 2x2, são relacionadas apenas 1 amostra de crominância vermelha e outra azul, ou seja, 3 amostras das cores azul e vermelha são descartadas. No formato 4:2:2, para cada 4 amostras de Y, são relacionadas 2 amostras de Cb e Cr. No formato 4:4:4 não há descarte de informação. (Figura 2.1) A subamostragem de cores é uma ferramenta de pre-processamento (parte não integrante do codificador/decodificador) importante para os compressores de vídeos, visto que reduz pela metade a quantidade informação (no formato 4:2:2 a redução é de 33%) antes mesmo de qualquer processo de codificação ser efetivamente realizado.

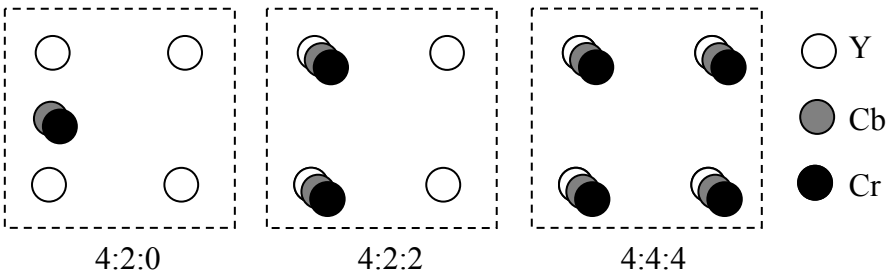

Figura 2.1: Subamostragem de cores (RICHARDSON, 2003)

2.2 Redundância de Dados na Representação de Vídeos

O vídeo digital utiliza de uma quantidade muito grande de dados para ser representado. Entretanto esses dados, em cenas naturais, apresentam elevada redundância, ou seja, não contribuem com informações relevantes para a representação do vídeo (RICHARDSON, 2003). A principal função dos codificadores de vídeo é

identificar e remover essas redundâncias do sinal de vídeo de modo que a quantidade de dados para representá-lo seja a menor possível. Em aplicações de vídeo digital, existem três tipos diferentes de redundância que são exploradas pelos codificadores de vídeo as quais estão detalhadas a seguir.

Redundância Espacial – também conhecida como redundância intraquadro (GHANBARI, 2003) – *pixels* vizinhos tendem a ter valores muito parecidos ou até mesmo iguais, ou seja, a mesma informação é repetida em uma determinada área dentro da matriz de *pixels* de um quadro, caracterizando a redundância de informação entre os *pixels* dessa área.

Redundância Temporal – também chamada de redundância interquadro (GHANBARI, 2003) – a redundância de dados temporal é identificada quando blocos de *pixels* não modificam seu valor entre um quadro e outro ou ainda, apresentam uma pequena variação de valores ou até, têm seu posicionamento deslocado em relação a outro quadro.

Redundância Entrópica – está relacionada com a forma de representação computacional dos símbolos codificados e não se relaciona diretamente ao conteúdo da imagem. Consiste em representar símbolos mais frequentes com menor cadeia de dados. A redundância entrópica existe quando as frequências de ocorrência dos símbolos são diferentes entre si. Assumindo que a distribuição de ocorrência de símbolos seja uniforme, ou seja, uma média, então não há redundância entrópica a ser explorada. Se a distribuição dos dados é uma curva não-uniforme como, por exemplo, uma curva normal ou exponencial, a redundância entrópica existe e quanto menor a variância dos dados nas curvas não uniformes, mais redundância entrópica existe entre os dados e mais esses dados poderão ser comprimidos.

2.3 Organização do padrão H.264/AVC

A primeira versão do padrão H.264/AVC (INTERNATIONAL, 2003) foi lançada em 2003 contendo suporte para os perfis *Baseline*, *Main* e *Extended*. Em seguida, foram lançadas várias revisões da norma que constituem na adição de novos perfis e ferramentas além da criação de extensões do padrão. A última versão do padrão H.264/AVC é de março de 2009 e detalha as funcionalidades dos perfis *High* (INTERNATIONAL, 2005), além das extensões para codificação de vídeo escalável (INTERNATIONAL, 2007) e multivisão (INTERNATIONAL, 2009). Os perfis do padrão serão explicados mais adiante ainda nessa Seção, enquanto que os anexos que tratam de escalabilidade e multivisão estão fora do escopo deste trabalho.

O padrão foi desenvolvido pela JVT (*Joint Video Team*), uma união de especialistas da ITU-T (*International Telecommunication Union – Telecommunication*) e da ISO/IEC (*International Organization for Standardization/International Electrotechnical Commission*) (RICHARDSON, 2003). A JVT disponibilizou um texto normatizando todo o processo de decodificação e transmissão do vídeo codificado segundo o padrão H.264/AVC (INTERNATIONAL, 2003). O processo de codificação não é descrito nesse texto. Além da norma do H.264, um software que realiza a codificação e a decodificação de vídeos no formato H.264/AVC escrito na linguagem de programação C foi disponibilizado e seu código fonte é aberto e livre (SUHRING, 2009). Com estes dois documentos é possível realizar o desenvolvimento e a validação de *software* e *hardware* de vídeo para o padrão H.264/AVC.

2.3.1 Estrutura de codificação

No H.264/AVC, cada quadro de vídeo pode ser, ou não, dividido em *slices* de acordo com a necessidade identificada na codificação. O *slice* é a menor estrutura sem dependência de dados de vizinhança possível e o codificador pode particionar o quadro em diversos *slices* para aumentar a robustez à perda de dados durante o processo de transmissão, o que também é útil para que possam ser processados de forma independente pelo codificador, dependendo de uma determinada necessidade de desempenho ou qualidade.

Na Figura 2.2(a) um quadro repartido em dois *slices* é representado. Para codificar o vídeo, um conjunto de *pixels* agrupados em uma matriz de dimensão menor que o *slice* é processado por vez. No padrão H.264/AVC, essa matriz de *pixels* é chamada de bloco e o bloco de maior tamanho possível é chamado de macrobloco e tem 16x16 *pixels*. Os macroblocos são organizados nos *slices* na ordem *raster scan*. Cada macrobloco pode ser partido em blocos menores de 8x16, 16x8 e 8x8 *pixels* e, caso o tamanho escolhido seja o de 8x8, este pode ser dividido novamente em blocos de 4x8, 8x4 e 4x4 *pixels*. Os blocos de tamanho 8x8 são chamados de sub-macroblocos. A Figura 2.2(b) ilustra todas as divisões de bloco previstas pelo padrão H.264/AVC.

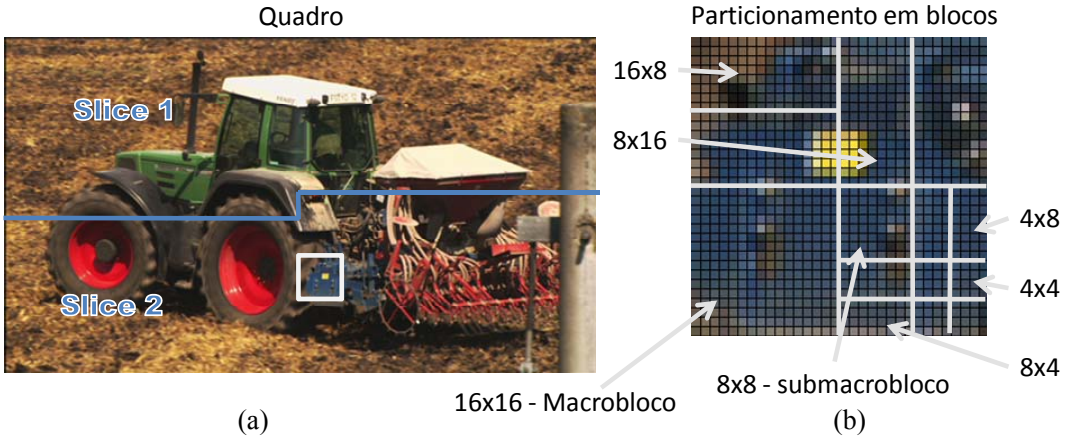

Figura 2.2: Particionamento de quadros e blocos segundo o padrão H.264/AVC.

O padrão H.264/AVC prevê cinco tipos diferentes de *slices:* I (Intra), P (preditivo), B (bi-preditivo), SI (*switch*-I) e SP (*switch*-P). Um *slice* do tipo I (Intra) foi codificado com predição intraquadros e deve conter somente macroblocos do tipo I. Um *slice* P pode conter macroblocos do tipo P e I e um *slice* B pode conter macroblocos do tipo B e I. Os *slices* SI e SP são análogos aos *slices* I e P e são utilizados para permitir a variação dinâmica no fluxo de bits em aplicações de *streaming* (WIEGAND, 2003). Um quadro do vídeo pode ser codificado em um ou mais *slices*, cada um contendo um número inteiro de macroblocos e os *slices* dentro de um mesmo quadro podem ser de diferentes tipos entre si. O número de macroblocos em um *slice* pode variar de um até o número total de macroblocos do quadro (RICHARDSON, 2003).

Cada quadro codificado recebe uma numeração e é armazenado em até duas listas de quadros para que seja explorada a redundância interquadros. Posto isso, os macroblocos do tipo P são codificados usando a codificação interquadros, a partir de quadros de referência previamente codificados. Cada partição de macrobloco é codificada utilizando como referência um quadro da lista 0. Os macroblocos do tipo B também são codificados usando a codificação interquadros. Cada partição de macrobloco pode ser codificada utilizando um ou dois quadros de referência, um quadro

proveniente da lista 0 e outro da lista 1. As partições dentro de um mesmo macrobloco são independentes, ou seja, as partições não precisam referenciar quadros da mesma lista, podem referenciar quadros diferentes dentro de uma mesma lista e ainda, uma partição pode ser bi-preditiva e a outra partição não. A única restrição que existe nos macroblocos B é que todas suas partições sejam inter-preditivas. Esse conceito é análogo para macroblocos P e isso representa um acréscimo importante de complexidade ao codificador. (RICHARDSON, 2003).

Além dos tipos de macroblocos já citados, existe o macrobloco *skip*, presente apenas nos *slices* P e B, e o macrobloco *direct*, presente nos *slices* B. O macrobloco *skip* é um macrobloco que não envia informação nenhuma de resíduos ou predição. A presença de um macrobloco *skip* significa que o decodificador deve calcular as informações deste macrobloco baseado nos macroblocos vizinhos e nos quadros armazenados nas listas. No macrobloco *direct* há apenas a informação de resíduos e não existe qualquer informação relacionado à predição interquadros (CORRÊA, 2010). Essa informação deve ser calculada com base nos macroblocos vizinhos. A lista dos tipos de *slice* e tipos de macroblocos que podem ser encontrados em cada tipo de *slice* estão na Tabela 2.1.

Tabela 2.1: Listagem de slices, macroblocos e particionamento do H.264/AVC

Slice	**Macrobloco**	**Particionamento**
I/SI	I	4x4, 8x8 (high), 16x16
P/SP	I, P, skip	4x4, 4x8, 8x4, 8x8,
B	I, B, skip, direct	8x16, 16x8, 16x16

2.3.2 Perfis e Níveis do H.264/AVC

Os vídeos digitais são utilizados nas mais variadas aplicações e contextos. Para cada aplicação, existem requisitos que devem ser atingidos e limitações que devem ser respeitadas que estão relacionadas com o tipo de dispositivo que está (de)codificando o vídeo e com o meio de transmissão do vídeo. Assim, o H.264/AVC é categorizado de acordo com uma série de perfis que definem um subconjunto de ferramentas de (de)codificação previstas pelo padrão.

O perfil *Baseline* é o mais básico por ter um conjunto limitado de ferramentas suportando apenas *slices* I e P e o codificador de entropia CAVLC (*Context-Based Adaptive Variable Length Coding* - códigos de comprimento variável adaptativos ao contexto). O perfil *Baseline* foi definido para ser utilizado em aplicações como videoconferência, vídeo telefonia e vídeos sem fio. O perfil *Main* é direcionado à aplicações de televisão digital e em armazenamento de vídeo. O perfil *Main* suporta *slices* B e o codificador de entropia CABAC (*Context-Based Adaptive Binary Arithmetic Coding* - codificação aritmética adaptativa ao contexto). O perfil *Extended* é voltado para aplicações em *streaming* de vídeo, melhorando a robustez a erros de transmissão. Estes três perfis foram inicialmente propostos pelo padrão H.264/AVC no lançamento da primeira versão da norma, mas estes perfis não incluíram suporte para vídeos com resoluções mais elevadas, tampouco ferramentas necessárias para ambientes profissionais.

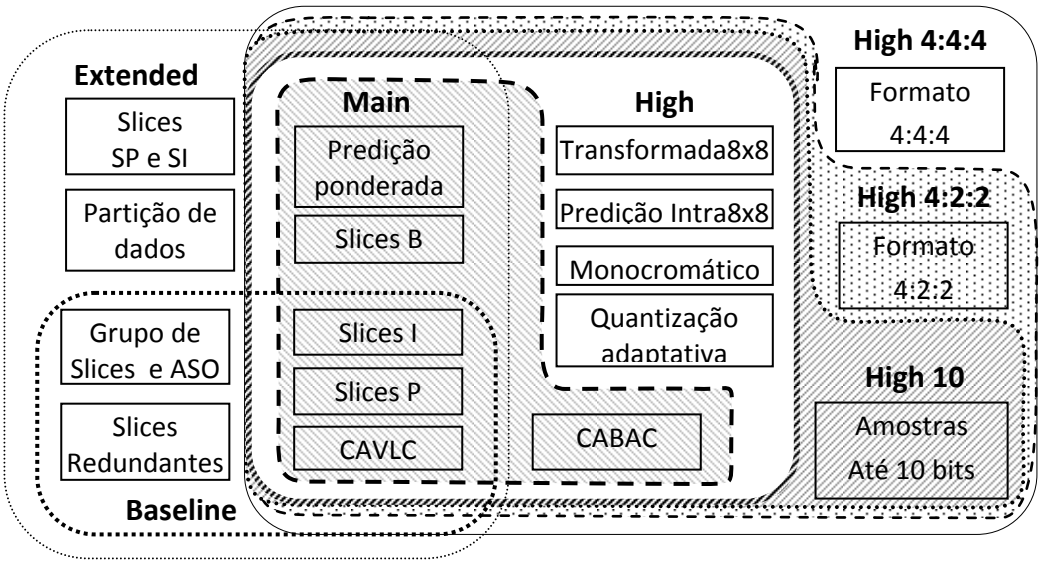

Figura 2.3: Relação entre ferramentas e os diferentes perfis definidos pelo padrão H.264/AVC.

Os perfis *High* possuem uma série de ferramentas inovadoras em relação aos perfis originais: suportam transformada adaptativa 4x4 e 8x8, predição intraquadros 8x8 e matrizes de quantização baseadas em percepção (SULLIVAN, 2004). O perfil *High* inclui vídeo com 8 bits por amostra e com subamostragem de cores 4:2:0, assim como os outros três perfis citados. O perfil *High* 10 suporta todas as ferramentas do perfil *High*, mas suas amostras podem ter 8, 9 ou 10 bits. O perfil *High* 4:2:2 suporta todas ferramentas do *High* 10 além do suporte à subamostragem de cor 4:2:2 (INTERNATIONAL, 2005). O perfil *High* 4:4:4 suporta todas as ferramentas do perfil *High* 4:2:2 e a subamostragem 4:4:4. As ferramentas de codificação agrupadas em perfis estão resumidas na Figura 2.3.

Além dos perfis, o padrão categoriza a codificação em níveis de acordo com a resolução ou tamanho do vídeo, taxa de processamento e perfil, estratificando e limitando os requisitos de processamento e memória do decodificador. A Tabela 2.2 ilustra todos os níveis do padrão H.264/AVC. Essa tabela é muito útil para determinar as configurações da (de)codificação dada uma determinada limitação de largura de banda, processamento ou de perfil. Por exemplo, quando um vídeo é codificado, é preciso escolher qual a resolução (tamanho do quadro) e o perfil desejado, porém é preciso saber se a largura de banda de transmissão de dados disponível é suficiente para suportar as configurações desejadas. No contexto de projeto de arquiteturas de *hardware*, no qual esse trabalho se enquadra, essa tabela é utilizada para determinar o nível de suporte das arquiteturas desenvolvidas. As taxas de processamento em Kbtis/s para os perfis *Baseline* e *Extended* são iguais ao perfil *Main*.

Tabela 2.2: Níveis do padrão H.264/AVC.

Nível	Tamanho: macroblocos por quadro	Processamento: macroblocos/s	Bitstream: Kbits/s			
			Main	High	High 10	High 4:2:2/4:4:4
1	99	1485	64	80	192	256
1b	99	1485	128	160	384	512
1.1	396	3000	192	240	576	768
1.2	396	6000	384	480	1152	1536
1.3	396	11880	768	960	2304	3072
2	396	11880	2000	2500	6000	8000
2.1	792	19800	4000	5000	12000	16000
2.2	1620	20250	4000	5000	12000	16000
3	1620	40500	10000	125000	30000	40000
3.1	3600	108000	14000	175000	42000	56000
3.2	5120	216000	20000	25000	60000	80000
4	8192	245760	20000	25000	60000	80000
4.1	8192	245760	50000	50000	150000	200000
4.2	8704	522240	50000	50000	150000	200000
5	22080	589824	135000	168750	405000	540000
5.1	36864	983040	240000	300000	720000	960000

2.4 Núcleo do padrão H.264/AVC

Nesta Seção serão apresentados todos os módulos que compõe o codificador H.264/AVC. Será dada uma ênfase especial nas informações de interesse ao codificador de entropia, foco deste trabalho.

Na Figura 2.4 está apresentado o diagrama de blocos tradicional do codificador H.264/AVC. O bloco de predição interquadros composto pela Estimação de Movimento (ME) e Compensação de Movimento (MC) explora as redundâncias temporais do vídeo comparando o quadro atual com quadros já codificados. Paralelamente ao processo de inter-predição, ocorre a predição intraquadros a qual visa eliminar as redundâncias espaciais do quadro. Em seguida há uma escolha, representada na Figura 2.4 pela chave, sobre qual predição conseguiu melhor resultado e apenas a informação de um tipo de predição é levada adiante para os próximos passos da codificação.

A informação do quadro predito e do quadro atual é subtraída e a informação resultante é chamada de resíduo. As amostras de resíduos são enviadas para o bloco de transformadas (T) para que sejam convertidas do domínio espacial para o domínio da frequência. Em seguida, o bloco de quantização (Q) determina a intensidade de descarte de informação que será realizado nas amostras transformadas e, finalmente, todas as informações geradas durante o processo de codificação do quadro são enviadas para a codificação de entropia encontrar as redundâncias estatísticas e encerrar o processo de codificação do quadro.

Para realizar a predição dos dados de forma precisa, o codificador necessita obter a mesma informação que o decodificador terá disponível em sua entrada. Para isso, o codificador não pode utilizar os quadros originais para realizar a predição para não haver incompatibilidade dos dados entre o codificador e o decodificador. Portanto, o codificador utiliza quadros reconstruídos para realizar a predição. O processo de reconstrução consiste em processar os dados provenientes da quantização e enviá-los para o processo de quantização e transformação inversa (IT e IQ) para que sejam somados com a imagem predita. Em seguida, são processados por um filtro que reduz o

efeito de bloco causado pelo processo de codificação e finalmente são guardados em memória para que possam ser utilizados pela predição.

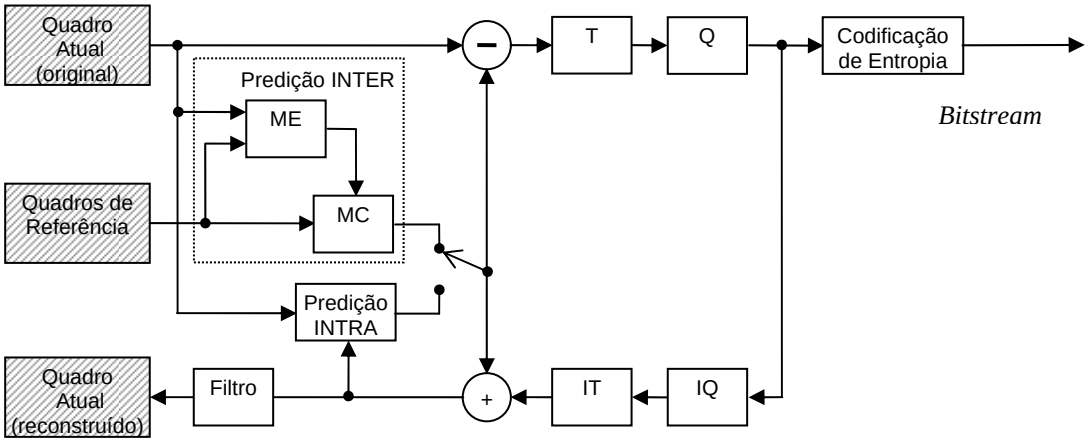

Figura 2.4: Diagrama de blocos de um codificador H.264/AVC.

2.4.1 O Bloco de Predições

O bloco de predições é o responsável por realizar a construção do quadro predito para que este seja subtraído pelo quadro atual resultando no quadro residual. O padrão H.264/AVC prevê dois tipos de predição de movimento conhecidas como preditores Intraquadros e Interquadros que exploram as redundâncias espaciais e temporais respectivamente. As duas predições são realizadas simultaneamente e o controle realiza a escolha do melhor modo de predição baseado em uma série de critérios que levam em consideração especialmente o compromisso entre qualidade e *bitrate*. Essa escolha não é trivial, pois precisa levar em consideração o *tradeoff* entre qualidade do vídeo e a taxa de bits esperada para a saída considerando os vários modos de predição disponíveis no padrão. Mais detalhes sobre como o codificador realiza a escolha para o melhor quadro predito pode ser encontrada em Corrêa (2010).

A Figura 2.5 mostra um exemplo hipotético de um quadro de resíduos gerado pela predição. Sob o ponto de vista da entropia, esse processo é de extrema importância, pois evidencia a redundância da informação intrínseca das amostras. Em cenas naturais, a taxa de ocorrência de cada valor possível de símbolos (ou *pixel*, ou amostra) de um quadro não predito é aproximadamente igual. Considerando um preditor de quadros eficiente, no bloco residual isso não acontece, pois a redundância dos dados foi evidenciada pelos preditores e, portanto, a taxa de ocorrência dos símbolos é altamente variável

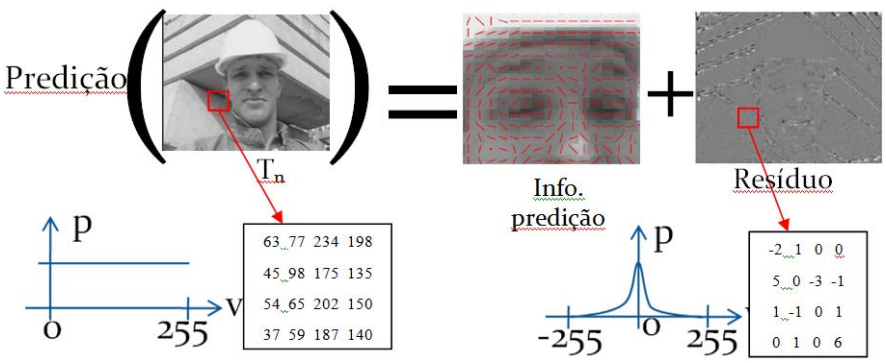

Figura 2.5: Exemplo genérico de um quadro residual predito. (adaptado de ROSA, 2010)

A predição é muito importante na geração da informação residual: quanto maior a precisão da predição, menor a quantidade de informação contida nos resíduos e, consequentemente, maior compressão poderá ser obtida. Nas próximas duas subseções será apresentado um panorama sobre o funcionamento das predições interquadros e intraquadros.

2.4.1.1 Predição Interquadros

A predição interquadros é formada pelos blocos da Estimação de Movimento (ME – *Motion Estimation*) e da Compensação de Movimento (MC – *Motion Compensation*) os quais estão estritamente relacionados.

A ME tem que procurar entre os macroblocos dos quadros que estão nas listas de referência qual o que mais se assemelha ao macrobloco do quadro atual. É definida uma área de busca ao redor do macrobloco atual para procura do melhor casamento (*match*) possível. Assim que o bloco com melhor *match* é encontrado, é gerado um vetor de movimento que localiza o quadro com melhor casamento e também fornece a informação de qual quadro das listas 0 e 1 esse vetor de movimento referencia (Figura 2.6). Os vetores de movimento e as listas de referência são enviados diretamente para codificação de entropia.

Para a realização da estimação de movimento é considerado apenas o componente de luminância do macrobloco. Cada componente de crominância possui a metade da resolução horizontal e vertical do componente de luminância, então as componentes horizontais e verticais de cada vetor de movimento são divididas por dois para serem aplicados aos blocos de crominância.

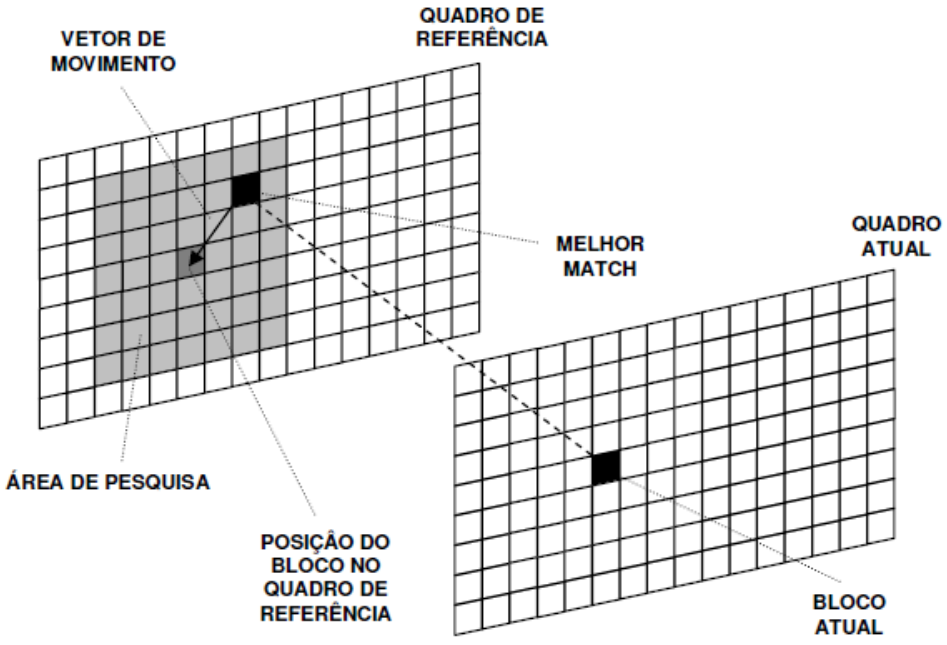

Figura 2.6: Busca pela área de máxima semelhança em um quadro de referência. (PORTO, 2007)

Definir o melhor casamento do macrobloco atual na estimação de movimento não é uma tarefa trivial e é considerada a mais complexa do codificador H.264/AVC (PURI, 2004). A principal inovação proporcionada pela ME é a utilização de tamanhos de blocos variáveis. A estimação de movimento pode ser feita utilizando qualquer tamanho

de bloco previsto pelo padrão H.264/AVC. Os diferentes particionamentos de blocos estão explicados na Seção 2.3.1 e ilustrados na Figura 2.1(b). Em contrapartida, este grande custo computacional, fruto das inovações inseridas neste bloco, juntamente com o bloco MC, caracterizam o preditor interquadros como a principal fonte de ganhos de compressão do H.264/AVC em relação aos demais padrões de compressão de vídeo padronizados existentes (WIEGAND, 2003; RICHARDSON, 2003). Maiores detalhes sobre a estimação de movimento não serão abordados neste texto, podendo ser encontrados em Porto (2007).

O Compensador de Movimento é o montador do quadro predito pela predição interquadros que será subtraído do quadro atual para gerar o quadro de resíduos. O MC precisa levar em consideração para montar o quadro predito as informações de vetor de movimento e quadros de referência gerados pela ME. Além disso, o MC tem que considerar uma série de complexas funcionalidades previstas no padrão para construir o bloco predito como, por exemplo, tratar múltiplos tamanhos de blocos, construir corretamente os macroblocos *skip*, *direct* e os macroblocos que usam predição bi-preditiva. Mais detalhes sobre o processo de Compensação de Movimento podem ser encontrados em Zatt (2006)

2.4.1.2 Predição Intraquadro

Este tipo de predição é uma novidade do padrão H.264/AVC, aplicada a macroblocos do tipo I. Para as amostras de luminância, existem três tipos de predição intra: Intra 16x16 (o macrobloco é predito por inteiro), Intra 8x8 (o macrobloco é predito em partições 8x8) ou Intra 4x4 (o macrobloco é predito em partições 4x4). Nos três tipos de predição, o macrobloco é predito com base nos vizinhos já processados (amostras A-M na Figura 2.7 e H e V na Figura 2.8). Não é possível que coexistam em um mesmo macrobloco, blocos preditos com predição Intra 4x4 e Intra 8x8, pois todo o macrobloco deve utilizar o mesmo tipo de predição, a variação é permitida somente nos modos de predição. A Figura 2.7 mostra os 9 modos definidos pelo padrão para a predição Intra 4x4. A predição Intra 8x8 suporta os mesmo 9 modos da Intra 4x4, porém é preciso armazenar 16 amostras da vizinhança acima e 8 amostras da vizinhança à esquerda do bloco 8x8 que está sendo predito.

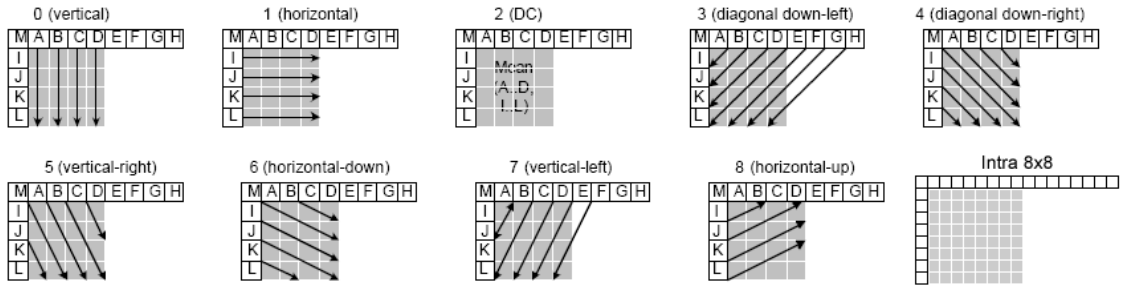

Figura 2.7: Modos de predição para Intra 4x4 (adaptado de DINIZ, 2009) e Intra 8x8

O padrão H.264/AVC também prevê 4 modos para a predição Intra 16x16, ilustrados na Figura 2.8. Os blocos de crominância também recebem predição Intra. Os modos de predição Intraquadros para crominância são semelhantes aos modos de predição Intra 16x16, porém aplicados a blocos de dimensão 8x8. Mais detalhes sobre a predição Intraquadros podem ser encontrados em Diniz (2009) e Lee (2008).

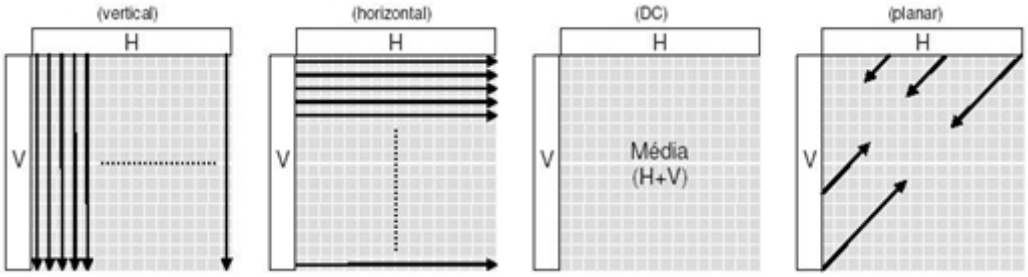

Figura 2.8: Modos de predição para Intra 16x16 e Croma. (DINIZ, 2009)

2.4.2 Transformadas

A aplicação de transformadas em blocos de resíduos consiste em arranjar a informação que está distribuída de forma equivalente dentro dos blocos para os elementos mais acima e à esquerda da matriz de amostras enquanto que os elementos mais afastados desse ponto ficam sem ou quase sem informação. A Figura 2.9 ilustra um exemplo hipotético da transformada nos dados de um bloco 4x4. Esse processo tem por objetivo melhorar a eficiência da codificação de entropia e da quantização.

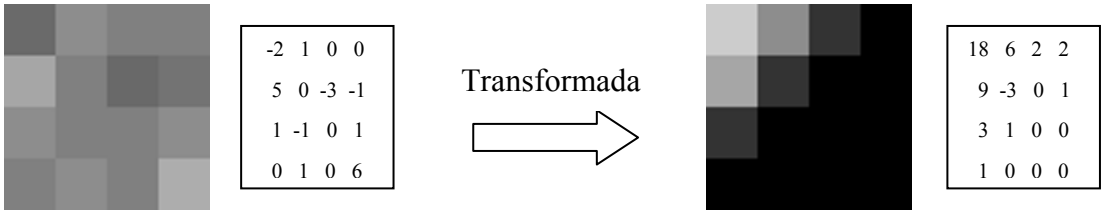

Figura 2.9: Processo de transformada em um bloco 4x4 (adaptado de RAMOS, 2010)

Ao quadro de resíduos é aplicada uma aproximação inteira da transformada discreta do cosseno (DCT). Os resíduos são divididos em blocos 4x4 e todos estes passam pela DCT 4x4. As saídas desse processo são os coeficientes que representam o sinal no domínio da frequência. Aos blocos de crominância, é aplicada a transformada Hadamard 2x2 nos coeficientes DC, enquanto que os coeficientes AC recebem a DCT 4x4. Nos macroblocos em que a predição é Intra 16x16, aos coeficientes DC do bloco de luminância é aplicada a transformada Hadamard 4x4, como ilustrado na Figura 2.10. No perfil *High*, há ainda a possibilidade de utilização da transformada DCT 8x8 nos blocos de luminância, como ilustrado na Figura 2.11. No codificador há ainda a presença do bloco de transformadas inversas, que realiza a recuperação do quadro de resíduo para que este possa ser reconstruído e utilizado como referência para a predição. Detalhes sobre o exato equacionamento das transformadas podem ser encontrados em Agostini (2007).

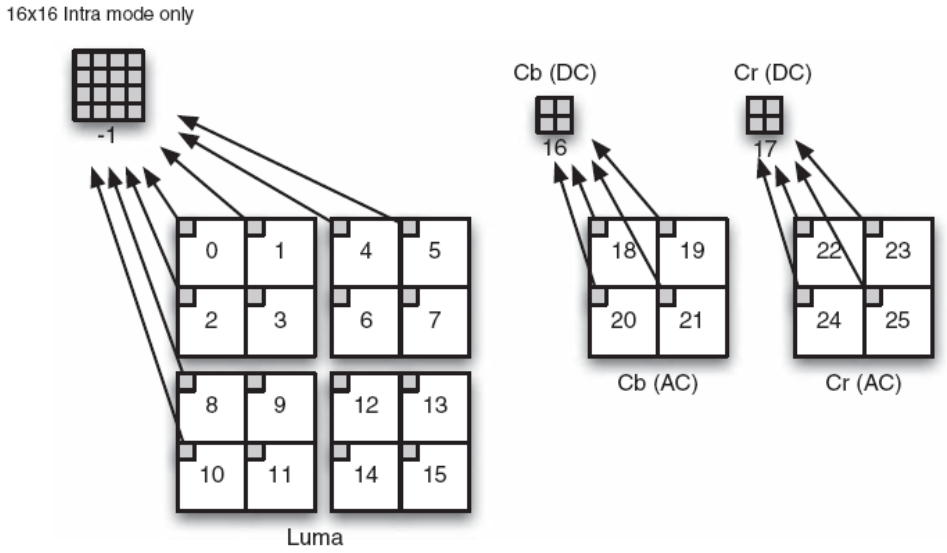

Figura 2.10: Ordem de varredura dos blocos utilizando transformada 4x4 e amostragem 4:2:0. (RICHARDSON, 2010)

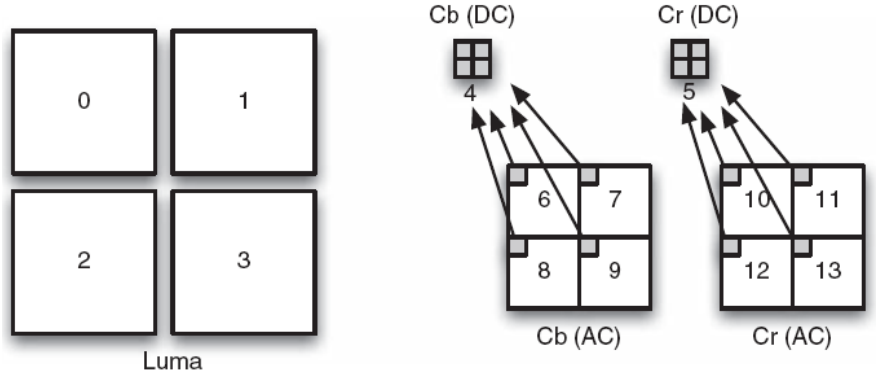

Figura 2.11: Ordem de varredura dos blocos utilizando transformada 8x8 e amostragem 4:2:0. (RICHARDSON, 2010)

2.4.3 Quantização

Na etapa de quantização, o codificador descarta informações de resíduos provenientes das transformadas DCT. A quantização realiza a inserção de perdas nas amostras de resíduos a fim de eliminar ou reduzir as informações de alta frequência, pois elas são menos sensíveis ao olho humano e com isso, é possível obter grande redução na taxa de bits ao custo de uma pequena perda de qualidade subjetiva do vídeo (RICHARDSON, 2003). A Figura 2.12 apresenta o bloco da Figura 2.9 quantizado. Fica evidente que muitos dos valores que estavam próximos a zero tendem a ser zerados e os demais tem seu valor reduzido.

Figura 2.12: Processo de quantização em um bloco 4x4 (adaptado de RAMOS, 2010)

Os cálculos da quantização variam de acordo com o modo de predição e se a informação é de luminância ou crominância. O parâmetro de quantização (QP) determina a intensidade do passo de quantização que será aplicado ao bloco de resíduos transformados. Quanto maior o passo de quantização, mais perdas serão inseridas. O QP varia de 0 à 51. Depois de quantizados, os coeficientes são enviados para a codificação de entropia na ordem definida nas Figuras 2.10 e 2.11, conhecida como duplo-Z. No codificador também há o bloco de quantização inversa, que corrige a escala do cálculo das transformadas para que este possa ser reconstruído e utilizado como referência para a predição. Mais detalhes sobre o processo de quantização podem ser encontrados em Agostini (2007).

2.4.4 Filtro de deblocagem

A fim de reduzir o efeito de bloco criado pelo processo de codificação e melhorar a qualidade subjetiva da imagem reconstruída, um filtro é aplicado a cada macrobloco decodificado. O efeito de bloco é bem visível quando o QP utilizado para a codificação é elevado, como no exemplo da Figura 2.13, onde há duas imagens, uma sem e outra com o filtro, e esse efeito pode ser percebido facilmente. O filtro de deblocagem é aplicado após a transformada inversa no codificador (antes de reconstruir e armazenar o macrobloco para predições) e no decodificador (antes da reconstrução e exibição das imagens), afetando até 3 amostras de cada lado na fronteira entre os blocos dependendo da força de filtragem aplicada ao macrobloco. O filtro tem por objetivo suavizar as bordas, melhorando a aparência das imagens decodificadas.

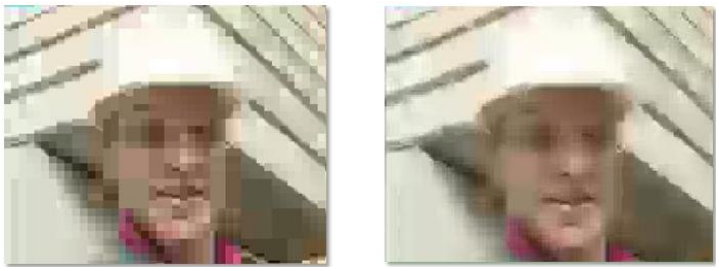

Figura 2.13: Imagens com e sem filtro de borda respectivamente (ROSA, 2010).

2.4.5 Codificação de Entropia

No padrão H.264/AVC a codificação de entropia é o processo responsável pelo tratamento das redundâncias estatísticas encontradas sob o conjunto de informações produzidas por outras ferramentas aplicadas pelo padrão H.264/AVC. A norma de especificação do padrão H.264/AVC define codificação de entropia usando códigos binários fixos e usando VLC (código de tamanho variável – *variable-length codes*). Os códigos VLC previstos são: Exp-Golomb, CAVLC (*Context-Adaptive Variable Length Coding*) e CABAC (*Context-Adaptive Binary Arithmetic Coding*) (RICHARDSON, 2003). O CABAC, por ser o foco deste trabalho, será detalhado no próximo capítulo.

2.4.5.1 Códigos Exp-Golomb

Códigos Exp-Golomb são uma forma de VLC com construção regular para símbolos inteiros sem sinal e têm sido muito utilizados em diferentes aplicações de áudio, vídeo e imagem (CHEN, 2005). Para transformar o valor do símbolo, identificado como *cod*, em um inteiro sem sinal (caso este seja um valor inteiro que possa ser negativo), a equação dada em (2.1) é utilizada. Os códigos Exp-Golomb (2.2) são construídos pela concatenação de um prefixo (P) correspondente ao número de zeros que antecede o

valor ‘1’ no código, dado em (2.3), um símbolo binário ‘1’ e um sufixo correspondente à representação binária do símbolo codificado, dado em (2.4). Os códigos Exp-Golomb são aplicados na maioria da informação não-residual gerada pelo codificador como, por exemplo, os vetores de movimento. A Tabela 2.3 apresenta um conjunto de valores sobre os quais são aplicados os códigos Exp-Golomb produzindo diferentes formatos de *bitstring*.

$$cod = (-1)^{x+1} * Ceil(x/2) \quad (2.1)$$

$$\underbrace{0 \dots 0}_{P}\ 1\ \underbrace{x_{M-1} \dots x_0}_{S} \quad (2.2)$$

$$P = \log_2(x+1) \quad (2.3)$$

$$S = x + 1 - 2^P \quad (2.4)$$

Tabela 2.3: Exemplo de aplicação dos códigos Exp-Golomb

cod	x	Intervalo de x	Formato Exp-Golomb	Bitstring
0	0	0	0	0
1	1	1-2	0 1 X^0	0 1 0
-1	2	...	…	0 1 1
2	3	3-6	00 1 X^1 X^0	00 1 00
-2	4	...	...	00 1 01
3	5	...	...	00 1 10
-3	6	...	...	00 1 11
4	7	7-14	000 1 X^2 X^1 X^0	000 1 000
…	…	...	...	…
-8	16	15-30	0000 1 X^3 X^2 X^1 X^0	0000 1 0001
…	…	...	...	…
-16	32	31-62	00000 1 X^4 X^3 X^2 X^1 X^0	00000 1 00001

2.4.5.2 CAVLC

Os blocos de resíduos, que por sua vez passaram pelo processo de predição, transformada e quantização, devem ser reordenados em forma de ziguezague antes de serem utilizados pela codificação de entropia, como está apresentado na Figura 2.14. Se a predição intra 16x16 foi utilizada, então existe um bloco 4x4 extra, composto de coeficientes Luma DC que devem lidos em ziguezague (Figura 2.10) e existem 15 amostras para cada bloco de Luma AC. A amostra destacada na Figura 2.14(a) corresponde à amostra que seria ignorada na leitura ziguezague de blocos Luma AC com predição intra16x16; a amostra destacada estaria compondo o bloco de Luma DC. Se a predição **não** é intra 16x16, então não há diferenciação entre amostras DC e AC e 16 amostras por bloco 4x4 são lidas na ordem ziguezague. Para os coeficientes Croma DC, a ordem ziguezague é representada na Figura 2.14 (b). Os coeficientes Croma AC são lidos da mesma maneira que os blocos Luma AC intra 16x16, ignorando a amostra na extremidade acima e à esquerda. Por último, se foi utilizada a transformada 8x8 no macrobloco, então a ordem ziguezague de leitura das amostras Luma corresponde à Figura 2.14 (c).

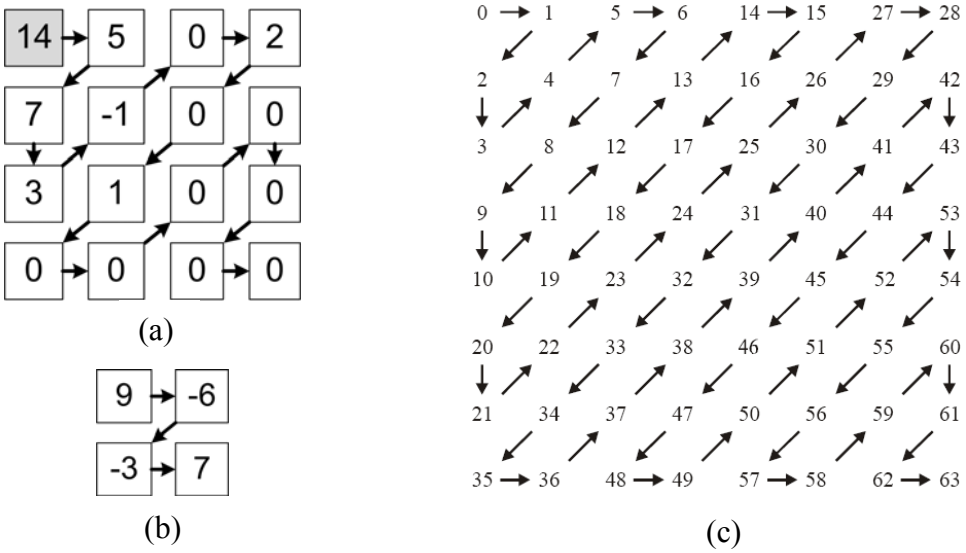

Figura 2.14: Ordenamento ziguezague para blocos Luma e Croma AC (a), Croma DC (b) e blocos Luma transformados com DCT 8x8 (c) (INTERNATIONAL, 2009).

O CABAC e o CAVLC são os codificadores de entropia responsáveis por processar os resíduos e ambos lêem a informação residual em ziguezague como apresentado na Figura 2.14. Os blocos de resíduos são geralmente matrizes que apresentam uma grande quantidade de coeficientes zero localizados no canto inferior direito dos blocos. A ordem ziguezague força que a maioria dos zeros presente nos blocos sejam lidos após a leitura de todos os coeficientes não-zero. Esses zeros posteriores às amostras não-nulas são simplesmente descartados no CABAC e no CAVLC. Assim, para o bloco 4x4 mostrado na Figura 2.14(a) as informações relevantes podem ser armazenadas e tratadas como o vetor da Tabela 2.4.

Tabela 2.4: Resultado do re-ordenamento ziguezague para o exemplo da Figura 2.13(a).

Posição Escaneada	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Coeficiente residual	14	5	7	3	-1	0	2	0	1							

O CAVLC explora ainda mais três características inerentes dos blocos de resíduos:

- Os últimos coeficientes não-nulos lidos em ziguezague no bloco de resíduos são geralmente ±1;
- A quantidade de coeficientes não-zero presente em um bloco de resíduos é similar à quantidade de coeficientes não-zero presente nos blocos vizinhos;
- A magnitude dos primeiros coeficientes não-nulos lidos em ziguezague tende a ser maior comparado aos últimos coeficientes não-nulo obedecendo assim, a uma ordem, na média, decrescente.

Para explorar as características dos blocos de resíduos e atingir uma taxa de compressão satisfatória, o algoritmo do CAVLC inicia com a geração de uma série de estatísticas sobre os coeficientes que não foram descartados pela leitura ziguezague (exemplo da Tabela 2.4). Essas estatísticas podem ser resumidas em: total de coeficientes zero, total de coeficientes não-zero, total de coeficientes não-zero e não-um e total de uns (±1) seguidos desconsiderando amostras nulas lidas na ordem reversa de varredura. Além dessas estatísticas, os coeficientes não-zero e não-um, conhecidos como *Levels* e o posicionamento dos zeros não descartados são geradas.

A partir dos dados estatísticos, são processadas 5 informações que, concatenadas, compõem o *bitstream* do CAVLC: *Coeff Token, Trailing Ones Sign Flag, Level, Total Zero, Run Before.* Essas informações são geradas a partir da consulta de tabelas e de operações aritméticas simples aplicadas a alguns dados estatísticos. Além dos dados estatísticos do bloco, o processamento de informações pelo CAVLC depende de dados de blocos vizinhos e do tipo de bloco que está sendo codificado (por exemplo, Croma AC). Mais detalhes sobre o funcionamento do CAVLC podem ser encontrados em Ramos (2010).

O principal diferencial introduzido na codificação de entropia pelo padrão H.264/AVC é a técnica de codificação adaptativa baseada em contextos, a qual é aplicada tanto no CAVLC quanto no CABAC. Com esta técnica, a maneira pela qual as informações são codificadas depende das informações codificadas em passos anteriores e da fase em que o algoritmo de codificação se encontra (RICHARDSON, 2003).

2.5 Hierarquia e *parsing* do H.264/AVC

Uma interessante inovação do padrão H.264/AVC é que ele faz uma clara distinção entre a organização do vídeo codificado e da sua transmissão via rede. A organização hierárquica da sintaxe do H.264/AVC está ilustrada na Figura 2.15. A camada abstrata de rede (NAL – *network abstraction layer*) consiste em uma série de unidades NAL. As NALs do tipo SPS (*Sequence Parameter Sets*) e PPS (*Picture Parameter Sets*) são unidades NAL que controlam determinados parâmetros comuns de codificação que devem ser informados ao decodificador. Os dados de saída do processo de codificação estão na camada VCL, sendo formados por uma seqüência de bits que representam os dados do vídeo codificado e classificados por *slices*. A camada de *slices* é separada por *Slice Header* e *Slice Data*. Interna à camada *Slice Data* há uma série de macroblocos codificados e, finalmente, cada macrobloco contém as seguintes informações:

- Tipo de macrobloco (I, P ou B)
- Informação de predição: modo de predição intra para macroblocos I ou quadros de referência e vetores de movimento para quadros P e B.
- Padrão do bloco codificado (CBP – *Coded Block Pattern*) indicando os blocos de resíduos de Luma e Croma com coeficientes não-zero
- Parâmetro de Quantização (QP) para macroblocos com CBP diferente de zero
- Amostras de resíduos para blocos contendo valores não-zero

O CODEC de vídeo conforme o padrão H.264/AVC pode ser dividido em dois processos: o processo de *parsing* e a o processo de (re)construção da imagem. Cada uma das ferramentas de codificação descritas pelo padrão H.264/AVC visam tratar um tipo específico de redundância e ao realizar suas tarefas geram um conjunto de informações de controle, as quais são denominadas de elementos sintáticos - SE (*Syntax Element*). O processo de *parsing* consiste em identificar e (de)codificar cada SE (através de alguma ferramenta de decodificação de entropia) e o processo de reconstrução tem como entrada estes elementos sintáticos para reconstrução da imagem, através de um processo de predição adicionado aos resíduos desta predição.

O processo de *parsing* é mais complexo no H.264/AVC em relação a outros padrões, devido às técnicas inovadoras de codificação de entropia (Exp-Golomb, CAVLC e CABAC) e à grande quantidade de elementos sintáticos produzidos como resultado da maior flexibilidade das ferramentas do padrão. A Figura 2.15 ilustra a

hierarquia do processo de *parsing* do *bitstream* do H.264/AVC, quanto à forma de codificação.

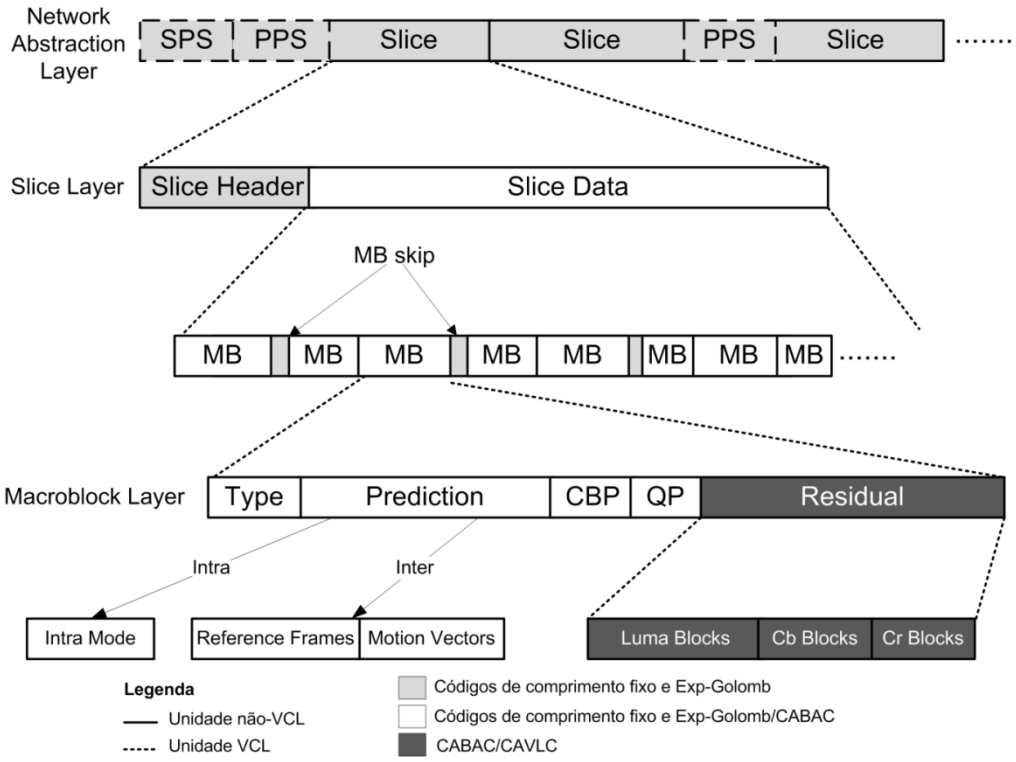

Figura 2.15: Organização hierárquica do bitstream no H.264/AVC. (baseado em RICHARDSON, 2010)

A Figura 2.15 mostra os dados contidos dentro de uma unidade NAL. Esta unidade pode ser VCL ou não-VCL. As unidades VCL contêm necessariamente um *slice* codificado para cada unidade NAL. Exemplos de informações VCL no nível hierárquico de *slice* são: tipo de *slice* e posição do primeiro macrobloco do *slice.* Os dados marcados em cinza (parâmetros não-VCL e cabeçalho de *slice*) são representados por códigos de comprimento fixo e por códigos Exp-Golomb. Informações como a definição do perfil, nível e resolução do vídeo são informações classificadas como não-VCL.

Ainda na Figura 2.15, os SEs de dados de *slice*, os SEs que representam as decisões tomadas pelo preditor, pelas transformadas e pela quantização são representados por códigos de comprimento fixo e por códigos Exp-Golomb ou CABAC. A escolha entre Exp-Golomb/CABAC é realizada através de um SE não-VCL proveniente da camada superior chamado *entropy_coding_mode_flag* (0=Exp-Golomb, 1=CABAC). Nesse nível da hierarquia (*Macroblock Layer*) encontram-se informações sobre o tipo de macrobloco, vetores de movimento, quadros de referência, modos de predição intra, CBP e QP.

Finalmente, a informação destacada em tom mais escuro na Figura 2.15 compreende os dados residuais de luminância e crominância para cada macrobloco. A informação residual é codificada utilizando CAVLC ou CABAC. A escolha entre cada modo de codificação de entropia também se dá pelo SE *entropy_coding_mode_flag* (0=CAVLC, 1=CABAC).

3 CODIFICAÇÃO ARITMÉTICA BINÁRIA ADAPTATIVA AO CONTEXTO

Este capítulo tem por objetivo apresentar a técnica e as ferramentas que compõem o codificador aritmético binário adaptativo ao contexto (CABAC) presente no padrão H.264/AVC. A Seção 3.2 ilustra com detalhes o processo de ordenamento e gerenciamento de informações, chamado de *parsing*. A Seção 3.3 apresenta uma descrição detalhada do processo de binarização para cada elemento sintático. A modelagem de contextos para todas as informações processadas com o CABAC é contemplada na Seção 3.4. Finalmente, a Seção 3.5 discute o codificador aritmético utilizado no CABAC.

3.1 Visão Geral do CABAC

O Codificador Aritmético Binário Adaptativo ao Contexto (CABAC - *Context-Adaptive Binary Arithmetic Coding*) é o principal codificador de entropia disponível nos perfis Main e High do padrão H.264/AVC. A principal função do CABAC é codificar elementos sintáticos (SE – *Sintax Elements*) para o *bitstream* (MARPE, 2003). O CABAC atinge alta eficiência de compressão através da seleção de modelos distintos de probabilidade para cada SE de acordo com o contexto do SE, da adaptação das estimativas de probabilidade baseado em estatísticas locais e da utilização de um codificador aritmético ao invés de um codificador de tamanho variável (RICHARDSON, 2010). O CABAC é o estado-da-arte em termos de eficiência de codificação, produzindo, em média, uma taxa de bits 7% menor comparado ao CAVLC, considerando apenas a codificação de resíduos. Entretanto, o CABAC pode ocupar 9,6% do tempo total de codificação de um vídeo e seu *throughput* é limitado devido às dependências de dados em nível de bit (LIN, 2010) caracterizando alta complexidade computacional e dependência de dados devido a inserção desses algoritmos.

A codificação de um SE consiste nos seguintes passos:

1. Binarização: CABAC utiliza um codificador aritmético binário, ou seja, apenas símbolos binários (0 ou 1) são codificados. Um valor não-binário, como um vetor de movimento, por exemplo, é 'binarizado' ou convertido em um código binário para que possa ser codificado. Cada bit gerado pela binarização é chamado de *bin*. Os próximos passos são repetidos para cada símbolo binarizado:
2. Seleção de um modelo de contexto: Um 'modelo de contexto' é um modelo de probabilidade, ou escala de ocorrência, que representa as estatísticas de distribuição e/ou ocorrência para um ou mais *bins* de um símbolo binarizado e é escolhido de uma seleção de modelos disponíveis dependendo das estatísticas

dos símbolos recentemente codificados. O modelo de contexto contém a probabilidade de cada *bin* 0 ou 1;

3. Codificação aritmética: um codificador aritmético codifica cada *Bin* de acordo com o modelo de probabilidade selecionado no passo anterior;
4. Atualização das Probabilidades: O modelo de contexto selecionado é atualizado baseado no último valor codificado. Exemplo: Se o valor de um *bin* é '1', então o contador de frequência de '1' é incrementado.

3.2 Parsing do CABAC

3.2.1 Elementos Sintáticos

O CABAC é responsável por codificar os dados de *slice* na camada de *slice* e todas as informações descendentes na hierarquia do H.264/AVC, como ilustrado na Seção 2.5. Essas informações são provenientes de todos os módulos que compõem o processo de codificação (visto que o Filtro, IT e IQ estão presentes no codificador apenas para realizar a reconstrução do quadro predito) como ilustrado na Figura 3.1 e, em conjunto com as amostras de resíduos, constituem as informações necessárias para reconstrução da imagem pelo decodificador. Essas informações devem ser tratadas pela codificação de entropia de forma genérica e são chamadas nesse escopo de elementos sintáticos (SE - *Syntactic Element*).

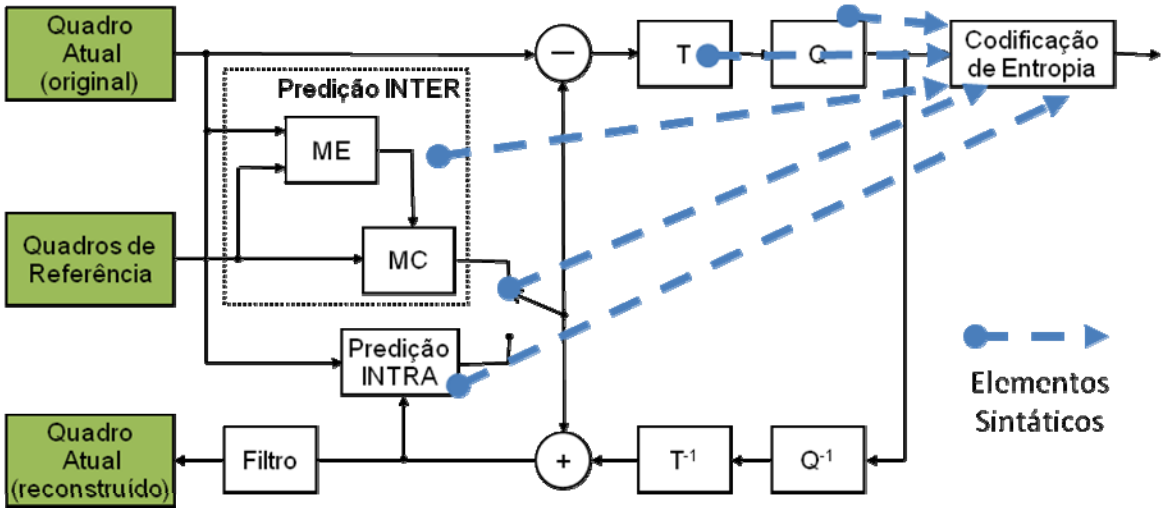

Figura 3.1: Origem dos elementos sintáticos.

Para a compreensão plena dos algoritmos que compõem o escopo do CABAC, é necessário detalhar cada SE codificado pelo CABAC, pois cada um deles tem tratamento diferenciado pelos algoritmos do CABAC. Os SEs podem ser de vários tipos e cada tipo de SE tem um intervalo de valores possíveis. O tipo do SE é utilizado principalmente pelo controle do CABAC para determinar o método de binarização e para calcular o modelo probabilístico que será utilizado pelo codificador aritmético. O valor é a informação que será processada pelo CABAC e seu significado depende do tipo de SE. Todos os tipos de SEs codificados através do CABAC serão descritos nesta Seção.

Os SEs devem ser codificados em uma determinada ordem para que o decodificador possa identificar corretamente qual o SE está sendo lido no momento para reconstruir o vídeo. Além disso, há SEs que sempre são codificados e outros SEs que são codificados

apenas em determinadas situações, sendo que o cenário que determina a ocorrência de um SE geralmente é delimitado por configurações globais externas ao CABAC e por um valor de SE codificado anteriormente. Para realizar a codificação do vídeo de forma correta é necessário, portanto, ordenar corretamente a transmissão dos SEs, escolher quais os SEs devem ser codificados e quantos SEs de cada tipo devem ser enviados para os próximos passos de codificação. O processo responsável por realizar o ordenamento e a escolha dos SEs que devem ser processados pelo CABAC é chamado de *Parsing*.

A ocorrência dos SEs é dependente da forma como o vídeo foi codificado, ou seja, de como o codificador explorou as redundâncias para gerar a codificação. Apesar disto, dividindo-se o *bitstream* em camadas, é possível estimar o melhor e o pior caso da ocorrência dos SEs de cada camada. Os dados não-residuais contêm informações relacionadas ao tipo de macrobloco, submacroblocos e ferramentas de compressão utilizadas durante o processo de codificação realizado anteriormente a entropia. Esses parâmetros podem ser classificados no nível de macrobloco, predição inter e predição intra. Assim, os SEs não-residuais tratados pelo CABAC são descritos quanto a sua função e processo de *parsing*:

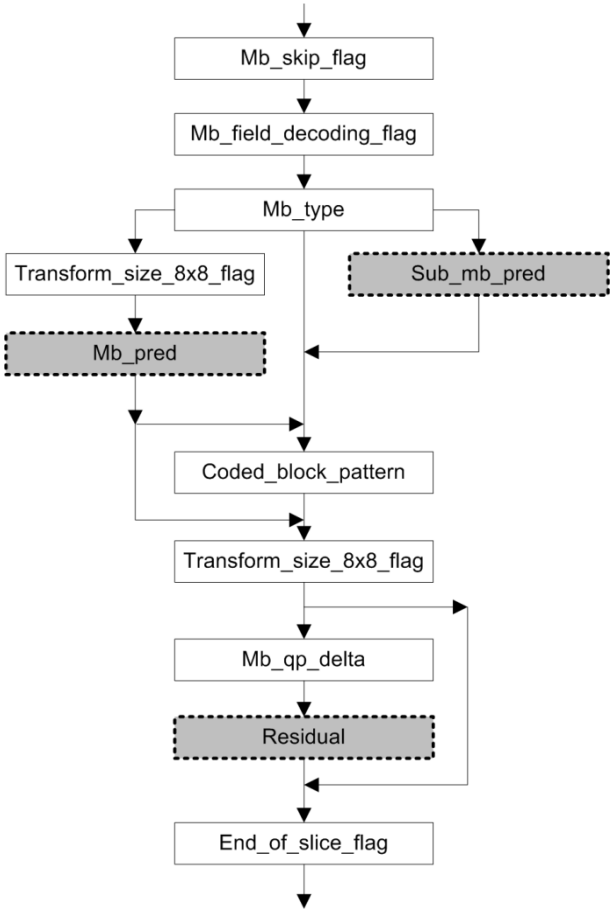

Figura 3.2: *Parsing* dos SEs transmitidos pelo CABAC

- **mb_skip_flag –** especifica se o macrobloco corrente é do tipo *skip* ou não. Caso afirmativo, nenhuma informação adicional sobre o macrobloco atual precisa ser processada pelo CABAC e o processamento do macrobloco termina. *mb_skip_flag* só ocorre em macroblocos P e B;
- **mb_field_decoding_flag –** especifica se o macrobloco corrente é um par de macrobloco do tipo quadro ou campo (*frame* ou *field*);

- **mb_type –** especifica o tipo de predição intra ou interquadros que foi utilizado pelo codificador. Há três tabelas, uma para cada tipo de macrobloco (I, P e B) e em cada tabela, diferentes informações sobre a predição e codificação do macrobloco são indicadas por esse SE.
  - Para macroblocos do tipo I, quando *mb_type*=0 indica que o macrobloco é intra 4x4 ou 8x8 enquanto que os demais valores indicam combinações entre os quatro diferentes modos Intra 16x16 e informações de *coded_block_pattern* que não são transmitidas para o *bitstream*, porém são utilizadas na etapa de modelagem de contextos;
  - Para macroblocos P, *mb_type* contém a informação de particionamento do macrobloco (16x16, 8x16, 16x8 ou 8x8). Se a partição é 8x8, o SE *sub_mb_type* também é requisitado.
  - Nos macroblocos B, *mb_type* contém a informação de particionamento do macrobloco (16x16, 8x16, 16x8 ou 8x8) e a informação de predição de cada partição (predição utilizando lista 0, lista 1 ou bi-predição). O SE *mb_type* pode ainda indicar se o macrobloco é do tipo *direct*. Analogamente aos macroblocos P, se a partição é 8x8, o SE *sub_mb_type* também é requisitado. O SE mb_type (em conjunto com o *sub_mb_type*, se for o caso) indica quantos SE dos tipos Vetores de Movimento e Quadros de Referência devem ser transmitidos.
- **mb_pred –** sub-processo que especifica o tipo de predição intra ou inter quadros que é utilizado pelo macrobloco, exceto para macroblocos 8x8 P e B. Se o macrobloco é do tipo *direct*, então o processo *mb_pred* não é realizado. O processo de *parsing* de *mb_pred* será detalhado na próxima Seção.
- **sub_mb_pred –** sub-processo que especifica a predição inter quadros que é utilizado pelo macrobloco P ou B particionado em blocos 8x8. O processo de *parsing* de *sub_mb_pred* será detalhado na próxima Seção.
- **transform_size_8x8_flag** – indica se a transformada DCT 8x8 foi utilizada nas amostras de luminância neste macrobloco ao invés da transformada DCT 4x4. Este SE só ocorre no *bitstream* quando há suporte ao perfil *high* e quando não é utilizada a predição Intra 16x16. Se o macrobloco é Intra 4x4 ou Intra 8x8, esse SE ocorre antes da informação de predição do macrobloco. Se o macrobloco é inter, a *flag* é posicionada após *coded_block_pattern*;
- **coded_block_pattern** – indica quais blocos 8x8 do macrobloco corrente têm blocos 4x4 significativos. Não ocorre quando a predição é Intra 16x16. Será mais detalhado na Seção 3.2.3;
- **mb_qp_delta** – especifica a variação do parâmetro de quantização em relação ao macrobloco anteriormente codificado. Essa variação pode ser positiva, negativa ou nula;
- **residual –** Se *coded_block_pattern* não é nulo, então há informação residual para ser enviada ao *bitstream.* O processo de codificação de resíduos será detalhado na Seção 3.2.3.
- **end_of_slice** – especifica se é o último macrobloco do *slice* atual.

3.2.2 Codificação dos Dados de Predição

Os elementos sintáticos (SE) de predição de macrobloco indicam como a predição inter ou intraquadros foi realizada para o macrobloco atual. O processo de *parsing* para esses SEs é ilustrado na Figura 3.3. Se o *mb_type* indica que o macrobloco é P ou B e

utiliza partições 8x8, então o processo de *parsing* para esse macrobloco é especificado na Figura 3.4.

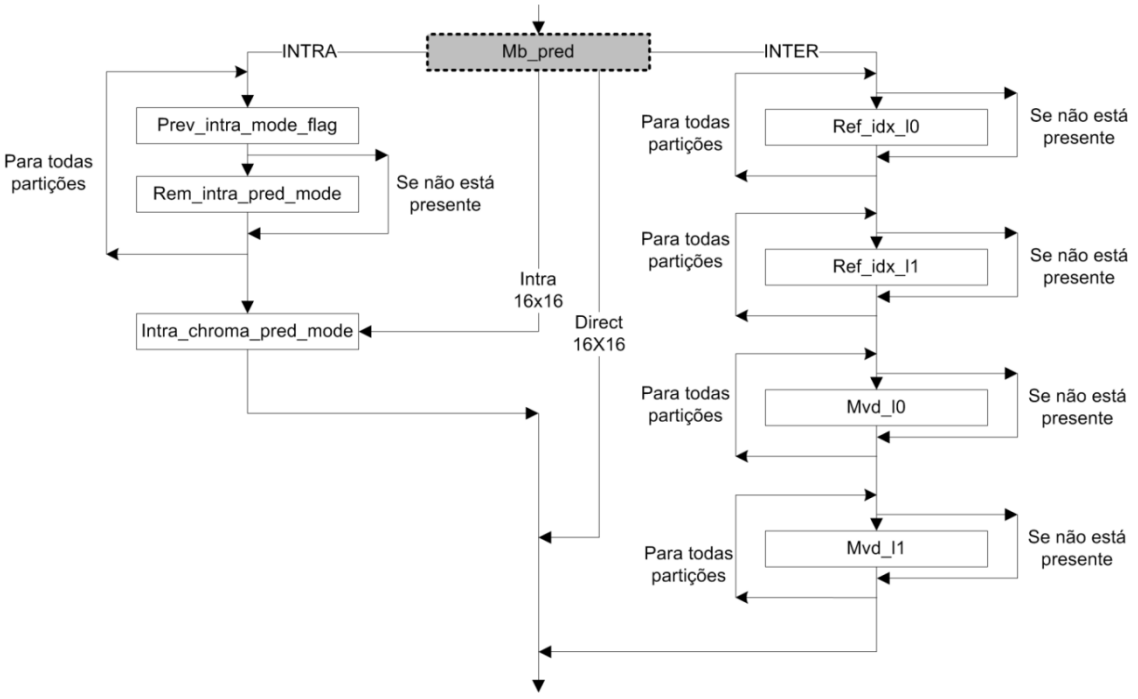

Figura 3.3: *Parsing* dos SEs que descrevem a predição do macrobloco

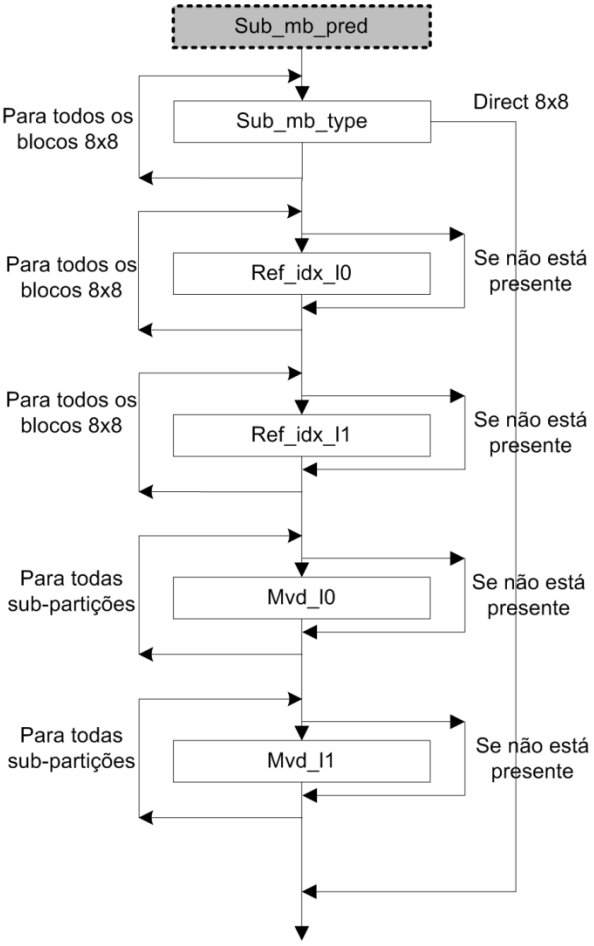

Figura 3.4: *Parsing* dos SEs que descrevem a subpredição do macrobloco

Quando a predição é intra 16x16, apenas uma informação descrevendo um dos 4 modos de predição Croma é enviada ao *bitstream*, visto que o *mb_type* define o modo de predição Intra 16x16. Para os outros casos, os SEs que descrevem a predição Intra são os seguintes:

- **prev_intra4x4_pred_mode** e **prev_intra8x8_pred_mode –** *flag* que especifica se o bloco NxN atual(onde N=4 ou N=8) utilizou o mesmo modo de predição do bloco NxN previamente codificado. Se o modo de predição do bloco atual é igual ao bloco anterior, então o SE *rem_intraNxN_pred_mode* não precisa ser transmitido. O SE *prev_intra8x8_pred_mode* apenas ocorre no perfil *high;*
- **rem_intra4x4_pred_mode** e **rem_intra8x8_pred_mode –** especifica qual o modo de predição Intra NxN do bloco de luminância NxN dentre as nove suportadas pelo padrão. O SE *rem_intra8x8_pred_mode* apenas ocorre no perfil *high;*
- **intra_chroma_pred_mode –** especifica qual dentre os quatro tipos de predição Intra para blocos de crominância foi usada no macrobloco atual.

Quando a predição do macrobloco é interquadros, os SEs que descrevem a predição interquadros são os seguintes:

- **sub_mb_type –** especifica os tipos de submacroblocos existentes para macroblocos P e B. Se o macrobloco é interquadros e foi dividido em partições 8x8, então este SE está presente e deve ser lido 4 vezes, um para cada partição 8x8. De forma análoga ao *mb_type*, é necessário verificar o particionamento (8x8, 4x8, 8x4 ou 4x4) e a predição (lista 0, lista 1 ou bi-predição) de cada submacrobloco através de tabelas. O submacrobloco pode ser do tipo *direct* também;
- **ref_idx_l0 -** especifica o índice da imagem de referência usada para predição na lista 0 de imagens de referência. Se há apenas um quadro de referência na lista 0, *ref_idx_l0* não é transmitido;
- **ref_idx_l1 -** possui o mesmo significado de *ref_idx_l0*, mas relativo à lista 1;
- **mvd_l0 -** especifica a diferença entre um vetor de movimento a ser usado e sua predição. A componente horizontal é a primeira a ser decodificada, seguida pela componente vertical;
- **mvd_l1 -** possui o mesmo significado de *mvd_l0*, mas relativo à lista 1.

Se o macrobloco é do tipo *direct*, nenhum SE de predição é enviado e a reconstrução do macrobloco predito é realizada pelo decodificador através de vetores de movimento e quadros de referência de macroblocos previamente codificados. Se o submacrobloco é *direct*, então nenhuma informação de predição será transmitida apenas para o submacrobloco dado com *direct*.

Para os macroblocos B, cada partição (16x16, 16x8, 8x16 e 8x8) pode ter um quadro de referência (lista 0 ou lista 1) ou dois quadros de referência (Bipredição), um para cada lista. Além dos quadros de referência, se a partição não é bipreditiva, um par de vetores (horizontal e vertical) de movimento para partição deve ser transmitido. Por outro lado, se a partição é bipreditiva, então dois pares de vetores de movimento, um para cada lista, são enviados. É importante observar que o mesmo quadro de referência é utilizado para todo o submacrobloco ao passo que os vetores de movimento acontecem em toda subpartição do submacrobloco, ou seja, para cada submacrobloco não bipreditivo, apenas um quadro de referência e até quatro pares de vetores de movimento podem ser enviados ao passo que para submacroblocos bipreditivos, essa

quantidade é dobrada. Na Tabela 3.1 estão ilustradas as infomações contidas neste parágrafo.

Tabela 3.1: Relação entre tipo de predição Inter e ocorrência dos SEs *ref_idx* e *mvd*

Partição	**Tipo de predição em cada partição**		**Qtde ref_idx**	**Qtde mvd** (par)
	Tipo 0	**Tipo 1**		
16x16	Lista 0 ou Lista 1	-	1	1
16x16	Bipreditivo	-	2	2
16x8 e 8x16	Lista 0 ou Lista 1	Lista 0 ou Lista 1	2	2
16x8 e 8x16	Lista 0 ou Lista 1	Bipreditivo	3	3
16x8 e 8x16	Bipreditivo	Lista 0 ou Lista 1	3	3
16x8 e 8x16	Bipreditivo	Bipreditivo	4	4
SubPartição	**Tipo de predição em cada subpartição**		**Qtde ref_idx**	**Qtde mvd** (par)
8x8	Lista 0 ou Lista 1		1	1
8x8	Bipreditivo		2	2
8x4 ou 4x8	Lista 0 ou Lista 1		1	2
8x4 ou 4x8	Bipreditivo		2	4
4x4	Lista 0 ou Lista 1		1	4
4x4	Bipreditivo		2	8

3.2.3 Codificação dos Dados Residuais

A codificação de resíduos inicia pela leitura dos resíduos através da interpretação do SE *coded_block_pattern* ou CBP que contém 6 bits, conforme mostrado na Figura 3.5. Inicialmente, o macrobloco de luminância (16x16) é particionado em quatro blocos 8x8, o mesmo tamanho dos blocos de crominância azul e vermelho, que não são particionados nesse processo. Os 4 bits menos significativos de CBP representam as amostras de Luma organizadas em blocos 8x8 que devem ser lidas. No exemplo da Figura 3.5(a) de acordo com CBP Luma, apenas o bloco P3 contém informações significantes, então será realizado a geração dos elementos sintáticos de resíduos, para esse bloco. Os blocos P0, P1 e P2 serão simplesmente ignorados pelo CABAC e não serão codificados. No exemplo da Figura 3.5(b), apenas o bloco P1 será descartado e na Figura 3.5(c), nenhum bloco será descartado.

Os 2 bits mais significativos do CBP codificam a informação de Croma. Quando CBP *Chroma* é 0, então não deve existir amostras válidas nos blocos Croma DC e AC e portanto, as amostras desses blocos não serão lidas pelo CABAC. Se CBP *Chroma* é 1, então somente as amostras Croma DC devem ser lidas pelo CABAC. Finalmente, se o valor de CBP *Chroma* é 2, as amostras Croma DC e AC devem ser lidas pelo CABAC. CBP *Chroma* nunca pode ser 3, pois é um valor inválido.

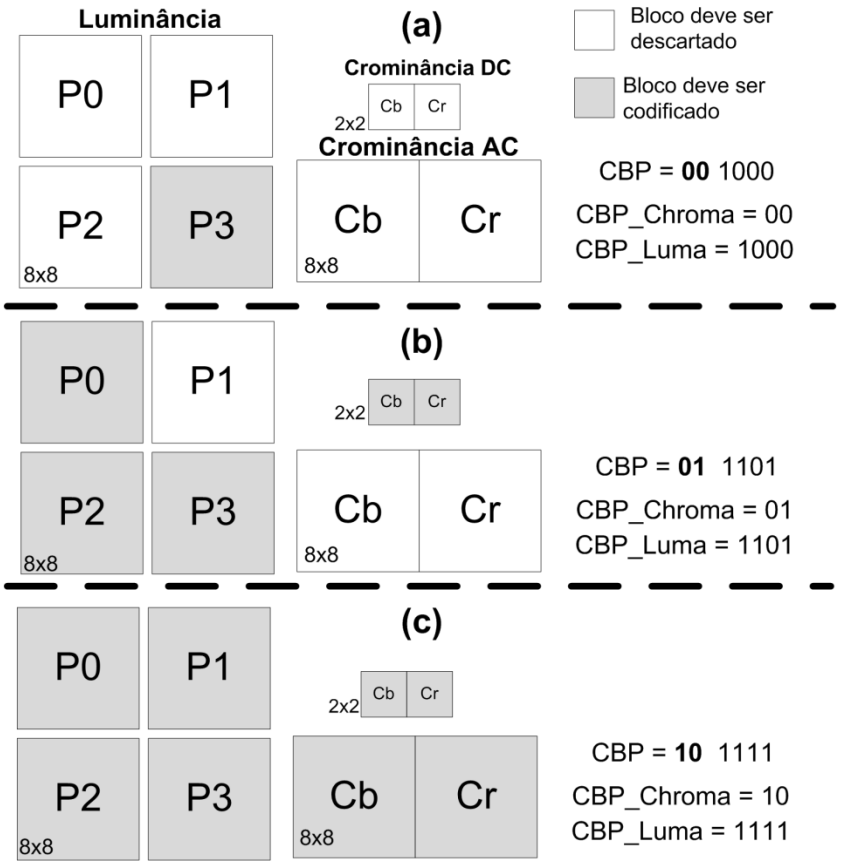

Figura 3.5: Exemplos de processamento dos resíduos segundo o SE *coded_block_pattern* (baseado em RICHARDSON, 2010)

O CABAC envia para processamento apenas os blocos Luma 8x8 e Croma indicados pelo SE *coded_block_pattern*. As amostras de resíduos são classificadas de acordo com seu tipo, como apresentado no capítulo anterior, e esta característica é considerada para o envio correto dos resíduos para a codificação do CABAC. A Figura 3.6 ilustra o ordenamento de envio para os blocos de resíduos. O primeiro bloco a ser enviado, se houver, é o bloco Intra 16x16 DC de tamanho 4x4. Em seguida, são enviados os blocos de luminância com CBP diferente de zero. Se o macrobloco utilizou transformada DCT 4x4, então as amostras serão enviadas para o CABAC em blocos 4x4 na ordem duplo-Z, como apresentado na Seção 2.4.5 deste trabalho. Se o macrobloco foi transformado com a DCT 8x8, então o bloco 8x8 é transmitido diretamente para o CABAC. Após o envio da informação de luminância, as amostras de crominância são enviadas, também de acordo com CBP. Primeiramente, são enviados os blocos 2x2 de crominância vermelha e azul DC e em seguida, os blocos 8x8 de crominância AC são transmitidos em blocos 4x4 na ordem duplo-Z.

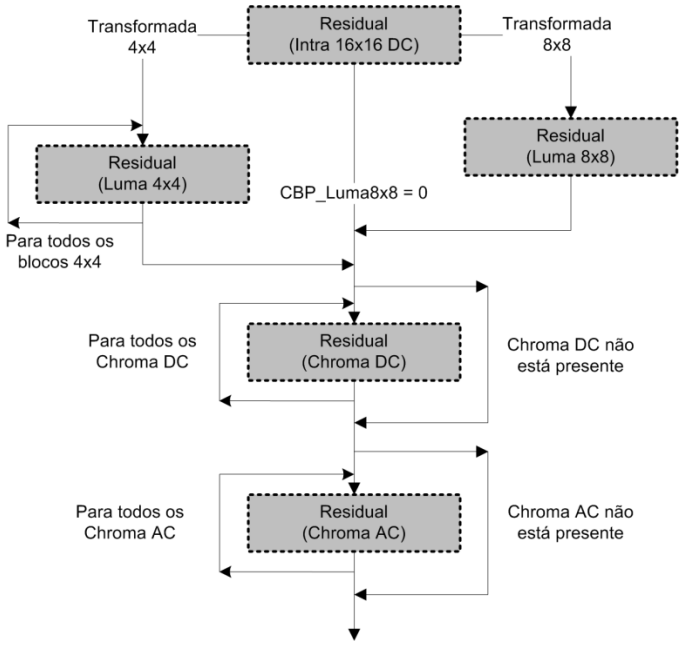

Figura 3.6: *Parsing* das amostras de resíduos de acordo com sua categoria

Para a codificação dos dados de resíduos, elementos sintáticos foram especificamente desenvolvidos para serem utilizados pelo CABAC:

- **coded_block_flag** (CBF) **–** especifica se há amostras de resíduos significantes no bloco 4x4 de resíduos atual. Não está presente quando nos blocos Luma que utilizaram a transformada 8x8;
- **significant_coeff_flag –** especifica se a amostra é diferente de zero;
- **last_significant_coeff_flag –** indica se a amostra é a última diferente de zero para o bloco 4x4 atual;
- **coeff_abs_level_minus1 –** contém o valor absoluto (sem sinal) subtraído de uma unidade da amostra que está sendo processada;
- **coeff_sign_flag –** especifica se o valor da amostra é negativo ou positivo.

A Figura 3.7 (a) ilustra o esquema de codificação do CABAC para os blocos de residuos. Primeiramente, o SE CBF é transmitido para o bloco de coeficientes que não utilizam transformada 8x8 de acordo com o mapeamento de CBP. Se CBF é nulo, nenhuma informação adicional sobre o bloco atual é enviada, caso contrário, um Mapa de Significância indicando a posição dos blocos significantes é codificado. Em seguida, a Informação de *Level* contendo o valor absoluto da amostra e o sinal da amostra são enviados ao próximo passo de processamento na ordem inversa de varredura. A Figura 3.7 (b) tem o mesmo significado da Figura 3.7 (a), porém ela conclui o processo de *parsing* sob a mesma perspectiva ilustrada na Figura 3.2, apresentando o processo chamado de Residual.

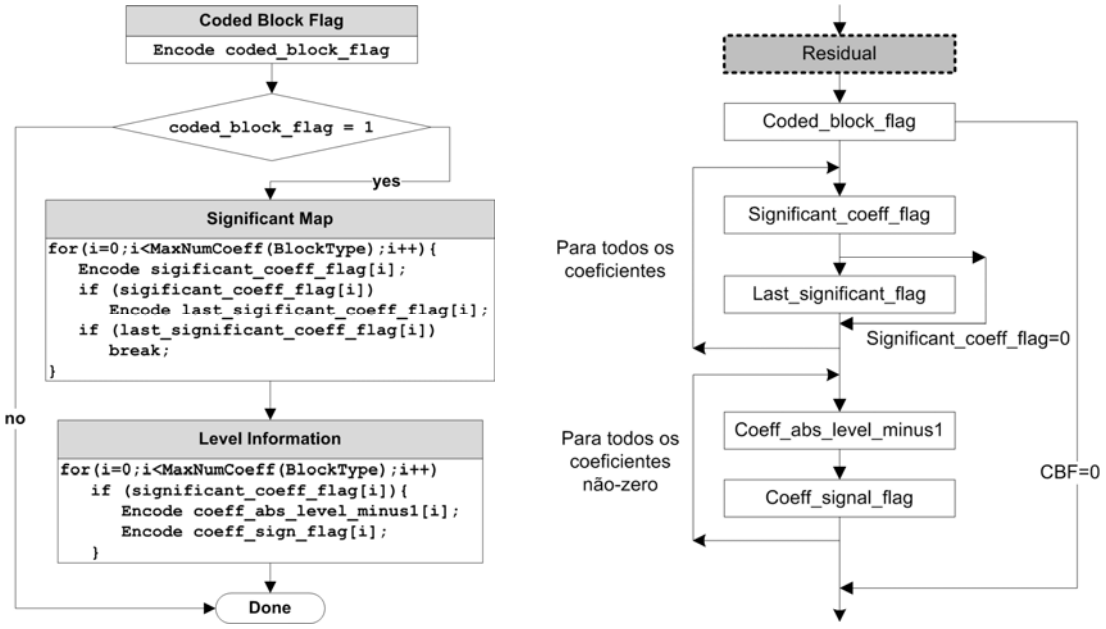

Figura 3.7: Fluxo de codificação para os blocos de resíduos (baseado em MARPE, 2003 e RICHARDSON, 2003).

Aplicando o algoritmo da Figura 3.7 ao macrobloco de exemplo da Figura 3.5(a), temos que apenas o bloco 8x8 P3 contém amostras de resíduos válidas. Supondo que a transformada 4x4 foi utilizada nesse macrobloco, o bloco P3 é entregue ao CABAC na ordem duplo-Z em blocos 4x4 para que suas amostras sejam lidas em ziguezague (Figura 3.8). Conforme o algoritmo dado na Figura 3.7, o bloco F0 não tem coeficientes significantes, então o elemento sintático *coded_block_flag* é 0. Já o bloco F1 tem coeficientes significantes, então o próximo *coded_block_flag* é 1 e, portanto, é necessário realizar os processos de Mapa de Significância e Informação de *Level* para completar a geração de SEs correspondentes aos coeficientes de resíduos do bloco F1.

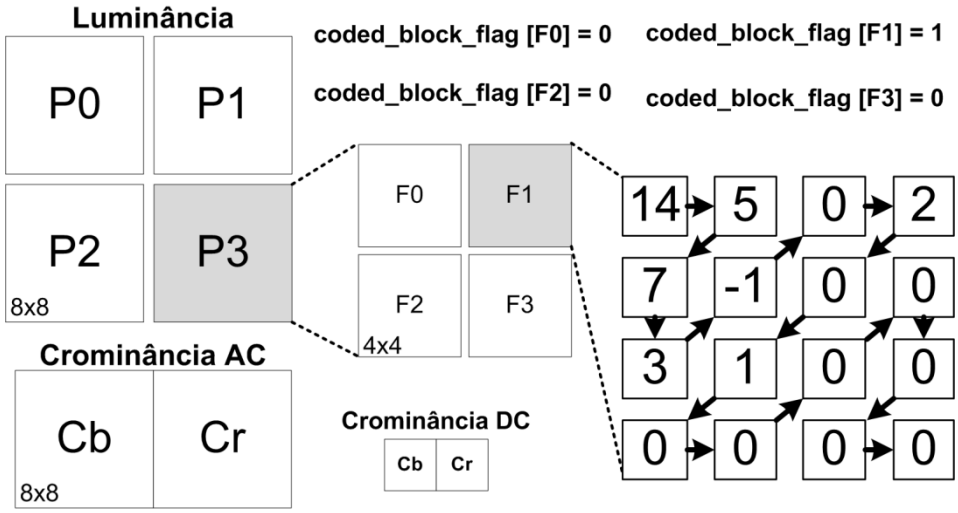

Figura 3.8: Processamento dos resíduos utilizando CABAC.

No processo de Mapa de Significância, para cada coeficiente na ordem de varredura, um SE do tipo *significant_coeff_flag* é transmitido. Se a amostra é válida (*significant_coeff_flag*=1) então um elemento do tipo *last_significant_coeff_flag* é

transmitido. Quando o último coeficiente válido é encontrado (*last_significant_coeff_flag*=1), este processo é encerrado. No bloco de exemplo da Figura 3.8 é possível verificar através do Mapa de Significância que o último coeficiente válido foi o nono a ser lido e que há dois coeficientes não significantes. Em seguida, com a exata localização dos coeficientes significantes do bloco de resíduos atual, há o processamento da Informação de *Level*. Esse processo envia na ordem reversa de varredura ziguezague os SEs *coeff_abs_level_minus1* e *coeff_sign_flag* apenas para as amostras consideradas relevantes. Assim, para o exemplo da Figura 3.8, o conjunto de SEs necessários para representar os dados de resíduos estão listados na Tabela 3.2. Observa-se que para cada valor não-zero da amostra, outros três SEs são criados de acordo com o fluxograma da Figura 3.7.

Tabela 3.2: Demonstração dos SEs gerados pelo exemplo da Figura 3.8.

Posição Escaneada	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
significant_coeff_flag	1	1	1	1	1	0	1	0	1							
last_significant_coeff_flag	0	0	0	0	0		0		1							
coeff_abs_level_minus1	14	5	7	3	-1		2		1							
coeff_sign_flag	0	0	0	0	1		0		0							

3.3 Binarização

O processo de binarização consiste no mapeamento de valores inteiros em uma sequência de bits que representa o valor original. Esse mapeamento é feito para reduzir o tamanho do alfabeto de símbolos, simplificando assim os custos da modelagem de contexto e facilitando a tarefa de codificação aritmética. Cada bit gerado por este processo é chamado de "*bin*". O tamanho do "*binstring*" gerado para cada valor de entrada é variável e depende do tipo da SE que está sendo processado e do contexto atual. O objetivo de usar o mapeamento binário com tamanhos variáveis de *bins* é gerar a menor representação possível para os valores de SE que ocorrem com mais frequência e ainda ter um modelo de probabilidade consistente para cada *bin*, aproximando-se ao limite da entropia (MARPE, 2003).

A binarização pode ser considerada um método de compressão baseada em códigos de comprimento variável que usa as probabilidades estáticas estabelecidas *a priori* pelos desenvolvedores do padrão (INTERNATIONAL, 2003) de ocorrência dos símbolos no conjunto de dados a ser comprimido para determinar códigos de tamanho variável para cada símbolo. O objetivo principal da binarização é converter os valores de elementos sintáticos não-binários em sequencias de valores binários (*binstring*) para posterior processamento pelo Codificador Aritmético Binário. Esta conversão foi planejada de forma que, em muitos casos, é possível estabelecer modelos de contexto para um *bin* individual na sequência. O binarizador considera que a probabilidade dos símbolos com grande largura de *binstring* é tipicamente pequena e que a probabilidade dos símbolos com pequena largura de *binstring* é tipicamente grande. Se essas estimativas estão corretas, haverá compressão, ou seja, o binarizador isoladamente já é um processo de compressão de dados. No entanto, o esquema de binarização aplicado ao CABAC opta pela utilização de algumas poucas árvores de código básico, cuja estrutura permite um cálculo simples e direto de todas as palavras de código sem a necessidade de armazenamento de tabelas.

Para implementar o processo de mapeamento binário, o CABAC define um conjunto de sete métodos de binarização, alguns deles compostos pela combinação de dois métodos através de um esquema de prefixo-sufixo. A decisão da forma de binarização que será utilizada é baseada no tipo de SE, tipo de macrobloco (ou sub-macrobloco), tipo de *slice*, e do valor do SE que está sendo processado. Os principais métodos de binarização são compostos por quatro técnicas básicas: Unário, Unário Truncado, de Tamanho Fixo e Unário/Exp-Golomb parametrizável. Além dessas, outras três formas de binarização especiais são definidas para alguns específicos SEs: Macroblocos e Submacrobloco, binarização do padrão do bloco codificado e binarização para a variação do parâmetro de quantização.

3.3.1 Binarização para Códigos Unário, Truncado Unário e Tamanho Fixo

O método Unário converte o valor inteiro sem sinal de cada símbolo numa cadeia de uns (“1”) com comprimento igual ao valor do símbolo acrescido de um zero (“0”) no final da cadeia. No método Truncado Unário, o bit zero (“0”) não deve ser acrescido ao final da cadeia de bits se o símbolo for maior ou igual a uma variável de controle chamada de *cMax*. O *binstring* produzido com o método Tamanho Fixo é a representação binária deste símbolo com um número fixo de bits fornecido pela variável *lMax*. O método Tamanho Fixo não recodifica o valor do SE, apenas limita o número de bits da cadeia de forma a tornar possível a codificação de todos os valores do SE. A Tabela 3.3 mostra um exemplo do mapeamento de elementos sintáticos para os métodos Unário, Truncado Unário e Tamanho Fixo. As variáveis *cMax* e *lMax* denotam a largura máxima de bits para os métodos Truncado Unário e Tamanho Fixo e a aplicação destes métodos considera a existência de um alfabeto finito de valores possíveis para o valor do SE a ser codificado.

Tabela 3.3: Binarização utilizando os métodos Unário, Truncado Unário e Tamanho Fixo

SE	**Bin**		
	Unário	**Truncado Unário (cMax=4)**	**Tamanho Fixo (lMax=4)**
0	0	0	0000
1	10	10	0001
2	110	110	0010
3	1110	1110	0011
4	11110	1111	0100
5	111110	1111	0101

3.3.2 Binarização Unário/Exp-Golomb Parametrizável

O *binstring* do método de binarização Unário/Exp-Golomb Parametrizável é dado pela concatenação de um prefixo e um sufixo (quando presente). O *binstring* do prefixo é definido pelo método Truncado Unário e o *binstring* do sufixo é definido por uma extensão parametrizável do método Exp-Golomb apresentado no capítulo anterior. No entanto, o *binstring* do Exp-Golomb Parametrizável não tem um comportamento linear como o apresentado na Tabela 2.3, pois é dependente das variáveis *uCoff* e *k*.

Os códigos Exp-Golomb Parametrizáveis no H.264/AVC são compostos por um prefixo (P) e um sufixo (S) como definido em (3.1), (3.2) e (3.3). Esses códigos são formados de maneira similar aos códigos Exp-Golomb apresentado na Seção anterior, porém o prefixo é formado por uma sequência de uns (‘1’), ao invés de zeros, há um zero (‘0’) separando o prefixo e o sufixo ao contrário do ‘1’ e, por último, o prefixo e o

sufixo não necessitam ter o mesmo tamanho devido à presença do parâmetro *k* na equação.

$$\underbrace{1 \ldots 1}_{P}\ 0\ \underbrace{x_{M-1} \ldots x_0}_{S} \quad (3.1)$$

$$P = \log_2(x/2^k + 1) \quad (3.2)$$

$$S = x + 2^k(1 - 2^P) \quad (3.3)$$

A binarização Unário/Exp-Golomb Parametrizável é dependente de dois parâmetros: *k* define a ordem do sufixo Exp-Golomb Parametrizável; e *uCoff* define o tamanho máximo do prefixo e o valor mínimo de entrada para o qual o sufixo existe. A Tabela 3.4 apresenta exemplos de *binstrings* gerados através da binarização Unário/Exp-Golomb Parametrizável. Na Seção 9.3.2.3 da norma de especificação do padrão H.264/AVC (INTERNATIONAL, 2009) há um algoritmo que gera a mesma palavra de saída para o método Unário/Exp-Golomb Parametrizável apresentado na Tabela 3.4.

Tabela 3.4: Exemplos de binarização utilizando o método Exp-Golomb Parametrizável

SE	**Truncado Unário** (uCoff=14)	**Exp-Golomb parametrizável** (k=0)		
		Prefixo		**Sufixo**
0	0			
1	10			
...	...	...	...	...
12	1111111111110			
13	11111111111110			
14	11111111111111		0	
15	11111111111111	1	0	0
16	11111111111111	1	0	1
17	11111111111111	11	0	00
18	11111111111111	11	0	01
...	...	...	...	...
SE	**Truncado Unário** (uCoff=9)	**Exp-Golomb parametrizável** (k=3)		
		Prefixo		**Sufixo**
0	0			
1	10			
...	...	...	...	...
8	111111110			
9	111111111		0	
10	111111111	1	0	000
11	111111111	1	0	001
12	111111111	1	0	010
13	111111111	1	0	011
14	111111111	1	0	100
15	111111111	1	0	101
16	111111111	1	0	110
17	111111111	1	0	111
18	111111111	11	0	0000
...	...	...	...	...

3.3.3 Outros Métodos de Binarização e Exceções

Os métodos de binarização apresentados até agora são os principais e geram mais de 90% dos *bins* que precisam ser processados pelo CABAC (OSORIO, 2006). Porém, alguns SEs têm tratamento diferenciado.

Uma das exceções é a binarização do SE **coded_block_pattern**, que consiste na concatenação de um prefixo gerado com a binarização de Tamanho Fixo (*lMax*=4) e um sufixo (quando presente) binarizado com o método Truncado Unário (*cMax*=2). O sufixo está presente quando uma *flag* denominada *ChromaArrayType* não é 0 ou 3.

Outro SE com método de binarização próprio, é a variação do parâmetro de quantização (*mb_qp_delta*). Este SE pode apresentar valores negativos e, como a binarização só é realizada para valores positivos, é preciso aplicar um pré-mapeamento, apresentado em (2.1) para que os dados possam ser processados pelo método Unário.

A binarização para os SE que especificam o tipo de macrobloco e submacrobloco (*mb_type* e *sub_mb_type*) é feita por mapeamento de tabelas em memória ROM. O valor do SE serve como índice de endereço e a saída de uma das cinco tabelas de memória é selecionado como o *bin* de saída. Há três tabelas ROM para tratar o SE do tipo macrobloco, uma para cada tipo (I, P ou B). As duas tabelas restantes são para retornar o *bin* proveniente do SE tipo de submacrobloco, pois o submacrobloco pode ser do tipo P ou B. A seleção de qual tabela deve ser selecionada depende do tipo de *slice* e da predição que foi utilizada no macrobloco. Se o *slice* é I, então só há a possibilidade de consultar a tabela de macroblocos I. Se o *slice* é P, então o macrobloco pode ser P ou I. Se o macrobloco for P, então a consulta é realizada diretamente na tabela de macrobloco P. Se o macrobloco for I, então um sufixo é adicionado ao *bin* gerado pela tabela I. De forma análoga ao macrobloco P, o mesmo caso é aplicado se o macrobloco for B.

3.3.4 Ordenamento e análise de ocorrência

A distinção dos SEs por grupos é importante, pois o conjunto de informações muda significativamente de acordo com o tipo de macrobloco que estiver sendo processado enquanto as informações de resíduos são, basicamente, as mesmas para todos os tipos de macroblocos, como apresentado na Seção 3.1. Outra observação relacionada com o que está apresentado na Seção 3.1, indica que os SEs não-residuais, em geral, ocorrem poucas vezes dentro de um macrobloco gerando poucos *bins* em cada ocorrência enquanto os SEs de resíduos ocorrem várias vezes e produzem muitos *bins* em cada ocorrência. Para todos os SEs é atribuído um dos sete métodos de binarização, os quais irão gerar um *binstring* de tamanho variável, de acordo com o valor do SE e os parâmetros de limitação (*cMax*, *lMax*, *k* e *uCoff*). A Tabela 3.5 apresenta todos os SEs processados pelo CABAC organizados em grupos, classificando o tipo de informação que esses elementos representam. As ocorrências mínimas e máximas de cada SE em cada tipo de macrobloco, o número mínimo e máximo de *bins* (separados em prefixo e sufixo) e o método de binarização correspondente a cada SE também são contemplados na Tabela 3.5.

Tabela 3.5: Relação dos tipos de SE de macrobloco, taxa de ocorrência por tipo de macrobloco, número de *bins* produzidos e o método de binarização correspondente.

	Tipo de Elemento Sintático	Ocorrência por MB			Δlargura do binstring		Método de binarização
		I	P	B	Prefixo	Sufixo	
info. de Slice e MB	mb_skip_flag	-	1	1	1		FL (*lMax=1*)
	mb_field_decoding_flag	0-1	0-1	0-1	1		FL (*lMax=1*)
	transform_size_8x8_flag	0-1	0-1	0-1	1		FL (*lMax=1*)
	mb_type	1	1	1	1-6	0-7	Tabela ROM
	sub_mb_type	-	0 ou 4	0 ou 4	1-6		Tabela ROM
	coded_block_pattern	0-1	1	1	4	0-2	FL (*lMax=4*) e TU (*cMax=2*)
	mb_qp_delta	0-1	1	1	1-26		U (após passar pela equação 2.1)
	end_of_slice	1	1	1	1		FL (*lMax=1*)
Inter	ref_idx_l0	-	0-4	0-4	1-5		U
	ref_idx_l1	-	-	0-4	1-5		U
	mvd_l0 (par)	-	0-16	0-16	1-18	0-9	UEG (*k=3, uCoff=9*)
	mvd_l1 (par)	-	-	0-16	1-18	0-9	UEG (*k=3, uCoff=9*)
Intra	intra_chroma_pred_mode	0-8	-	-	1-2		TU (*cMax=2*)
	prev_intra4x4_pred_mode	0 ou 16	-	-	1		FL (*lMax=1*)
	rem_intra4x4_pred_mode	0-16	-	-	3		FL (*lMax=3*)
	prev_intra8x8_pred_mode	0 ou 4	-	-	1		FL (*lMax=1*)
	rem_intra8x8_pred_mode	0-4	-	-	3		FL (*lMax=3*)
Resíduos	coded_block_flag	0-27	0-27	0-27	1		FL (*lMax=1*)
	significant_coeff_flag	0-384	0-384	0-384	1		FL (*lMax=1*)
	last_significant_coeff_flag	0-384	0-384	0-384	1		FL (*lMax=1*)
	coeff_ sign_flag	0-384	0-384	0-384	1		FL (*lMax=1*)
	coeff_ abs_level_minus1	0-384	0-384	0-384	1-25	0-13	UEG (*k=0, uCoff=14*)

3.4 Modelagem de Contexto

Um modelo de contexto é um modelo probabilístico que representa as estatísticas de distribuição e/ou ocorrência de um determinado símbolo, com base na análise da frequência de ocorrência daquele símbolo para a fonte em questão e da observação dos símbolos processados anteriormente durante um determinado processo de codificação. Na etapa de modelagem uma determinada distribuição probabilística é associada a cada símbolo (*bin*) para que a etapa subsequente (etapa de codificação aritmética) possa gerar a sequência de bits, que seja a representação codificada daquele símbolo, de acordo com o seu modelo de distribuição probabilístico para uma determinada fonte.

Para que o codificador aritmético funcione adequadamente, atingindo compressão significativa dos dados, a escolha do modelo de contexto precisa ser a mais próxima possível da distribuição probabilística real de cada símbolo. Para cada *bin* do *binstring* é associado um modelo de contexto que deve ser corretamente calculado pelo CABAC. Realizar essa escolha de maneira ótima em tempo de execução seria extremamente custoso computacionalmente (MARPE, 2003), por isso o CABAC definiu uma quantidade limitada de modelos de contexto. No perfil *Main*, há 399 modelos de probabilidades distintos enquanto que no perfil *High*, há 1024 modelos de contextos os quais são utilizados para estimar a probabilidade de um *bin*. Cada contexto contém a informação do símbolo mais provável de ocorrer (0 ou 1) e um índice de probabilidade. O CABAC utiliza essas informações para realizar a codificação aritmética através do motor de codificação *regular*. Para *bins* de SEs considerados equiprováveis pelos proponentes do CABAC, não há modelagem de contexto e esses *bins* são processados

utilizando o motor de codificação *bypass.* Os *bins* de SEs equiprováveis e que não utilizam a modelagem de contextos são: **mvd_l0** (sufixo), **mvd_l1** (sufixo), **coeff_abs_level_minus1** (sufixo), e **coeff_sign_flag**. A utilização da informação de contexto e dos motores aritméticos será descrita na Seção 3.5

Encontrar o modelo de contexto para determinado símbolo, consiste em calcular o índice para o modelo de contexto desejado. Há duas equações para calcular o modelo de contexto (*ctxIdx*), sendo a primeira (3.4) relacionada aos SEs não-residuais e a segunda (3.5) atribuída aos SEs de resíduos.

$$ctxIdx = ctxOffset + ctxInc \quad (3.4)$$

$$ctxIdx = ctxOffset + ctxInc + ctxCat \quad (3.5)$$

O valor de *ctxOffset* depende exclusivamente do tipo do SE e marca o início da porção da memória de probabilidades com os contextos para um determinado SE. Os valores *ctxInc* e *ctxCat* definem qual dentre os contextos para um determinado tipo de SE deve ser escolhido.O cálculo do *ctxCat* depende do tipo de SE e do tipo de bloco de dados que está sendo processado. A Tabela 3.6 apresenta a descrição das 14 categorias de bloco existentes para a realização do cálculo do modelo de contexto. A categoria 5 ocorre somente no perfil *High* e as categorias maiores que 6 (inclusive) somente ocorrem no perfil *High* com subamostragem 4:4:4.

Tabela 3.6: Descrição das categorias de bloco para a modelagem de contextos.

Categoria	**Máx. de Coef.**	**Descrição de Bloco**
0	16	Luma DC 16x16
1	15	Luma AC 16x16
2	16	Luma 4x4
3	4*b_8x8	Croma DC
4	15	Croma AC
5	64	[HIGH] Luma 8x8
6	16	[HIGH] Croma azul DC 16x16 – 4:4:4
7	15	[HIGH] Croma azul AC 16x16 – 4:4:4
8	16	[HIGH] Croma azul 4x4 – 4:4:4
9	64	[HIGH] Croma azul 8x8 – 4:4:4
10	16	[HIGH] Croma vermelho DC 16x16 – 4:4:4
11	15	[HIGH] Croma vermelho AC 16x16 – 4:4:4
12	16	[HIGH] Croma vermelho 4x4 – 4:4:4
13	64	[HIGH] Croma vermelho 8x8 – 4:4:4

Para realizar o cálculo do *ctxInc*, o CABAC adota cinco estratégias distintas para realizar a modelagem de contextos com o objetivo de diminuir custo computacional das operações sem sacrificar significativamente a eficácia na codificação. A adoção de cada estratégia depende do tipo de SE e do índice do *bin* no *binstring*. Basicamente, as estratégias para o cálculo do *ctxInc* são realizadas considerando:

1. Os SEs dos macroblocos vizinhos;
2. Os *bins* ou SEs previamente codificados;
3. A posição do coeficiente que está sendo tratado no momento;
4. Estatísticas relativas à Informação de *Level* no bloco de resíduos;
5. Atribuição fixa. Nenhuma informação adicional é utilizada além do tipo do SE e do índice do *bin*.

Nas próximas subseções será detalhada a modelagem de contexto para cada *bin* de todos os SEs tratados pelo CABAC de acordo com as estratégias descritas. Isso significa que o cálculo dos parâmetros *ctxOffset*, *ctxCat* e *ctxInc* serão descritos um a um até formar a especificação completa dos 1024 modelos de contexto previstos no padrão H.264/AVC.

3.4.1 Construção da Tabela de Contextos

No começo da codificação de um novo *slice*, o codificador deve construir uma nova tabela de contextos, com os 1024 modelos probabilísticos dos *bins*. O Padrão H.264/AVC prevê a existência de valores iniciais estabelecidos de acordo com um algoritmo para a inicialização dos modelos de contextos. O padrão H.264/AVC fornece uma série de tabelas para a inicialização dos contextos que produzem como saída um valor denominado *m* e outro denominado *n*. Para cada número de contexto, existem quatro pares *m* e *n*, sendo um par com valores de contexto para *slices* do tipo I, e os outros três pares que dependem de um parâmetro externo *Init IDC*. Uma vez selecionado qual o conjunto de pares *m* e *n* que será utilizado para a inicialização da tabela de contextos, esses valores são utilizados em um algoritmo composto por algumas operações aritméticas que levam em consideração o valor do QP inicial utilizado para o *slice*. O resultado da seleção da coluna de valores *m* e *n* e as operações aritméticas com o valor inicial de QP são utilizados como valor inicial para a tabela de contextos.

3.4.2 Modelagem de Contextos Considerando Elementos Sintáticos de Blocos Vizinhos

Devido à elevada correlação entre blocos vizinhos, o padrão H.264/AVC define que modelos de contextos para alguns *bins* sejam definidos com base nos dados dos blocos vizinhos. Entretanto, há uma série de maneiras diferentes de especificar a vizinhança entre os blocos, chamada tecnicamente de "processo de derivação de vizinhança", onde para cada tipo de SE é associado um jeito de especificar (ou um processo de derivação) a vizinhança. Para realizar corretamente a modelagem de contexto para os *bins* que precisam consultar dados de vizinhança é necessário, portanto, considerar a relação entre os dados vizinhos para cada tipo de SE.

A Figura 3.9 resume os processos de derivação de vizinhança previstos para os SEs processados com CABAC. Para o cálculo de contextos são considerados apenas os vizinhos acima e à esquerda do macrobloco atual. Em outros módulos do codificador H.264/AVC, como no preditor interquadros, informações de vizinhos nas diagonais acima e à direita e acima e à esquerda também são utilizados no processamento de dados.

A Figura 3.9(a, b e c) ilustra o processo de derivação de vizinhança de blocos 16x16, 8x8 e 4x4, que acontece, geralmente, para SEs com ocorrência única por macrobloco, submacrobloco e bloco 4x4, respectivamente. Porém, não são todos os SEs que possuem um modo estático de derivação de vizinhança. A Figura 3.9(d) mostra um exemplo em que a correlação de vizinhos em blocos não é regular apresentando partições de vários tamanhos diferentes. A Figura 3.9(d) mostra que as partições vizinhas não precisam ter o mesmo tamanho para serem consideradas pela modelagem de contextos e que, se houver mais de um vizinho acima, então, apenas o vizinho acima mais à esquerda é considerado e se houver mais de um vizinho à esquerda, apenas o vizinho à esquerda mais acima é considerado. A escolha do vizinho de bloco quando o

bloco tem mais de um vizinho está claramente indicado pelo posicionamento das setas na Figura 3.9.

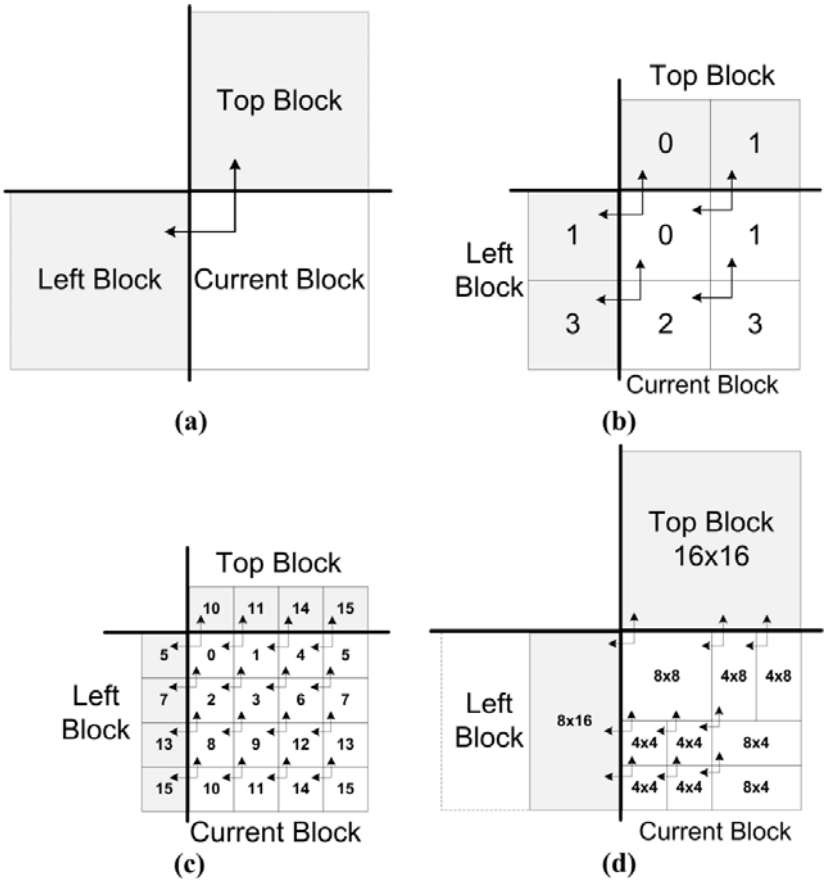

Figura 3.9: Processos de derivação de vizinhança para a modelagem de contextos do CABAC.

A apresentação dos vários tipos de vizinhança do H.264 evidencia que a granularidade da vizinhança tem relação direta com a abrangência dos SEs dentro do macrobloco. Por exemplo, o SE *coded_block_pattern* utiliza o processo de derivação de vizinhança da Figura 3.9(c) porque este SE ocorre a cada bloco 4x4 enquanto que o SE *mb_skip_flag* utiliza o processo de derivação de vizinhança da Figura 3.9(a), pois sua granularidade é em blocos 16x16. Além disso o SE procurado pode não existir no bloco/macrobloco vizinho ou simplesmente não há bloco vizinho, devido ao macrobloco atual encontrar-se numa borda. Um exemplo de falta de vizinho pode acontecer quando é consultado o vizinho de um vetor de movimento (*mvd_l0* ou *mvd_1*), mas o macrobloco vizinho foi codificado com predição intraquadros. O cálculo do incremento de contexto (*ctxInc*) será detalhado a seguir para cada SE que necessitam de dados de vizinhos.

A equação que define o *ctxInc* para o *bin* do SE *mb_skip_flag* é definida em 3.6 e a equação que define o *ctxInc* para o primeiro *bin* do *binstring* dos SEs *mb_type*, *mb_field_decoding*, *intra_chroma_pred_mode* e *transform_size_8x8_flag* é definida em 3.7. O parâmetro *SEtype* é definido como um dos quatro SEs que utilizam a equação 3.7, analogamente à sintaxe utilizada por *mb_skip_flag* na equação 3.6. De acordo com essas equações, existem três valores possíveis para *ctxInc* e, portanto, três modelos de contexto diferentes para cada um destes cinco SEs. Como esses cinco SEs ocorrem apenas uma vez a cada macrobloco, então o processo de derivação de vizinhança

utilizado para eles é o da Figura 3.9(a). Se algum dos SEs vizinhos não estiver disponível, então *SEtype(X)* é igual a zero.

$$ctxInc = (mb\_skip\_flag(Left) \neq 0)?\,0:1 + (mb\_skip\_flag(Top) \neq 0)?\,0:1 \quad (3.6)$$

$$ctxInc = (SEtype(Left) = 0)?\,0:1 + (SEtype(Top) = 0)?\,0:1 \quad (3.7)$$

A equação que define o *ctxInc* para o primeiro *bin* do *binstring* dos SEs *ref_idx_l0* e *ref_idx_l1* e para todos os *bins* dos SEs *coded_block_pattern* (prefixo) é definida na equação 3.8. De acordo com a equação, existem quatro valores possíveis para *ctxInc* e, portanto, quatro modelos de contexto diferentes para cada um destes quatro SEs. Os SEs *ref_idx_l0* e *ref_idx_l1* podem ocorrer uma, duas ou quatro vezes por macrobloco, dependendo de como o macrobloco foi particionado (Seção 3.2.2), e serem dispostos de quatro maneiras diferentes por macrobloco, assim, o processo de derivação de vizinhança para *ref_idx* é o da Figura 3.9(d). O SE *coded_block_pattern* ocorre apenas uma vez por macrobloco, mas, como visto na Seção anterior, cada *bin* desse SE representa uma decisão de codificação para um bloco 8x8 de coeficientes (Seção 3.2.3) e o processo de derivação de vizinhança dos *bins* é o da Figura 3.9(b). Se algum dos SEs vizinhos não estiver disponível, então *SEtype(X)* é igual a zero.

$$ctxInc = 2 * (SEtype(Left) \neq 0)?\,0:1 + (SEtype(Top) \neq 0)?\,0:1 \quad (3.8)$$

As equações que definem o *ctxInc* para o sufixo de *bins* do SE *coded_block_pattern* (CBP) estão ilustradas em 3.9 e 3.10. O *binstring* de *coded_block_pattern* (sufixo) tem no máximo dois *bins* e de acordo com a equação, existem oito valores possíveis para *ctxInc*, sendo quatro modelos de contexto distintos para cada um dos dois *bins*. A equação 3.9 é referente ao primeiro *bin* de CBP (sufixo) e a equação 3.10 corresponde ao segundo *bin*. O SE *coded_block_pattern* ocorre apenas uma vez por macrobloco e, portanto, o processo de derivação de vizinhança dos *bins* é o da Figura 3.9 (a). Se CBP dos vizinhos não estiver disponível, então *CBPsuffix(X)* é igual a zero.

$$ctxInc = 2 * (CBPsuffix(Left) = 0)?\,0:1 + (CBPsuffix(Top) = 0)?\,0:1 \quad (3.9)$$

$$ctxInc = 4 + 2 * (CBPsuffix(Left) \neq 2)?\,0:1 + (CBPsuffix(Top) \neq 2)?\,0:1 \quad (3.10)$$

A equação 3.11 define o *ctxInc* para o *bin* do SE *coded_block_flag* (CBF). A correlação de vizinhança para o SE CBF depende do tipo (categoria) de bloco que CBF representa. Pode existir, no pior caso, 27 SEs do tipo CBF no *bitstream* (Seção 3.2.3) do CABAC por macrobloco: 16 CBFs correspondem a informação de luminância e para esses SEs o processo de derivação de vizinhança dos *bins* é o da Figura 3.9(c); 4 CBFs correspondem a informação de crominância azul e outros 4 correspondem à crominância vermelha e, considerando que os blocos de Croma são de 8x8 pixels, para esses CBFs o processo de derivação de vizinhança dos *bins* é o da Figura 3.9 (b). Ainda existe 1 CBF para cada bloco com resíduo DC sendo o processo de derivação de vizinhança da Figura 3.9 (a) associado à esses CBFs.

$$ctxInc = 2 * (CBF(Left) = 0)?\,0:1 + (CBF(Top) = 0)?\,0:1 \quad (3.11)$$

Como apresentado na Seção 3.1.2, cada vetor de movimento é composto por duas componentes, horizontal e vertical, que são enviadas separadamente para o *bitstream*. A modelagem de contexto do primeiro *bin* para cada componente (x e y) é realizado de forma individual. A equação que define o *ctxInc* para o primeiro *bin* dos SE *mvd_l0* e *mvd_l1* é definida em 3.12. O valor absoluto dos vetores de movimento dos blocos vizinhos acima e à esquerda da componente *cmp* são somados e para o resultado dessa soma é realizada uma comparação para a definição de qual, dentre os três modelos de contexto disponíveis será utilizado. Os pares de *mvd_l0* e *mvd_l1* podem ocorrer entre

uma e dezesseis vezes por macrobloco, dependendo de como o macrobloco foi particionado (Seção 3.2.2), e, por isso, dezenas de combinações entre partições de blocos para os macroblocos atual, acima e esquerdo são possíveis. Assim, o processo de derivação de vizinhança para *mvd* é dinâmico durante a codificação, sendo definido de acordo com a regra exposta no exemplo da Figura 3.9(d).

$$ctxInc = \begin{cases} 0, & |mvd(Top,cmp)| + |mvd(Left,cmp)| < 3 \\ 1, & 3 \leq |mvd(Top,cmp)| + |mvd(Left,cmp)| \leq 32 \\ 2, & |mvd(Top,cmp)| + |mvd(Left,cmp)| > 32 \end{cases} \quad (3.12)$$

3.4.3 Modelagem de Contexto Considerando Informações Previamente Codificadas

A modelagem de contexto para alguns *bins* é baseada em informações previamente codificadas, o que não significa que essas informações sejam vizinhas. O parâmetro *ctxInc* para o primeiro *bin* do SE *mb_qp_delta*, o qual pode ocorrer apenas uma vez por macrobloco, se baseia na informação de *mb_qp_delta* do macrobloco codificado anteriormente ao atual, sendo vizinho ou não. Alguns *bins* dos SEs *mb_type* e *sub_mb_type* tem o incremento *ctxInc* definido a partir do teste de *bins* anteriormente codificados. A Tabela 3.7 resume todos os casos que se enquadram nessa estratégia de modelagem de contexto, listando todos os cálculos de *ctxInc* para os *bins* que utilizam essa regra. É importante destacar que a modelagem de contextos dos *bins mb_type* e *sub_mb_type* é dependente do tipo de macrobloco que está sendo processado.

Tabela 3.7: Especificação de *ctxInc* para *bins* que utilizam informações previamente codificadas

Elemento Sintático	Índice do *bin*	ctxInc
mb_qp_delta	0	(mb_qp_delta(previously)=0)?0:1
mb_type (I)	4	(binstring(3)=1)?5:6
mb_type (I)	5	(binstring(3)=1)?6:7
mb_type (P – prefixo)	2	(binstring(1)=0)?2:3
mb_type (P – sufixo)	4	(binstring(3)=1)?2:3
mb_type (B – prefixo)	2	(binstring(1)=1)?4:5
mb_type (B – sufixo)	4	(binstring(3)=1)?2:3
sub_mb_type (B)	2	(binstring(1)=1)?2:3

3.4.4 Modelagem de Contexto para Elementos Sintáticos De Resíduos

A modelagem de contextos para os SEs *significant_coeff_flag* e *last_significant_coeff_flag* é dependente da posição de varredura utilizada no Mapa de Significâncias (Seção 3.2.3), do número máximo de coeficientes (*MaxNumCoeff*) e da categoria do contexto conforme a Tabela 3.8. Para os blocos de luminância que utilizaram transformada DCT 8x8, ou seja, as categorias 5, 9 e 13, o incremento *ctxInc* para ambos SEs é tabelado (Tabela 9-43 da norma de especificação do H.264/AVC – INTERNATIONAL, 2009). Se o bloco de resíduos é de crominância AC, ou seja, categoria 3, então o parâmetro *ctxInc* é dado pela equação 3.13, onde *levelIdx* é o índice de varredura do Mapa de Significância e *Num8x8* é referente à amostragem do vídeo (*Num8x8*=1 se amostragem é 4:2:0 e *Num8x8*=2 se amostragem é 4:2:2). Para as outras categorias, *ctxIdx* é calculado diretamente através do índice de varredura do Mapa de Significância (Equação 3.14).

$$ctxInc = Min(\frac{levelIdx}{Num8x8}, 2) \quad (3.13)$$

$$ctxInc = levelIdx \quad (3.14)$$

Para codificar o SE *coeff_abs_level_minus1* (prefixo), dois conjuntos de modelos de contextos foram projetados baseados em duas informações estatísticas sobre o bloco atual: *numEq1* e *numGt1*. A estatística *numEq1* denota o número acumulado de coeficientes de transformada com valor absoluto igual a 1. A estatística *numGt1* denota o número acumulado de coeficientes de transformada com valor absoluto maior que 1. Essas Estatísticas são geradas durante o processo de coleta de Informações de *Level*, após o Mapa de Significância (Seção 3.2.3). Os contadores de *numEq1* e *numGt1* iniciam com valor nulo e durante a leitura das amostras de resíduos, quando os coeficientes são lidos na ordem reversa. Se o coeficiente é 1, então *numEq1* é incrementado em uma unidade. Se o coeficiente é maior que 1, então *numEq1* é zerado e *numGt1* é incrementado em uma unidade. Cada coeficiente terá, ao final da varredura reversa, um conjunto de estatísticas único. A Tabela 3.8 descreve o valor das estatísticas *numEq1* e *numGt1* para o bloco de exemplo da Figura 3.8

Tabela 3.8: Determinação de estatísticas de resíduos para o bloco da Figura 3.8.

Posição Escaneada	0	1	2	3	4	5	6	7	8
coeff_abs_level_minus1	14	5	7	3	-1		2		1
numEq1	0	0	0	0	1		0		1
numGt1	5	4	3	2	1		1		0

Independente da categoria do bloco que está sendo processado *numGt1* e *numEq1* são utilizados para a modelagem de contextos. Assim, para o primeiro *bin* do prefixo de *coeff_abs_level_minus1* o incremento *ctxInc* é dado em 3.15 e para os demais *bins* o parâmetro *ctxInc* é dado em 3.16. É preciso ressaltar que os *bins* do sufixo desse SE são equiprováveis, por isso não há modelagem de contexto para esses *bins* e eles são enviados diretamente para o codificador aritmético.

$$ctxInc = (numGt1 \neq 0)?\, 0 : Min(4, 1 + numEq1) \quad (3.15)$$

$$ctxInc = 5 + Min(4 + (blockCat = 3)?\, 1 : 0, numGt1) \quad (3.16)$$

3.4.5 Modelagem de Contexto Estática

A última e mais simples estratégia para modelagem de contexto dos *bins* é baseada apenas no tipo de SE e no índice do *bin*. Na Tabela 3.9 estão listados na última coluna (considerando que a tabela tem três colunas) todos os *ctxInc* possíveis na modelagem de contextos para todos *bins* de todos SEs processados com CABAC. Para alguns *bins* dessa tabela, mais de um contexto pode ser admitido por causa do cálculo de *ctxInc*. Os incrementos *ctxInc* destacados em cinza claro na Tabela 3.9 utilizam a informação de vizinhos e as equações da Seção 3.4.1 para escolha do modelo de contexto. Os parâmetros *ctxInc* marcados em preto na Tabela 3.9 são determinados com base em informações previamente codificadas, como mostrado na Seção 3.4.2. Finalmente, os incrementos *ctxInc* destacados em cinza escuro são exclusivos de *bins* de resíduos e são calculados através de estatísticas e da posição do coeficiente no bloco de resíduos, conforme demonstrado na Seção 3.4.4. O SE *end_of_slice* e, em alguns casos, o primeiro *bin* de *mb_type* recebem diretamente o contexto de número 276, indicando que esses *bins*, ao contrário de todos os outros, deve ser codificado utilizando o motor *terminate* do codificador aritmético, ao invés do motor *regular*.

Tabela 3.9: Especificação de *ctxInc* e *ctxOffset* para para os *bins* dos SE que utilizam modelagem de contexto

Elemento Sintático	**ctxOffset**	**ctxInc de acordo com o Índice do *bin***						
		0	**1**	**2**	**3**	**4**	**5**	**>=6**
mb_type (I)	3	**0,1,2**	ctxIdx=276	3	4	**5,6**	**6,7**	7
mb_skip (P)	11	**0,1,2**	-	-	-	-	-	-
mb_type (P prefix)	14	0	1	**2,3**	-	-	-	-
mb_type (P suffix)	17	0	ctxIdx=276	1	2	**2,3**	3	3
sub_mb_type (P)	21	0	1	2	-	-	-	-
mb_skip (B)	24	**0,1,2**	-	-	-	-	-	-
mb_type (B prefix)	27	**0,1,2**	3	**4,5**	5	5	5	5
mb_type (B suffix)	32	0	ctxIdx=276	1	2	**2,3**	3	3
sub_mb_type (B)	36	0	1	**2,3**	3	3	3	-
mvd_l0 (prefix)	40	**0,1,2**	3	4	5	6	6	6
mvd_l1 (prefix)	47	**0,1,2**	3	4	5	6	6	6
ref_idx_l0, ref_idx_l1	54	**0,1,2,3**	4	5	5	5	5	5
mb_qp_delta	60	**0,1**	2	3	3	3	3	3
intra_chroma_pred_mode	64	**0,1,2**	3	3	-	-	-	-
prev_intraNxN_pred_mode	68	0	-	-	-	-	-	-
rem_intraNxN_pred_mode	69	0	0	0	-	-	-	-
mb_field_decoding	70	**0,1,2**	-	-	-	-	-	-
CBP (prefix)	73	**0,1,2,3**	**0,1,2,3**	**0,1,2,3**	**0,1,2,3**	-	-	-
CBP (suffix)	77	**0,1,2,3**	**4,5,6,7**	-	-	-	-	-
end_of_slice	276	0	-	-	-	-	-	-
transform_size_8x8_flag	399	**0,1,2**	-	-	-	-	-	-
coded_block_flag	Tab. 3.10	**0,1,2,3**	-	-	-	-	-	-
significant_coeff_flag	Tab. 3.10	**0-15**	-	-	-	-	-	-
last_significant_coeff_flag	Tab. 3.10	**0-15**	-	-	-	-	-	-
coeff_abs_level_minus1	Tab. 3.10	**0-4**	**5-9**					

Na Tabela 3.9 é possível consultar o deslocamento *ctxOffset* para cada SE. Analisando a tabela é possível perceber que *ctxOffset* é variável para os SEs *mb_skip_flag*, *mb_type* e *sub_mb_type* de acordo com o tipo de macrobloco (I, P ou B). A especificação do *ctxOffset* para os SEs de resíduos depende de qual dentre as 14 categorias, o bloco residual transformado se enquadra. Além disso, conforme a equação 3.3, o modelo de contexto para SEs de resíduos necessita do parâmetro *ctxCat*, variável com a categoria do bloco de resíduos. A especificação de *ctxOffset* e *ctxCat* para os SEs de resíduos é apresentada na Tabela 3.10. Assim, através das Tabelas 3.9 e 3.10, estão apresentados resumidamente o cálculo de todos os 1024 contextos para todos os *bins* de todos os SEs codificados através do CABAC.

Tabela 3.10: Especificação de *ctxInc* e *ctxCat* para *bins* de SEs de resíduos

Elemento Sintático	Info. p/ o contexto	Categoria													
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
coded_block _flag	**ctxCat**	0	4	8	12	16	0	0	4	8	4	0	4	8	8
	ctxOffset	85					1012	460			1012	472			1012
Significant _coeff_flag	**ctxCat**	0	15	29	44	47	0	0	15	29	0	0	15	29	0
	ctxOffset*	105/277					402/436	484/776			660/675	528/820			718/733
last_significant _coeff_flag	**ctxCat**	0	15	29	44	47	0	0	15	29	0	0	15	29	0
	ctxOffset*	166/338					417/451	572/864	690/699	616/908	748/757				
coeff_abs_ level_minus1	**ctxCat**	0	10	20	30	39	0	0	10	20	0	0	10	20	0
	ctxOffset	227					426	952			708	982			766

*o ctxOffset varia se o bloco codificado é progressivo/entrelaçado.

Analisando a Tabela 3.9, a modelagem de contexto dos *bins* que consideram vizinhos geralmente ocorre para o primeiro *bin* do *binstring* de cada SE (mb_type, mvd, ref_idx, *intra_chroma_pred_mode*) ou para SEs relacionados a codificação de resíduos (CBP e CBF). Essa característica está diretamente relacionada com o processo de binarização e com a complexidade computacional da modelagem de contextos. O *binstring* gerado tem tamanho variável e considerando que *binstrings* de menores cadeias de bits são muito mais prováveis que *binstrings* com grandes cadeias de bits, então é mais importante realizar a modelagem de contexto de forma mais precisa possível nos *bins* que ocorrem com maior frequência, enquanto que nos *bins* com menor frequência, a modelagem de contexto pode ser menos precisa. O custo computacional em especificar seria extremamente alto, do ponto de vista de operações aritméticas e tamanho de memória, para um ganho de taxa de compressão mínimo. Isso ocorre porque existe um compromisso entre precisão na modelagem de contexto e complexidade computacional (MARPE, 2003).

3.5 Codificador Aritmético Binário

O algoritmo de codificação aritmética pode ser visto como uma divisão interativa de um intervalo baseada na probabilidade dos símbolos de um alfabeto que são codificados. A codificação é feita selecionando o subintervalo associado a cada símbolo. Eventualmente, um subintervalo final é obtido, onde qualquer valor dentro do intervalo final caracteriza toda a sequência codificada.

O codificador aritmético binário do CABAC aplica uma subdivisão recursiva do intervalo de probabilidades como mostrado na Figura 3.10 para produzir uma sequência de bits. Duas variáveis, *LOW* representando o limite inferior do intervalo e *RANGE* determinando a dimensão do subintervalo, especificam o intervalo. Além disso, todos os *bins* são classificados como Símbolo Mais Provável (*Most Probable Symbol* – MPS) ou Símbolo Menos Provável (*Least Probable Symbol* – LPS). O intervalo é dividido em duas partes: *RangeMPS* (rMPS) e *RangeLPS* (rLPS). rMPS é utilizado para o cálculo de um novo subintervalo de probabilidades se o *bin* é um MPS, senão, rLPS é usado para este fim.

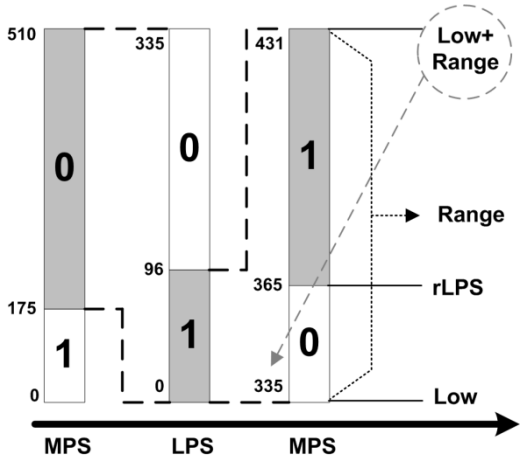

Figura 3.10: Exemplo do processamento do codificador aritmético para a mensagem "011".

Antes que qualquer símbolo seja transmitido, um intervalo de probabilidades do alfabeto que pode estar presente na mensagem é definido entre [0, 510] (*RANGE* = 510; *LOW* = 0). Conforme cada símbolo é processado o espectro do intervalo é reduzido para a porção do intervalo alocada pelo símbolo em questão, de acordo com seu modelo de ocorrência probabilística. O algoritmo de codificação aritmética binária no CABAC que determina o funcionamento do esquema ilustrado na Figura 3.10 pode ser classificado em cinco passos distintos, os quais devem ser repetidos para cada novo *bin* que é codificado e estão descritos a seguir:

1 – Leitura do modelo de contexto: o número de contexto calculado no estágio de modelagem de contexto (Seção 3.4) é um endereço de memória onde cada posição contém a estimativa de probabilidade de LPS, o qual é caracterizado por um índice entre 0 e 63 e o valor do LPS, que é 0 ou 1. A estatística, representada por , é parte de um endereço de memória para o cálculo do rLPS no passo 2.

2 – Subdivisão do intervalo de probabilidades: uma vez determinado o modelo estatístico e o símbolo MPS, é necessário subdividir o intervalo de probabilidades. O índice de probabilidade é utilizado na atualização do intervalo, ou seja, na definição de uma nova porcentagem de ocorrência para LPS. A definição do novo intervalo depende do estado atual de codificação e da probabilidade de cada *bin*. O novo intervalo de LPS (rLPS) consiste na consulta a uma tabela de intervalos LPS indexada por e pelo sétimo e oitavos bit mais significativos de RANGE ((RANGE>>6)&3). O novo intervalo de MPS (rMPS) é o complemento da probabilidade de LPS. O codificador aritmético assume que o MPS será codificado e atualiza RANGE com o valor de rMPS. Mas se o *bin* codificado é LPS, a variável LOW deve ser atualizada e RANGE recebe rLPS.

3 – Estimativa de probabilidade: O índice de probabilidades proveniente da memória de contextos (passo 1) é referente a uma tabela que contém um conjunto de valores para as probabilidades de LPS. A cada *bin* codificado, é necessário atualizar a probabilidade de ocorrência do próximo *bin*. A atualização de probabilidade consiste no seguinte conceito: se o codificador tem acertado o valor dos *bins* codificados anteriormente, a probabilidade de acerto do próximo *bin* aumenta, caso contrário, diminui. No CABAC, a atualização de probabilidade é realizada através da consulta de duas *lookup tables* (LUTs) indexadas pelo estado atual de codificação, onde o estado é o valor da probabilidade LPS e as LUTs representam sua regra de transição para atualizar

os índices de estados (DEPRÁ, 2009). Baseado na estimativa de probabilidade dada pelas LUTs é que o rLPS irá variar de valor, se o *bin* é MPS, a probabilidade de outro *bin* MPS ocorrer aumenta, senão a probabilidade diminui.

4 –Renormalização: Conforme os *bins* são processados, o intervalo necessário para representá-los torna-se cada vez menor (pois a probabilidade de ocorrência de um símbolo é sempre menor do que 1). A renormalização no CABAC consiste ampliar o intervalo, dobrando o valor das variáveis *LOW* e *RANGE*, emitindo parte da informação codificada para o *bitstream*. O processo é repetido até que o valor do intervalo esteja limitado em [256, 511]. Apenas na etapa de renormalização bits são transmitidos ao *bitstream*. No exemplo da Figura 3.10, para realizar o processamento do terceiro *bin*, a renormalização será executada.

5 – Atualização do modelo de contexto: A atualização de uma nova probabilidade deve ser realizada a cada *bin*. O valor de MPS também pode ser atualizado, se a probabilidade do *bin* codificado mudar de LPS para MPS ou vice-versa. Assim, todos os modelos de probabilidade utilizados pelo CABAC são modelos adaptativos, para os quais uma etapa de atualização de estatísticas e de MPS precisa ser realizada após a codificação de cada *bin.* O novo índice de probabilidade e o MPS atualizado devem ser escritos na memória de contextos, atualizando assim a estimativa de probabilidade para o *bin* com o número de contexto obtido no passo 1.

A Figura 3.11 apresenta um fluxograma simplificado contemplando os cinco passos do algoritmo de codificação aritmética binária presente no CABAC.

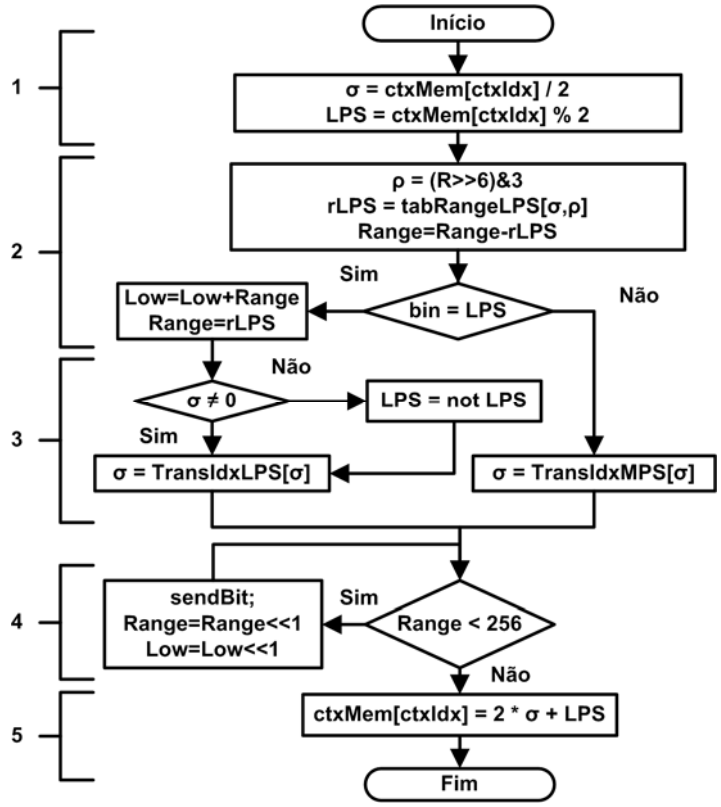

Figura 3.11: Fluxograma do algoritmo de codificação aritmética binária do CABAC.

O modo *regular* de codificação aritmética foi apresentado e é o que utiliza os modelos de probabilidades adaptativos para definir os limites dos valores de intervalo e deslocamento para o cálculo aritmético. Há também um modo especial para *bins* com

distribuição probabilística considerada pelo CABAC como uniforme. Esse modo de codificação aritmética é conhecido como *bypass*, o qual processa os *bins* que não precisam de modelagem de contexto, pois são considerados equiprováveis. Portanto este modo de codificação aritmética apresenta simplificações em relação ao *regular* acelerando a velocidade de codificação de *bins.* O intervalo deveria ser dividido em dois subintervalos iguais para realizar esta codificação, mas ao invés de reduzir explicitamente o intervalo *RANGE* pela metade, o valor da variável *LOW* é dobrado para eliminar a necessidade de atualização da variável *RANGE* e, consequentemente, algum passo de renormalização. Em seguida, o limite superior ou inferior é atualizado baseado no valor do símbolo codificado. A Figura 3.12 apresenta o diagrama de blocos para o motor *bypass*.

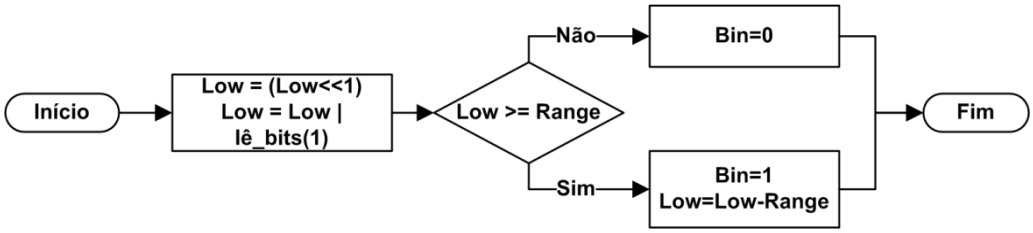

Figura 3.12: Fluxo de execução do motor de codificação *bypass*. (baseado em DEPRÁ, 2009)

Além do modo *bypass*, existe outro modo especial de codificação aritmética utilizado apenas para codificar a palavra de código que marca o final da sequência de codificação de um *slice* chamado de *terminate*. Ao modo *terminate* é associado um estado de probabilidade fixo, ou seja, não há leitura nem atualização de modelo de contexto neste modo de codificação. A Figura 3.13 ilustra o processo de codificação aritmética do motor *terminate.*

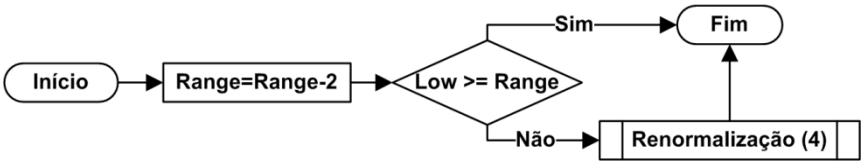

Figura 3.13: Fluxo de execução do motor de codificação *terminate.* (baseado em DEPRÁ, 2009)

O codificador aritmético binário não faz parte do escopo deste trabalho, mas compõe o conjunto de algoritmos do CABAC. Uma descrição mais aprofundada sobre o funcionamento do codificador aritmético binário pode ser encontrados em Deprá (2009) e Rosa (2010)

4 ARQUITETURA DESENVOLVIDA PARA O BCM DO CABAC

Este capítulo tem por objetivo descrever as arquiteturas de *hardware* desenvolvidas durante a realização deste trabalho. Em linhas gerais o funcionamento do subconjunto de ferramentas que compõem o CABAC foi apresentado ao longo do Capítulo 3, no qual a complexidade e extensão dos seus algoritmos foram discutidos. Foi dada uma ênfase maior para o Parser, o Binarizador e o Modelador de Contextos, constituintes do bloco denominado BCM (*Binarizing and Context Modeling* - Binarização e Modelagem de Contextos) e que compõem o conjunto de funcionalidades para as quais foram projetadas arquiteturas de *hardware* neste trabalho. A proposta arquitetural do CABAC (contendo o BCM) é exposta em conjunto com os requisitos de projeto do BCM e apresentado na Seção 4.1. Na Seção 4.2, o projeto completo da arquitetura do BCM é apresentado, contendo uma descrição detalhada das funcionalidades dos principais módulos, justificando as principais decisões de projeto tomadas durante o desenvolvimento. Na Seção 4.3, a metodologia para a validação do CABAC é descrita. A Seção 4.4 apresenta os resultados de síntese da arquitetura desenvolvida. A Seção 4.5 contém uma revisão dos principais trabalhos relacionados na literatura que propõem arquiteturas de *hardware* para o CABAC. Finalmente, a Seção 4.6 realiza uma série de comparações dos trabalhos relacionados com o trabalho desenvolvido nesta dissertação.

4.1 Proposta Arquitetural do CABAC

O desenvolvimento da arquitetura para o CABAC é dividido em duas partes: o BCM (Binarização e Modelagem de Contextos – *Binarizing and Context Modeling*) e o BAE (Codificador Aritmético Binário - *Binary Arithmetic Encoder*), como ilustrado na Figura 4.1. As funcionalidades do BCM, módulo desenvolvido neste trabalho e pontilhado na Figura 4.1, foram descritos detalhadamente nas Seções 3.2, 3.3 e 3.4. Os algoritmos do BAE foram descritos na Seção 3.5.

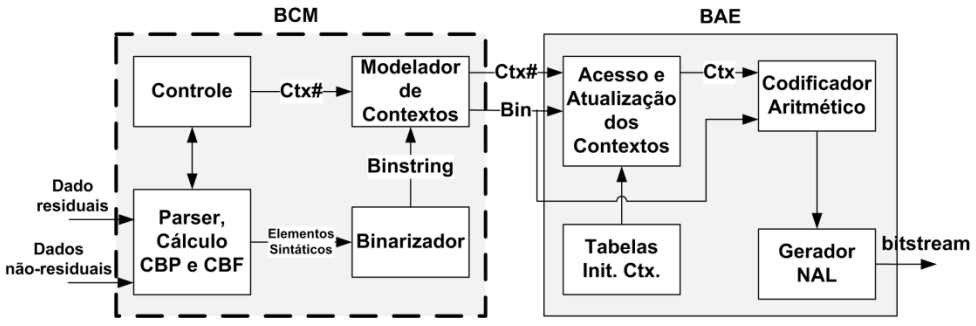

Figura 4.1: Proposta arquitetural para o codificador CABAC. (baseado em ROSA, 2010)

Os algoritmos definidos pelo CABAC são essencialmente sequenciais contendo dependências de dados intrínsecas dificultando a exploração de paralelismo. A taxa de execução do BAE é 0.6 *bin*/ciclo de acordo com Bingbo (2007). Diversas propostas arquiteturais para o BAE com taxas de processamento (bins/ciclo) variadas são encontradas na literatura. O trabalho desenvolvido por Rosa (2010) apresenta arquiteturas para o BAE com taxa de processamento que pode alcançar até 1 *bin*/ciclo. Esse trabalho também demonstra que para atingir taxas de processamento maiores que 1 *bin*/ciclo tem se mostrado uma solução extremamente complexa e ineficiente em área e energia. Logo, para o desenvolvimento do BCM, é assumido que o BAE tem taxa de processamento máxima de 1 *bin*/ciclo. Para alcançar o processamento de vídeos em alta resolução (ex. 1080p), assumindo que a taxa de processamento é 1*bin*/ciclo, é necessário que o CABAC atinja altas frequências de *clock*, pois CABAC processa dados bit-a-bit e também é preciso garantir o equilíbrio na taxa de processamento entre os módulos, pois o BCM a ser desenvolvido deve ter na sua saída uma taxa similar à taxa de consumo do BAE.

Além disso, outro fator que apóia essa decisão é o CABAC não apresentar dependências de dados no nível de *slice*, permitindo que múltiplos *slices* sejam processados com múltiplas instâncias do CABAC. Essa característica pode ser facilmente expandida e aplicada para a codificação de múltiplos quadros, suportando a mais nova extensão do H.264/AVC, o MVC (Codificação de Vídeo Multivista - *Multiview Video Coding*), o qual prevê a codificação de entre 2 até 100 vistas (DING, 2010). O CABAC é o principal codificador de entropia do H.264/MVC e para processar cada vista, uma nova instância de um módulo de *hardware* do CABAC pode ser utilizada sem qualquer tipo de alteração nos algoritmos (DING, 2010).

Nesse contexto, este trabalho propõe o desenvolvimento de arquiteturas de *hardware* para o BCM do H.264/AVC que com apenas uma instância (sem paralelismo em nível de *slice*) sejam capazes de processar vídeos em tempo real no perfil *Main* de resolução 1080p (1920x1080 *pixels*), visando atingir grande eficiência no consumo de energia e na utilização dos recursos de *hardware* em relação aos trabalhos relacionados na literatura. Assim, propostas arquiteturais para o BAE podem ser encontradas em Rosa (2010), Wu (2009) e Chen (2007b), enquanto o projeto da arquitetura de *hardware* do BCM será apresentado a seguir.

4.2 Arquitetura desenvolvida para o BCM

A proposta arquitetural detalhada para o BCM da Figura 4.1 é detalhada na Figura 4.2. Primeiramente, as linhas tracejadas e as FIFOs na vertical representam barreiras temporais que dividem a arquitetura em três componentes principais, caracterizando um *pipeline* de três estágios. Nas FIFOs verticais são atribuídos dados que precisam de sincronização e nas linhas tracejadas verticais, os dados são armazenados em registradores, constituindo a barreira temporal.

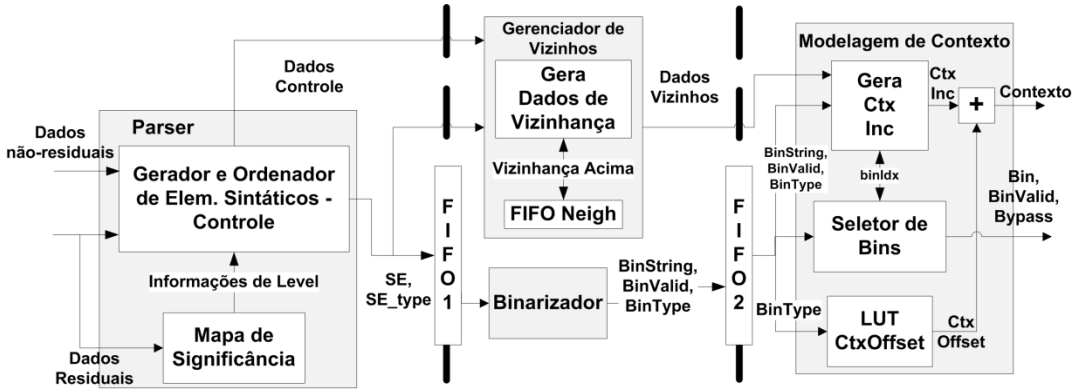

Figura 4.2: Proposta arquitetural para o codificador BCM.

No primeiro estágio de *pipeline* da arquitetura, está o *Parser* composto pelo *Controle* e pelo *Mapa de Significância*. Os dados provenientes dos outros módulos do codificador de vídeo (Figura 3.1) são recebidos pelo *Parser* que deve escolher quais as informações devem ser enviadas baseado nas decisões tomadas pelo codificador de vídeo. A entrada chamada *Dados não-residuais* constitui uma porta individual para cada informação não-residual tratada pelo CABAC, como as informações de predição e tipo de macrobloco. A entrada chamada de *Dados Residuais* é referente às amostras de resíduos, as quais necessitam passar pelo *Mapa de Significância* que, por sua vez, mapeia quais as amostras devem ser processadas pelo *Controle* e quais devem ser descartadas. O *Controle* também deve escolher, gerar e ordenar as informações que são enviadas ao próximo estágio de codificação, baseado nas decisões de tipo de macrobloco, *slice* e predição tomadas pelo codificador de vídeo, rotulando as informações na forma de Elementos Sintáticos (SE) e armazenando-as na *FIFO1*.

No segundo estágio de *pipeline*, encontram-se o *Binarizador* e o *Gerenciador de Vizinhos*. O *Binarizador* mapeia os SEs para valores binários, chamados de *binstring*, utilizando um dos métodos de binarização previstos de acordo com o tipo de SE (Seção 3.3). O *Gerenciador de Vizinhos* é um conjunto de registradores responsável por armazenar e gerenciar os dados de vizinhança necessários para o cálculo dos modelos de contexto (Figura 3.9). O *Binarizador* foi alvo de uma exploração do espaço de projeto, sendo propostas três diferentes arquiteturas para este módulo.

No terceiro estágio de *pipeline*, está a *Modelagem de Contextos*, composto pelo *Gera CtxInc*, *Seletor de Bins* e pelo *LUT CtxOffset*. O *Seletor de Bins* lê o *binString* de tamanho *binValid* da *FIFO2* e envia um *bin* por ciclo para o BAE processar. O *Contexto* de cada *bin* é composto pela soma de um incremento e um deslocamento (Equações 3.4 e 3.5). O deslocamento *CtxOffset* é calculado pelo *LUT CtxOffset* que considera o tipo de *bin* e a categoria do *bin*, caso este *bin* seja de um tipo residual. O *Gera CtxInc* calcula o incremento *CtxInc* baseado nos dados de vizinhança, índice e tipo do *bin* (Seção 3.4). Após a definição de *ctxOffset* e *ctxInc*, a soma desses dois sinais determina o número de contexto para o *bin* proveniente do *Seletor de Bins*.

Nas subseções seguintes, as arquiteturas de todos os módulos que constituem o BCM serão descritas detalhadamente, incluindo as três diferentes arquiteturas para o *Binarizador*.

4.2.1 Esquema para o Acesso Simplificado de Dados

Algumas informações enviadas pelos blocos de predição podem ser de um mesmo tipo, ocorrendo mais de uma vez por macrobloco, como os dados de predição intra 4x4, vetores de movimento e quadros de referência. Portanto, é necessário tomar uma decisão sobre a interface de entrada desses dados. Como a predição é altamente paralelizável, é assumido que todos os dados não-residuais estão prontos no início do processamento do CABAC, eliminando a necessidade de gerenciamento de vários pequenos *buffers*, mas trazendo a necessidade de criação de várias portas de entrada de um mesmo tipo de informação.

Devido à necessidade de armazenar vizinhos e às diferentes modalidades de vizinhança (Seção 3.4.2), a organização dos dados é decisiva numa solução arquitetural otimizada para o cálculo correto dos dados de vizinhança. Os dados para os SEs *ref_idx*, *mvd, coded_block_pattern* e *coded_block_flag* são organizados em matrizes conforme a numeração apresentada na Figura 3.9-b (*ref_idx* e *coded_block_pattern*) e Figura 3.9-c (*mvd* e *coded_block_flag*), facilitando o gerenciamento dos dados pelo BCM.

Contudo, no caso do *mvd*, para esta organização de dados funcionar efetivamente é preciso considerar que os *mvd* podem ocorrer entre uma e dezesseis vezes por macrobloco e cada *mvd* pode estar contido em blocos de tamanho entre 4x4 e 16x16 *pixels* (Seções 3.2.2 e 3.4.2). Cada elemento da matriz de *mvd* representa um bloco 4x4, pois no pior caso de ocorrência de *mvd* há 16 vetores distribuídos em 16 blocos 4x4. Blocos com tamanhos maiores que 4x4 são representados na matriz de *mvd* através de combinações de mais de um elemento da matriz *mvd*. Os SEs *mb_type* e *sub_mb_type* determinam a configuração geométrica da partição de blocos utilizada pela predição interquadros e são utilizados para controlar os valores na matriz. Esta organização de dados foi pensada para facilitar o acesso à informação de vizinhança. No exemplo da Figura 4.3, a matriz de *mvd* está em branco e a informação necessária para o cálculo de vizinhos está destacada em cinza.

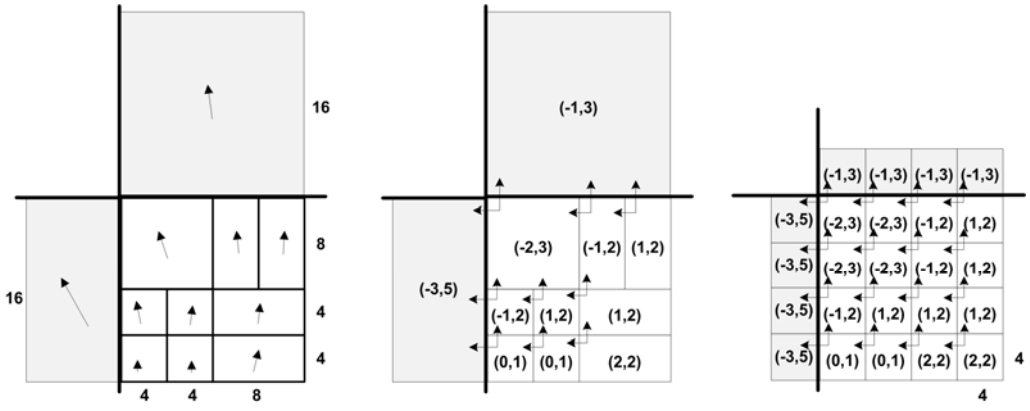

Figura 4.3: Exemplo de mapeamento em matriz dos vetores de movimento

O SE CBF representa um bloco de amostras 4x4 ou 2x2 (Croma DC) de determinada categoria (Luma DC, Luma, Croma AC...), conforme apresentado na Seção 3.2.3. Dependendo da categoria do bloco, CBF tem diferentes processos de derivação de vizinhança. Além disso, é preciso considerar o ordenamento duplo-Z dos blocos 4x4 e o mapeamento de amostras dado pelo SE CBP (Seção 3.2.3) para o correto ordenamento de SEs e mapeamento de vizinhos. Na arquitetura proposta, todas as informações que caracterizam cada CBF são obtidas a partir da categoria do bloco, do número do bloco 8x8 (exceto blocos DC), o número do bloco 4x4 (exceto blocos DC) e a cor do bloco de

Croma, se for o caso (Figura 4.4). Todas essas variáveis são calculadas no *Parser* e, como CBF é uma informação de um bit e a binarização não altera a largura de bits de CBF, os dados necessários para o processamento de CBF são transmitidos aos demais estágios de codificação utilizando a largura de dados restante disponível das FIFOs.

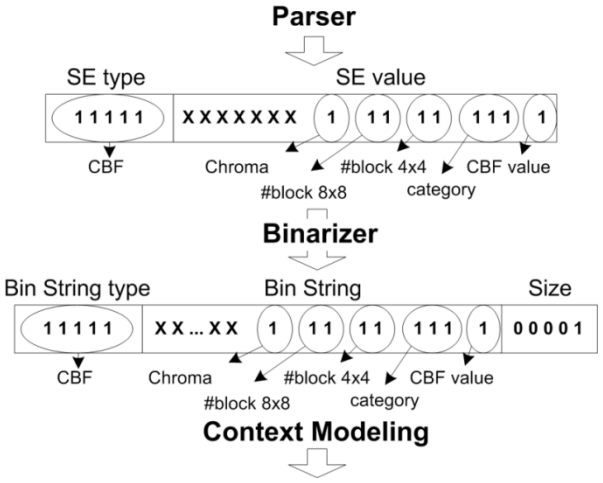

Figura 4.4: Transmissão dos dados de CBF através das FIFOs

Para os dados de resíduos existe um *buffer* na entrada do BCM com largura suficiente para armazenar 16 amostras em uma linha. Com isso, a cada requisição de dados neste *buffer*, um bloco 4x4 com 16 amostras é disponibilizado na entrada do BCM. Cada amostra não-residual tem 16 bits e a entrada para os dados residuais (assim como o *buffer*) tem largura de 256 bits. A definição do *buffer* de entrada foi baseada na decisão apresentada em Ramos (2010).

4.2.2 Arquitetura do Parser

Na Figura 4.5 está ilustrado o fluxo completo de codificação dos SEs da arquitetura desenvolvida para o *Parser*. Os processos destacados em branco geram SEs para o próximo passo de codificação, enquanto que os processos destacados em cinza apenas gerenciam sinais de controle. A sequência de blocos entre *Mb_skip_flag* e *Mb_qp_delta* realiza a codificação de SEs não-residuais. No início da codificação de resíduos, o bloco cinza *New Block 4x4* requisita de um bloco 4x4 de amostras que é categorizado na etapa seguinte, para então passar pelo processo de *Mapa de Significância*. A penúltima etapa testa o fim do bloco de amostras e, consequentemente, o fim do macrobloco, enquanto a última etapa testa se o macrobloco processado é o último do *slice* e envia um sinal ao "Gerenciador de Vizinhos" para atualização dos dados de vizinhança.

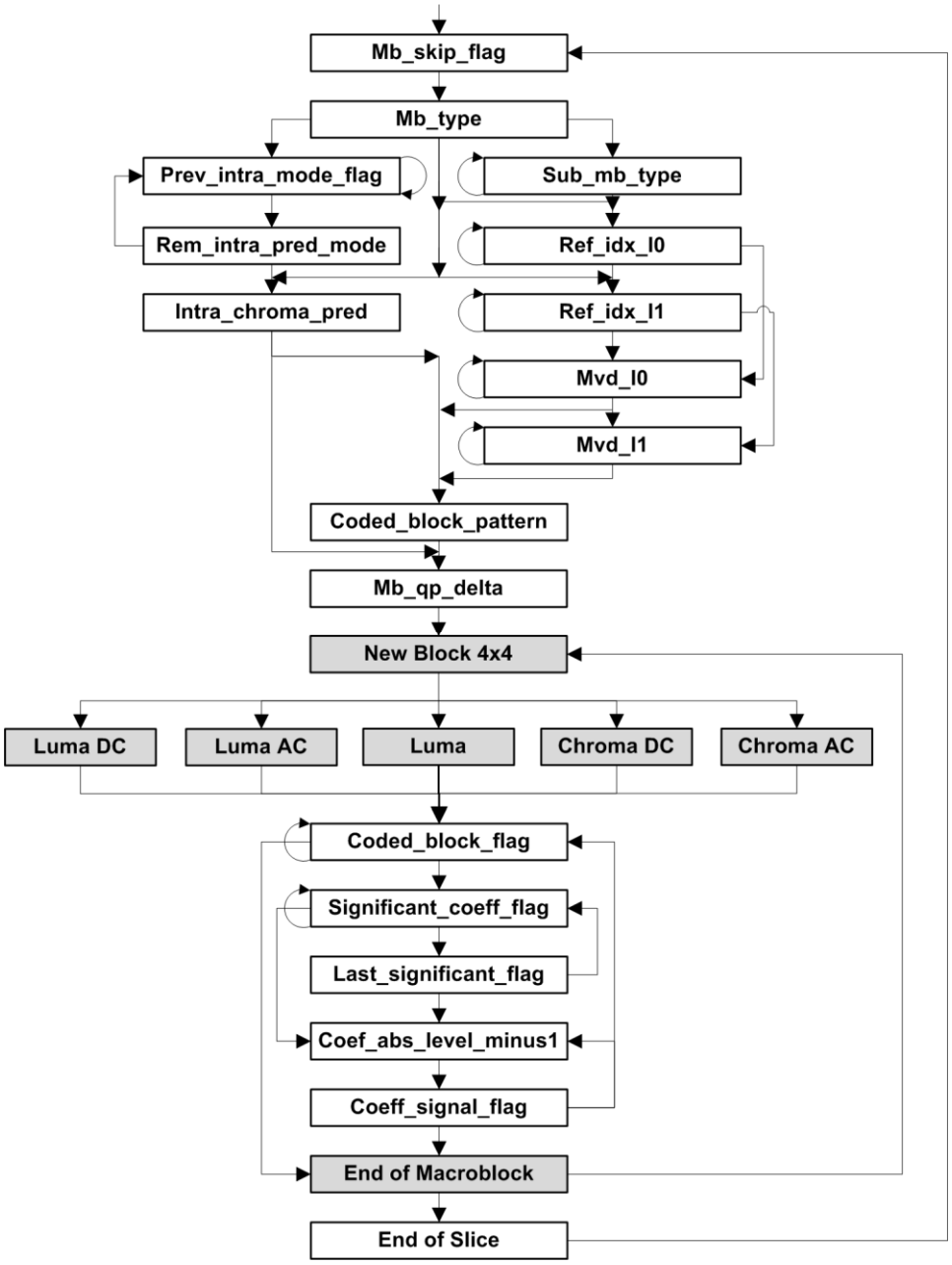

Figura 4.5: Fluxo de Codificação da arquitetura do *Parser*

Na Figura 4.6 estão apresentados os principais módulos arquiteturais do *hardware* do *Parser* da Figura 4.5, incluindo a interface da arquitetura. Todos os SEs da Figura 4.5 são enviados serialmente para FIFO 1 através das portas SE_value e SE_type. O *Parser* contém dois módulos que gerenciam o envio dos dados de predição chamados de Escalonador de SEs Intra4x4, responsável por controlar os SEs *prev_intra4x4_pred_mode* e *rem_intra4x4_pred_mode*, e de Escalonador de Ref_Idx MVD, o qual realiza o acesso aos dados da predição interquadros de acordo com esquema apresentado na Seção 4.2.1.

Para o gerenciamento dos resíduos, o *Parser* contém o Contador de Blocos e Amostras, responsável por realizar a contagem dos blocos 8x8, 4x4 e das amostras. A contagem de blocos 8x8 e 4x4 utiliza o mesmo padrão apresentado na Seção 3.2.3 considerando o SE *coded_block_pattern* e a categoria do bloco corrente. A contagem de

amostras é realizada de acordo com o mapeamento dado pelo módulo Mapa de Significância. O Mapa de Significância gera 48 SEs, um *significant_coeff_flag*, um *last_significant_coeff_flag* e um *coeff_signal_flag* para cada amostra do bloco 4x4 corrente, enquanto o controle decide quais SEs devem ser transmitidos de acordo com o algoritmo apresentado na Figura 3.7. O Gerador de CBF organiza os SEs *coded_block_flag* considerando a contagem dos blocos 8x8 e 4x4 e a categoria do bloco e envia esses SEs para o Gerenciador de Vizinhos através das portas CBF_array, CBF_LumaDC e CBF_ChromaDC.

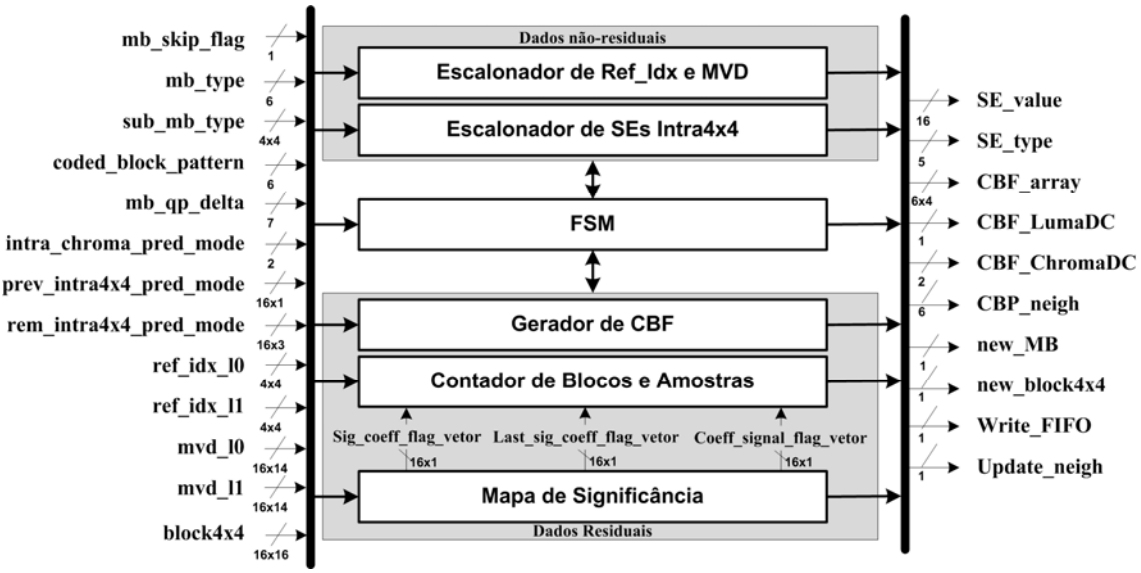

Figura 4.6: Principais Módulos e Interface da Arquitetura do *Parser*

4.2.3 Arquiteturas desenvolvidas para o Binarizador

O binarizador é o estágio de codificação responsável por mapear os SEs para cadeias binárias chamadas *binstrings*. O próximo estágio da codificação do BCM é a modelagem de contexto que processa os *bins* do *binstring* individualmente. Assim, a binarização não é considerada um dos gargalos do sistema do CABAC e, por isso, os projetos propostos para o binarizador na literatura são incompletos e/ou ineficientes. Na literatura há vários trabalhos que visam desenvolver arquiteturas de *hardware* para o CABAC, porém são poucos os trabalhos que implementam o binarizador com todas as suas funcionalidades. No entanto, o trabalho de Osório (2004) mostra que o binarizador pode utilizar 52% da área do CABAC com *throughput* de até 1*bin*/ciclo e, numa extensão deste trabalho de 2004, o trabalho de Osório (2006) apresenta o mesmo binarizador ocupando 33.4% dos recursos de *hardware* para uma arquitetura de CABAC que processa 2 *bins*/ciclo. Portanto, não há razão para acelerar o binarizador, mas há um grande potencial para economizar recursos de *hardware* neste módulo.

4.2.3.1 Processo de Binarização Unário/Exp-Golomb Parametrizável

Para obter uma redução do percentual de área do binarizador o comportamento do *binstring* gerado pelo processo de binarização Unário/Exp-Golomb Parametrizável (UEGk) apresentado na Seção 3.3.2 foi analisado com o objetivo de otimizar e/ou substituir o algoritmo que gera o *binstring* de UEGk, o qual é o método de binarização que se destaca em complexidade em relação aos demais métodos.

Analisando o *binstring* de UEGk, foi identificado que a concatenação do prefixo da binarização UEGk construído pelo método Truncado Unário (TU), o prefixo e o zero (‘0’) do meio do código Exp-Golomb Parametrizável (Equações 3.1 e 3.2) formam o mesmo padrão de *binstring* dado pela binarização Unária (U), como destacada, em pontilhado, a Tabela 4.1. Consequentemente, destacado em linha contínua na Tabela 4.1, o sufixo do código Exp-Golomb Parametrizável pode ser dado pela binarização Tamanho Fixo (FL – *Fixed Length*).

Tabela 4.1: Exemplos de binarização utilizando o método Exp-Golomb Parametrizável

SE	**Truncado Unário** (uCoff=14)	**Exp-Golomb parametrizável** (k=0)		
		Prefixo		**Sufixo**
0	0			
1	10			
...	...	...	...	...
13	11111111111110			
14	11111111111111		0	
15	11111111111111	1	0	0
16	**11111111111111**	**1**	**0**	**1**
17	**11111111111111**	**11**	**0**	**00**
18	11111111111111	11	0	01
...	...	...	...	...

SE	**Unário**	**Tamanho Fixo**
0	0	
1	10	
...	...	...
13	11111111111110	
14	11111111111111 0	
15	11111111111111 1 0	0
16	**11111111111111 1 0**	**1**
17	**11111111111111 11 0**	**00**
18	11111111111111 11 0	01
...	...	...

Assim, através da reutilização de métodos de binarização já existentes, um novo equacionamento é proposto para gerar o mesmo *binstring* produzido pela binarização UEGk. O prefixo do *binstring*, apresentado na Tabela 4.1, é formado pelo método U e recebe como parâmetro para a binarização o valor do SE quando o valor do SE for menor que o parâmetro *uCoff*. Neste caso (*SE* < *uCoff*), o sufixo não existe. Caso contrário, quando o valor do SE for maior ou igual à *uCoff*, então o prefixo é determinado pela soma de *k*, *uCoff* e a largura do sufixo binarizado. O sufixo do *binstring*, dado em 4.2, é codificado com o método de binarização FL e seu valor depende do SE de entrada, *uCoff* e *k*. O parâmetro *lMax*, dado em 4.3, determina o comprimento do *binstring*. O *binstring* final do método UEGk corresponde à concatenação do prefixo e do sufixo (quando este existir).

$$prefixo(SE) = \begin{cases} U(SE), & SE \leq uCoff \\ U(uCoff + size(sufixo) - k - 1), & SE > uCoff \end{cases} \quad (4.1)$$

$$sufixo(SE) = FL(2^k + SE - uCoff) \quad (4.2)$$

$$lMax_{sufixo} = floor(\log_2(sufixo)) = size(sufixo) \quad (4.3)$$

Onde:

SE é o elemento sintático ou entrada do *EGk*;

uCoff é o coeficiente que determina se o *binstring* deve ter sufixo;

k é o parâmetro que determina o tamanho mínimo do sufixo;

U() é o *binstring* gerado pelo método de binarização Unário;

FL() é o *binstring* gerado pelo método de binarização Tamanho Fixo;

$lMax_{sufixo}$ é o tamanho do *binstring* gerado pelo método de binarização *FL()*.

4.2.3.2 Arquiteturas desenvolvidas para o Binarizador

Para desenvolver o binarizador utilizando as melhorias do algoritmo UEGk, focando na redução dos recursos de *hardware*, uma arquitetura multiciclo é proposta por permitir a reutilização de alguns operadores. A arquitetura multiciclo do binarizador desenvolvida é apresentada na Figura 4.7 e suporta todos os métodos de binarização. O módulo chamado LOC é um Contador de Uns a Frente (*Lead One Counter*) e calcula o tamanho do *binstring* gerado pelo método U. Os registradores *prefix* e *suffix* são utilizados para calcular *binstrings* que precisam de duas palavras de código, mas esses registradores também são utilizados como variáveis auxiliares. O registrador *size* determina o tamanho do *binstring*. Cinco LUTs são utilizadas para mapear *binstrings* com construção irregular, gerados a partir dos SEs *mb_type* e *sub_mb_type*. O parâmetro *cMax* é determinado pelo controle para SEs que utilizam binarização TU, FL e *coded block pattern* (CBP). O módulo *Unary* é utilizado para realizar os processos de binarização U, TU, UEGk e variação do parâmetro de quantização (QPd). O *shifter* é utilizado na binarização TU.

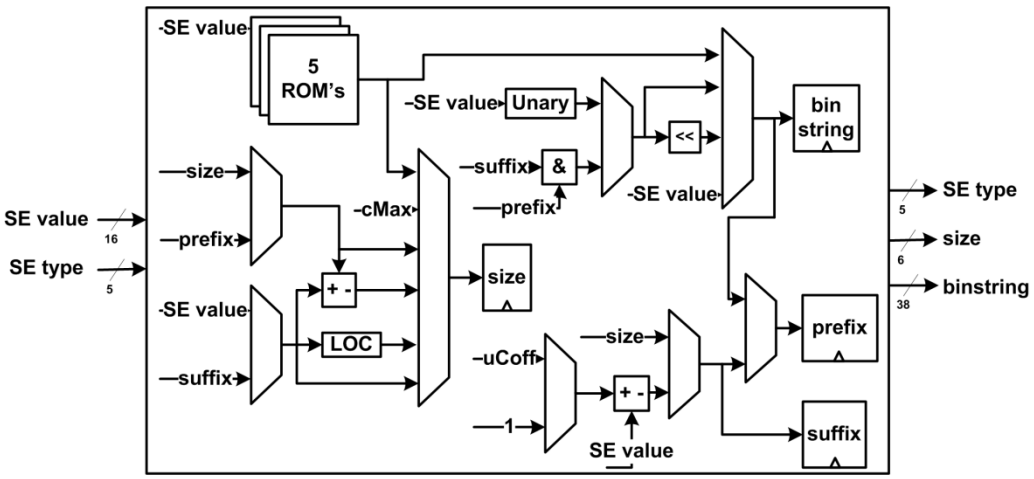

Figura 4.7: Arquitetura multiciclo para o binarizador.

Com o objetivo de comparar quais as reais melhorias atingidas pela arquitetura multiciclo que utiliza a técnica apresentada na Seção 4.3.2.1, foi desenvolvida uma arquitetura ciclo-único sem a otimização da binarização UEGk. A arquitetura ciclo-único, ilustrada na Figura 4.8, também suporta todos os métodos de binarização. O módulo denominado "&" concatena o *binstring* de dois métodos de binarização quando este é formado por prefixo e sufixo. O módulo Kth Exp-Golomb (ausente na arquitetura multiciclo) é um codificador que calcula o sufixo da binarização UEGk. O módulo QPd mapeia os SEs para valores sem sinal que podem ser posteriormente binarizados com o método U.

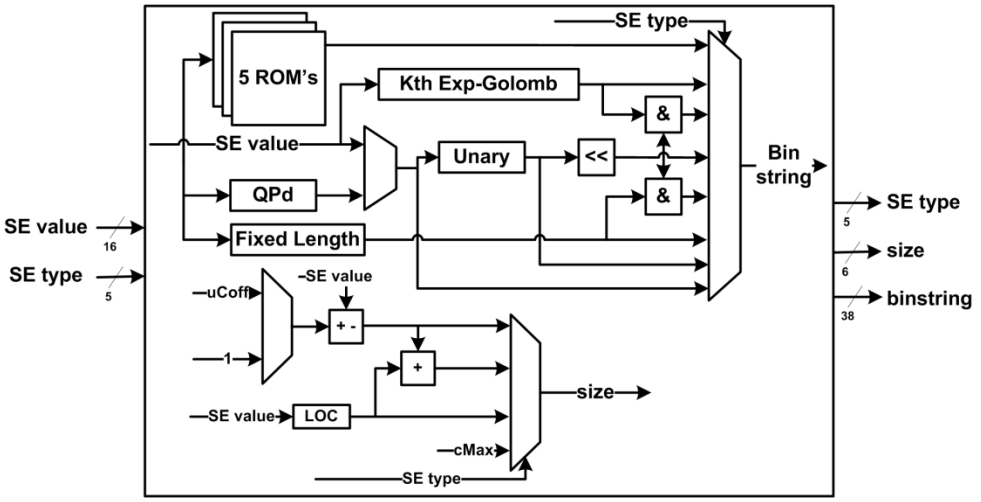

Figura 4.8: Arquitetura ciclo-único para o binarizador.

Finalmente, uma terceira arquitetura para o binarizador, chamada de *context-aware*, é proposta (Figura 4.9) com o objetivo de facilitar a modelagem de contextos, próximo estágio de codificação do BCM. Sabendo que os *binstrings* formados por prefixo e sufixo têm modelagem de contextos diferenciada (Seção 3.4), é vantajoso que as duas partes destes *binstrings* sejam enviadas separadamente e devidamente identificadas, facilitando o controle da modelagem de contextos. A principal diferença desta arquitetura para a ciclo-único é um circuito adicional que identifica o tipo de *binstring*, quando este for sufixo. Assim, este binarizador leva dois ciclos para processar SEs que geram *bins* formados por prefixo e sufixo e um ciclo para os demais *bins*.

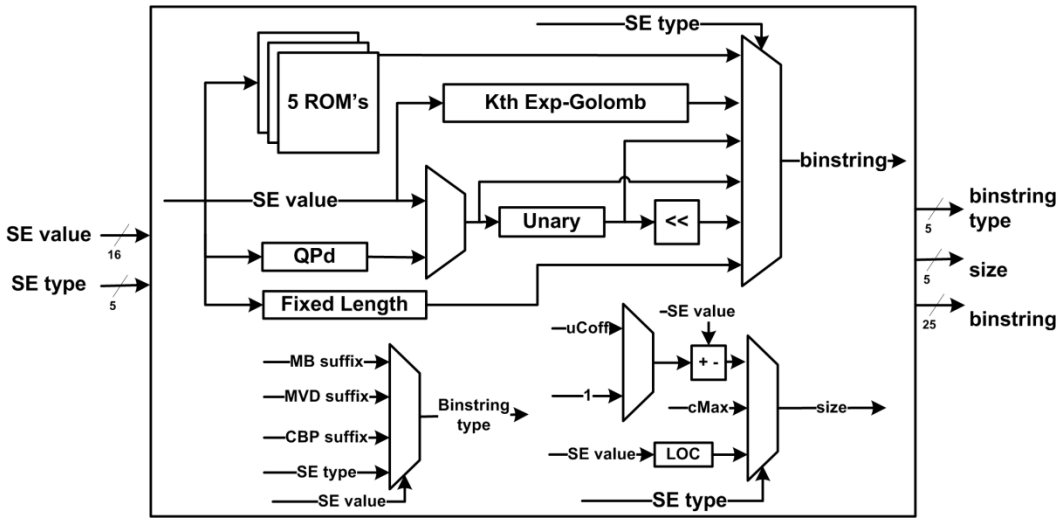

Figura 4.9: Arquitetura *context-aware* para o binarizador.

A arquitetura multiciclo executa a binarização FL em um ciclo, enquanto que os outros métodos precisam de dois ciclos para finalizar. As binarizações QPd e UEGk podem levar até três ciclos para serem concluídas. Uma análise de *bitstream* para sete sequências de vídeo HD1080p, apresentada em Deprá (2009), é utilizada para verificar se esta arquitetura está criando um novo gargalo de processamento. Os SEs são agrupados de acordo com o número de ciclos que cada SE necessita para executar neste projeto. Foi assumido o pior caso para todas as ocorrências de binarização UEGk e QPd. Os resultados obtidos estão mostrados na Tabela 4.2. A taxa de SE na terceira

coluna é a porcentagem de SE agrupados na segunda coluna em relação a todos os SE enviados para processamento do binarizador. A quarta coluna está a porcentagem de *bins* produzidos para cada tipo de SE em relação ao total de *bins* produzidos pelo binarizador.

Tabela 4.2: Avaliação de taxa de processamento do binarizador multiciclo para vídeos 1080p

Método de Binarização	**Ciclos**	**SE (%)**	**BINS (%)**
FL	1	58,22	30,38
CBP, U, TU, MB&SUBMB	2	1,46	2,36
UEGk, QPd	3	40,32	67,26

Com o objetivo de comparar as arquiteturas desenvolvidas com o trabalho de Osório (2006), as arquiteturas descritas em VHDL foram sintetizadas para o dispositivo FPGA Virtex II XC2VP30 (VIRTEX, 2010). A Tabela 4.3 resume os resultados obtidos. Os resultados mostram que a arquitetura de Osório (2006) tem o melhor desempenho, mas não suporta todos os métodos de binarização. A arquitetura multiciclo atingiu uma redução área de 58,3% em relação à arquitetura ciclo único e 47,3% em relação à arquitetura de Osório (2006). A taxa típica de compressão no H.264/AVC do codificador aritmético é 1 bit para cada 1.3 *bins* produzidos (CHEN, 2007). Baseados nos resultados de *throughput* calculados a partir da Tabela 4.2, o binarizador multiciclo atinge a taxa de processamento de 103,9 Mbins/s, sendo suficiente para processar dados no nível 4.2 da especificação do H.264/AVC que define neste nível um taxa de bits de 50 Mbits/s ou 65 Mbins/s.

Tabela 4.3: Resultados de síntese e comparação de resultados das arquiteturas de binarizador

Binarizador	**Funções**	**Área**	***Throughput***	**Frequência**
OSORIO, 2006	UEGk e U	Slices: 403	2 *binstring*/ciclo	185 MHz
Ciclo-único	Full- binarizer	Slices: 509	1 *binstring* /ciclo	142.4 MHz
Context-aware	Full- binarizer	Slices: 311	0.83 *binstring* /ciclo	250.9 MHz
Multiciclo	Full- binarizer	Slices: 212	0.42 *binstring* /ciclo	247.5 MHz

As modificações no algoritmo UEGk implementadas na arquitetura multiciclo surtiram grande efeito na redução dos recursos de *hardware*, pelo fato de eliminar o algoritmo mais complexo através da substituição de algoritmos já existentes na aplicação do binarizador e, ainda, sem criar um novo gargalo no processamento dos dados. Além disso, a arquitetura *context-aware* é uma boa alternativa no consumo de área, pois a largura de dados e operadores torna-se menor à medida que não existe a necessidade de concatenação de prefixo e sufixo para o envio do *binstring*. Essa largura menor dos dados irá refletir na largura da FIFO de saída do binarizador. Além disso, a arquitetura *context-aware* facilita a modelagem de contextos e, por isso, esta é a arquitetura escolhida para compor a arquitetura global do BCM.

Uma versão extendida das arquiteturas desenvolvidas para binarização foram publicadas e estão disponíveis em Martins (2010)

4.2.4 Arquitetura desenvolvida para o Gerenciador de Vizinhos

O Gerenciador de Vizinhos é executado em paralelo ao Binarizador na arquitetura do BCM sendo um módulo auxiliar na modelagem de contexto. Enquanto o Binarizador fornece o *binstring* para a Modelagem de Contextos, o Gerenciador de Vizinhos

disponibiliza toda a informação de vizinhança necessária para o cálculo de contexto de *bins*. O sinal para a atualização dos dados de vizinhança é fornecido pelo *Parser* ao codificar o SE *end_of_slice* marcando o fim do macrobloco.

A Figura 4.10 apresenta os principais módulos que compõem o Gerenciador de Vizinhos. Um conjunto de registradores armazena a vizinhança à esquerda, atualizada a todo novo macrobloco. Uma FIFO armazena a informação dos vizinhos acima. O Gerenciador de Bordas verifica se os macroblocos vizinhos acima e à esquerda estão ou não disponíveis e controla a leitura e a escrita da FIFO de Vizinhos Acima.

Figura 4.10: Principais Módulos e Interface da Arquitetura do Gerenciamento de Vizinhos

Considerando os SEs que precisam ter seu valor armazenado, a cada macrobloco são guardados 4 *mvd*, 4 CBF, 2 *ref_idx* e 1 dos demais SEs ilustrados na Figura 4.10, constituindo uma largura de dados de 152 bits para o conjunto de registradores e a FIFO. A largura de bits dos dados dos SEs pode ser observada na interface de entrada da Figura 4.6 e foi definida a partir do intervalo de valores possíveis para cada tipo de dado disponível na norma que define o padrão H.264/AVC (INTERNATIONAL, 2009), exceto para os quadros de referência (*ref_idx*) e vetores de movimento (*mvd*). De acordo com experimentos apresentados em Tian (2007), *ref_idx* e *mvd* podem ser representados com 4 e 7 bits, respectivamente, sem perda significativa na representação dos dados.

A numeração dos macroblocos da Figura 4.11 representa a ordem de processamento de macroblocos. Quando a borda de cima está sendo processada, a informação dos macroblocos vai sendo salva na FIFO. Quando o macrobloco não pertence às bordas de cima ou abaixo, as operações de leitura e escrita são executadas simultâneamente na FIFO. Quando a borda de baixo é processada, a informação dos vizinhos é lida da FIFO e nada é escrito até o fim do *slice*. Na Figura 4.11, os macroblocos em branco estão armazenados na FIFO e o macrobloco atual irá assumir o lugar de seu vizinho acima na FIFO ao final do processamento.

Figura 4.11: Esquema do Gerenciamento de Vizinhos

4.2.5 Arquitetura desenvolvida para a Modelagem de Contextos

A Modelagem de Contextos é o último estágio de processamento da arquitetura do BCM. A Modelagem de Contextos deve ler uma *string* de *bins* e atribuir a cada *bin* um número de contexto. O número do contexto (Ctx) é formado pela soma de um deslocamento (CtxOffset) e um incremento (CtxInc). O deslocamento depende apenas

do tipo do *binstring*, e da categoria caso o binstring seja de um tipo de informação de resíduo. O incremento é calculado individualmente para cada *bin* de acordo com o índice do *bin* no *binstring* e do tipo do *binstring* (Seção 3.4). A arquitetura da Modelagem de Contextos está ilustrada na Figura 4.2.

Para atingir o requisito de alta frequência determinado na especificação da arquitetura do BCM, as operações de cálculo de deslocamento, incremento e contexto foram individualizadas. O processamento dos *bins* é feito em quatro etapas distintas. Primeiro o *binstring* é lido da FIFO. Em seguida, há o cálculo do deslocamento paralelamente ao incremento do primeiro *bin* do *binstring*. Na terceira etapa, ocorre a soma do deslocamento e do incremento, formando o número de contexto do primeiro *bin,* concomitantemente com o cálculo do incremento do segundo *bin*. A terceira etapa é repetida até a identificação do último *bin*. Na quarta e última etapa ocorre o cálculo do número de contexto do último *bin.* O Seletor de Bins monitora se o *bin* que está sendo processado é o último *bin* do *binstring*, se o primeiro *bin* também for o último bin, a terceira etapa não acontece.

O cálculo do contexto dividido em quatro etapas não garante que a arquitetura do BCM alcance uma taxa de processamento de 1 *bin*/ciclo portanto, foi necessário implementar um esquema de *pipeline* de três estágios, ilustrado na Figura 4.12. Cada estágio de *pipeline* processa um *binstring* diferente executando as quatro etapas supracitadas. Apenas um *bin* de um dos três *binstrings* é enviada por ciclo. A primeira, a segunda e a quarta etapa são executadas em paralelo para três *binstrings* distintos. Quando a terceira etapa precisa ser executada, os outros estágios ficam em estado de espera. Os blocos de cálculo de contexto destacados em cinza na linha temporal de processamento ilustram o *throughput* esperado de 1 *bin*/ciclo.

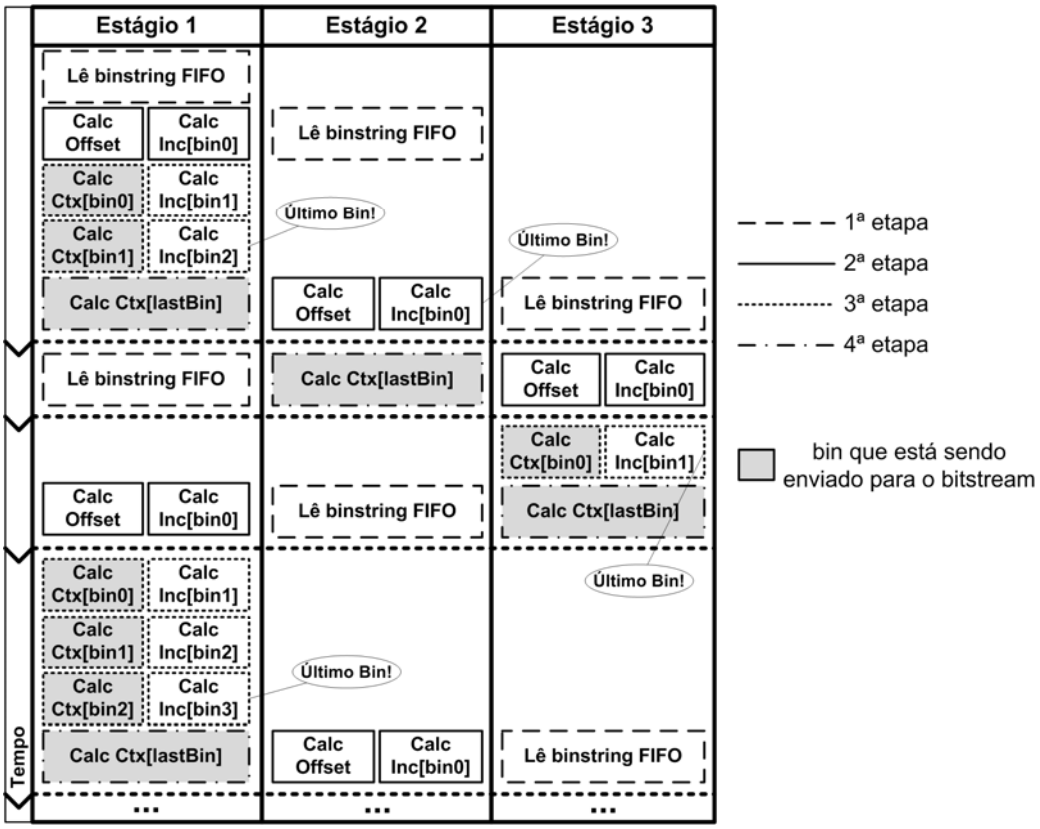

Figura 4.12: *Pipeline* do Modelador de Contextos: garantia de 1 *bin*/ciclo

4.3 Metodologia de Validação do CABAC

A validação do BCM foi dividida em duas fases de acordo com uma metodologia *bottom-up*. Na primeira fase, o *Parser*, o Binarizador, o Gerenciador de Vizinhos e o Modelador de Contextos são validados individualmente, como se fossem o topo da arquitetura. O primeiro módulo a ser validado é o *Parser*, pois é o primeiro estágio de codificação. Em seguida, o Binarizador e o Gerenciador de Vizinhos podem ser validados concomitantemente e, finalmente, a Modelagem de Contextos é verificada. A ordem de verificação é importante, pois, por exemplo, os mesmo vetores gerados na saída do *Parser* são utilizados, após verificados, na entrada do Binarizador e do Gerenciador de Vizinhos. Após o término da primeira fase com a validação da Modelagem de Contextos, ainda não é possível garantir que o BCM funciona perfeitamente. A segunda fase consiste em verificar a arquitetura do BCM integrada. Nesta fase, os problemas de sincronização, como sinais de *handshaking* e problemas de temporização, são verificados e dados importantes da arquitetura, como o tamanho das FIFOs, são extraídos com precisão nesta fase.

O processo de validação inicia a partir da escolha do software em que serão extraídos os vetores de teste. O *software* de referência (SUHRING, 2009) é um *software* oficial do padrão H.264/AVC disponível na internet com código fonte aberto e uma implementação sabidamente correta do padrão servindo como guia para o desenvolvimento de outras aplicações que utilizam o padrão H.264/AVC. Devido à enorme dificuldade que desenvolvedores de *hardware* encontram para extrair os vetores de teste, o Laboratório de Processamento de Sinais e Imagens (LaPSI) da UFRGS desenvolveu o *software* PRH264, baseado no *software* de referência. Entretanto, o PRH264 também não se mostrou uma boa alternativa, apesar da estrutura modular e interface amigável, pois este *software* implementa apenas o decodificador H264/AVC e o BCM é um módulo do codificador. Finalmente, o software x264 foi avaliado como alternativa para extração dos vetores de teste e, devido à sua modularidade, clareza de código e suporte as funções de codificação, este é o software escolhido para a extração dos vetores de teste. O x264 é um *software* de código aberto para codificação de vídeo H.264/AVC e está disponível em X264 (2010) em conjunto com um guia de intruções de uso e configurações, disponível em X264 (2010b).

O x264 não é um software feito para o desenvolvimento de aplicações, por isso não há qualquer interface ou *script* que gere arquivos *trace* para serem utilizados na construção de vetores de teste. Logo, toda a extração dos vetores de teste foi feita manualmente através da inserção de rotinas de escrita em arquivos em pontos de interesse do código original. A Figura 4.13 mostra um trecho do código do x264 com rotinas para a extração de vetores de teste.

```
293  static void x264_cabac_mb_intra4x4_pred_mode( x264_cabac_t *cb, int i_pred, int i_mode )
294  {
295      if( i_pred == i_mode ){
296          x264_cabac_encode_decision( cb, 68, 1 );
297          fprintf(fparserIn,"1 ");
298          fprintf(fbinIn,"SE_PREV_INTRA 1\n");
299          fprintf(fctxIn,"SE_PREV_INTRA 1 1\n");
300      }
301      else
302      {
303          x264_cabac_encode_decision( cb, 68, 0 );
304          fprintf(fbinIn,"SE_PREV_INTRA 0\n");
305          fprintf(fctxIn,"SE_PREV_INTRA 0 1\n");
306          if( i_mode > i_pred  )
307              i_mode--;
308          fprintf(fparserIn,"%d ", (i_mode<0 ? 1 : 2*i_mode));
309          fprintf(fbinIn,"SE_REM_INTRA %d\n", i_mode);
310          fprintf(fctxIn,"SE_REM_INTRA %d%d%d 3\n", (i_mode)&0x01, (i_mode >> 1)&0x01, (i_mode >> 2)&0x01);
311          x264_cabac_encode_decision( cb, 69, (i_mode    )&0x01 );
312          x264_cabac_encode_decision( cb, 69, (i_mode >> 1)&0x01 );
313          x264_cabac_encode_decision( cb, 69, (i_mode >> 2)&0x01 );
314      }
315  }
```

Figura 4.13: Extraindo Vetores de Teste no x264

Após a extração dos vetores de teste, os *testbenchs* são descritos utilizando a linguagem de descrição de *hardware* VHDL. Os *testbenchs* excitam o circuito com os vetores de entrada e armazenam a saída em arquivos texto. Os arquivos texto gerados pelo *testbench* devem ter a mesma organização dos *traces* extraídos pelo *software*, pois é através da comparação de arquivos que a verificação é feita. Diferentemente do software de referência, onde já existem dados de *trace* prontos e corretos, utilizando o x264 não há como certificar que os dados de traces foram extraídos corretamente ou se os dados estão organizados de acordo com a interface de entrada do RTL. Logo, é necessário avaliar a cada passo de verificação se o problema de incompatibilidade de arquivos reside no arquivo *trace*, na descrição do *testbench* ou na descrição do RTL em si. Este ambiente de validação está ilustrado na Figura 4.14.

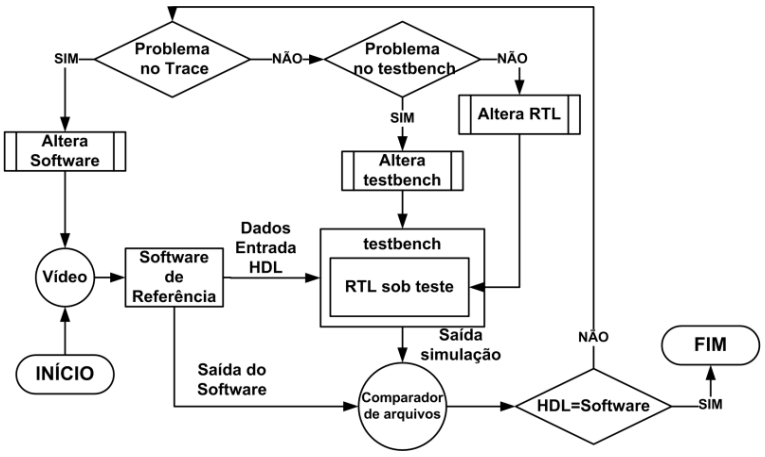

Figura 4.14: Ambiente de validação do BCM

Na Figura 4.15 está ilustrado à direita um trecho do arquivo *trace* de saída do *Parser* gerada pela simulação do *testbench*. Através da simulação dos *testbenchs* podem ser gerados alguns dados importantes da arquitetura. A ferramenta de simulação dos *testbenchs* utilizada neste trabalho é o ModelSim 6.3 (MODELSIM, 2010). Para comparar os arquivos gerados em simulação e os arquivos extraídos do software x264, utilizou-se o aplicativo WinMerge, disponível na internet para *download* em Winmerge (2010).

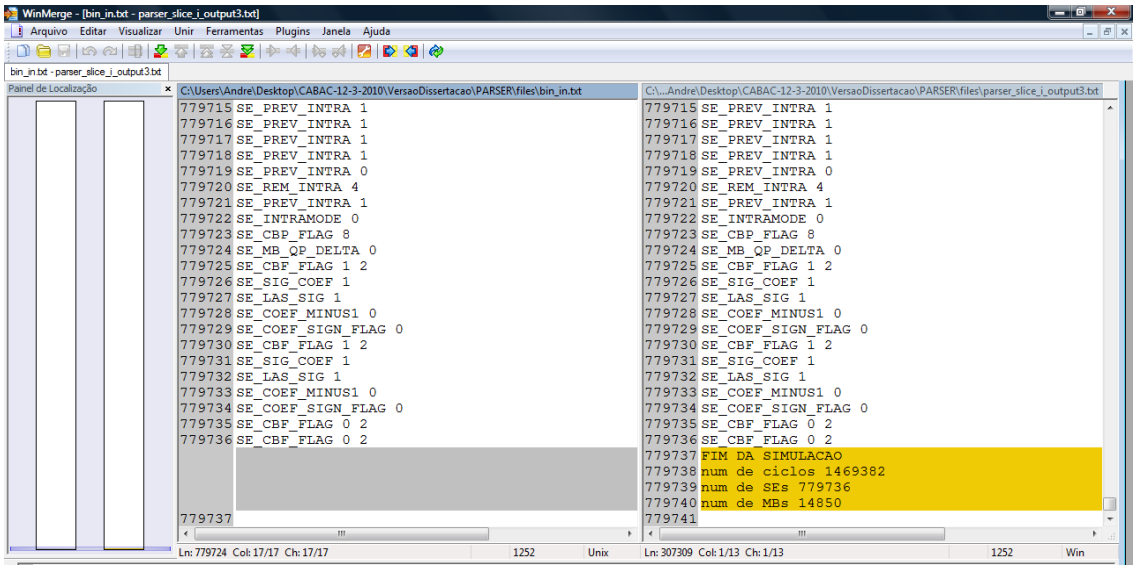

Figura 4.15: WinMerge apontando a diferença entre os arquivos do x264 e de simulação.

Até o momento da entrega do texto desta dissertação, a primeira fase de verificação do BCM para vídeos com *slice* I foi concluída, ou seja, o Parser, o Binarizador, o Gerenciador de Vizinhos e o Modelador de Contextos foram verificados individualmente. A segunda fase de validação, com o BCM integrado, está em andamento. Após o término da segunda fase, o BCM passará por um novo laço de validação para vídeos com *slice*s P e B.

4.4 Apresentação e Análise dos Resultados de Síntese

A arquitetura proposta neste trabalho foi desenvolvida para processar vídeos de resolução 1080p (1920x1080) rodando a 30 quadros/s no perfil *Main* do padrão H.264/AVC, ou seja, é necessário processar os dados de 244800 macroblocos por segundo. Como as arquiteturas de Parser e do Binarizador são multiciclo, para verificar a frequência mínima a ser atingida pela arquitetura para alcançar processamento em tempo real, foram retirados arquivos *traces* de três sequências de vídeo 1080p (BlueSky, Sunflower e RushHour) para simulação no BCM. A contagem do número de ciclos é, na média, aproximadamente de 50 milhões para processar um segundo de vídeo 1080p no perfil *Main*. Considerando apenas o BCM, uma frequência de 50MHz é suficiente para processar vídeos 1080p.

Todos os módulos desenvolvidos estão descritos em VHDL e uma síntese para o dispositivo FPGA Virtex V xc5vlx110t (VIRTEX5, 2010) foi realizada utilizando a ferramenta de síntese ISE 10.1 (ISE, 2010). As memórias FIFOs foram criadas a partir da utilização do *software* CORE Generator (COREGENERATOR, 2010) fornecido junto com o ISE 10.1. No dispositivo alvo desta síntese, um bloco de memória RAM tem 18Kbits (VIRTEX5, 2010). O binarizador utilizado é a versão *context-aware*. Os resultados da síntese FPGA estão na Tabela 4.4. Com o intuito de facilitar futuras comparações com trabalhos futuros e permitir uma avaliação de proporcionalidade na área ocupada, foi feita uma síntese individual dos módulos Parser, Gerenciador de Vizinhos, Binarizador e Modelador de Contextos e os resultados também estão na Tabela 4.4.

Tabela 4.4: Resultados de síntese FPGA da arquitetura do BCM

	Slice Registers	**Slice LUTs**	**LUT-FF pairs**	**BRAMs**	**Frequência**	**MB/s**	**Bins/s**
Parser	397	1140	1151	-	277 MHz	-	-
Ger. Vizinhos	224	1039	1073	5	360 MHz	-	-
Binarizador	90	389	390	-	350 MHz	-	-
Mod. Ctx.	218	920	1019	-	229 MHz	-	-
BCM	**1221**	**3258**	**3506**	**7**	**216 MHz**	**~979 K**	**~216 M**

A frequência de operação atingida foi de 216 MHz, atingindo o desempenho necessário para processar vídeos no nível 5 do padrão H.264/AVC. O BCM ocupou 5% dos recursos de *hardware* da FPGA, onde 27% desses recursos estão sendo utilizadas simultâneamente por registrador e LUT. As FIFOs foram mapeadas para blocos de memória RAM internos da FPGA.

Duas sínteses ASIC para *standard-cells* do BCM nas tecnologias TSMC 90nm e TSMC 0.18um foram feitas utilizando a ferramenta Synopsys Design Compiler (SYNOPSYS, 2010). Nestas sínteses, as FIFOs foram retiradas do BCM, pois não há ferramenta disponível para a geração dos IPs de memória. Os resultados desta síntese estão resumidos na Tabela 4.5.

Tabela 4.5: Resultados de síntese para *standard-cells* TSMC 90nm da arquitetura do BCM

Tecnologia	**Área**	**Potência**	**Frequência**	**MB/s**	**Bins/s**	**(KBins/s)/gate**
90 nm	19,2 kgates	11,1 mW	600 MHz	~2,9 M	~600 M	31.25
0.18 um	19,1 kgates	54,3 mW	400 MHz	~1,6 M	~400 M	20.9

Os resultados da síntese ASIC mostram que o BCM atingiu uma frequência 12 vezes maior que a necessária para processar vídeos no nível 4 do padrão H.264/AVC, ocupando uma área de 19,2 Kgates equivalentes, com um consumo de 11,1 mW de potência.

4.5 Trabalhos Relacionados

Muitos trabalhos propondo soluções arquiteturais para o CABAC podem ser encontrados na literatura. Entretanto, em sua maioria, esses trabalhos focam no BAE. Portanto, a revisão bibliográfica apresentada nesta seção também contém trabalhos que implementam somente o BAE.

4.5.1 Trabalho de Shojania – IEEE-NEWCAS 2005

O trabalho de Shojania é o primeiro a propor uma solução para a aceleração do processo de renormalização, o qual é o principal gargalo para atingir boa performance em termos de números de ciclos por *bin* processado. O processo de renormalização pode exigir vários ciclos para ser concluído e enquanto esse processo não termina, nenhum outro *bin* pode ser processado. Assim, Shojania utiliza um circuito detector de uns consecutivos para realizar a renormalização em um ciclo. A arquitetura proposta por Shojania prevê apenas o codificador aritmético, a memória de contextos e um circuito de empacotamento para a camada de rede.

A arquitetura sintetizada para FPGA Altera Startix 1S80 ocupou 1320 células lógicas e alcançou a frequência de 163MHz, garantindo ao codificador aritmético

proposto uma taxa de processamento de 54 Mbps. Uma síntese ASIC utilizando a tecnologia TSMC 0.18µm e os resultados de simulação indicam que o circuito ocupa 0.423mm^2, consumindo 48mW de potência e podendo rodar a uma frequência de 263 MHz, o que garante a taxa de codificação de 87 Mbps. Em ambos cenários o projeto proposto por Shojania é capaz de processar vídeos HDTV em tempo real no perfil *Main*.

4.5.2 Trabalho de Chen – ISCAS 2005

O trabalho de Chen trata de um codificador aritmético que processa até quatro *bins* por ciclo com memória de contexto. Ao contrário de outros trabalhos que propõem um codificador aritmético multisímbolos, Chen propôs o projeto incluindo a memória de contextos. Por conta da inclusão da memória, o autor focou seu trabalho em resolver o gerenciamento das operações de leitura e escrita, problema que não é abordado em outros trabalhos.

Toda vez que um *bin* é codificado utilizando o motor *regular*, uma operação de leitura e outra de escrita na memória de contextos é realizada acarretando em bolhas no *pipeline*. Para garantir o alto *throughput* da arquitetura, o autor propõe uma série de técnicas de gerenciamento de memória, como a inserção de bancos de registradores que funcionam como uma pequena memória *cache,* suporte a técnicas de *forward* e *backward* e operação de leitura/escrita explicitamente separadas para evitar a disputa em um mesmo endereço de memória.

O autor apresentou uma síntese ASIC na tecnologia TSMC 0.18 µm para três versões da arquitetura. Mesmo com o sofisticado esquema de gerenciamento de memória, o codificador aritmético de Chen capaz de processar 4 *bins* em paralelo alcança, na média, 3.32 *bins*/ciclo a 192MHz ocupando 32.1Kgates. A versão que processa até 2 *bins* em paralelo, ocupa 18.9 Kgates e atinge a frequência de 344MHz sob o *throughput* médio de 1.84 *bins*/ciclo. A versão de ciclo único ocupa 13.2 Kgates e atinge a frequência de 625 MHz. Todas as arquiteturas são capazes de processar vídeos 1080p à 30fps no perfil *Main*.

4.5.3 Trabalho de Kuo – APCCAS '06

O trabalho de Kuo é uma arquitetura que implementa apenas o codificador aritmético, mas foca na redução de potência. A principal otimização consiste em uma *cache* de tamanho variável para reduzir o acesso às memórias de atualização de probabilidades, pois de acordo com experimentos realizados pelo autor é a principal fonte de consumo de potência. O autor compara a arquitetura desenvolvida com e sem o esquema especial de *cache* e comprova a redução de 50% no consumo de potência em diferentes vídeos sob diferentes condições de codificação. Apesar dos excelentes resultados atingidos em termos de potência, como o projeto não prevê a implementação de vários módulos do CABAC, então não é possível prever se esses ganhos são relevantes em comparação ao restante do módulo.

O autor utiliza *pipeline*, circuito para detectar zeros seguidos e uns seguidos na etapa de renormalização para alcançar desempenho suficiente para processar vídeos HDTV em tempo real. A frequência que o projeto alcança não é descrita no artigo, mas o autor garante que na síntese realizada para TSMC 0.18µm o CABAC proposto atinge 200Mbps ocupando 0.31mm^2 de área e consumindo 20.7 mW de potência.

4.5.4 Trabalho de Li – APCCAS 2006

A arquitetura proposta por Li contém apenas o codificador aritmético, a memória de contextos e um circuito empacotador para a camada de rede, pois o foco do trabalho é propor um esquema dinâmico de *pipeline* que acelere o codificador aritmético, principal gargalo do CABAC. Primeiramente, as dependências de dados entre as tarefas da codificação aritmética são analisadas em conjunto com uma aceleração da busca de *bins* consecutivos com contextos que se repetem e dos *bins* equiprováveis. O *pipeline* é composto de três estágios: no primeiro estágio há a leitura do modelo de contexto; no segundo acontece a divisão do intervalo e a estimativa de probabilidade e, finalmente no terceiro estágio a renormalização. Neste projeto a renormalização não é acelerada causando bolhas no *pipeline* quando mais de um ciclo para este processo são necessários.

A arquitetura de Li foi sintetizada para tecnologia ROHM 0.35μm alcançando a frequência de 150 MHz e ocupando 4.57 Kgates equivalentes, excluindo a memória de contextos. A arquitetura de Li é capaz de processar vídeos 1080p em tempo real pois atinge uma taxa de codificação de 80Mbps.

4.5.5 Trabalho de Osório – IEEE Trans. Circuits Syst. Video Technol., 2006

O trabalho de Osório publicado na IEEE Trans. Circuits Syst. Video Technol. é uma extensão da primeira arquitetura de *hardware* desenvolvida para o CABAC que data de 2004. O desenvolvimento da arquitetura proposta por Osório inicia no nível algorítmico, onde o autor apresenta algumas modificações no codificador aritmético para que ele seja capaz de processar um *bin* no motor *bypass* concomitantemente a outro no motor *regular* ou dois *bins* no motor *bypass.* Quando há a ocorrência de 4 *bins*, sendo 3 deles equiprováveis, a arquitetura é capaz de processá-los em paralelo.

A arquitetura proposta prevê um binarizador parcial, apenas para os elementos sintáticos de resíduos. Todos os *bins* para cada amostra residual são gerados em um ciclo para suportar as otimizações do codificador aritmético. A modelagem de contextos, assim como a binarização, também é adaptada para fornecer mais de um contexto por ciclo e também só é desenvolvida para dados residuais. Um processador de propósito geral é previsto no projeto para realizar o processamento dos dados não-residuais e para a inicialização da tabela de contextos.

O projeto foi sintetizado para ASIC utilizando tecnologia AMS 0.35 μm ocupando 19426 gates com a frequência de 192 MHz. Outra síntese para o dispositivo FPGA Virtex II foi feita e, neste caso, 1113 *slices* da FPGA foram utilizados e a frequência de 92 MHz foi atingida. O CABAC proposto por Osorio tem um *throughput* que varia entre 1.9 e 2.3 *bins*/ciclo (considerando apenas os *bins* de resíduos) atingindo o desempenho necessário para processar vídeos 1080p à 30 fps no perfil *Main* na síntese ASIC.

4.5.6 Trabalho de Chen – ICASSP 2007

Chen propõe uma arquitetura para o CABAC praticamente completa, excluindo o *parser*. A arquitetura tem 3 estágios de *pipeline*. Para acelerar o processamento dos dados a binarização e a modelagem de contextos são processadas em paralelo e estão no primeiro estágio de *pipeline*. A atualização e a leitura da memória de contextos apenas são possíveis devido à utilização de uma memória *dual-port*, onde uma porta é dedicada somente para leitura e a outra somente para a escrita de contexto. Não é detalhado no

trabalho como o projeto gerencia a memória no caso da escrita e leitura ocorrerem no mesmo endereço de memória. No segundo estágio está o codificador aritmético que prevê um circuito de detecção de zeros seguidos para acelerar a renormalização. No último estágio está o empacotador de rede que consiste em uma FIFO que agrupa os bits com taxa variável na saída do codificador aritmético em palavras de 8 bits.

Em uma síntese sem a memória de contextos utilizando tecnologia 0.15µm, a arquitetura proposta por Chen ocupou 13.3Kgates e atingiu a frequência de 333MHz. As melhorias visando a otimização de desempenho garantem ao projeto um *throughput* de 0.55 *bins*/ciclo. Experimentos realizados pelo autor mostram que, em média, há uma redução de 10% no número de ciclos necessários para codificar vídeos devido às melhorias propostas e garantem que a arquitetura é capaz de rodar vídeos HDTV em tempo real.

4.5.7 Trabalho de Tian - IEEE RSP 2007

É proposta uma arquitetura que suporte todos os modos do CABAC e RDO (*Rate-Distortion Optimization*). O suporte a RDO aumenta ordens de grandeza a exigência por memória, criando um novo gargalo. Na arquitetura proposta, um esquema complexo de memória que conta com 5 RAM's e 4 tabelas em uma ROM é proposto. No projeto apresentado, o CABAC é dividido em 4 módulos principais: bloco 1 para binarização e seleção do modelo de contexto; bloco 2 para acesso ao modelo de contexto e codificador aritmético; bloco 3 para gerenciamento das memórias e; um bloco contendo todas as memórias. O autor assume que o *parser* é realizado por um processador externo ao projeto.

Para o projeto proposto por Tian, foi realizado uma síntese pós-*layout* para a tecnologia TSMC 0.13µm com todas as ferramentas RDO ativadas e para o pior caso foi atingida a frequência de 327MHz à 1 bin/ciclo com área total de 1.41mm^2 (ou 44.6Kgates). Está apto para rodar vídeos 720p em tempo real. Para alcançar HDTV é sugerida redução dos testes de modo para RDO e adicionar novas instâncias do CABAC em paralelo para codificar mais de um *slice* de um mesmo *frame* ao mesmo tempo. Não fica claro em nenhum momento qual o perfil que a arquitetura suporta.

A principal contribuição deste trabalho é a codificação aritmética com suporte à RDO. De acordo com o autor, o CABAC utilizando RDO pode reduzir em 10% o *bitrate* na saída do codificador, mas o incremento de complexidade do projeto é muito significativo, dificultando a codificação de vídeos de alta resolução. Além disso, o suporte a RDO inclui a necessidade de um esquema complexo de memória, ou seja, o consumo de potência é muito elevado. O projeto de Tian foi desenvolvido com técnicas de otimização de potência e, para vídeos 720p, o consumo de potência chega aos 34.3mW codificando à 5.34 fps com RDO ativo. Por outro lado, quando RDO está desativado, o mesmo design é capaz de codificar um vídeo 720p à 60 fps com consumo de potência de 2.55mW. De acordo com os resultados mostrados pelo autor, conclui-se que o suporte à RDO (principal contribuição do artigo de Tian) não é uma boa alternativa.

4.5.8 Trabalho de Liu – VLSI-DAT 2007

Liu propõe um CABAC dividido em 4 módulos: binarizador e modelador de contexto (BCM), um módulo que obtém dados de vizinhança, um módulo de

gerenciamento de contextos (PISO) e um codificador aritmético de múltiplo modo de 4 estágios de *pipeline*.

O processo de binarização pode gerar de 0 a 14 *bins* por elemento sintático e atribuir a cada um desses *bins* um número de contexto. O par formado pelo *bin* e pelo valor de contexto formam o 'par de contexto'. O PISO recebe os até 14 pares de contexto e repassa serialmente, um por um, os pares para o codificador aritmético. Os pares de contextos são guardados em um *buffer* de contextos para otimizar e reduzir o total de ciclos da codificação, de modo que o codificador aritmético nunca tenha que esperar por um par de *bin*/contexto.

O codificador aritmético é dividido em 4 estágios de *pipeline*, sendo que os dois primeiros estágios servem apenas para calcular parâmetros para codificação do motor regular, quando o modo de codificação é bypass, os registradores específicos do motor regular são setados em 0. No terceiro estágio está a renormalização onde é proposta uma expansão do *loop* interno ao algoritmo para que este processo seja realizado em um ciclo. No quarto estágio é gerado o bit de saída.

Mesmo com o constante abastecimento de pares *bin*/contexto, há bolhas no *pipeline,* garantindo um *throughput* médio de 0.67 *bins*/ciclo. Foram obtidos dados para tecnologia TSMC 0.13um e a arquitetura ocupa 34.265 gates equivalentes e roda à 200MHz. A arquitetura proposta por Liu é capaz de processar vídeos 1080p à 30 fps em tempo real, de acordo com o autor.

4.5.9 Trabalho de Chen – VLSI-DAT 2007

Chen inicia o trabalho expondo os pontos negativos de acelerar o CABAC através da inserção vários estágios de *pipeline* entre o caminho crítico do codificador aritmético, pois, de acordo com o autor, os ganhos com esta técnica são limitados e é impossível conseguir um *pipeline* balanceado devido à característica recursiva do algoritmo. Em seguida, Chen destaca que as soluções que procuram processar vários *bins* precisam de mudanças muito intensas nos algoritmos presentes no CABAC para serem realizadas e o projeto, nesse caso, é mais crítico que nas soluções que exploram o *pipeline* ao máximo.

Para balancear os caminhos críticos entre as etapas de codificação e conseguir uma melhor eficiência na utilização do *hardware* enquanto atinge altas taxas de processamento, o autor sugere uma decomposição da codificação LPS e da codificação MPS (decomposição ML), que nos outros trabalhos têm sido implementadas em conjunto. Isso significa que ao invés de processar novos intervalos de MPS e LPS em paralelo e realizar a escolha entre um deles, é proposto que os intervalos sejam calculados serialmente em um circuito combinacional que tem o cálculo de MPS e LPS como parte do caminho crítico.

A proposta de arquitetura utilizando células de decomposição ML consiste em arranjar essas células em forma de cascata sempre intercalando uma célula M e outra L. A arquitetura é facilmente configurável. Supondo que dois símbolos, um M e outro L, sejam checados, a arquitetura conseguirá processar ambos se os símbolos forem MPS e LPS. Se os símbolos são do mesmo tipo, a arquitetura irá processar um símbolo. Para cada célula há uma *flag* indicando se o contexto precisa ser atualizado ou não. Se um mesmo contexto foi utilizado em mais de uma célula, então a última ocorrência do contexto será utilizada para a atualização.

O autor apresenta cinco sínteses para a tecnologia TSMC 0.18μm de sua arquitetura em cascata contendo entre duas e seis células ML. Os resultados mostram que para 2 células, a arquitetura ocupa 4,5 Kgates, atingindo a taxa de processamento de 1.3 *bins*/ciclo sob a frequência de 520MHz. Para a arquitetura configurada com seis células, o codificador aritmético pode processar até 3.9 *bins*/ciclo à 210MHz ocupando a área de 12 Kgates. Em qualquer configuração, o projeto pode processar vídeos *FullHD* à 30fps em tempo real.

4.5.10 Trabalho de Lo - TENCON 2007

O artigo de Lo foca na melhora do *throughput* do codificador aritmético para conseguir aceleração da codificação, permitindo que múltiplos *bins* possam ser processados ao mesmo tempo. Para isso, é feito um levantamento sobre a distribuição dos *bins* binarizados e descobriu-se que os *bins* de resíduos podem corresponder até 75% do total de *bins*. Após essa importante constatação, o autor monta um esquema para melhorar o *tradeoff* entre geração e produção de bits e evitar a criação de buffers adicionais para armazenamento de *bins*. Essa arquitetura pode manipular dois *bins* por ciclo. A memória de contextos é dividida em duas partes (uma para os dois tipos de *bins* da pesquisa e outra para os outros *bins*) e é menor em tamanho. O processo que garante um melhor *throughput* na saída do codificador aritmético consiste em uma modificação no processo de renormalização: há uma função em paralelo que adiciona 'zero' a uma FIFO sempre que o símbolo é LPS, esses bits podem ser válidos ou não, quando são consumidos do *buffer*.

O projeto apresentado é um dos poucos que contém *parser*, porém o projeto não utiliza de *buffers* para sincronizar a produção de elementos sintáticos pelo *parser* com o binarizador nem a produção de *bins* do binarizador para a modelagem de contextos. A arquitetura tem *throughput* variável entre 1 e 2 *bins*/ciclo. A síntese da arquitetura proposta por Lo para tecnologia TSMC 0.18 alcançou a frequência de 135MHz e ocupa uma área 0.451mm^2. O desempenho é suficiente para processar vídeos 1080p à 30 fps no perfil *Main*.

4.5.11 Trabalho de Pastuzsak - IEEE Trans. Circuits Syst. Video Technol., 2008

Este trabalho apresenta uma arquitetura do CABAC com o binarizador compartilhado com CABAC+CAVLC e compatível com HDTV. O projeto suporta todos os formatos de crominância, codificação de vídeo entrelaçado e transformada 8x8. O projeto está em conformidade com o perfil *High* H.264/AVC e suporta dois modos de codificação binária: CABAC e CAVLC.

A arquitetura do binarizador com CABAC e CAVLC permite poupar recursos de *hardware*, como as memórias, e utilizar entradas e saídas comuns sem grandes esforços. Para o projeto é proposto um esquema de *pipeline* de 4 estágios. No primeiro estágio, os dados de entrada são lidos e há a identificação dos elementos sintáticos e armazenamento de dados na memória de vizinhos. O segundo estágio mapeia os elementos sintáticos para suas representações binárias através de circuitos combinacionais. O terceiro estágio encaminha o *bitstring* gerado pelos outros dois estágios anteriores e escolhe se o *bitstring* será codificado pelo CABAC ou pelo CAVLC. O quarto estágio encapsula os elementos sintáticos produzidos pela binarização e pelo fluxo do CABAC na NAL (*Network Abstract Layer*).

Em seguida é apresentada uma proposta arquitetural para o codificador aritmético e modelador de contextos. Para relacionar o *tradeoff* entre freqüência e área do CABAC, variações de alguns módulos foram testadas. Dois modeladores de contextos diferentes foram projetados, diferenciando-se, basicamente, na quantidade de contextos na saída: um ou dois contextos. Para o processo de inicialização do CABAC, é utilizada uma estratégia de geração contínua de pares de contexto e três estágios de *pipeline*. O bloco de inicialização é colocado em paralelo ao modelador de contextos e um multiplexador escolhe qual módulo fornecerá os dados para serem codificados. O objetivo principal é que a parada da codificação causada pelo processo de inicialização seja mínima.

A estrutura sequencial do codificador aritmético é dividido em 9 ou 10 estágios de *pipeline*, dependendo da quantidade de motores *bypass*: um ou três. A arquitetura proposta é capaz de processar dois *bins* simultaneamente devido às otimizações na modelagem de contexto.

O autor criou cinco versões de CABAC a partir das variações arquiteturais do codificador aritmético e modelador de contexto e as comparou entre si e verificou a mais eficiente. O autor testou apenas dois vídeos em CIF e 720p, variando o QP de 12 a 42 incrementando-o de 6 em 6 para verificar a performance do projeto. Verificou-se que a arquitetura é compatível com HDTV. O autor apresenta uma síntese excluindo o CAVLC do projeto que alcança os 192 MHz de frequência e 36k*gates* equivalentes para tecnologia AMS 0.35um.

O projeto de Pastuszak promete o suporte à transformada 8x8, porém sua entrada para dados residuais é de 16x16 bits. Para suportar transformada 8x8, a entrada de resíduos deveria ser de 64x16 bits, ou conter um esquema de armazenamento de amostras quando a transformada é 8x8. O autor não apresenta detalhes sobre isso, mas como a arquitetura contém *parser*, assume-se que as amostras já estão ordenadas de maneira correta em ziguezague no *buffer* de entrada. Pela largura da memória de macroblocos (49 bits) fica claro que o projeto não suporta quadros P e B.

4.5.12 Trabalho de Zheng - Electronics, IEEE Transactions 2008

Nesse artigo é proposto o projeto de um CABAC de 4 estágios de pipeline que pode ser facilmente incorporado em uma SoC (*System on Chip*). Para reduzir o custo de *hardware*, um esquema simples de controle e técnicas de *forwarding* são incorporadas no projeto. Para conseguir um *pipeline* balanceado e atingir o *throughput* de 1 *bin*/ciclo, o tempo de consumo de bits, a iteração variável da renormalização e a geração de *bitstream* são realizados em 3 estágios de *pipeline*.

Para implementar esse esquema, a binarização e o seletor de contextos são executados por um processador de propósito geral. Um barramento faz a comunicação entre o codificador, o processador RISC e a RAM que contém os elementos sintáticos. O pipeline do CABAC apresentado por Zheng é assim dividido: 1 – Lê memória de contextos; 2 – Lê memória LPS e processa novos rLPS e rMPS; 3 – codificação aritmética e renormalização e 4 – geração de bits e atualização de contextos. O *forwarding* aplicado evita a dependência de dados e as bolhas no *pipeline*. Os estágios 2 e 3 contam com *pipeline* interno de 3 estágios.

Em relação ao controle do CABAC proposto, foram estudadas técnicas complexas e com alto custo de *hardware*, mas descobriu-se que para um projeto adaptável com SoC, um *pipeline* balanceado e um *throughput* estável são as melhores alternativas. O projeto ocupa 14000 *gates* e 15 Kbit de SRAM, podendo alcançar até 300MHz de frequência

para tecnologia SMIC 0.18 um com *throughput* de 1 símbolo/ciclo, exceto quando a tabela de contextos está sendo inicializada.

4.5.13 Trabalho de Wu – ISCAS 2009

Neste trabalho, uma arquitetura com alto *throughput* para resoluções altíssimas (3840x2160) de vídeo é proposta. Esta arquitetura é composta por um esquema otimizado de acesso à memória de contextos, um codificador aritmético binário multi-símbolo e uma modelagem de contexto otimizada de acordo com o tipo de elemento sintático.

Primeiramente, uma análise da distribuição dos *bins* para algumas sequências de vídeos foi apresentada com a finalidade de justificar as decisões arquiteturais tomadas no projeto. A binarização e a modelagem de contexto para os elementos sintáticos mais frequentes de acordo com o experimento foram acelerados. Para os *bins* dos SE binarizados com Exp-Golomb parametrizável, o codificador aritmético pode atingir taxas de processamento de 2 *bins* regulares e dois *bypass* ou 8 *bins bypass*, pois esses *bins* são frequentes e que geram a maior informação que deve ser processada. Para outros casos, dois *bins* regulares podem ser processados em paralelo. Assim, o codificador aritmético baseado na distribuição de *bins* proposto por Wu possui taxa de processamento média de 2.36 *bins*/ciclo, enquanto que a arquitetura completa do CABAC alcança uma taxa de 1.3 *bins*/ciclo.

Em uma síntese utilizando a biblioteca de células TSMC 0.13 μm para o fluxo ASIC, o circuito alcançou a frequência de 222MHz ocupando 45.9 Kgates. Resultados experimentais mostram que a arquitetura é capaz de atingir o máximo *bitrate* definido no perfil *Main,* nível 5.1.

4.5.14 Trabalho de Rosa – UFRGS 2010

Até o trabalho de Rosa (2010), os trabalhos estavam propondo soluções arquiteturais para o CABAC visando aumentar o *throughput* através da exploração de paralelismo no bloco BAE e, assim, alcançar desempenho suficiente para processar vídeos nos níveis mais altos do padrão H.264/AVC. O trabalho de Rosa (2010) questiona a necessidade de paralelizar o BAE e propõe três arquiteturas de BAE de *throughput* baixo capazes de processar vídeos nos níveis mais altos do padrão H.264/AVC.

A primeira arquitetura de BAE proposta por Rosa (2010) é totalmente serial e, mesmo com um *throughput* de 0.68 *bins*/ciclo, essa arquitetura alcança desempenho suficiente para processar vídeos no nível 4.2 do padrão H.264/AVC. Rosa propõe ainda uma arquitetura com *pipeline* que alcança *throughput* de 0,86 bin/ciclo e uma terceira arquitetura com aceleração do processo de renormalização que garante ao BAE 1 bin/ciclo. Todas as arquiteturas propostas possuem alta frequência e são muito eficientes em área e energia. Surpreendentemente, a arquitetura mais eficiente é também a mais simples, a arquitetura totalmente serial. Assim, através de suas propostas arquiteturais e comparações com trabalhos relacionados, o trabalho de Rosa (2010) prova que paralelizar o CABAC é ineficiente, complexo e desnecessário.

4.6 Comparação com Trabalhos Relacionados

Os trabalhos discutidos na Seção 4.5 são considerados as principais referências disponíveis na literatura em termos de arquiteturas de *hardware* destinadas ao CABAC.

Entretanto, a comparação de desempenho entre esses trabalhos não é trivial, pois na maioria dos trabalhos não é possível encontrar informações suficientes para compor todo o cenário de avaliação com os parâmetros que podem interferir no processo. De qualquer modo, uma comparação dos resultados dos trabalhos relacionados com a arquitetura proposta está resumida na Tabela 4.6. A comparação leva em conta somente dados de síntese ASIC, visto que alguns trabalhos não apresentam resultados em FPGA e, quando apresentam, os dispositivos alvos de síntese são os mais variados tornando a comparação injusta.

Tabela 4.6: Comparação de resultados entre arquiteturas desenvolvidas para o CABAC

Autor	**Tec. (um)**	**Potência (mW)**	**bins/ciclo**	**Freq. (MHz)**	**Área BCM**	**Área BAE**	**Mbins/s**
Sho., 2005	0.18	48	0,33	263	-	0.423 mm²	87
Kuo, 2006	0.18	20,7	-	-	-	0.31 mm²	200
Li, 2006	0.35	-	0,53*	150	-	4.57 kgates	80
Osório, 2006	0.35	-	1,9~2,3	186	19.4 kgates	17 kgates	353
Chen, 2006	0.18	-	3,3	192	-	32.1 kgates	633*
Chen, 2007	0.15	-	0,55	333	13.3 kgates		183*
Liu, 2007	0.13	-	0,67	200	29.1 kgates	5.1 kgates	134*
Chen, 2007b	0.18	-	3,9	210	-	12 kgates	819*
Lo, 2007	0.18	-	1~2*	135	0.451mm²		135~270
Past., 2008	0.18	-	2,2	140	14.8 kgates	44.9 kgates	308
Zheng, 2008	0.18	-	1	300	-	14 kgates	300*
Wu, 2009	0.13	-	1,3	222	31.2 kgates	14.7 kgates	288
$Rosa_1$, 2010	0.18	1,9	0,68	292	-	1,2 kgates	200
$Rosa_2$, 2010	0.18	9,6	1	250	-	31,4 kgates	250
Este Trabalho	**0.09**	**11,1**	**1**	**600**	**19,2 kgates**	**-**	**600**

**Informação calculada com base nas informações disponíveis no trabalho*

A Tabela 4.6 mostra que poucos trabalhos propõem soluções arquiteturais para o BCM. Os resultados das arquiteturas CABAC de Chen (2007) e Lo (2007) não fazem distinção entre BAE e BCM. Nenhum dos trabalhos de BCM da Tabela 4.6 suporta transformada 8x8 (presente apenas no perfil *High*) ou taxa de amostragem maior que 4:2:0.

Todos os trabalhos da literatura listados na tabela são capazes de processar vídeos no nível 4 do perfil *Main* do padrão H.264/AVC, enquanto que os trabalhos de Osório (2006), Chen (2006), Chen (2007b) e este trabalho são os únicos capazes de alcançar os requisitos máximos de processamento do padrão H.264/AVC no perfil *Main*.

A arquitetura proposta neste trabalho é a única que apresenta resultados de potência no BCM, logo qualquer comparação em termos de potência e energia é impossível.

As arquiteturas que apresentam maior *throughput* e melhor desempenho são Chen (2006) e Chen (2007b), entretanto essas arquiteturas não contêm BCM. Wu (2009) propõe um BAE com *throughput* de 2.37 *bins*/ciclo, mas a implementação do BCM acaba limitando esse parâmetro em 1.3 bin/ciclo na arquitetura global do CABAC, reforçando a hipótese paralelizar o BAE não é uma boa alternativa (Rosa, 2010).

A Tabela 4.7 apresenta uma comparação de resultados apenas entre os trabalhos que propõem arquiteturas para o BCM. Com o objetivo de normalizar os resultados e não ser beneficiado por fatores tecnológicos, a síntese para *standard-cells* 0.18um é utilizada nesta comparação.

Tabela 4.7: Comparação de resultados entre arquiteturas de BCM

Autor	**Tec.** (um)	**Pot.** (mW)	**bins/ciclo**	**Freq.** (MHz)	**Área** (kgates)	**Mbins/s**	**Efic. em Área** (Kbins/s)/gate	**Parser**	**Binarizador**	**Mod. De Contexto**
Osório,2006	0.35	-	1,9~2,3	186	19,4	353	18,2	Não	Parcial	Sim
Chen, 2007	0.15	-	0,55	333	13,3	183	13,7	Não	Sim	Sim
Liu, 2007	0.13	-	0,67	200	29,1	134	4,6	Não	Sim	Sim
Past., 2008	0.18	-	2,2	140	14,8	308	20,8	Sim	Sim	Não
Wu, 2009	0.13	-	1,3	222	31,2	288	9,2	Sim	Sim	Sim
Este Trabalho	**0.18**	**54,3**	**1**	**400**	**19,1**	**400**	**20,9**	**Sim**	**Sim**	**Sim**

Nos trabalhos que apresentam arquiteturas de BCM é preciso destacar que Osório (2006) não contém *Parser* e suporta apenas as binarizações U e UEGk, a modelagem de contexto de Pastuzsak (2008) está contida no BAE, Chen (2007) e Liu (2007) não contém *Parser* e apenas o trabalho de Wu (2009) e este trabalho apresentam soluções completas para o BCM.

Entre as arquiteturas que apresentam soluções para o BCM, este trabalho é o melhor em termos de tempo de computação, sendo 13,3% mais rápido que Osório (2006) e 29,9% mais rápido que Pastuzsak (2008), os quais eram os melhores trabalhos neste quesito. Como mostra a Tabela 4.6, os trabalhos de Pastuzsak (2008) e Wu (2009) consistem em arquiteturas completas para o CABAC, portanto o desempenho do BCM dessas arquiteturas pode estar sendo comprometido por caminhos críticos presentes no BAE. A arquitetura proposta é desenvolvida para atender os requisitos de um BAE como o descrito em Rosa (2010). Supondo a integração do trabalho de Rosa (2010) e este trabalho, o caminho crítico estaria localizado no BAE e a frequência do BCM cairia para 250MHz.

Em relação aos resultados de área, as arquiteturas de Chen (2007) e Pastuzsak (2008) são as que utilizam menos recursos de *hardware*, sendo 30,3% e 22,5%, respectivamente, menores que a arquitetura proposta neste trabalho. Todavia, como dito anteriormente, Pastuzsak (2008) atribui a modelagem de contexto ao BAE e o trabalho de Chen (2007) não contém *Parser*.

Em termos de eficiência em área, dado em (kbins/s)/gate, o BCM desenvolvido neste trabalho atingiu os melhores resultados em comparação aos trabalhos relacionados, alcançando um índice similar ao de Pastuzsak (2008), melhor trabalho da literatura nesse quesito e 12,7% melhor que Osório (2006). Assim, considerando que Pastuzsak (2008) não contém Modelagem de Contextos e analisando o *tradeoff* existente em qualquer projeto de *hardware* entre tempo de computação e área, o BCM apresentado neste trabalho é a opção mais equilibrada das arquiteturas encontradas na literatura.

5 CONCLUSÕES

Esta dissertação apresenta inicialmente conceitos básicos sobre compressão de vídeo digital. Além disso, um panorama sobre as principais ferramentas e características do estado-da-arte em codificadores de vídeo, o padrão H.264/AVC, é descrito. Em seguida, é apresentado um estudo sobre o conjunto de algoritmos que constituem o CABAC, detalhando minuciosamente os processos de *Parsing*, Binarização e Modelagem de Contextos, os quais, juntos, são conhecidos na literatura como BCM.

A partir do estudo do padrão H.264/AVC e do CABAC, uma nova arquitetura de *hardware* para o BCM é proposta. Esta arquitetura é desenvolvida para codificar vídeos de alta resolução em tempo real para o perfil *Main* do padrão H.264/AVC de modo eficiente em área e energia.

O projeto do BCM é desenvolvido basedo em análises de trabalhos da literatura que descrevem arquiteturas para o módulo posterior ao BCM, chamado de BAE. A partir dessa análise, a estratégia adotada para atingir os objetivos propostos é desenvolver uma arquitetura com alta frequência e *throughput* máximo de *1bin*/ciclo.

A arquitetura descrita em VHDL é sintetizada para o dispositivo FPGA Virtex 5 da Xilinx e para *standard-cells*. Os resultados mostram que a arquitetura proposta é a mais eficiente em área e é capaz de processar vídeos no nível mais alto (nível 5.1) do padrão H.264/AVC, atingindo os objetivos traçados para este projeto.

O BCM proposto é tão eficiente em área quanto Pastuzsak (2008), mas o trabalho de Pastuzsak não contempla a modelagem de contexto, módulo que consome maior porcentagem de área. A arquitetura proposta também apresenta o melhor desempenho dentre os trabalhos relacionados. Comparado à única arquitetura completa de BCM da literatura (Wu, 2009), o BCM proposto é melhor em consumo de recursos de *hardware*, tempo de computação e eficiência em área.

Uma exploração de espaço de projeto para arquiteturas do Binarizador também foi apresentada. Foram propostas três arquiteturas para o binarizador: uma multiciclo, uma ciclo único e outra chamada de *context-aware*. Uma das inovações dessa exploração arquitetural é a implementação do algoritmo de binarização Exp-Golomb Parametrizável utilizando uma combinação dos códigos Unário e Tamanho Fixo.

5.1 Trabalhos Futuros

A finalização da Validação do BCM é o trabalho que está em andamento atualmente. Posteriormente, será feita a extensão da arquitetura do BCM para o perfil *High* e para vídeos escaláveis. Para suportar o perfil *High*, as principais alterações se concentram no Parser que deverá processar os resíduos de transformadas 8x8 e realizar a montagem e o mapa de significância desses resíduos corretamente. Para processar

vídeos escaláveis, o BCM terá que tratar mais três elementos sintáticos exclusivos dessa extensão do padrão H.264/AVC (INTERNATIONAL, 2007).

Outro trabalho previsto é a adição de técnicas de redução de potência na arquitetura do BCM. Pretende-se aplicar técnicas de *power gating* no BCM. Por exemplo, o circuito que processa dados de predição interquadros pode ser desligado se o *slice* atual é intraquadros e vice-versa.

Um trabalho futuro que segue a linha de *software* é a experimentação e proposta de uma mudança na modelagem de contextos para os *bins* gerados com binarização Unário/Exp-Golomb Parametrizáveis através da utilização de códigos Unário e Tamanho Fixo propostas nesta dissertação. Os *bins* gerados com códigos Exp-Golomb não são passíveis de compressão, pois utilizam o motor de codificação aritmética *bypass*. Aplicando a nova técnica proposta para a geração do *binstring* Unário/Exp-Golomb Parametrizável, metade dos *bins* de código Exp-Golomb podem receber um número de contexto sendo, portanto, passíveis de compressão pelo codificador aritmético. A expectativa é que, aplicando essa técnica, principalmente para vídeos de altíssima resolução, haja uma redução na taxa de bits sem comprometer o custo computacional de operações.

REFERÊNCIAS

AGOSTINI, L. **Desenvolvimento de Arquiteturas de Alta Performance Dedicadas à Compressão de Vídeo Segundo o Padrão H.264/AVC.** 2007. 173 f. Tese (Doutorado em Ciência da Computação) - Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

BINGBO, L. et al. A high-performance VLSI architecture for CABAC decoding in H.264/AVC. In: INTERNATIONAL CONFERENCE ON ASIC, ASICON, 2007 Guilin, China. **Proceedings…** [s.n.], 2007 pp. 790-793.

CHEN, J-L. et al. A low cost context adaptive arithmetic coder for H.264/MPEG-4 AVC video coding. In: INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING. **Proceedings…** Hawaii, USA: IEEE, 2007, pp 105–108

CHEN, J-W. et al. A hardware accelerator for context-based adaptive binary arithmetic decoding in H.264/AVC. In: INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS. **Proceedings…** , Kobe, Japan: IEEE, 2005, pp 4525–4528

CHEN, J-W. et al. A high-performance hardwired CABAC decoder. In: INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING. **Proceedings…** Hawaii, USA: IEEE, 2007, pp 37–40

CHEN, Y-J. . et al. Novel configurable architecture of ML-decomposed binary arithmetic encoder for multimedia applications. In: INTERNATIONAL SYMPOSIUM ON VLSI DESIGN, AUTOMATION, AND TEST. **Proceedings…** Hsinchu, Taiwan: IEEE, 2007, pp 1–4

CHEN, S. Efficient bit-rate estimation technique for CABAC. In: ASIA PACIFIC CONFERENCE CIRCUITS AND SYSTEMS, APCCAS, 2008. **Proceedings…** [S.l]: IEEE**,** 2008, pp. 514-517

COREGENERATOR System. **XILINX**. Disponível em: < http://www.xilinx.com/tools/coregen.htm>. Acesso em: Novembro de 2010c.

CORRÊA, G. R. **Estudo e desenvolvimento de heurísticas e arquiteturas de hardware para decisão rápida do modo de codificação de bloco para o padrão H.264/AVC.** 2010. 95 f. Dissertação (Mestrado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

DEPRÁ, D. A. **Algoritmos e Desenvolvimento de Arquitetura para a Codificação Binária Adaptativa ao Contexto para o Decodificador H.264/AVC.** 2009. 152 f. Dissertação (Mestrado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

DEPRA, D. A. et al. **Design and Implementation of a High-Performance Architecture for Binarization Methods of Defined by H.264/AVC Standard**. In: XIV Workshop Iberchip, 2008

DI, W. et al. An Exp-Golomb encoder and decoder architecture for JVT/AVS. In: 5TH INTERNATIONAL CONFERENCE ON ASIC **Proceedings…** [S.l: s.n.], 2003, pp. 910–913.

DING, L-F. et al. A 212 MPixels/s 4096 × 2160p Multiview Video Encoder Chip for 3D/Quad Full HDTV Applications. In: IEEE JOURNAL OF SOLID CIRCUITS **Proceedings…** [S.l]: IEEE, 2010, vol. 45, no. 1.

DINIZ, C. M. **Arquitetura de hardware dedicada para a predição intraquadro em codificadores do padrão H.264/AVC de compressão de vídeo.** 2009. 96 f. Dissertação (Mestrado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

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

GONZALEZ, R. et al. Processamento de Imagens Digitais. São Paulo: **Edgard Blucher**, 2003.

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

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

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

INTERNATIONAL Telecommunications Union. **ITU-T Home**. Disponível em: <www.itu.int/ITU-T/>. Acesso em: ago. 2009a.

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

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.

ISE Foundation. **XILINX**. Disponível em: <http://www.xilinx.com/ise/ logic_design_prod/foundation.htm>. Acesso em: dezembro de 2010.

KUO, C-C. et al. Design of a Low Power Architecture for CABAC Encoder in H.264. In: ASIA PACIFIC CONFERENCE ON CIRCUITS AND SYSTEMS, APCCS, 2006. **Proceedings…** Singapore: IEEE, 2006, pp 243 – 246

LEE, J. B. et al. **The VC-1 and H.264 Video Compression Standards for Broadband Video Services**, Springer, 2008.

LI, L.; SONG. et al. A CABAC encoding core with dynamic pipeline for H.264/AVC main profile. In: ASIA PACIFIC CONFERENCE ON CIRCUITS AND SYSTEM **Proceedings…** Singapore: IEEE, 2006, pp 760–763

LIN, Y-L. et al. **VLSI Design for Video Coding– H.264/AVC Encoding from Standard Specification to Chip**. Springer Science US, 2010

LIU, P-S. et al. A hardwired context-based adaptive binary arithmetic encoder for H.264 advanced video coding. In: INTERNATIONAL SYMPOSIUM ON VLSI DESIGN, AUTOMATION, AND TEST. **Proceedings…** Hsinchu, Taiwan: IEEE, 2007, pp 1–4

LO C-C. et al. Design and test of a high-throughput CABAC encoder. In: INTERNATIONAL TECHNICAL CONFERENCE OF IEEE REGION 10. **Proceedings…** Taipei, Taiwan: IEEE, 2007, pp 1–4

MARPE. et al. Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard. In: **IEEE Transactions on Circuits and Systems for Video Technology**, 2003, vol. 13, no 7.

MARPE. et al. A highly efficient multiplication-free binary arithmetic coder and its application in video coding. In: INTERNATIONAL CONFERENCE ON IMAGE PROCESSING, ICIP, 2003. **Proceedings…** [S.l: s.n.], 2003, v. 2, pp. 263-266.

MARTINS, A. L. M. et al, A Low-Cost Hardware Architecture Binarizer Design for the H.264/AVC CABAC Entropy Coding. In: IEEE International Conference on Electronics, Circuits and Systems, ICECS, 2010. **Proceedings…** Athenas, Greece, IEEE, 2010, vol. 1.

MARTINS, A. L. M. et al, A Low-Cost Hardware Architecture Design for Binarizer Defined by H.264/AVC Standard. In: South Symposium on Microeletronics, SIM, 2010 **Proceedings…** [S.l: s.n.], 2010b.

MODELSIM, **Mentor Graphics INC**. Disponível em: <http://www.altera.com/products/*software*/products/model/eda-ms.html>. Acesso em: outubro 2008.

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

OSORIO, R. R. et al. Arithmetic coding architecture for H.264/AVC CABAC compression system. In: EUROMICRO SYMPOSIUM, 2004. **Proceedings…** [S.l: s.n.], 2004, pp 62-69.

OSORIO, R. R. et al. High-Throughput architecture for H.264/AVC CABAC compression system. In: **IEEE Transactions on. Circuits and Systems for Video Technology**, 2006, vol. 16, no. 11, pp. 1376–1384.

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

PENNEBAKER, W. B. et al. An overview of the basic principles of the Q-Coder adaptive binary arithmetic coder. **IBM 1. RES. DEVELOP**, 1988, vol. 32 no. 6.

PORTO, R. **Desenvolvimento Arquitetural para Estimação de Movimento de Blocos de Tamanhos Variáveis Segundo o Padrão H.264/AVC de Compressão de Vídeo Digital**, 2008b. 96f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

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

RAMOS, F. L. L. Implementação de um codificador CAVLC para o codificador H.264/AVC, visando aplicações para HDTV em tempo real. In: XV WORKSHOP IBERCHIP 2009. **Proceedings…** Buenos Aires, Argentina: [s.n.], 2009, pp. 540–544.

RICHARDSON, I. E. **The H.264 Advanced Video Compression Standard**. 2.ed. Wiley, 2010. 346p.

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

ROSA, V. S. **Arquiteturas de hardware dedicadas para codificadores de vídeo H.264: filtragem de efeitos de bloco e codificação aritmética binária adaptativa a contexto,** 2010. 171 f. Tese (Doutorado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

SAID, A. **Introduction to Arithmetic Coding - Theory and Practice**. Palo Alto: Imaging Systems Laboratory - HP Laboratories. 2004.

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

SHI, Y. et al. **Image and Video Compression for Multimedia Engineering: Fundamentals, Algorithms and Standards**. Boca Raton, FL: CRC Press, 1999.

SHOJANIA, H. et al. A high performance CABAC encoder. In: INTERNATIONAL NORTHEAST WORKSHOP ON CIRCUITS AND SYSTEMS, 2005. **Proceedings…** Ville de Quebec, Canada: IEEE, 2005, pp 315–318.

SUHRING, K. H.264/AVC Reference Software. **In: Fraunhofer Heinrich-Hertz-Institute**. Disponível em: <http://iphome.hhi.de/suehring/tml/download/>. Acesso em: maio 2009.

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

SULLIVAN, G. et al. **Video Compression – From Concepts to the H.264/AVC Standard**. Proceedings of the IEEE, [S.l.]: IEEE, 2005, v. 93, n. 1, p. 18-31.

SUN, C. et al. An Efficient Context Modeling Algorithm for Motion Vectors in CABAC. In: INTERNATIONAL SYMPOSIUM ON SIGNAL PROCESSING AND INFORMATION TECHNOLOGY, 2007 **Proceedings…** [S.l]: IEEE, 2007.

SYNOPSYS INC. **Synopsys Design Compiler Software**, 2010. Disponível em: <http://www.synopsys.com/Tools/Implementation/RTLSynthesis/Pages/DesignCompiler2010-ds.aspx>. Acesso em: dezembro 2010.

SZE, V. Parallel CABAC for low power video coding. INTERNATIONAL CONFERENCE ON IMAGE PROCESSING, ICIP, 2008. **Proceedings…** [S.l]: IEEE**.** 2008, pp. 2096-2099

TIAN, X. H. et al. A CABAC Encoder Design of H.264/AVC with RDO Support. In: RAPID SYSTEM PROTOTYPING, RSP, 2007. **Proceedings…** [S.l]: IEEE, 2007, pp. 167-173.

VIRTEX II pro support. **XILINX**. Disponível em: <http://www.xilinx.com/support/mysupport.htm#Virtex-II%20Pro>. Acesso em: Novembro de 2010.

VIRTEX5 Multi-platform FPGA. **XILINX**. Disponível em: <http://www.xilinx.com/products/virtex5/>. Acesso em: Dezembro de 2010.

WIEGAND, T. et al. Overview of the H.264/AVC Video Coding Standard. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.]: IEEE, 2003, v. 13, n. 7, p. 560-576.

WINMERGE. **WinMerge 2.12.4**. Disponível em: <http://winmerge.org/>. Acessado em: Maio de 2010

WU, L-C. et al. A high throughput CABAC encoder for ultra high resolution video. In: INTERNATIONAL SYMPOSIUM ON CIRCTUITS AND SYSTEMS, ISCAS, 2009. **Proceedings…** [S.l]: IEEE, 2009, pp. 1048-1051.

X264. **x264**. Disponível em: <http://x264.nl/>. Acesso em: Julho de 2010

X264. **x264 Settings**. Disponível em: <http://mewiki.project357.com/wiki/X264_Settings>. Acesso em: Julho de 2010b

YANG, E. Soft Decision Quantization for H.264 With Main Profile Compatibility. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.]: IEEE, 2009, v. 19, n. 1, p. 122-127.

ZATT, B. **Preditor de vetores de movimento para o padrão H.264/AVC perfil Main,** 2006. 73 f. Trabalho de Conclusão (Graduação em Engenharia de Computação) - Instituto de Informática, UFRGS, Porto Alegre.

ZATT, B. **Modelagem de hardware para codificação de vídeo e arquitetura de compensação de movimento segundo o padrão H.264/AVC**, 2008. 120 f. Trabalho de Conclusão (Mestrado em Ciência da Computação) - Instituto de Informática, UFRGS, Porto Alegre.

ZHENG, W. et al. Efficient pipelined CABAC encoding architecture. **IEEE Transactions on Consumer Electronics.** [S.l.]: IEEE 2008, v. 54, pp. 681-686.