Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Arquitetura Dedicada para Decodificação CABAC H.264/AVC em Sistema em Silício

José Porfírio Albuquerque de Carvalho

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Orientador
Prof. Dr. Ricardo Pezzuol Jacobi

Coorientador
Prof. Dr. Pedro de A. Berger

Brasília
2009

Universidade de Brasília – UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Mestrado em Informática

Coordenador: Prof. Dr. Li Weigang

Banca examinadora composta por:

Prof. Dr. Ricardo Pezzuol Jacobi (Orientador) – CIC/UnB
Prof. Dr. Sergio Bampi – Instituto de Informática/UFRGS
Prof. Dr. Bruno L. Macchiavello E. – CIC/UnB

CIP – Catalogação Internacional na Publicação

José Porfírio Albuquerque de Carvalho.

Arquitetura Dedicada para Decodificação CABAC H.264/AVC em Sistema em Silício/ José Porfírio Albuquerque de Carvalho. Brasília : UnB, 2009.
93 p. : il. ; 29,5 cm.

Tese (Mestre) – Universidade de Brasília, Brasília, 2009.

1. CABAC, 2. H.264, 3. Decodificação, 4. SoC

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro – Asa Norte
CEP 70910–900
Brasília – DF – Brasil

Agradecimentos

Primeiramente gostaria de agradecer aos meus pais e família que são a base primordial da minha vida e que me apoiaram e estimularam desde o início, sempre enaltecendo a educação e o conhecimento.

Agradeço ao Juan Fernando Eusse pela disposição e paciência em avaliar minha primeira descrição VHDL e fornecer dicas de como melhorá-la. Também agradeço aos colegas João Vitor Cavalcanti e Cauê Dobbin por toda a ajuda nas versões iniciais da descrição VHDL, nos primeiros testes funcionais e pela explicação do ModelSim. Um agradecimento especial ao Cauê por me ajudar a ver a solução de utilizar freqüências distintas para as memórias de contextos.

Agradeço aos meus amigos e colegas de trabalho do Banco Central do Brasil e da ECT que me apoiaram durante o período do mestrado e possibilitaram a continuação dos meus estudos. Obrigado pelas palavras de motivação e pela confiança.

Agradeço aos docentes do departamento de Ciência de Computação e aos funcionários da secretaria de Mestrado em Informática da Universidade de Brasília, principalmente a Rosa Amariles Vilar de Azevedo, por todo o apoio durante todos esse anos.

Um agradecimento especial para meu amigo e companheiro de universidade Roberto Silva Cantanhede, o qual sempre me apoiou em todas atividades acadêmicas nos últimos 11 anos. Obrigado por conversar comigo sobre minhas idéias e dúvidas do mestrado e por ajudar na revisão do texto. Sua ajuda, críticas, dicas e amizade foram e são muito importantes para mim.

Agradeço ao Prof. Dr. Pedro de A. Berger por participar da minha orientação. Seus conselhos, idéias, dicas, paciência e disponibilidade foram essenciais para a elaboração de todo o trabalho.

Um profundo agradecimento ao meu orientador Prof. Dr. Ricardo Pezzuol Jacobi por toda a confiança depositada em mim durante todo o período do mestrado. Agradeço todos os esforços, ensinamentos transmitidos, a liberdade e o tempo disponibilizados. Espero que todo meu empenho dedicado a esse trabalho faça jus aos meus dois orientadores.

Finalmente, gostaria de dedicar esse trabalho ao amor da minha vida e esposa Simone Keiko Yoshida de Carvalho. Agradeço por todo apoio, pela paciência e esforço em compensar minha ausência em tantos momentos. Obrigado por me dar e cuidar deste grande presente que é o nosso filho Lucas Daisuke. Saiba que estou eternamente grato por todo o sacrifício feito por você.

Resumo

O padrão de codificação de vídeo ITU-T H.264/MPEG-4 Part 10 (*Advanced Video Coding*), publicado originalmente em 2003 pela equipe de especialistas do *Joint Video Team* (JVT), está presente em tecnologias como a Blu-Ray, HD-DVD e o Sistema de Televisão Digital Brasileira. Este padrão trouxe algumas melhorias quando comparada aos seus antecessores (MPEG-1, MPEG-2, MPEG-4, H.261 e H.263), entre elas, um inovador codificador entrópico baseado em codificação aritmética, chamado *Context-based Adaptive Binary Arithmetic Coding* (CABAC). Tais melhorias trouxeram uma grande eficiência para o padrão, reduzindo em até 50% a taxa de *bits* (*bit-rate*) necessária para a transmissão de um mesmo vídeo codificado quando comparada com o padrão MPEG-4, com a penalização de ser mais complexo computacionalmente.

A complexidade do processo CABAC é exemplificada por estudos prévios que relatam a necessidade de freqüência de *clock* superior a 100MHz para que um Processador Digital de Sinais (DSP) típico suporte, de forma dedicada, a implementação do decodificador CABAC para a decodificação em tempo-real de um vídeo com resolução D1 (720x480) a taxa de 30 *frames*/segundo. O processo do (de)codificador CABAC é primordialmente seqüencial e para acelerar sua execução, opta-se por desenvolver soluções otimizadas, inclusive em hardware dedicado.

Este trabalho realizou um estudo sobre o processo de decodificação CABAC, propondo uma arquitetura dedicada que utiliza otimizações para obter paralelismo na decodificação. A solução permite a decodificação de vídeos codificados pelo padrão H.264 em resolução 1080p a taxa de 30 *frames*/segundo. O decodificador poderá ser integrado em um projeto de SoC (*System-on-a-Chip*) para ser utilizado em dispositivos de decodificação de vídeos H.264/AVC.

Palavras-chave: CABAC, H.264, Decodificação, SoC

Abstract

The ITU-T H.264/MPEG-4 Part 10 (Advanced Video Coding) video coding standard, originally published in 2003 by the specialists of Joint Video Team (JVT), is used in technologies like Blu-Ray, defunct HD-DVD standard and the Brazilian Digital Television System. This Standard brings some improvements when compared with previous video standards (MPEG-1, MPEG-2, MPEG-4, H.261 e H.263), among them, an innovative entropy coder based on a binary arithmetic coding, called Context-based Adaptive Binary Arithmetic Coding (CABAC). These new improved features are responsible for the great efficiency of the standard, reducing up to 50% the bit-rate needed for transmission of a coded video stream when compared with the MPEG-4 video standard, at the price of greater computational complexity.

The complexity of CABAC process is exemplified by previous works that reports the need of a clock frequency above 100 MHz for a typical Digital Signal Processor (DSP) to support a CABAC decoder implementation for real-time decoding of a video with a D1 resolution (720x480) at frame-rate of 30 frames per second. The CABAC decoder process is primordially sequential and the acceleration of the execution is normally reached by optimized solutions including dedicated hardware.

In this work it is presented a study about the CABAC decoding process and propose a dedicated hardware architecture based on optimized modules to produce parallel decoding. The solution supports decoding of H.264/AVC videos with 1080p resolution (1920x1080) at frame-rate of 30 frames per second. The decoder may be integrated with a complete H.264/AVC video decoder as a component of a System-on-a-Chip (SoC).

Keywords: CABAC, H.264, Decoder, SoC

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviações

ALUT	Adaptive Lookup-Table
ANSI	American National Standards Institute
ASO	Arbitrary Slice Ordering
AVC	Advanced Video Coding
CABAC	Context-based Adaptive Binary Arithmetic Coding
CAVLC	Context-based Adaptive Variable Length Coding
CIF	Commom Intermediate Format
CM	Compensação de Movimento
Cb	Chroma blue
Cr	Chroma red
DSP	Digital Signal Processor
EGk	k-th order Exponential Golomb
FIFO	First-In, First-Out
FL	Fixed-Length
FMO	Flexible Macroblock Ordering
FOD	First One Detector
FPGA	Field Programmable Gate Array
HDTV	High Definition Television
IBM	International Business Machines
IEEE	Institute of Electrical and Electronics Engineers
IP	Intellectual Property
ISO	International Organization for Standardization
ITU	International Telecommunication Union
JBIG	Joint Bi-level Image Experts Group
JPEG	Joint Photographic Experts Group
JVT	Joint Video Team
LIFO	Last-In, First-Out
LPS	Least Probable Symbol
LZA	Leading Zero Antecipator
MB	Macroblock
MBAFF	Macroblock Adaptive Frame-Field
MIPS	Milhões de Instruções por Segundo
MPEG	Movie Picture Experts Group
MPS	Most Probable Symbol
NAL	Network Abstraction Layer
PCM	Pulse-code Modulation
PicAFF	Picture Adaptive Frame-Field
RAM	Randon-access Memory
ROM	Read-only Memory
SoC	System-on-a-Chip
UEGk	Unary / k-th order Exponential Golomb
ULA	Unidade Lógica e Aritmética
VCEG	Video Coding Experts Group
VCL	Video Coding Layer
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
VLC	Variable Length Coding

Capítulo 1

Introdução

A alta demanda por conteúdo visual para ser acessado pelos atuais meios digitais provoca a necessidade da criação de formatos de codificação de vídeo que possibilitem o armazenamento e transmissão dos sinais de vídeos, os quais possuem cada vez maior qualidade e definição. Esses formatos buscam reduzir a quantidade de informação que precisa ser armazenada ou transmitida para reconstruir uma versão semelhante ao vídeo original. Dentre os formatos de codificação de vídeo mais recentes, destaca-se o definido pelo padrão ITU-T H.264 / MPEG4 *Part* 10 *Advanced Video Coding*. Conhecido abreviadamente como H.264 / AVC, este formato trouxe uma redução de aproximadamente 50% na taxa de *bits* de informação por segundo necessária para armazenar/transmitir um vídeo de qualidade equivalente, quando comparado com os padrões anteriores e para resoluções específicas. (Wiegand et al. (2003))

O grupo de trabalho *Joint Video Team* (JVT), formado por integrantes dos dois principais grupos responsáveis pelo principais padrões de vídeos atuais, gerou o H.264 / AVC a partir do formato anterior H.263+, acrescentando e alterando alguns de seus componentes. Um de seus novos componentes é apontado como um dos principais responsáveis pela melhoria alcançada, trata-se do codificador entrópico CABAC (*Context-based Adaptive Binary Arithmetic Coding*). Essa melhoria é paga pelo aumento da complexidade computacional necessária para sua execução. (Chen, Chang e Lin (2005))

O CABAC foi criado especialmente para o padrão H.264 / AVC e portanto possui uma série de customizações. Essas customizações possibilitam a codificação com redução eficiente no tamanho dos elementos processados, mas também acarretam a necessidade de processamento massivo para cada *bit* processado. O resultado é a necessidade de processadores de propósito geral com alta freqüência para realizar a decodificação de vídeos de alta definição em tempo real. (Yi e Park (2007)). Assim, busca-se desenvolver alternativas para processadores específicos ou criar dispositivos de *hardware* capazes de atender tal demanda.

A natureza seqüencial do processo de codificação ou decodificação do CABAC, impossibilita alternativas simples de otimização por paralelismo. O que provoca o surgimento de várias propostas de arquiteturas de hardware

dedicado que buscam dar uma solução ao problema. A maioria das soluções avaliadas relatam que a chave para a otimização de todo o processo está na aceleração do processamento dos elementos codificáveis ou decodificáveis mais freqüentes e/ou de maior comprimento. O processamento desses elementos é acelerado através de especulação, a qual é suportada por uma série de mecanismos implementados. A idéia de otimização geral dos elementos é rechaçada pelo custo de complexidade avaliado.

O trabalho a ser apresentado nessa dissertação tem como objetivo gerar uma arquitetura alternativa para um decodificador CABAC otimizado que suporte pelo menos os perfis *High* simples e *Main* e os 3 principais tipos de *slices*. O desempenho a ser alcançado será o necessário para permitir que um decodificador H.264 / AVC reconstrua vídeos de resolução 1080p codificados utilizando-se o perfil *main* nível 4.0 em tempo real.

Para tanto, este documento traz em seu conteúdo os diversos elementos que formaram o trabalho executado. O texto está dividido em 6 capítulos. O primeiro capítulo introduz o assunto e o objetivo do trabalho. O segundo capítulo traz a fundamentação teórica que permite compreender o padrão H.264 / AVC, a codificação aritmética e detalhes mais específicos do CABAC. O terceiro capítulo apresenta uma seleção de trabalhos semelhantes, que foram pesquisados, avaliados e formaram base de experiência para a arquitetura desenvolvida. O quarto capítulo descreve a arquitetura, seu funcionamento e operação. O quinto capítulo relata a metodologia utilizada e quais foram os resultados obtidos. O sexto capítulo conclui o trabalho com um resumo e avaliação dos resultados obtidos e apresenta sugestões de melhorias na arquitetura, que podem ser desenvolvidas como trabalhos futuros.

Capítulo 2

Fundamentação

2.1 Padrão ITU-T H.264 / MPEG4 *Part* 10 *Advanced Video Coding*

O padrão de codificação de vídeo H.264/AVC foi desenvolvido pelo grupo de trabalho *Joint Video Team* (JVT). Este grupo é formado por integrantes dos dois principais grupos de estudo de codificadores de vídeo: *Moving Picture Experts Group* (MPEG) da *International Organization for Standardization* (ISO) e o *Video Coding Experts Group* (VCEG) da *International Telecommunications Union* (ITU). O MPEG é responsável pelos padrões de vídeo MPEG-1, MPEG-2 e MPEG-4 e o padrão de codificação de áudio MPEG-2 *Layer* 3 (MP3). O VCEG é responsável pelos padrões de codificação de vídeo para telecomunicação H.261, H.263 e H.263+.

O H.264/AVC surgiu a partir do H.26L, sucessor do H.263+. Em 2001 durante o desenvolvimento do MPEG-4, o MPEG estava procurando uma ferramente para codificação avançada de vídeo, após inicialmente optar pelo H.263, decidiu adotar o H.26L e criar o JVT, juntamente com o VCEG, para evoluir o padrão para o que mais tarde se tornaria o H.264/AVC. O trabalho produzido pelo JVT gerou originalmente em 2003 duas publicações: ITU-T H.264 e MPEG-4 Part 10 Advanced Video Coding. Esse padrão geralmente é referenciado pelo seu nome abreviado H.264/AVC.

As melhorias introduzidas por essa nova técnica de codificação de vídeo permite codificar um vídeo com uma redução de aproximadamente 50% na taxa de bits necessária para uma mesma qualidade visual, em relação aos padrões anteriores e resoluções específicas. (Wiegand et al. (2003))

Semelhante aos padrões anteriores (MPEG-1, MPEG-2, MPEG-4, H.261 e H.263), o H.264 / AVC possui basicamente os seguintes elementos funcionais:

- Módulo de Predição (Estimativa de Movimento - EM, Compensação de Movimento - CM e Predição intra);
- Transformada - T;
- Quantizador - Q;

- Codificador Entrópico. (Codificador que permite compressão dos dados sem perda de informação, baseado em códigos adaptáveis à distribuição de probabilidades dos símbolos a serem codificados) (Moffat, Bell e Witten (1995))

Assim como seus antecessores (H.261, H.263 e H.263+), o H.264/AVC processa vídeos com quadros retangulares (*frames*) e é baseado na codificação de blocos de amostras, chamados Macroblocos. Cada Macrobloco é composto de blocos de 16x16 amostras de luma (luminância) e seus dois respectivos blocos de amostras de chroma (crominância). Cada imagem (*frame*) do vídeo em processamento é dividida em um ou mais conjuntos de Macroblocos, chamados *Slices*. Uma inovação introduzida no H.264/AVC é o particionamento dos Macroblocos em blocos menores, que podem ser empregados na otimização da codificação dependendo do tipo de imagem que está sendo processada.

Cada grupo *Slice* possui as principais informações necessárias para a codificação ou decodificação de seus Macroblocos de imagens. O codificador H.264/AVC oferece um ordenamento flexível dos Macroblocos (*Flexible Macroblock Ordering* - FMO) para composição de *Slices*, permitindo codificação mais eficiente que os codificadores anteriores. São definidos cinco tipos de *Slices* para o padrão:

1. I (Intra) - Formado apenas por Macroblocos do tipo I, em que cada Macrobloco ou partição do Macrobloco é previsto a partir de dados, codificados previamente, do próprio *Slice*. Presente em todos os perfis;

2. P (*Predicted*) - Formado por Macroblocos I e/ou Macroblocos P, em que cada Macrobloco, ou partição do Macrobloco, é previsto de uma lista de imagens de referência. Presente em todos os perfis;

3. B (*Bi-predictive*) - Formado por Macroblocos I e/ou Macroblocos B, em que cada Macrobloco, ou partição de Macrobloco, é previsto a partir de imagens de referência contidas em até duas listas. Presente apenas nos perfis *Main*, *High* e *Extended*;

4. SP (*Switching* P) - *Slice* que facilita a troca entre *streams* codificados e que contém Macroblocos do tipo P e/ou I. Presente apenas nos perfis *High* e *Extended*;

5. SI (*Switching* I) - *Slice* que facilita a troca entre *streams* codificados e que contém Macroblocos SI, que é um tipo especial de Macrobloco codificado *intra-Slice*. Presente apenas nos perfis *High* e *Extended*.

O processo de codificação é baseado na predição da imagem a ser codificada. A predição consiste na definição ou cálculo de um Macrobloco similar, que é tomado como predição do Macrobloco a ser codificado. Uma vez definida a predição, a diferença entre o Macrobloco previsto e o real é calculada, produzindo uma matriz de diferenças dos valores das amostras. A seguir, essa diferença é operada por uma transformada, que re-expressa a matriz

das diferenças em termos de uma matriz com os respectivos componentes de freqüência. Os coeficientes da transformada são então quantizados para remover informações desprezáveis para a decodificação, obedecendo parâmetros e limites que determinam a distorção permitida no sinal. As diversas informações geradas pela codificação (Elementos Sintáticos), tais como os coeficientes quantizados, informações de controle da codificação, parâmetros do vídeo e dos dados de predição são posteriormente reorganizados e processados por um codificador entrópico que gera códigos binários de tamanhos reduzidos como representação do vídeo codificado em forma de um *bitstream*.Iain (2003)

O codificador H.264/AVC utiliza dois tipos básicos de predição: a intra-imagem e a inter-imagem. A predição intra-imagem utiliza informações da própria imagem sendo codificada para gerar a imagem prevista. A predição inter-imagem utiliza imagens previamente processadas para gerar a imagem prevista.

O modo de predição intra-imagem utiliza Macroblocos ou partes dos Macroblocos previamente codificados da própria imagem para executar a predição do bloco sendo codificado. Possui nove tipos de predição para blocos de 4x4 amostras luma, quatro tipos de predição para blocos de 16x16 amostras luma e quatro tipos de predição para blocos de 8x8 amostras chroma.

O modo de predição inter-imagem utiliza Macroblocos ou partes dos Macroblocos previamente codificados para determinar uma estimativa de movimento do Macrobloco ou parte do Macrobloco em codificação. A partir dos vetores de movimentos calculados é possível obter, através de compensação de movimento aplicado nos Macroblocos ou partes dos Macroblocos de imagens previamente codificadas, uma predição da imagem atual. A codificação H.264/AVC permite a divisão dos Macroblocos em até quatro tipos de partições de Macroblocos (sub-Macroblocos), que por sua vez ainda podem ser divididos novamente em outros quatro tipos de sub-Macroblocos. A precisão do vetor de movimento pode chegar até a um quarto de tamanho do bloco de amostra luma. (Iain (2003)).

O padrão apresenta as seguintes melhorias nos métodos para predição da imagem a ser codificada quando comparada aos padrões anteriores (Wiegand et al. (2003)):

- Maior flexibilização na seleção dos blocos que serão utilizados para a predição de movimento de blocos. Utilização de blocos de até 4x4 amostras luma (luminância);
- Compensação de movimento com acurácia de um quarto do tamanho da amostra;
- Vetores de movimento aplicados às bordas da imagem;
- Capacidade de utilizar múltiplas imagens de referência para a compensação de movimento da codificação inter. Utilização tanto para imagens geradas por predição ou bi-predição (utilização de duas imagens de referências);

- Desacoplamento da ordem de referência da ordem de exibição das imagens. Possibilitando a flexibilização das referências, limitada apenas pela capacidade de memória do decodificador;
- Desacoplamento da restrição do tipo de predição utilizado para a codificação da imagem e sua utilização como referência para outras imagens. Possibilidade de utilização de imagens codificadas com referência dupla como referências para outras imagens codificadas com inter predição;
- Possibilidade de utilização de “pesos” e "deslocamentos"aplicados aos vetores utilizados na compensação de movimento pelo codificador, auxiliando, por exemplo, a codificação de imagens com efeito de "*fade*";
- Aprimoramento na compensação de movimento com a possibilidade de inferência de regiões "ignoradas"(*Skipped*);
- Novas técnicas para predição intra-imagem;
- Filtro para redução de "efeito-de-bloco"aprimorado e interno ao *loop* de predição inter-imagem;
- Utilização padronizada de transformada de bloco com tamanho de 4x4 amostras, ao invés da tradicional transformada de 8x8 amostras. Sendo que o perfil *High* permite a utilização da transformada de 8x8 amostras;
- Transformada hierárquica de bloco, pela qual é possível variar o tamanho dos blocos da transformada para otimização de determinadas condições;
- Transformada com palavras de tamanho reduzido, necessitando de processamento apenas com palavras de 16 bits;
- Inversa exata da transformada. O padrão H.264/AVC é o primeiro a oferecer uma transformada com uma inversa que obtém valores exatos aos originais. Possibilita maior qualidade efetiva do vídeo e a igualdade na qualidade do vídeo para todos os decodificadores;
- Codificador Entrópico Aritmético. O padrão utiliza um codificador aritmético adaptativo otimizado (*Context-based Adaptive Binary Arithmetic Coding* - CABAC) que possibilita uma maior compressão quando comparado aos codificadores entrópicos dos padrões anteriores. Presente nos perfis *Main* e *High*;
- Codificador entrópico baseado em códigos de tamanhos variáveis com utilização de contextos adaptativos (*Context-based Adaptive Variable Length Coding* - CAVLC). Presente em todos os perfis.

Os fluxos principais de codificação e decodificação estão representados nas Figuras 2.1 e 2.2 respectivamente. Os elementos dos fluxos estão descritos na Tabela 2.1.

Elemento	Descrição
Fn (Atual)	*Frame* (ou campo de imagem entrelaçada) atual em processo de codificação.
F'n-1 (Referência)	*Frame* (ou campo de imagem entrelaçada) previamente codificado e decodificado e utilizado como referência para predição inter.
F'n (Reconstruído)	*Frame* (ou campo de imagem entrelaçada) reconstruído a partir do processo de decodificação para ser utilizado na lista de referência para a predição inter-imagem.
EM	Módulo de Estimativa de Movimento. Calcula os vetores de movimento.
CM	Módulo de Compensação de Movimento. Calcula o Macrobloco ou sub-Macrobloco de predição a partir das referências e os vetores de movimento.
P	Macrobloco ou sub-Macrobloco gerado pelo processo de predição.
Dn	Diferença residual do Macrobloco ou sub-Macrobloco previsto P e o original.
T	Transformada. São utilizadas 3 tipos de transformadas específicas para cada tipo de resíduo: Transformada Hadamard 4x4; Transformada Hadamard 2x2; Transformada baseada na Transformada de Cosseno Discreta.
Q	Quantizador.
X	Coeficientes quantizados da transformada.
NAL	Camada de abstração de rede (*Network Abstraction Layer*). Definição de empacotamento das informações codificadas.
Q-1	Inverso do quantizador.
T-1	Inversa da transformada.
D'n	Diferença residual reconstruída.
uF'n	*Frame* (ou campo de imagem entrelaçada) reconstruído sem o uso do filtro de redução de efeito de bloco.
Filtro	Filtro de redução de efeito de bloco.

Tabela 2.1: Elementos da codificação/decodificação H.264 / AVC

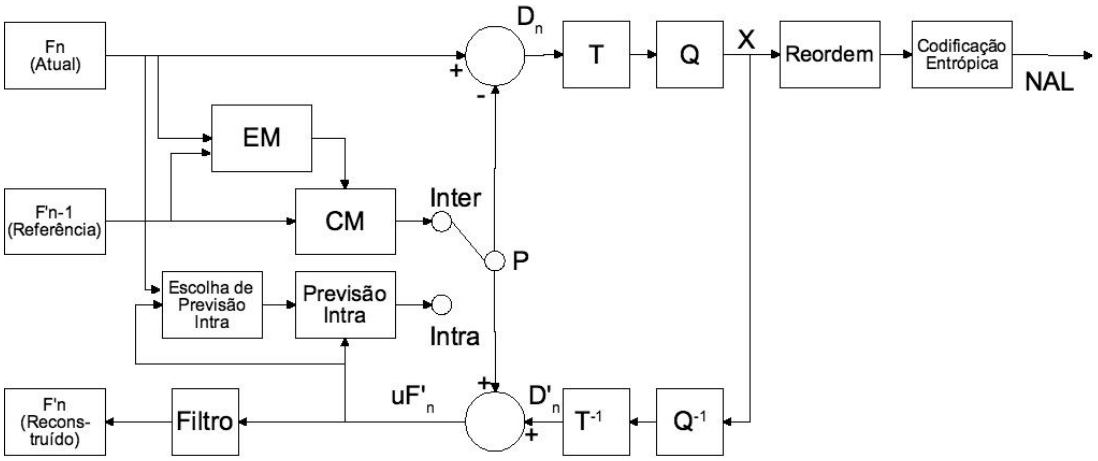

Figura 2.1: Fluxo de codificação H.264 / AVC

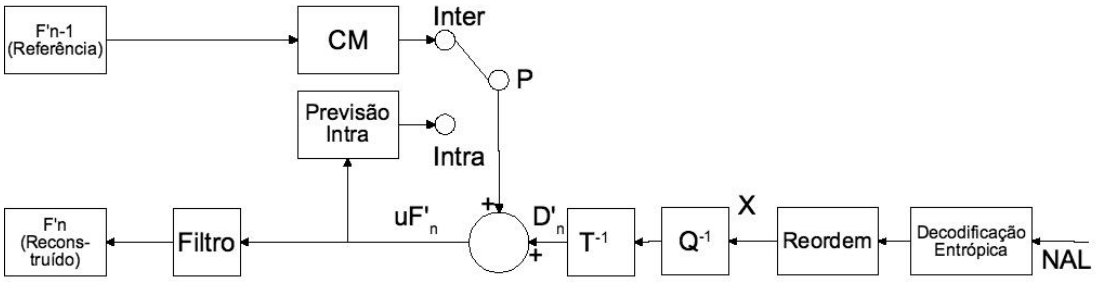

Figura 2.2: Fluxo de decodificação H.264 / AVC

O padrão H.264/AVC descreve dois tipos de codificação entrópica para processamento dos elementos sintáticos. O tipo mais simples se chama *Context-based Adaptive Variable Length Coding* - CAVLC e utiliza Tabelas de códigos binários de tamanhos variáveis para representar os diversos valores dos elementos sintáticos. A escolha da Tabela e código depende de elementos sintáticos previamente codificados. Este tipo de adaptação provê uma codificação mais otimizada que um simples mapeamento de código de tamanho variável tradicional.

O segundo codificador entrópico descrito se chama *Contex-based Adaptive Binary Arithmetic Coding* - CABAC e utiliza uma máquina de codificação aritmética binária para codificar os valores dos elementos sintáticos (ITU (2005)). Assim como o CAVLC, este codificador utiliza contextos para realizar a codificação dos elementos de forma adaptativa aos elementos previamente processados. O modelo de probabilidade é ajustado dinamicamente para cada *bit* processado e é registrado nos diversos contextos relacionados a cada elemento sintático. A máquina de codificação aritmética foi desenvolvida para ser de baixa complexidade, utilizando apenas operações de soma, subtração e deslocamentos. Comparado ao codificador CAVLC, o CABAC oferece uma redução de 5 a 15% na taxa de *bits*. (Wiegand et al. (2003))

O codificador H.264/AVC define ainda perfis e níveis de operação. Os

perfis definem conjuntos de ferramentas e algoritmos de codificação. Os níveis estabelecem restrições para os valores dos elementos sintáticos e parâmetros do *bitstream* codificado. Uma implementação de decodificador do padrão H.264/AVC que esteja em conformidade com determinado perfil, deve implementar todas as suas funcionalidades definidas. No caso do codificador, este não necessita implementar todas as funcionalidades, mas precisa produzir *bitstreams* que estejam em conformidade com os recursos disponibilizados pelo perfil para o qual foi codificado.

Originalmente o padrão define três tipos de perfis: (Wiegand et al. (2003), Sullivan, Topiwala e Luthra (2004))

- ***Baseline***; Entre outras funcionalidades, suporta codificação com predição do tipo intra-imagem e inter-imagem. *Slices* do tipo I e P e codificador entrópico do tipo CAVLC.
  Aplicações: Vídeo-Telefonia, Vídeo-Conferência e vídeo para aparelhos com comunicação sem fio.

- ***Main***; Entre outras funcionalidades, suporte para codificação de vídeo entrelaçado, predição inter-imagem através de *slices* do tipo B, predição inter-imagem com pesos e codificador entrópico do tipo CABAC ou CAVLC.
  Aplicações: Vídeo para TV de difusão e vídeo armazenado em mídia digital.

- ***High Profile***. Perfil adicional direcionado para uso de vídeos em alta resolução. Suporta todas as funcionalidades do perfil *Main* com adição do uso de transformada adaptável ao tamanho do bloco e matrizes escaláveis de quantização. Possui uma série de mais 3 sub-perfis (High 10, High 4:2:2, High 4:4:4) que entre outras funcionalidades, suportam uma maior quantidade de amostras de crominância e maior precisão das amostras.
  Aplicações: Vídeo em alta resolução para TV de difusão e vídeo em alta resolução armazenado em mídia digital.

- ***Extended Profile***. Entre outras funcionalidade, suporta *slices* do tipo SP e SI e possui recursos para aumentar a tolerância a erros. Codificador entrópico CABAC ou CAVLC.
  Aplicações: *Stream* de vídeo.

A Tabela 2.2 apresenta a relação de algumas funcionalidades e sua disponibilidade para cada um dos perfis do H.264 / AVC. (ITU (2005), Sullivan, Topiwala e Luthra (2004))

A Tabela 2.3 apresenta as restrições de codificação para os perfis/níveis. (ITU (2005))

	Perfis						
Funcionalidades	***Baseline***	***Extended***	***Main***	***High***	***High 10***	***High 4:2:2***	***High 4:4:4***
Slices I e P	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Slices B	Não	Sim	Sim	Sim	Sim	Sim	Sim
Slices SI e SP	Não	Sim	Não	Não	Não	Não	Não
Múltiplos *frames* de referência	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Filtro para redução "efeito-de-bloco"intra-*Loop*	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Codificador entrópico CAVLC	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Codificador entrópico CABAC	Não	Não	Sim	Sim	Sim	Sim	Sim
Ordenamento flexível de Macroblocos (FMO)	Sim	Sim	Não	Não	Não	Não	Não
Ordenamento arbitrário de *Slices* (ASO)	Sim	Sim	Não	Não	Não	Não	Não
Slices Redundantes (RS)	Sim	Sim	Não	Não	Não	Não	Não
Partição de dados	Não	Sim	Não	Não	Não	Não	Não
Codificação entrelaçada (PicAFF, MBAFF)	Não	Sim	Sim	Sim	Sim	Sim	Sim
Formato de Chroma 4:2:0	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Formato de vídeo monocromático (4:0:0)	Não	Não	Não	Sim	Sim	Sim	Sim
Formato de Chroma 4:2:2	Não	Não	Não	Não	Não	Sim	Sim
Formato de Chroma 4:4:4	Não	Não	Não	Não	Não	Não	Sim
Amostras com precisão de 8 *Bits*	Sim	Sim	Sim	Sim	Sim	Sim	Sim
Amostras com precisões de 9 e 10 *Bits*	Não	Não	Não	Não	Sim	Sim	Sim
Amostras com precisões de 11 e 12 *Bits*	Não	Não	Não	Não	Não	Não	Sim
Transformada adaptativa de 8x8 p/ 4x4	Não	Não	Não	Sim	Sim	Sim	Sim
Matrizes de quantização	Não	Não	Não	Sim	Sim	Sim	Sim
Controle de quantização separado para Cb e Cr	Não	Não	Não	Sim	Sim	Sim	Sim
Transformada para resíduos de amostras de cor	Não	Não	Não	Não	Não	Não	Sim
Codificação preditiva sem perda	Não	Não	Não	Não	Não	Não	Sim
Funcionalidades	***Baseline***	***Extended***	***Main***	***High***	***High 10***	***High 4:2:2***	***High 4:4:4***

Tabela 2.2: Tabela de funcionalidades do H.264 para cada um dos perfis.

Nível	**Máximo de Macroblocos por segundo**	**Máximo do tamanho do frame (Macroblocos)**	**Máximo bit rate do vídeo (VCL) para perfis *Baseline*, *Extended* e *Main*.**	**Máximo *bit rate* do vídeo (VCL) para perfil *High***	**Máximo *bit rate* do vídeo (VCL) para perfil *High* 10**	**Máximo *bit rate* do vídeo (VCL) para perfis *High* 4:2:2 e *High* 4:4:4**	**Exemplos de resoluções @ *frame rate* (Qtd. máxima de *frames* armazenados) para o nível**
1	1485	99	64 kbit/s	80 kbit/s	192 kbit/s	256 kbit/s	128x96@30.9(8) 176x144@15.0(4)
1b	1485	99	128 kbit/s	160 kbit/s	384 kbit/s	512 kbit/s	128x96@30.9(8) 176x144@15.0(4)
1.1	3000	396	192 kbit/s	240 kbit/s	576 kbit/s	768 kbit/s	176x144@30.3(9) 320x240@10.0(3) 352x288@7.5 (2)
1.2	6000	396	384 kbit/s	480 kbit/s	1152 kbit/s	1536 kbit/s	320x240@20.0(7) 352x288@15.2(6)
1.3	11880	396	768 kbit/s	960 kbit/s	2304 kbit/s	3072 kbit/s	320x240@36.0(7) 352x288@30.0(6)
2	11880	396	2 Mbit/s	2.5 Mbit/s	6 Mbit/s	8 Mbit/s	320x240@36.0(7) 352x288@30.0(6)
2.1	19800	792	4 Mbit/s	5 Mbit/s	12 Mbit/s	16 Mbit/s	352x480@30.0(7) 352x576@25.0(6)
2.2	20250	1620	4 Mbit/s	5 Mbit/s	12 Mbit/s	16 Mbit/s	352x480@30.7(10) 352x576@25.6(7) 720x480@15.0(6) 720x576@12.5(5)
3	40500	1620	10 Mbit/s	12.5 Mbit/s	30 Mbit/s	40 Mbit/s	352x480@61.4(12) 352x576@51.1(10) 720x480@30.0(6) 720x576@25.0(5)
3.1	108000	3600	14 Mbit/s	14 Mbit/s	42 Mbit/s	56 Mbit/s	720x480@80.0(13) 720x576@66.7(11) 1280x720@30.0(5)
3.2	216000	5120	20 Mbit/s	25 Mbit/s	60 Mbit/s	80 Mbit/s	1280x720@60.0(5) 1280x1024@42.2(4)
4	245760	8192	20 Mbit/s	25 Mbit/s	60 Mbit/s	80 Mbit/s	1280x720@68.3(9) 1920x1080@30.1(4) 2048x1024@30.0(4)
4.1	245760	8192	50 Mbit/s	62.5 Mbit/s	150 Mbit/s	200 Mbit/s	1280x720@68.3(9) 1920x1080@30.1(4) 2048x1024@30.0(4)
4.2	522240	8704	50 Mbit/s	62.5 Mbit/s	150 Mbit/s	200 Mbit/s	1920x1080@64.0(4) 2048x1080@60.0(4)
5	589824	22080	135 Mbit/s	168.75 Mbit/s	405 Mbit/s	540 Mbit/s	1920x1080@72.3(13) 2048x1024@72.0(13) 2048x1080@67.8(12) 2560x1920@30.7(5) 3680x1536@26.7(5)
5.1	983040	36864	240 Mbit/s	300 Mbit/s	720 Mbit/s	960 Mbit/s	1920x1080@120.5(16) 4096x2048@30.0(5) 4096x2304@26.7(5)
Nível	**Máximo de Macroblocos por segundo**	**Máximo do tamanho do frame (Macroblocos)**	**Máximo bit rate do vídeo (VCL) para perfis *Baseline*, *Extended* e *Main*.**	**Máximo *bit rate* do vídeo (VCL) para perfil *High***	**Máximo *bit rate* do vídeo (VCL) para perfil *High* 10**	**Máximo *bit rate* do vídeo (VCL) para perfis *High* 4:2:2 e *High* 4:4:4**	**Exemplos de resoluções @ *frame rate* (Qtd. máxima de *frames* armazenados) para o nível**

Tabela 2.3: Relação de perfis/níveis e restrições de suporte.

As sessões seguintes irão apresentar de forma mais detalhada o funcionamento do codificador entrópico CABAC.

2.2 Codificação aritmética

2.2.1 Visão geral

A codificação aritmética utiliza como princípio um modelo de probabilidades de ocorrência dos símbolos que podem compor a mensagem a ser codificada. Iniciando a partir do intervalo [0,1), subdivide-se este intervalo proporcionalmente às probabilidades dos símbolos. (MacKay (2004))

Inicia-se o processo através da identificação do primeiro símbolo da mensagem, então escolhe-se o subintervalo correspondente a probabilidade do símbolo identificado. Toma-se o subintervalo escolhido como o novo intervalo principal, subdivide-se este proporcionalmente as probabilidades dos símbolos, identifica o símbolo seguinte e escolhe o subintervalo correspondente. Executa este processo para todos os símbolos da mensagem. Ao final obtém-se um subintervalo que corresponde as probabilidade da ocorrência de toda a seqüência de símbolos da mensagem. É possível então escolher um valor deste último subintervalo, de forma que sua representação seja aquela que necessite da menor quantidade de informação para ser transmitida.

A Figura a seguir exemplifica a codificação da mensagem "PALAVRA"para o modelo de probabilidade apresentado na Tabela.

Símbolo	Probabilidade
A	$\frac{3}{7}$
P	$\frac{1}{7}$
L	$\frac{1}{7}$
V	$\frac{1}{7}$
R	$\frac{1}{7}$

Tabela 2.4: Modelo probabilidade

Quando se utiliza precisão fixa, o processo precisa realizar normalização dos valores dos intervalos.

O processo de codificação aritmética binária corresponde a situação onde somente dois símbolos compõe o alfabeto da mensagem. A codificação aritmética adaptativa corresponde a situação onde a probabilidade dos símbolos não é estática, variando e adaptando-se a medida que a mensagem é codificada. (MacKay (2004))

A decodificação aritmética funciona de forma semelhante à codificação: as entradas para o processo são o valor de probabilidade final da codificação e o modelo de probabilidade dos símbolos que compõem a mensagem a

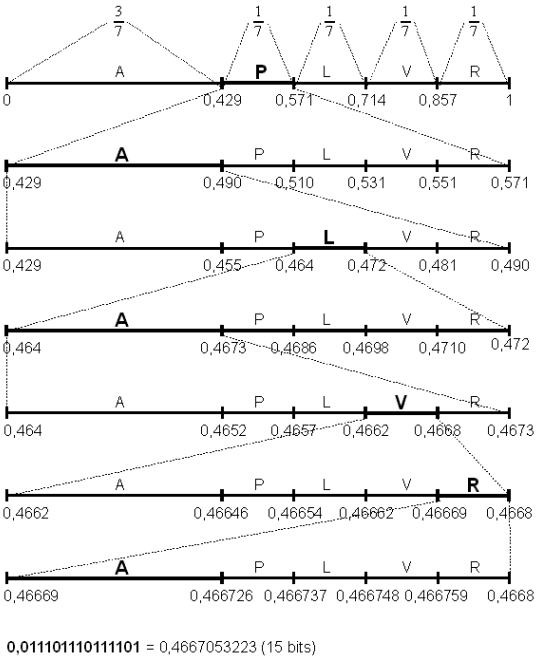

Figura 2.3: Exemplo codificação aritmética da mensagem (PALAVRA)

ser decodificada. Inicia-se com um intervalo principal [0,1), subdividindo-o em subintervalos proporcionais às probabilidades dos símbolos. Identifica-se o subintervalo no qual o valor de probabilidade esteja incluso, e então obtém-se o símbolo correspondente a este subintervalo. Toma-se o subintervalo identificado como o novo intervalo principal, subdividindo-o da mesma forma anterior, e continuando o processo da mesma forma que o passo anterior para cada símbolo. O término do processo pode ser através da decodificação de um símbolo exclusivo de finalização, ou através de um tamanho total pré-definido ou analisando os resultados finais a cada passo até encontrar um valor desejado.

2.2.2 Histórico

Os primórdios da codificação aritmética datam de 1948, quando em um artigo, Claude Shannon observou que mensagens de tamanho N símbolos poderiam ser codificadas, ordenando-as inicialmente pelo valor de probabilidades dos símbolos e enviando o valor da probabilidade acumulada. O código resultante era uma fração binária que poderia ser decodificada através de comparação de magnitude. (Jr. (1984))

Em 1963, Peter Elias observou que o processo descrito por Shannon também funcionava sem a necessidade de ordenação dos símbolos. Observou também que a probabilidade cumulativa de uma mensagem de N símbolos poderia ser calculada de forma iterativa a partir das probabilidades individuais dos símbolos e a probabilidade acumulada da mensagem de N-1 símbolos.

Essas abordagens iniciais necessitavam de precisão incremental propor-

cional ao comprimento das mensagens, ou quando utilizando precisão fixa, o tempo de codificação dos símbolos aumentava linearmente ao tamanho da mensagem codificada.

Em 1976, Jorma Rissanen propôs uma codificação aritmética LIFO (*Last In First Out*) onde o tempo para codificar um símbolo era fixo, utilizando também aritmética de precisão fixa. No mesmo ano, R. Pasco propôs um codificador aritmético FIFO (*First In First Out*) com precisão fixa, utilizando a mesma idéia aplicada por Rissanen. Neste trabalho, Pasco manteve a *string* do código na memória do computador até o final da codificação, dessa forma um *carry-over* pode ser propagado por toda a cadeia de *bits* necessários.

Nos trabalhos citados anteriormente, são utilizadas as probabilidades, obtidas ou presumidas, dos símbolos sendo codificados. Em trabalhos publicados em 1979 e 1981, G. G. Langdon e Rissanen introduziram a idéia de modelo de codificação, os quais eram apenas baseados nas probabilidades dos símbolos. A separação dos modelos de codificação e as probabilidades dos símbolos permitiu a simplificação da implementação do codificador e uma maior margem de ajustes de compensação para os modeladores de futuros codificadores. As propostas de Langdon e Rissanen também demonstraram a possibilidade de se bloquear o *carry-over*, o que foi feito através de um método chamado de *bit-stuffing*. A análise realizada nos trabalhos permitiram, a Langdon e Rissanen, generalizar a codificação aritmética como uma família que engloba vários outros codificadores pré-aritméticos.

2.2.3 Tipos de Codificadores aritméticos

Apesar do princípio original do codificador aritmético apresentado por Langdon e Rissanen em 1979 ser utilizado nos codificadores aritméticos posteriores, algumas melhorias e alterações foram propostas, criando sub-categorias de codificadores aritméticos (Marpe, Kirchhoffer e Marten (2006), Marpe, Schwarz e Wiegand (2003)):

- *Q-Coder*
  O codificador aritmético original, proposto por Langdon e Rissanen; (Jr. e Rissanen (1981))

- *QM-Coder*
  Proposta de codificador utilizado nos padrões JBIG e JPEG *Arithmetic*; (Barni (2006))

- *MQ-Coder*
  Proposta de codificador utilizado nos padrões JBIG2 e JPEG2000; (Acharia e Tsai (2005))

- *M-Coder*
  Proposta de codificador aritmético apresentado no apêndice E do padrão H.263 e o codificador CABAC, utilizado no padrão H.264/AVC. (Marpe, Kirchhoffer e Marten (2006), Marpe, Schwarz e Wiegand (2003))

As sub-sessões seguintes relatam maiores informações sobre o *Q-Coder* e o *M-Coder*. Os codificadores *QM-Coder* e *MQ-Coder* são variações do *Q-Coder* e não são alvo de pesquisa neste trabalho. (Marpe, Kirchhoffer e Marten (2006))

2.2.4 *Q-Coder*

Proposto por Glen G. Langdon Jr. e Jorma Rissanen em 1981 (Jr. e Rissanen (1981)), este codificador aritmético descrito em um artigo sobre compressão de imagens preto-e-branco, utiliza aritmética com precisão finita, aproxima as probabilidades dos símbolos menos prováveis através de quantização (Q) para frações potências de $\frac{1}{2}$ (substituindo as multiplicações por deslocamentos de *bits* para direita) e propõe um método chamado de "*bit stuffing*"para impedir uma longa propagação de *bits* de *carry-over*.

Ao adotar o uso de precisão fixa, o *Q-coder* adota a idéia de renormalização do intervalo para manter os valores dentro da precisão utilizada. A regra adotada é manter sempre o *bit* mais significativo do tamanho do intervalo como igual a 1, realizando deslocamentos para a esquerda quando necessário. A renormalização é realizada tanto pelo codificador quanto pelo decodificador e o deslocamento deve ser realizado também no ponteiro de início do intervalo, para manter as operações aritméticas operando no mesmo intervalo de significância.

A utilização de operações aritméticas acarreta, em algumas circunstâncias, na propagação de um *bit* 1 para as casas mais significativas. Para limitar o tratamento da propagação, adotou-se o mecanismo de "*bit-stuffing*". Do ponto de vista do codificador, esse mecanismo monitora a quantidade de *bits* consecutivos que possuem o valor 1. Caso essa *string* de *bits* 1 tenha tamanho igual ao tamanho de um *buffer* pré-definido, um *bit* 0 é inserido no *bit* de posição seguinte. Dessa forma a propagação futura de um *bit* 1 será "bloqueada"pelo *bit* 0, permitindo que o codificador não tenha que tratar a propagação de *bits* em uma *string* de comprimento superior ao tamanho definido do *buffer*. O decodificador precisa monitorar os valores dos *bits*. Caso ocorra a situação de uma *string* de *bits* 1 com tamanho igual ao do *buffer*, deverá verificar o próximo *bit* (o *stuffed bit*), caso o mesmo seja igual a 0, não ocorreu a propagação de *bits* 1, então deve removê-lo e continuar o processamento. Caso o *bit* seja igual a 1, deve removê-lo e somar 1 ao intervalo menos significativo da *string* já lida.

A decodificação é realizada através de comparações do valor da fração codificada e os intervalos representantes dos elementos. Caso o ponteiro esteja interno a um dos intervalos, o elemento referente será o decodificado. No caso de ser o elemento menos provável, ajusta-se o ponteiro subtraindo o tamanho do intervalo correspondente ao intervalo do mais provável.

2.2.5 *M-Coder*

O *M-Coder* pode ser visto como uma generalização do *Q-Coder*, pois o segundo pode ser construído a partir do primeiro. Algumas diferenças são destacadas a seguir: (Marpe, Kirchhoffer e Marten (2006), Marpe, Schwarz e Wiegand (2003))

Ao invés de quantizar as probabilidades dos eventos menos prováveis, o codificador adota dois conjuntos:

1. Um conjunto para valores quantizados dos possíveis valores do domínio ao qual pertence os intervalos de codificação renormalizados;
2. Um conjunto com valores discretos das probabilidades dos eventos menos prováveis.

A partir da multiplicação dos valores desses dois conjuntos obtém-se uma Tabela bidimensional com os valores pré-calculados dos subintervalos que serão utilizados pelo processo de (de)codificação aritmética. O armazenamento é realizado no formato de inteiros de precisão finita. Assim a multiplicação dos intervalos pelas probabilidades é obtida através de uma simples consulta a Tabela. Os índices utilizados correspondem ao índice do estado de probabilidade e um índice calculado a partir do intervalo de probabilidade da codificação atual (k).

Considere k como o índice do intervalo utilizado para acesso a Tabela. O cálculo seria dado por (Marpe, Kirchhoffer e Marten (2006)):

k = (R $>>$ (b - 2 - x) $\&$ $(2^x - 1)$

onde:

- $>>$ = Deslocamento lógico a direita de *bits*;
- $\&$ = Operação de AND lógico;
- R = Intervalo de codificação atual;
- b = Tamanho em *bits* do registro que armazena os intervalos;
- x = Tamanho em *bits* do registro que armazena os intervalos quantizados definidos para o conjunto 1.

A parte final da fórmula pode ser vista como uma operação de módulo, daí o nome do codificador: *M-Coder*. Dessa forma os x *bits* mais significativos do intervalo (desconsiderando o *bit* mais significativo, que no caso é manipulado pela renormalização) são utilizados para a criação do índice.

Outra diferença está no mecanismo adotado para evitar a propagação do *carry-over*. É adotada uma estratégia de reter a emissão do *bit* atual, caso haja possibilidade de um *bit* futuro alterar o valor do atual, até que o *carry-over* possa ser resolvido. Um contador é utilizado para guardar a quantidade de *bits* retidos.

2.3 *Context-based Adaptive Binary Arithmetic Coding* (CABAC)

CABAC é um método de codificação entrópica que utiliza a abordagem de codificação aritmética binária com uso de contextos adaptativos de probabilidades. (Marpe, Schwarz e Wiegand (2003)). Foi definido inicialmente para o padrão H.26L e posteriormente finalizado no padrão H.264, sendo utilizado no seu perfil principal "*Main Profile*". Uma *flag* dos parâmetros do vídeo indica se este método deve ser utilizado. (*entropy_coding_mode* = 1). O processo de decodificação é semelhante ao de codificação e por ser alvo deste trabalho, será utilizado na descrição do método.

O processo de decodificação é acionado através da requisição de um valor para um elemento sintático informado. A saída do processo é um valor para o elemento sintático requisitado.

Alguns valores de elementos sintáticos extraídos previamente são utilizados no processo, como por exemplo: Tipo de *slice* do elemento sintático requisitado, Valor de quantização do *slice* e se o Macroboloco decodificado é o primeiro do *slice*. Além disso, é necessário guardar algumas informações entre acionamentos do processo, tais como o posição de decodificação do *bitstream*, contextos e intervalo e valores de probabilidade para a decodificação aritmética.

O método é composto de 3 blocos principais:

1. Binarização;
2. Modelagem de contextos;
3. Decodificador aritmético CABAC.

2.3.1 Binarização

A binarização do CABAC é um processo para gerar representações binárias únicas para cada elemento sintático não-binário do padrão H.264 / AVC. Os valores são representados em forma de strings de *bits*, chamados de strings de *bins*. O objetivo do processo é otimizar e reduzir a complexidade dos processos de cálculo de probabilidade e de codificação aritmética. Para tanto, reduz o alfabeto dos símbolos a serem codificados para representações previamente planejadas.

Este processo é acionado para cada elemento sintático requisitado e produz como saída:

- O método de binarização a ser utilizado;
- Um limite máximo para cálculo do índice do contexto correspondente (maxBinIdxCtx);
- Um índice de contexto base para o cálculo do índice de contexto correspondente (ctxIdxOffset);

- O valor do *flag* "*bypassFlag*", que indica a possibilidade de utilizar o método de decodificação aritmética do tipo "*bypass*".

Essas informações são passadas através de uma Tabela descrita no documento de especificação do padrão H.264/AVC (ITU (2005)).

Alguns métodos de binarização realizam o processo em duas etapas, definidas como prefixo e sufixo. Desta forma cada etapa possui seus próprios valores de variáveis.

O padrão H.264 / AVC (ITU (2005)) define 7 métodos de binarização:

1. *Unary* (U);
2. *Truncated Unary* (TU);
3. Concatenação de *Unary* e Exp-Golomb de ordem k (UEGk);
4. *Fixed-Length* (FL);
5. Binarização para elementos sintáticos *Macroblock Type* e *Sub-Macroblock Type*;
6. Binarização para elementos sintáticos *Coded Block Pattern* e
7. Binarização para elementos sintáticos 'mb_qp_delta'.

Após a decodificação de cada símbolo de uma *string* de *bins*, o método de binarização deve avaliar a *string* decodificada. Caso corresponda a uma representação válida para o elemento sintático sendo decodificado, o processo de decodificação é interrompido e o valor correspondente é retornado.

2.3.2 Modelagem de contextos

O codificador CABAC define modelos de contextos baseados em probabilidades condicionais utilizando as dependências estatísticas dinâmicas presentes entre os símbolos que compõem as informações do vídeo, visando otimizar a compressão do codificador aritmético. Além dos contextos baseados em probabilidades condicionais, também existem contextos baseados em probabilidades fixas.

Os modelos de probabilidade de cada contexto são definidos como dois valores, um estado de probabilidade (64 valores possíveis) e um valor do símbolo mais provável para o contexto (MPS - dois valores possíveis). Durante o processo de codificação/decodificação, os contextos são atualizados, alterando-se o estado de probabilidade seguindo uma seqüência pré-determinada. A Figura 2.4 representa o fluxo de atualizações dos estados de probabilidade.

Durante o processo de decodificação do CABAC, para cada *bit* operado é necessário definir qual o contexto que será utilizado pela máquina de decodificação aritmética. O cálculo para o índice do contexto (ctxIdx), depende do índice do *bin* atual (binIdx - com valor 0 para o primeiro *bin* da *string* ou

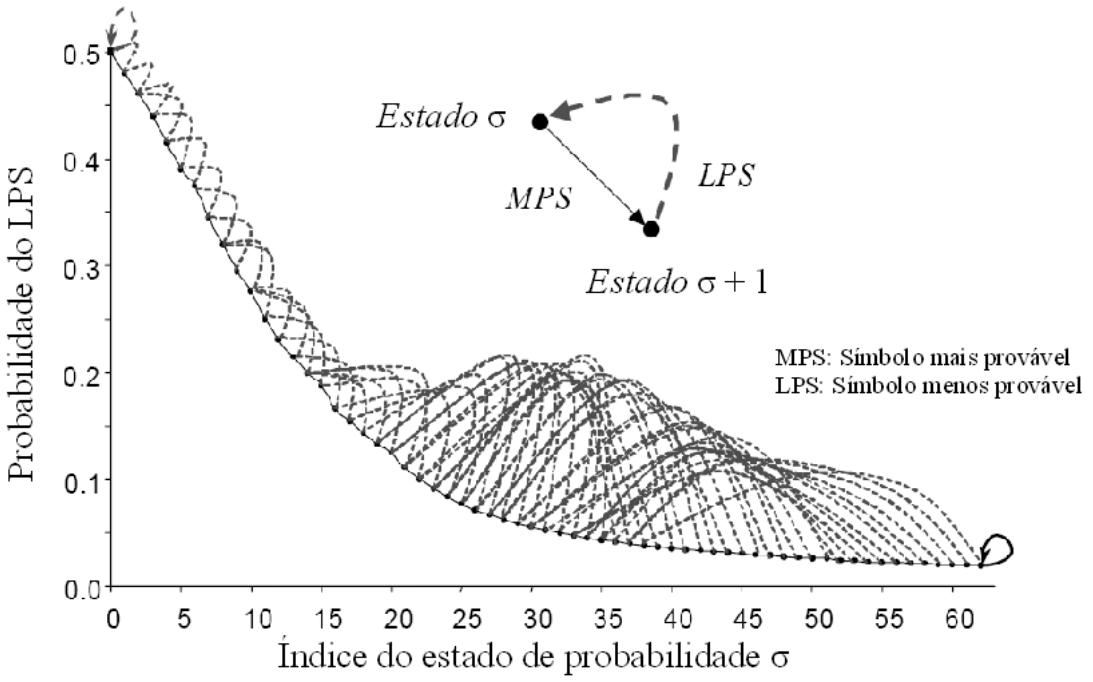

Figura 2.4: Fluxo dos estados de probabilidade

parte da *string*: prefixo e sufixo), o valor definido para a base ctxIdxOffset, o tipo de elemento sintático requisitado e informações do *slice* e de elementos de Macroblocos processados previamente. Caso o valor da *flag bypassFlag* seja '1', não é necessário calcular o índice de contexto, pois o processo de decodificação não será o da máquina de decodificação aritmética regular.

2.3.2.1 Cálculo de ctxIdx

O valor de ctxIdx pode ser obtido de três formas diferentes:

1. Obtendo o valor de forma direta;
2. Obtendo o valor de incremento (ctxIdxInc) (de forma direta ou através de processos específicos) e somando seu valor ao valor da base ctxIdxOffset. (ctxIdx = ctxIdxOffset + ctxIdxInc);
3. Obtendo o valor de outras duas variáveis (ctxIdxBlockCatOffset(ctxBlockCat) e ctxIdxInc(ctxBlockCat)) e somando seus valores ao valor de ctxIdxOffset. (ctxIdx = ctxIdxOffset + ctxIdxBlockCatOffset(ctxBlockCat) + ctxIdxInc(ctxBlockCat)).

A especificação (ITU (2005)) define Tabelas que informam como obter as variáveis incrementais respectivas. A primeira Tabela informa, para alguns valores de ctxIdxOffset e para alguns valores de binIdx, o valor direto de ctxIdx ou um conjunto de valores possíveis para ctxIdxInc. Pode também informar o respectivo processo utilizado para definir o valor de ctxIdxInc. Caso a condição atual não esteja definida na primeira Tabela, outros processos específicos são definidos para serem utilizados com auxílio de dados

fornecidos por Tabelas auxiliares. A especificação (ITU (2005)) define nove processos específicos para o cálculo dos incrementos:

1. ctxIdxInc para elementos 'mb_skip_flag';
2. ctxIdxInc para elementos 'mb_field_decoding_flag';
3. ctxIdxInc para elementos 'mb_type';
4. ctxIdxInc para elementos 'coded_block_pattern';
5. ctxIdxInc para elementos 'mb_qp_delta';
6. ctxIdxInc para elementos 'ref_idx_l0' e 'ref_idx_l1';
7. ctxIdxInc para elementos 'mvd_l0' e 'mvd_l1';
8. ctxIdxInc para elementos 'intra_chroma_pred_mode';
9. ctxIdxInc para elementos 'coded_block_flag';

O cálculo do índice ctxIdx deve ser feito até um limite máximo definido por maxBinIdxCtx.

2.3.3 Decodificador aritmético CABAC

O método CABAC define um codificador aritmético específico, otimizado para o padrão H.264 / AVC, sem multiplicadores, chamado de *M-Coder* (Módulo-*Coder*)(ver sub-sessão anterior). O *M-Coder* do CABAC utiliza os parâmetros x=2, b=10, 460 Contextos e 64 estados de probabilidade. (Marpe, Kirchhoffer e Marten (2006), ITU (2005)) O decodificador correspondente ao *M-Coder* utiliza os mesmos parâmetros.

O CABAC também define decodificadores específicos para símbolos que indicam final de *slice* e que utilizam modelos eqüiprováveis (*bypass bins*). Esses decodificadores reduzem a complexidade total do processo de decodificação.

2.3.3.1 Máquina de decodificação aritmética regular

A máquina de decodificação aritmética regular possui dois registradores (codIRange e codIOffset) que armazenam respectivamente o intervalo de probabilidade correspondente ao momento atual (ou um valor inicial) e o valor da posição da probabilidade atual. Esses mesmos registradores são comuns as máquinas de decodificação especializadas para símbolos finais e do tipo *bypass*.

O valor do índice de contexto ctxIdx, previamente calculado, aponta para o contexto correspondente ao *bin* que está sendo processado. O contexto traz informações sobre qual é o estado de probabilidade da máquina de decodificação regular para o estado atual (pStateIdx) e o valor do símbolo mais provável para o *bin* atual (valMPS).

O processo inicia obtendo o valor do índice k do *M-Coder* (qCodIRangeIdx) e que será utilizado para obter o valor do subintervalo correspondente ao símbolo menos provável do atual estado da máquina.

```
1. qCodIRangeIdx = (codIRange>>6) & 0x03
```

Obtêm-se então o subintervalo desejado (codIRangeLPS) através de uma consulta a uma Tabela pré-definida:

```
2. codIRangeLPS = rangeTabLPS[pStateIdx][qCodIRangeIdx]
```

Calcula-se o valor do subintervalo correspondente ao símbolo mais provável.

```
3. codIRange = codIRange - codIRangeLPS
```

Verifica-se se o valor da posição da probabilidade está fora do subintervalo correspondente ao símbolo mais provável:

```
4. (codIOffset >= codIRange) ?
```

Caso a verificação seja verdadeira, o símbolo decodificado não corresponde ao símbolo mais provável deste estado. Neste caso, define-se o valor decodificado do *bin* para a negação do símbolo mais provável (valMPS), toma-se o subintervalo corresponde ao símbolo menos provável como o intervalo atual e atualiza o valor da posição da probabilidade atual:

```
5. binVal = ! valMPS
6. codIOffset = codIOffset - codIRange
7. codIRange = codIRangeLPS
```

Se a verificação for falsa, o símbolo decodificado corresponde ao símbolo mais provável deste estado. Neste caso, define-se o valor decodificado do *bin* como o valor do símbolo mais provável e não é necessário definir o novo intervalo, pois o mesmo já foi definido para o subintervalo correspondente ao símbolo mais provável no passo anterior (passo 3).

```
5. binVal = valMPS
```

O processo atualiza o contexto, alterando o respectivo estado e no caso deste ser o primeiro estado da máquina (estado 0) e o símbolo decodificado não ser o mais provável, atualiza-se o valor do símbolo mais provável.

Em caso do símbolo decodificado ser diferente do símbolo mais provável:

```
8. se pStateIdx == 0, então valMPS = 1 - valMPS
9. pStateIdx = transidxLPS[pStateIdx]
```

Em caso do símbolo decodificado ser igual ao símbolo mais provável:

```
6. pStateIdx = transidxMPS[pStateIdx]
```

Onde, transidxLPS e transidxMPS correspondem a Tabelas pré-definidas com as transições dos estados de probabilidade da máquina de decodificação regular. Conforme Figura 2.4.

Após a atualização dos registradores codIRange e codIOffset, pode ser necessário realizar uma renormalização dos valores com a leitura de mais um bit do *bitstream* de dados.

A Figura 2.5 apresenta o fluxo do processo de decodificação de *bins* do tipo regular.

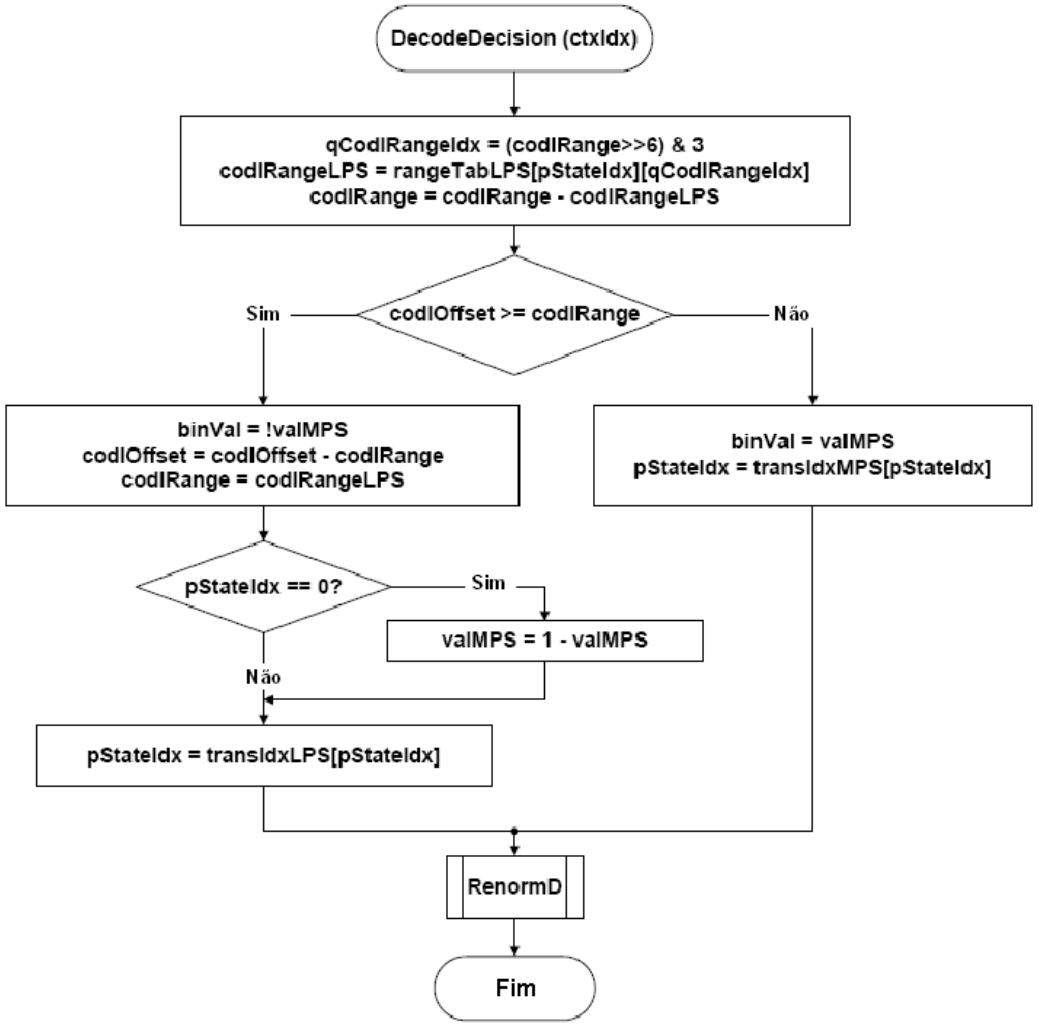

Figura 2.5: Fluxo de decodificação do *M-Coder* CABAC

Elemento	Descrição
DecodeDecision	Processo que determina o contexto para o símbolo sendo processado.
valMPS	Valor do símbolo mais provável do contexto.
codIRange	Intervalo de probabilidade atual.
binVal	Valor do símbolo decodificado.
qCodIRangeIdx	Parte do índice para consulta dos intervalos pré-calculados de probabilidades.
transIdxLPS	Tabela com a seqüência de transições do índice do estado de probabilidade do contexto, caso o símbolo decodificado for o menos provável.
pStateIdx	Índice do estado de probabilidade do contexto.
transIdxMPS	Tabela com a seqüência de transições do índice do estado de probabilidade do contexto, caso o símbolo decodificado for o mais provável.
codIRangeLPS	Subintervalo de probabilidade para o elemento menos provável. (LPS)
RenormD	Processo de renormalização.
codIOffSet	Ponteiro para o valor de probabilidade atual

Tabela 2.5: Elementos do processo de decodificação regular do CABAC

2.3.3.2 Renormalização

A renormalização é realizada, comparando-se o valor de codIRange:

```
1. (codIRange < 0x0100) ?
```

Se a verificação é falsa, não é necessário fazer as renormalizações e o processo termina, caso contrário, os registradores são dobrados e é feito a leitura de um *bit* do *bitstream*, o qual é adicionado na posição menos significativa do valor codIOffset:

```
2. codIRange = codIRange << 1
3. codIOffset = codIOffset << 1
4. codIOffset = codIOffset | read_bits(1)
```

A rotina se repete até que a verificação 1 seja falsa. A Figura 2.6 apresenta o fluxo do processo de renormalização.

2.3.3.3 Decodificação *ByPass*

Para este processo, os contextos e estados de probabilidade da máquina de decodificação não são utilizados. Apenas os registradores codIRange e codIOffset. Os registradores são dobrados e é realizada a leitura de um *bit* do *bitstream*, o qual é adicionado na posição menos significativa do valor codIOffset:

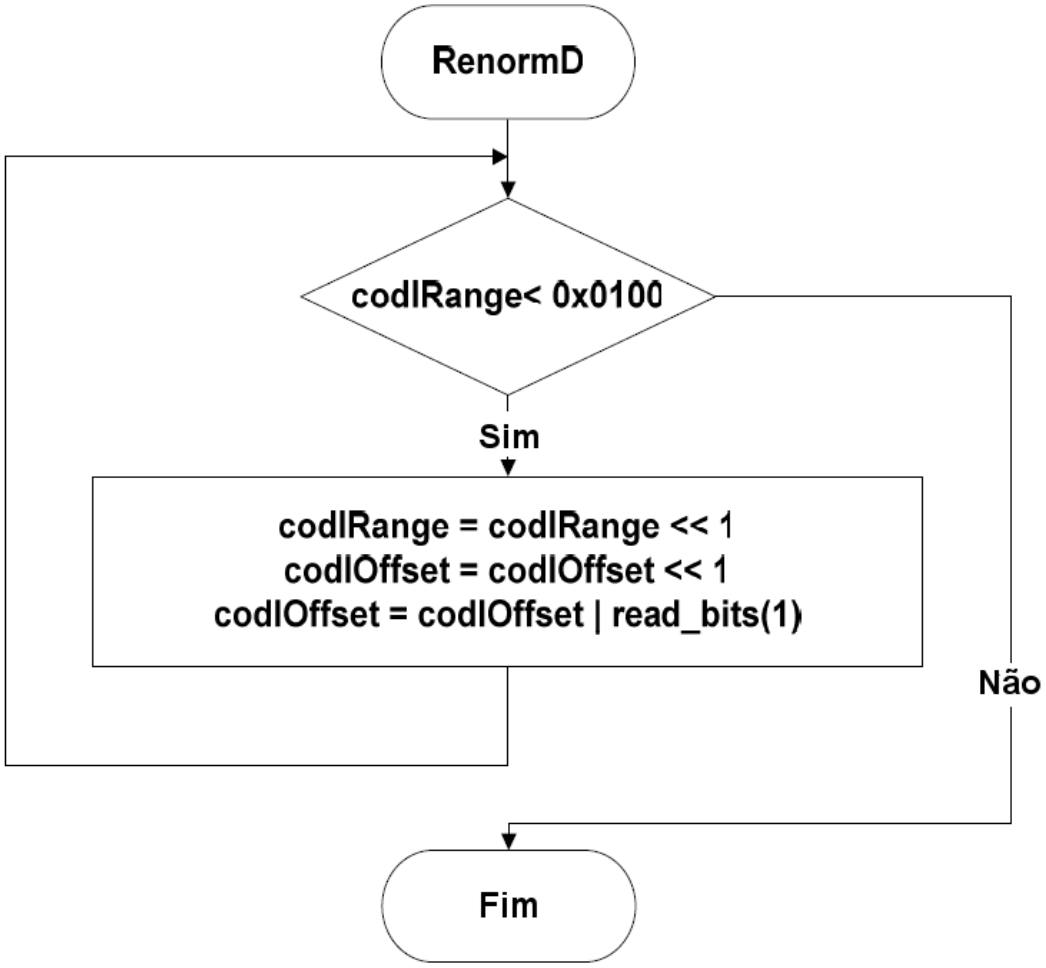

Figura 2.6: Fluxo da renormalização do *M-Coder* CABAC.

```
1. codIRange = codIRange << 1
2. codIOffset = codIOffset << 1
3. codIOffset = codIOffset | read_bits(1)
```

É realizada uma verificação para determinar o valor do *bin* decodificado:

```
4. (codIOffset >= codIRange) ?
```

Se a verificação é falsa, o valor do *bin* é igual a 0, caso contrário, o valor do *bin* é igual a 1 e se executa a atualização de codIOffset:
Se falsa:

```
5. binVal = 0
```

Se verdadeira:

```
5. binVal = 1
6. codIOffset = codIOffset – codIRange
```

A Figura 2.7 apresenta o fluxo do processo da máquina de decodificação aritmética do tipo *bypass*.

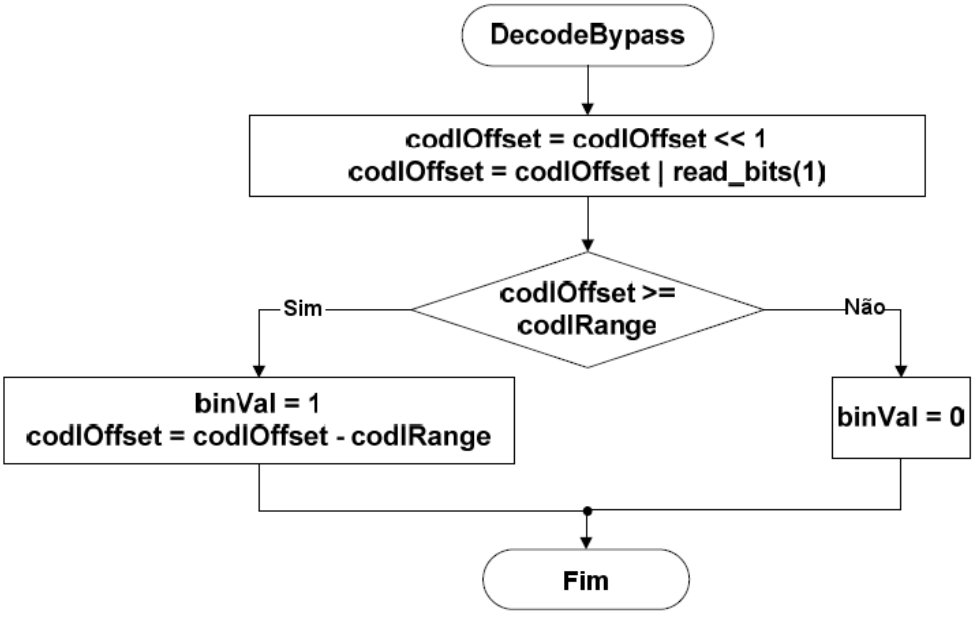

Figura 2.7: Fluxo da máquina de decodificação aritmética do tipo *bypass*.

2.3.3.4 Decodificação para símbolo final de *slice*

Este processo de decodificação é específico para o elemento sintático 'end_of_slice_flag' e para o *bin* que indica o modo I_PCM correspondente ao ctxIdx=276.

O valor do registrador codIRange é decrementado em 2 e é feito uma verificação do valor de codIOffset:

```
1. codIRange = codIRange – 2
2.( codIOffset >= codIRange) ?
```

Se a verificação é verdadeira o valor do *bin* decodificado é 1 e o processo termina, caso contrário, o valor do *bin* decodificado é 0, é realizada uma renormalização, como definido previamente, e o processo termina.

A Figura 2.8 apresenta o fluxo do processo da máquina de decodificação aritmética para símbolo final de *slice*.

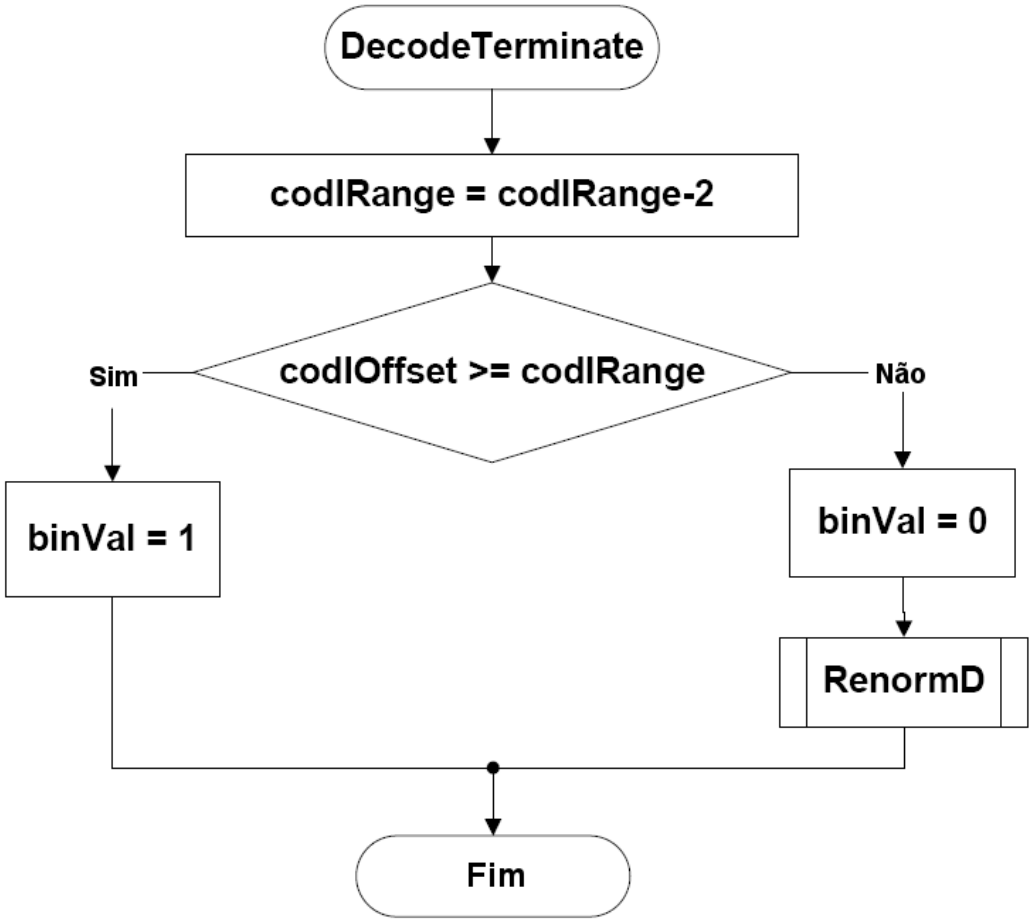

Figura 2.8: Fluxo da máquina de decodificação aritmética para símbolo final de *slice*.

2.4 SystemC

SystemC desde 2005 é um padrão definido pela IEEE composto de uma biblioteca de classes de ANSI C++ para descrição e modelagem de projetos de sistemas de software e hardware. (IEEE Computer Society (2005))

A biblioteca foi desenvolvida para dar suporte a projetos de sistemas, permitindo a modelagem funcional em diferentes níveis de descrições, abordando tanto o *software* quanto o *hardware*. Como complemento, também é utilizada como meio para compartilhamento de *Intellectual Properties* (IPs).

A modelagem funcional dos sistemas é provida por classes que permitem representar os seguinte conceitos:

- Decomposição hierárquica do sistema em módulos;
- Conexão estrutural entre os módulos do sistema através de elementos específicos;
- Agendamento e sincronização de processos concorrentes através de monitoramento de eventos;
- O transcorrer de tempo simulado;
- Separação de computação (processos) de comunicação (canais);
- Refinamentos independentes da interface de uso de computação e comunicação;
- Tipos de dados orientados para o hardware que permite a modelagem de lógica digital e aritmética de ponto fixo.

A biblioteca possibilita que um projetista descreva os módulos do sistema em forma de funções C++ que são executadas através de um escalonador que simula o comportamento de concorrência e comunicação existente em sistemas eletrônicos reais. Possui mecanismos que permitem imitar o transcorrer do tempo simulado e monitorar sinais de controle de forma útil ao processo de modelagem e verificação.

A arquitetura da biblioteca (Figura 2.9) é formada por:

1. Linguagem núcleo;
   Composto basicamente por um escalonador não-preemptivo e os elementos básicos de construção: *Modules*, *Ports*, *Exports*, *Channels*, *Processes*, *Events*, etc.
2. Tipos de dados do SystemC;
   Tipo de lógica 4-valores, Vetores de lógica 4-valores, Vetores de *bits*, Inteiros de precisão finita, Inteiros de precisão limitada, Tipos de ponto-fixo, etc.
3. Canais pré-definidos;
   Signal, *Clock*, FIFO, mutex, semáforos, etc.

4. Utilitários.
   Rastreamento de execução, auxiliar de relatórios, etc.

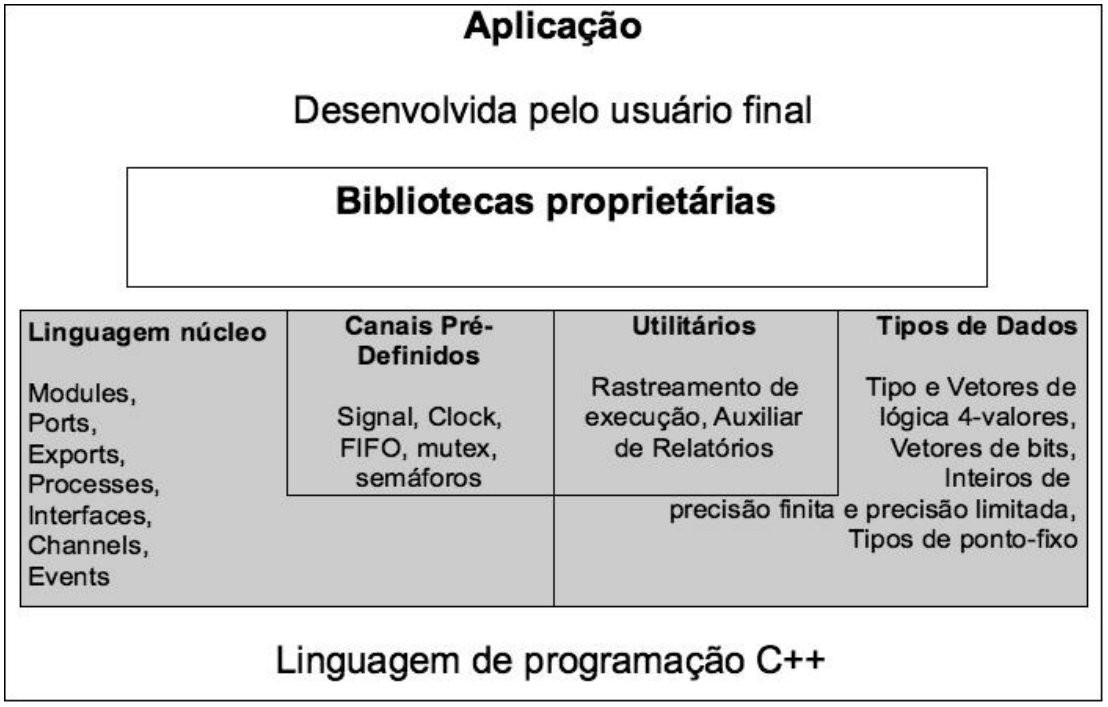

Figura 2.9: Arquitetura do SystemC.

2.5 Sistema em Silício (*System-on-a-Chip*)

O constante aprimoramento das tecnologias para confecção de circuitos digitais baseadas na miniaturização de transistores permitiu o desenvolvimento de sistema eletrônicos com maior poder de computação que seus antecessores utilizando o mesmo espaço em silício. Este incremento proveniente da maior quantidade de transistores, acarreta também uma grande complexidade no projeto desses sistemas, envolvendo todos os processos desde a modelagem até a verificação e fabricação.

Para um melhor aproveitamento dessa maior densidade de dispositivos, novas arquiteturas e modelos computacionais foram criados (*Pipeline*, Processadores Super-escalares, etc.). Uma das abordagens adotadas foi a de integrar diversos dispositivos e/ou *softwares* que compõem um sistema eletrônico, como um computador, em uma única pastilha de silício (SoC - *System-on-a-Chip*). Essa alternativa possibilita a construção de sistemas com maior densidade computacional através da utilização de unidades mais complexas e de maior capacidade (Processadores, redes de comunicação, conversores A/D, etc) que as unidades digitais básicas que constituem os *chips* tradicionais (ULAs, registradores, barramentos, etc).

Como os SoCs são constituídos geralmente de *software* e *hardware*, a utilização de uma ferramenta que integre a modelagem de ambos os mundos é de grande valia para os projetistas. A biblioteca SystemC possui estas características, possibilitando a utilização de uma linguagem comum (C++) para a modelagem em diversos níveis de abstrações, sendo inclusive adotado como padrão pela IEEE.

Devido as vantagens oferecidas pelos SoCs em comparação à abordagem tradicional de sistemas com múltiplos *chips*, (como por exemplo: menor consumo elétrico, maior robustez e menor custo financeiro para produção) esses sistemas acabam sendo utilizados na fabricação de dispositivos portáteis modernos. (Siwert (2005))

Este capítulo apresentou informações sobre o padrão de vídeo H.264/AVC e algumas de suas características, incluindo um dos codificadores utilizados no padrão, o CABAC. Descreve o funcionamento dos codificadores aritméticos e detalha o processo de operação do CABAC. Apresenta brevemente o SystemC, uma API C++ utilizada na descrição de hardware e o tipo de arquitetura alvo para o projeto, baseado no conceito de SoC.

Capítulo 3

Estado da Arte

Durante o período de execução do trabalho foram realizadas pesquisas em busca de propostas de otimização para decodificadores/codificadores CABAC em hardware. A partir do resultado, foram selecionados oito artigos como os trabalhos que apresentaram melhores propostas e/ou desempenhos.

No restante deste capítulo, cada uma das propostas citadas será descrita e comentada. Inicialmente serão abordados os trabalhos de Osorio e Bruguera e Flordal, Wu e Liu que tratam da codificação CABAC. Os trabalhos seguintes tratam do processo de decodificação.

3.1 Codificador de Osorio e Bruguera

Na abordagem adotada por Osorio e Bruguera (2004), o processo de codificação do CABAC é dividido em duas partes críticas, abordadas individualmente: Gerência de contextos e máquina de codificação aritmética. Para otimizar o processo, foi adotada solução de construir *hardware* dedicado, o qual ao final atingiu a capacidade de se produzir 1 símbolo (*bin*) por ciclo, em circuito que pode atingir freqüência superior a 200 MHz com um gasto de área considerado baixo, desempenho e características suficientes para a utilização em dispositivos portáteis que codifiquem vídeos.

O trabalho aponta que apenas uma pequena parte dos símbolos a serem codificados pelo CABAC é responsável pela sobrecarga, devido a alta freqüência de ocorrência e a quantidade de *bins* utilizados para representação. O aumento no desempenho será obtido na otimização da codificação via *hardware* destes símbolos especiais, deixando os menores e menos freqüentes para codificação via software. Os símbolos escolhidos para serem codificados por *hardware* são os coeficientes da transformada e os mapas de significância, responsáveis por gerar múltiplos *bins* por *frame*. A codificação de um *bin* por ciclo é obtida através de circuitos dedicados para o codificador aritmético, o renormalizador e o processo de *byte-stuffing*. O processo de renormalização utiliza circuito de LZA (*Leading Zero Antecipator*) e *barrel-shifters*.

Para permitir a codificação de um símbolo (*bin*) por ciclo, é necessária

a implementação de um sistema de gerência de contextos. Esta gerência é impactada por um gargalo de memória, visto que existem 399 contextos (460 na publicação de 2005 (ITU (2005))), dos quais um deve ser carregado e atualizado a cada codificação de *bin*. A abordagem adotada foi a utilização de uma *cache* para 16 contextos. Para os elementos especiais codificados em *hardware* é utilizada técnica de pré-carga especulativa da cache, através da carga de subconjuntos de contextos de 4 grupos de contextos necessários para os elementos especiais.

3.2 Codificador de Flordal, Wu e Liu

O trabalho executado por Flordal, Wu e Liu (2006), baseia-se na solução adotada por Osorio e Bruguera (2004). Nesta abordagem é mostrada uma solução baseada em *hardware* de DSP customizado para fornecer instruções e características de arquitetura que auxiliam o *software* de codificação CABAC de vídeos H.264/AVC de menor resolução. Apesar da otimização alcançada, a avaliação é de que o desempenho não é suficiente para codificação CABAC de vídeos H.264/AVC com resolução HDTV 1080p. Como alternativa o trabalho também propõe o projeto de um *hardware* dedicado para codificação CABAC de vídeos H.264/AVC de maior resolução.

O *hardware* dedicado para o CABAC apresenta soluções de otimização para 4 estágios da codificação:

1. Leitura dos índices e valores dos contextos com a utilização de uma cache de 64 contextos ao invés de 16;

2. Atualização do valor de *Range* a partir do processo de codificação de 2 *bins* simultâneos com a utilização de cálculos especulativos;

3. Atualização do valor de *Low* e propagação dos valores gerados pela codificação;

4. Renormalização e *byte outstanding* dos valores atualizados através de mecanismo com registradores de 34 *bits*.

A avaliação de desempenho do hardware dedicado indicam uma taxa de codificação de mais de 1,5 *bins* por ciclo com *clock* máximo superior a 190 MHz. O que possibilitaria a codificação de vídeos H.264/AVC em resolução HDTV 1080p com a taxa de 6 MIPS.

A Tabela 3.1 representa as comparações observadas entre essas diversas implementações do codificador CABAC.

3.3 Decodificador de Yu e He

O trabalho desenvolvido por Wei Yu e Yun He (Yu e He (2005)) apresenta uma solução de hardware dedicado a decodificação CABAC que permitiria a decodificação de todos os elementos sintáticos de um Macrobloco. O

desempenho obtido garante o mínimo de um *bin* decodificado por ciclo e a decodificação sem interrupção dos bins consecutivos de um mesmo elemento sintático. A arquitetura é baseada na freqüência e características dos elementos sintáticos que compõem o Macrobloco. Adotou-se a concatenação de duas máquias de decodificação aritmética regulares de *bins* (com duas máquinas de decodificação de *bypass bins* para sinal) e duas máquinas de decodificação de *bypass bins*. A concatenação das máquinas permite a decodificação dos elementos sintáticos: abs_mvd, significant_coeff_flag, last_significant_coeff_flag e coeff_abs_level_minus1 com taxas de até 3 *bins* (2 regulares e 1 *bypass*) por ciclo. Os demais elementos utilizam apenas parte do conjunto de máquinas, sendo decodificados em taxas de 1 *bin* por ciclo. O aumento do caminho crítico gerado pela concatenação das máquinas de decodificação é amenizada por estruturas que pré-calculam e carregam valores que serão utilizados na máquina seguinte.

Para possibilitar a decodificação de até 2 *bins* regulares por ciclo foi necessário adotar uma estratégia de gerenciamento de contextos e utilização de uma cache. Os 399 contextos utilizados foram organizados em 18 grupos de tamanhos variados e com um tamanho máximo de 44 contextos. Os grupos foram organizados a partir da ordem de decodificação dos elementos sintáticos em um Macrobloco. Esta estratégia de gerenciamento de contextos permite a decodificação de vários elementos sem a necessidade de acesso a memória, o que reduz drasticamente o gasto de ciclos do processo de decodificação. Diferente de propostas anteriores, o método de pré-carga dos contextos é simples, baseada na ordem dos elementos sintáticos. A *cache* possui 44 registradores de 7 *bits* e os contextos são organizados em memória RAM em linhas de 15 contextos, permitindo que todos os grupos menos um possam ser lidos e atualizados a partir do acesso de uma linha da memória.

A implementação em *hardware* possui caminho crítico de 6,7 ns e análises mostram que a média de decodificação de macroblocos seria de aproximadamente 500 ciclos/MB com taxas máximas de 3 bins / cliclo, possibilitando a decodificação de vídeos H.264/AVC em resolução D1 com taxa de *stream* de 4 Mbits/s com freqüência de *clock* de 18MHz.

3.4 Decodificador de Zheng et al.

O trabalho de Zheng et al. (2007) propõe a utilização de duas unidades de controle de decodificação do CABAC, em que uma delas seria especializada para a decodificação de alguns elementos sintáticos dos blocos de resíduos: significant_coeff_flag (Marca se o coeficiente da transformada é diferente de zero), last_significant_coeff_flag (Marca se o coeficiente da transformada é o último diferente de zero), coeff_abs_level_minus1 (Valor do coeficiente da transformada menos 1) e coeff_sign_flag (Marca do sinal do coeficiente da transformada). A proposta também inclui a otimização no controle e armazenamento dos parâmetros dos macroblocos de referência da decodificação e a utilização de renormalização em um ciclo através da determi-

nação prévia da quantidade de deslocamentos necessários a partir dos valores de codIRange e codIRangeLPS. Os elementos dos blocos de resíduos foram escolhidos por permitir que a determinação dos índices dos contextos possa ser feita a partir dos parâmetros decodificados no início do macrobloco. Diferente dos trabalhos anteriores, não há citação a utilização de *cache* e agrupamentos de contextos. Os contextos são acessados e atualizados diretamente em memória e para alguns elementos sintáticos a escrita é prorrogada para o fim de uma seqüência de *bins* e é realizada em paralelo com a decodificação de *bypass bins* do resto do elemento sintático. Os resultados obtidos permitem a decodificação do vídeo de referência Foreman com resolução CIF (352x240) com um taxa média de 4 ciclos por bin e operação com freqüência de *clock* de até 100 MHz. Declaram uma melhora de 27,9%, 18,2% e 48,8% para *frames* I, P e B respectivamente comparado ao proposto por Jian-Wen Chen, Cheng-Ru Chang, Youn-Long Lin (Chen, Chang e Lin (2005)).

3.5 Decodificador de Chen e Lin

A proposta de Chen e Lin (2007) apresenta uma técnica que paraleliza a decodificação dos elementos sintático Coded_block_flag (Marca de bloco de transformadas), coeficientes e os pares de elementos significant_coeff_flag e last_significant_coeff_flag com a preparação e carregamento de parâmetros necessários. Adotou uma máquina de decodificação de 2 *bins* paralelos para a decodificação dos coeficientes, semelhante a solução adotada por Yu e He (2005). A otimização da decodificação dos pares significant_coeff_flag e last_significant_coeff_flag foi obtida através da utilização de Tabelas de contextos separadas para os contextos destes dois elementos. O resultado obtido permitiria a decodificação de vídeos com resolução 1080p operando com freqüência de *clock* de 45 MHz em um circuito que permite freqüência de *clock* máxima de até 180 MHz A taxa de decodificação atingiria até 1,24 *bins* por ciclo. Uma melhora de 50% se comparado com o trabalho anterior do mesmo grupo (Chen, Chang e Lin (2005)) e 40% se comparado com o trabalho de Yu e He (2005).

3.6 Decodificador de Xu et al.

O trabalho de Xu et al. (2007) é semelhante ao trabalho de Yu e He (2005) com algumas alterações. O sistema possui uma máquina com 2 (dois) decodificadores de *bins* regulares e 2 (dois) decodificadores de *bypass-bins*. O processo de decodificação somente é otimizado para o elemento coeff_abs_level_minus1, o qual é decodificado em taxas de 2 *bins* regulares por ciclo. Os demais elementos sintáticos são decodificados em taxas de 1 *bin* regulares/*bypass* por ciclo. A renormalização utiliza um mecanismo de FOD (*Firts One Detector*). Uma ROM armazena de forma otimizada as informações sobre o próximo estado de probabilidade e o range para o sím-

bolo menos provável, tanto para o *bin* atual, como para o seguinte. O estado de probabilidade do símbolo mais provável é calculado ao invés de armazenado em ROM. É utilizada uma *cache* para até 44 contextos e o conjunto de 460 contextos foi dividido em 25 grupos, organizados por ordem de carregamento, e que possuem na sua maioria, até 14 contextos, sendo que um possui até 44 contextos. O resultado obtido permite operação com freqüência máxima de *clock* de aproximadamente 80 MHz e não informa a taxa de decodificação.

3.7 Decodificador de Zhang et al.

A versão proposta por Zhang et al. (2007) é diferente das demais e utiliza uma abordagem direcionada pela taxa de decodificação de *bits*, ao invés da abordagem tradicional de taxa de decodificação de *bins*. A partir da análise que a decodificação de 1 *bit* pode gerar até 16 *bins*, foram projetados módulos especializados na decodificação dos elementos sintáticos mais freqüentes: Intra mode, Motion info, Significance map e Coefficient. Esses módulos produzem as informações necessárias para alimentar outro módulo com 16 mecanismos de decodificação de *bins* regulares e *bypass*, que propagam os resultados sobre o estado da decodificação e renormalização de forma paralela. Os restantes dos elementos sintáticos são decodificados através de um processo menos otimizado para reduzir a complexidade total do decodificador. O resultado obtido permite a decodificação de vídeo de resolução 1080i em tempo-real operando com *clock* de 42,2 MHz.

3.8 Decodificador de Yi e Park

A proposta Yi e Park (2007) utiliza a idéia de um mecanismo de decodificação baseado em um *pipeline* de 2 níveis especializado nos elementos que possuem decodificação não-eqüiprovável. O *pipeline* está dividido em:

1. Seleção e Leitura de Contexto;
2. Decodificação aritmética e comparação de binarização.

O sistema possui uma estrutura semelhante a uma *cache* de contextos com capacidade para 8 contextos (64 *bits*). Os contextos foram organizados em memória com palavra de comprimento 64 *bits* de forma a permitir que o carregamento dos contextos necessários para um elemento sintático seja realizado, na maioria das vezes, em uma leitura simples. A utilização de uma cache para os contextos, permite que a atualização seja postergada até que o contexto necessário para a decodificação de um *bin* não esteja presente. Neste caso, uma nova leitura é precedida pela atualização na memória dos contextos previamente alterados. O resultado obtido pode executar a uma freqüência de *clock* máxima de 225 MHz. Possui uma taxa média aproximada de 3,93 ciclos por bin e possui desempenho suficiente para decodificar vídeos em resolução 1080p em tempo real.

A Tabela 3.2 representa as comparações observadas entre essas diversas implementações do decodificador CABAC.

Este capítulo trouxe descrições de alguns trabalhos que implementam codificadores e decodificadores CABAC, em hardware, de forma otimizada. As principais características e desempenhos foram relatadas de forma a permitir a exposição de alguns dos conceitos utilizados neste trabalho e possibilitar a comparação dos desempenhos.

	Características										
Trabalhos	***Cache* de Contextos**	**Tamanho *Cache***	**Grupos Utilizados**	***Prefetching* de Contextos**	**Renormalizador de 1 ciclo**	**Elementos Otimizados**	**Desempenho**	**Frequência máxima**	**Máquina dupla de decodificação (Regular + *Bypass*)**	**Resolução suportada estimada**	**OBS**
Osorio e Bruguera (2004)	Sim	16 Contextos (bytes)	4	Sim	Sim	Transformadas de Coeficientes e Mapas de Significância (significant_coeff_flag, last_significant_coeff_flag e coeff_abs_level_minus1)	1,1 ciclos/bin	> 200MHz	Não	NI	Codificador
Flordal, Wu e Liu (2006)	Sim	64 Contextos	NI	Sim	Sim	NI	> 1,5 bins/ciclo	190MHz	Não	1080p	Codificador

Tabela 3.1: Tabela comparativa das arquiteturas otimizadas para codificação CABAC

NI - Não informado.

	Características										
Trabalhos	***Cache* de Contextos**	**Tamanho *Cache***	**Grupos Utilizados**	***Prefetching* de Contextos**	**Renormalizador de 1 ciclo**	**Elementos Otimizados**	**Desempenho**	**Frequência máxima**	**Máquina dupla de decodificação (Regular + *Bypass*)**	**Resolução suportada estimada**	**OBS**
Yu e He (2005)	Sim	44 Contextos	18	Sim	Sim*	abs_mvd, significant_coeff_flag, last_significant_coeff_flag e coeff_abs_level_minus1	> 1 bin/ciclo (Máx de 3 bins/ciclo)	+/-150MHz	Sim	1080p	
Zheng et al. (2007)	Não	NA	NA	NI	Sim	significant_coeff_flag, last_significant_coeff_flag, coeff_abs_level_minus1 e coeff_sign_flag	média de 4 ciclos/bin	100 MHz	Não	1080p	
Chen e Lin (2007)	NI	NA	NA	NI	Sim*	Coded_block_flag, significant_coeff_flag, last_significant_coeff_flag e coeff_abs_level_minus1	média de 1,24 bins/ciclo	137 MHz	Sim	1080p	
Yi e Park (2007)	Sim	8 Contextos	Aprox. 1 para cada E.S.	Não	NI	Aqueles formados por bins não-eqüiprováveis	média de 3,93 ciclos/bin	225 MHz	Não	1080p	
Xu et al. (2007)	Sim	44 Contextos	25	Não	Sim*	coeff_abs_level_minus1	NI	+/-80 MHz	Sim	NI	
Zhang et al. (2007)	NI	NA	NA	NI	Sim	Modos de Intra, Informações de Motion Estimation, Mapas de significância e Coeficientes	Até 16 bins/ciclo	45 MHz	Não	1080p	Máquina de decodificação múltipla. Até 16 bins.

Tabela 3.2: Tabela comparativa das arquiteturas otimizadas para decodificação CABAC

NI - Não informado.
Sim* - Não informado, mas avaliado a partir do trabalho como sim.

Capítulo 4

Proposta de Arquitetura

De forma geral, a proposta visa acelerar o processo de decodificação através da extração de múltiplos *bins* em um único ciclo e para qualquer elemento sintático a ser processado. A arquitetura tem como base o artigo de Yu e He (2005) que utiliza o conceito de concatenação de máquinas de decodificação aritmética para possibilitar a decodificação dos múltiplos *bins*. Para este trabalho, optou-se por um projeto com dois tipos de módulos com máquinas concatenadas:

- **Tipo A** (dbl_decoder): Concatena três máquinas na seguinte combinação: duas máquinas de decodificação aritmética regular e uma máquina de decodificação aritmética especializada para o símbolo *terminate*.
- **Tipo B** (dbl_eq_decoder): Concatena duas máquinas de decodificação aritmética especializada em *bypass bins*.

O módulo do tipo A é utilizado de forma mais genérica e realiza a decodificação da maioria dos elementos sintáticos processados pelo decodificador CABAC. O módulo do tipo B é especializado na decodificação de sufixos de elementos sintáticos cuja binarização é do tipo *Exp-Golomb* de ordem k (EGk).

O restante da arquitetura é composta por diversos elementos que fornecem suporte aos dados e coordenam a execução dos módulos. A tabela 4.1 relaciona os principais componentes.

Os módulos foram projetados a partir da especificação do padrão ITU (2005) e do software de referência JM 10.2 (H.264/AVC Software Coordination (2009)). Adotou-se a ordem *little-endian* para a definição dos vetores de *bits* utilizados e a representação de complemento de dois para valores inteiros. Todas as memórias ROM utilizadas na arquitetura são memórias assíncronas e as memórias RAM são síncronas.

A Figura 4.1 apresenta de forma simplificada as conexões entre os diversos módulos da arquitetura.

Componente	**Descrição**
biari_decoder - (Regular Decoder)	Máquina de decodificação aritmética para *bins* do tipo regular.
biari_eq_prob_decoder - (Eq. Probability Decoder)	Máquina de decodificação aritmética para *bins* do tipo *bypass*.
biari_final_decoder - (Final Decoder)	Máquina de decodificação aritmética para *bins* do tipo final.
fast_renorm	Módulo que executa a renormalização de forma combinacional.
dbl_decoder - (Double Decoder)	Módulo que concatena máquinas de decodificação aritmética para decodificar até dois *bins* válidos em um único ciclo de *clock*.
dbl_eq_decoder - (Double Eq.Prob Decoder)	Módulo que concatena duas máquinas de decodificação aritmética do tipo biari_eq_prob_decoder para decodificar até dois *bins* válidos do tipo *bypass*.
cabac_decoder	Módulo principal de controle que inclui três máquinas de estado que conduzem o processo de decodificação para os diversos elementos sintáticos do H.264/AVC.
Módulos de cálculo de índice de contexto	Diversos módulos que calculam qual o índice de contexto a ser utilizado na decodificação aritmética. Estão citados na Tabela representada na Figura 4.15.
Unidades de controle	Módulos que realizam a coordenação de ativação e seleção de saídas dos diversos módulos operativos de decodificação.

Tabela 4.1: Componentes da arquitetura proposta

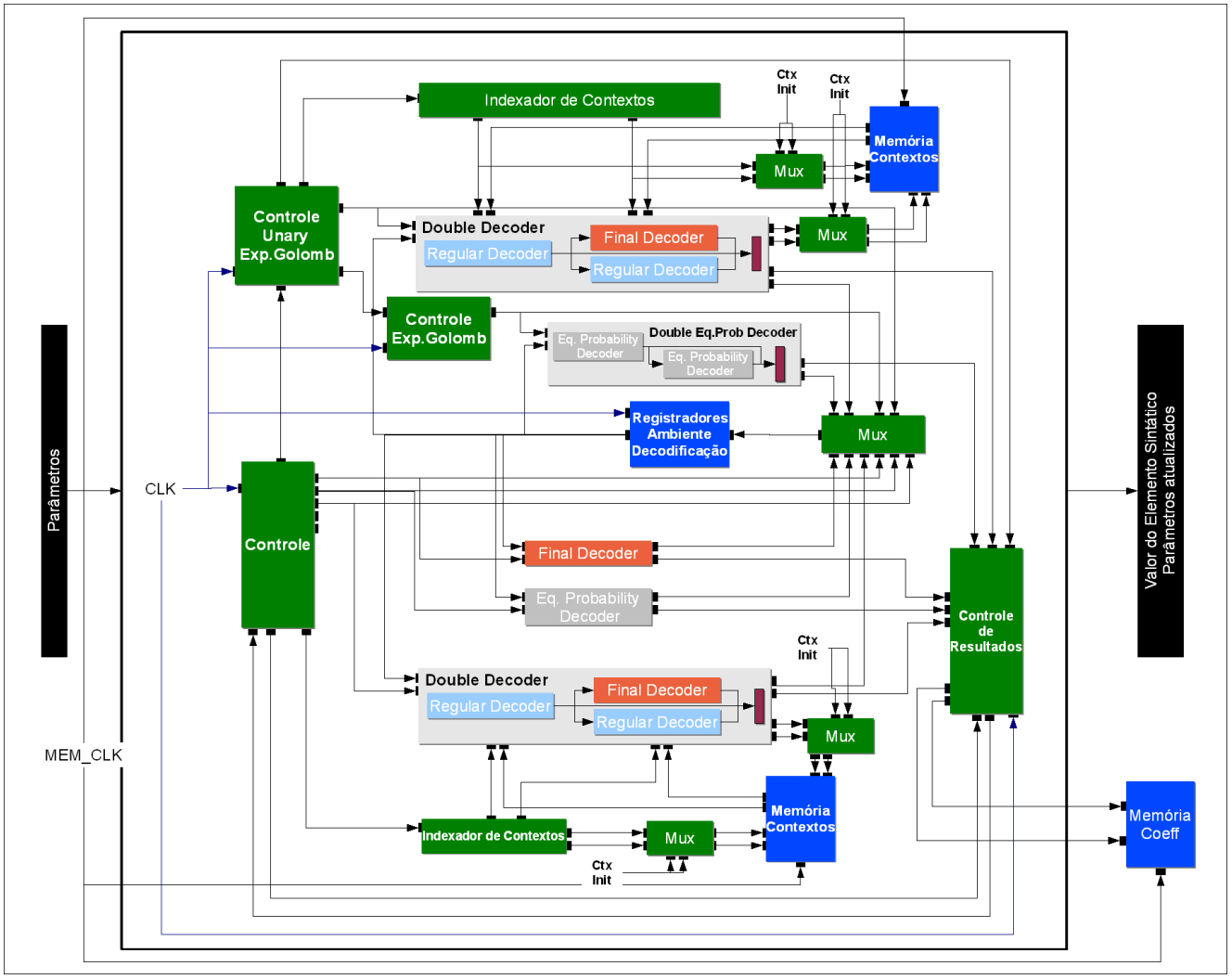

Figura 4.1: Esquema geral simplificado da arquitetura

Componente VHDL	Componente Arquitetura
biari_decoder	Regular Decoder
biari_eq_prob_decoder	Eq. Probability Decoder
biari_final_decoder	Final Decoder
fast_renorm	Não representado na figura
dbl_decoder	Double Decoder
dbl_eq_decoder	(Double Eq.Prob Decoder
cabac_decoder	Composto por todos os módulos da arquitetura
Módulos de cálculo de índice de contexto	Indexadores de contextos
Unidades de controle	Composto pelo módulos: Controle, Controle de Resultados, Controle Unary Exp. Golomb e Controle Exp. Golomb

Tabela 4.2: Relação componentes VHDL e componentes da arquitetura representados na figura

O módulo do decodificador CABAC apresenta portas de entrada e saída conforme a Figura 4.2 e as Tabelas 4.3 e 4.5.

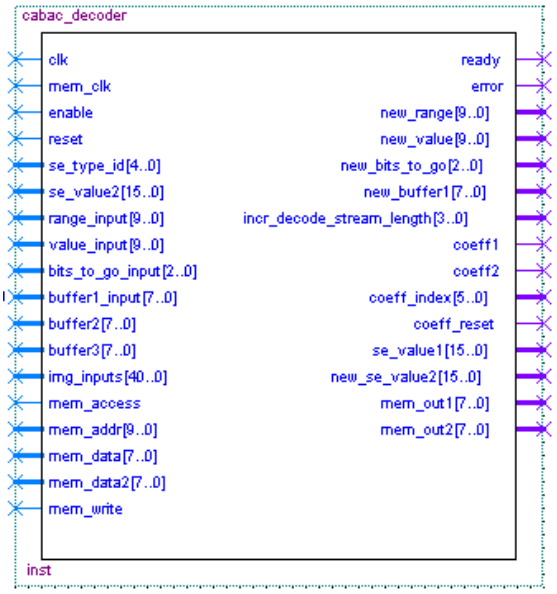

Figura 4.2: Interface do módulo decodificador CABAC

Entradas	
Porta	**Descrição**
clk	*Clock* geral para o módulo de decodificação.
mem_clk	*Clock* específico para as memórias de contextos.
enable	Ativa a decodificação do(s) elemento(s) sintático(s).
reset	Inicia os registradores e estados para os valores padrões de início de decodificação.
se_type_id	Identificador do tipo de elemento sintático a ser decodificado.
se_value2	Valor secundário pré-decodificação de alguns elementos sintáticos (baseado na JM 10.2).
range_input	Valor pré-decodificação do registrador *Range*. Essa entrada é atribuída ao registrador *Range* durante o acionamento do *reset*.
value_input	Valor pré-decodificação do registrador *Value*. Essa entrada é atribuída ao registrador *Value* durante o acionamento do *reset*.
bits_to_go_input	Índice do *bit* atual na leitura do *byte* atual do *bitstream*. Essa entrada é atribuída ao registrador *bits_to_go* durante o acionamento do *reset*.
buffer1_input	*Byte* atual do *bitstream*.
buffer2	Próximo *Byte* a ser consumido do *bitstream*.
buffer3	Terceiro *Byte* a ser consumido do *bitstream*.
img_inputs	Entrada de 41 *bits* com concatenação de diversos parâmetros formados por valores de elementos previamente decodificados do *slice* e imagem. A Tabela 4.4 apresenta de forma detalhada esta porta.
mem_acess	Sinal que aciona o controle externo para carga das memórias de contextos.
mem_addr	Porta de endereçamento das memórias de contextos.
mem_data	Porta para entrada de dados na carga das memórias de contextos. Relativo ao endereço determinado pela entrada mem_addr.
mem_data2	Porta para entrada de dados na carga das memórias de contextos. Relativo ao endereço determinado pela entrada mem_addr + 1.
mem_write	Sinal de controle para escrita nas memórias de contextos. Válido quando o sinal mem_acess está ativo.

Tabela 4.3: Portas de entrada do módulo de decodificação CABAC

Parâmetro	***Bits* mapeados da entrada**
blkA_avail	*bit* 0
blkB_avail	*bit* 1
blkA_fieldf	*bit* 2
blkB_fieldf	*bit* 3
mbaff_framef	*bit* 4
blkA_mvd	*bits* 5 ao 12
blkB_mvd	*bits* 13 ao 20
currMB_MVD_CompIdx	*bit* 21
blkA_skipf	*bit* 2
blkB_skipf	*bit* 3
blkA_tr_size_8x8_f	*bit* 2
blkB_tr_size_8x8_f	*bit* 3
blkA_mb_type	*bits* 5 ao 9
blkB_mb_type	*bits* 10 ao 14
blkA_sub_mode	*bits* 15 ao 18
blkB_sub_mode	*bits* 19 ao 22
blkA_sub_pdir	*bits* 23 ao 24
blkB_sub_pdir	*bits* 25 ao 26
blkA_ref_frame_array	*bits* 27 ao 31
blkB_ref_frame_array	*bits* 32 ao 36
img_type	*bits* 37 ao 39
currMB_fieldf	*bit* 40
mbA1_cbp	*bits* 15 ao 20
mbB1_cbp	*bits* 21 ao 26
last_dquant	*bits* 0 ao 6
structure	*bits* 34 ao 35
type_in	*bits* 36 ao 39
c1	*bits* 0 ao 4
c2	*bits* 5 ao 8
mbA_c_ipred_mode	*bits* 15 ao 17
mbB_c_ipred_mode	*bits* 18 ao 20
img_intra_block	*bit* 2
upper_bit	*bit* 3
left_bit	*bit* 4

Tabela 4.4: Mapeamento dos parâmetros para a entrada img_inputs.

Saídas	
Porta	**Descrição**
ready	Sinal que indica término da decodificação do elemento sintático.
error	Sinal que indica erro na decodificação causada por estado não esperado.
new_range	Valor atualizado do registrador *Range*.
new_value	Valor atualizado do registrador *Value*.
new_bits_to_go	Índice do *bit* atualizado no *byte* atual do *bitstream*.
new_buffer1	*Byte* atual do *bitstream*.
incr_decode_stream_length	Valor de incremento para o ponteiro do *byte* atual do *bitstream*.
coeff1	Sinal de significância do coeficiente relativo ao índice determinado pela saída coeff_index.
coeff2	Sinal de significância do coeficiente relativo ao índice determinado pela saída coeff_index + 1.
coeff_index	Índice do coeficiente sendo atualizado pelas entradas coeff1 e coeff2.
coeff_reset	Sinal que determina a atribuição de '0' para todos os sinais de significância do conjunto de até 64 coeficientes possíveis.
se_value1	Valor primário decodificado para o elemento sintático solicitado (baseado na JM 10.2).
new_se_value2	Valor secundário atualizado para o elemento sintático solicitado (baseado na JM 10.2).
mem_out	Valor do registro da memória 1 de contextos referente ao endereço definido pela entrada mem_addr. Saída para diagnóstico
mem_out2	Valor do registro da memória 2 de contextos referente ao endereço definido pela entrada mem_addr. Saída para diagnóstico

Tabela 4.5: Portas de saída do módulo de decodificação CABAC

Os componentes que formam esta arquitetura serão descritos mais detalhadamente na continuação deste capítulo.

4.1 Componentes da Arquitetura

4.1.1 BIARI_DECODER

É o módulo que realiza a decodificação binária para os *bins* do tipo regular. Sua interface está representada na Figura 4.3 e a descrição das entradas e saídas na Tabela 4.6.

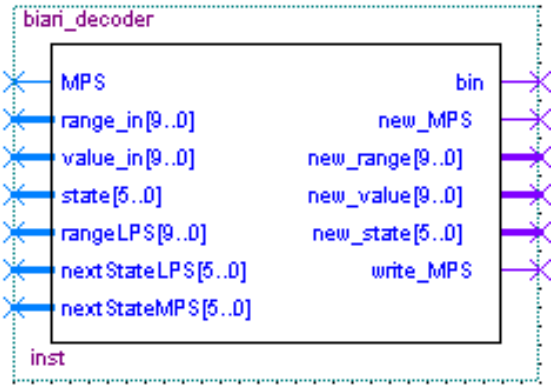

Figura 4.3: Interface do módulo biari_decoder

A decodificação é realizada a partir de dois subtratores, um somador e multiplexadores. A Figura 4.4 é a implementação gerada pela ferramenta de síntese a partir do VHDL.

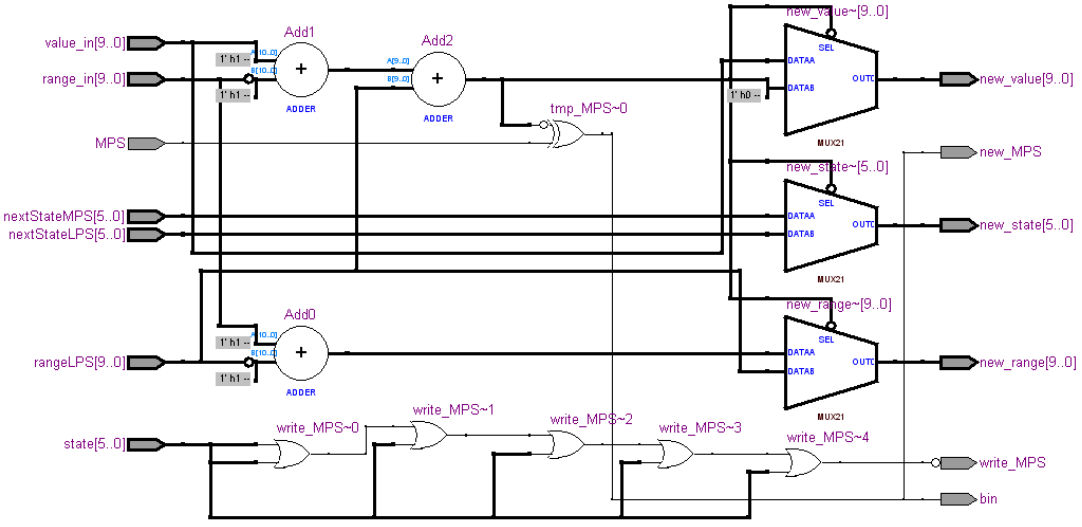

Figura 4.4: Detalhamento interno do módulo biari_decoder

Este módulo está presente dentro dos módulos de decodificação concatenada do tipo A.

Entradas	
Porta	**Descrição**
MPS	*(Most Probable Symbol)* Valor do símbolo mais provável para o processo de decodificação em execução pelo módulo. Informação é recuperada do contexto atual utilizado na decodificação.
range_in	Valor do *Range* a ser utilizado na decodificação.
value_in	Valor do *Value* (ou *Offset*) a ser utilizado na decodificação.
state	Estado da máquina de decodificação aritmética do CABAC. (64 estados possíveis).
rangeLPS	Valor do *Range* correspondente ao símbolo menos provável. Informação recuperada de uma Tabela pré-definida implementada em forma de ROM.
nextStateLPS	Novo estado da máquina de decodificação aritmética no caso do símbolo decodificado ser o menos provável. Informação recuperada de uma Tabela pré-definida implementada em forma de ROM.
nextStateMPS	Novo estado da máquina de decodificação aritmética no caso do símbolo decodificado ser o mais provável. Informação recuperada de uma Tabela pré-definida implementada em forma de ROM.
Saídas	
bin	Valor do símbolo decodificado.
new_MPS	Novo valor do MPS em caso de atualização do contexto.
new_range	Valor atualizado do *Range* após a operação de decodificação.
new_value	Valor atualizado do *Value* (ou *Offset*) após a operação de decodificação.
new_state	Novo estado da máquina de decodificação aritmética.
write_MPS	Sinal de controle para atualização do MPS no contexto.

Tabela 4.6: Portas de entrada e saída do módulo de biari_decoder

4.1.2 BIARI_EQ_PROB_DECODER

É o módulo que realiza a decodificação binária para os *bins* com possíveis valores equiprováveis. Sua interface está representada na Figura 4.5 e a descrição das entradas e saídas na Tabela 4.7.

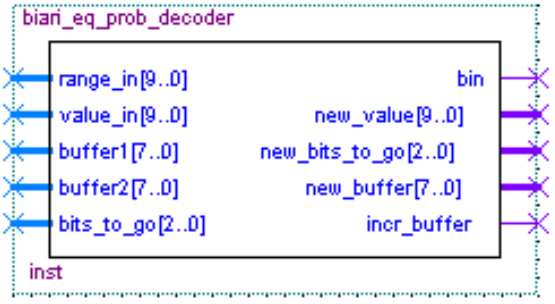

Figura 4.5: Interface do módulo biari_eq_prob_decoder

Entradas	
Porta	**Descrição**
range_in	Valor do *Range* a ser utilizado na decodificação.
value_in	Valor do *Value* (ou *Offset*) a ser utilizado na decodificação.
buffer1	*Byte* atual a ser consumido do *bitstream*.
buffer2	Próximo *Byte* a ser consumido do *bitstream*.
bits_to_go	Índice para o *bit* a ser consumido.
Saídas	
bin	Valor do símbolo decodificado.
new_value	Valor atualizado do *Value* (ou *Offset*) após a operação de decodificação.
new_bits_to_go	Índice atualizado do próximo bit a ser consumido do *bitstream*.
new_buffer	Valor atualizado do *buffer*, com o *byte* atual do *bitstream*.
incr_buffer	Valor de incremento no ponteiro de consumo do *bitstream*.

Tabela 4.7: Portas de entrada e saída do módulo de biari_eq_prob_decoder

A decodificação é realizada a partir de um somador e multiplexadores. A Figura 4.6 é a implementação gerada pela ferramenta de síntese a partir do VHDL.

Este módulo está presente no módulo de decodificação concatenada do tipo B.

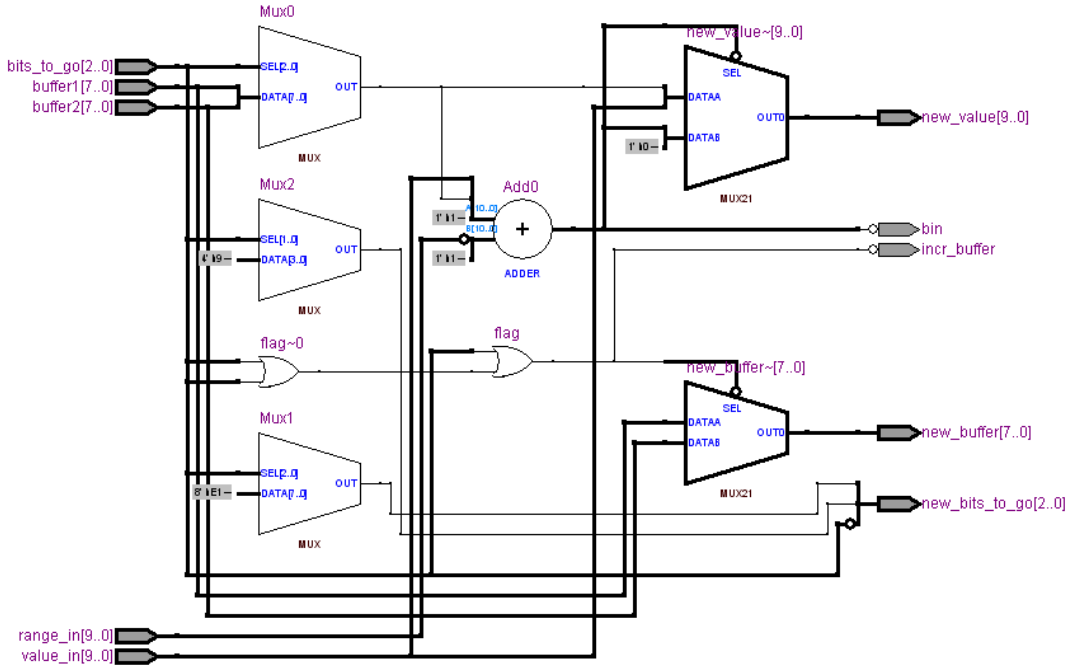

Figura 4.6: Detalhamento interno do módulo biari_eq_prob_decoder

4.1.3 BIARI_FINAL_DECODER

É o módulo que realiza a decodificação binária para os *bins* de término de slice. Sua interface está representada na Figura 4.7 e a descrição das entradas e saídas na Tabela 4.8.

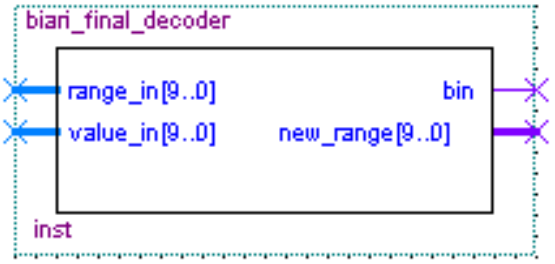

Figura 4.7: Interface do módulo biari_final_decoder

A decodificação é realizada a partir de dois subtratores. A Figura 4.8 é a implementação gerada pela ferramenta de síntese a partir do VHDL.

Este módulo está presente no módulo de decodificação concatenada do tipo A.

Entradas	
Porta	**Descrição**
range_in	Valor do *Range* a ser utilizado na decodificação.
value_in	Valor do *Value* (ou *Offset*) a ser utilizado na decodificação.
buffer1	*Byte* atual a ser consumido do *bitstream*.
buffer2	Próximo *Byte* a ser consumido do *bitstream*.
bits_to_go	Índice para o *bit* a ser consumido.
Saídas	
bin	Valor do símbolo decodificado.
new_range	Valor atualizado do *Range* após a operação de decodificação.

Tabela 4.8: Portas de entrada e saída do módulo de biari_final_decoder

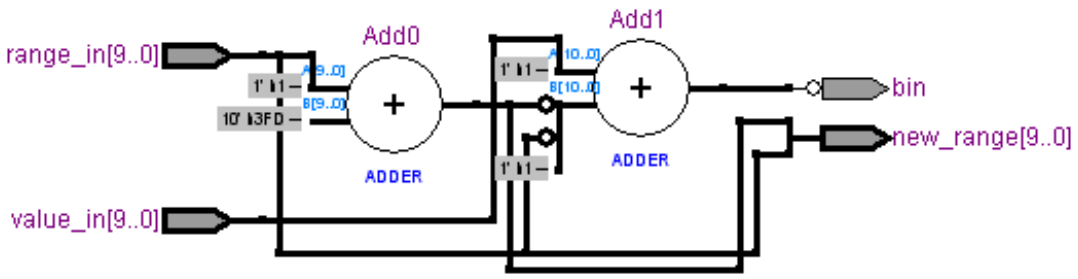

Figura 4.8: Detalhamento interno do módulo biari_final_decoder

4.1.4 FAST_RENORM

É o módulo que realiza a renormalização dos valores de *Range* e *Value* (ou *Offset*). Sua interface está representada na Figura 4.9 e a descrição das entradas e saídas na Tabela 4.9.

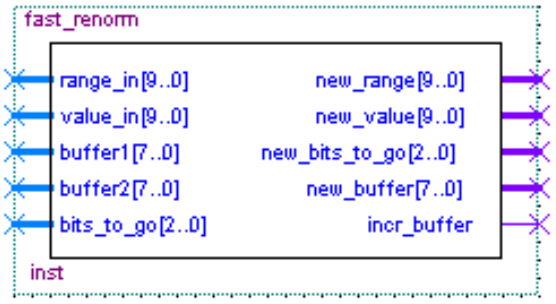

Figura 4.9: Interface do módulo fast_renorm.

Após o processo de decodificação de *bins* regulares e *bins* do símbolo *terminate* é necessário realizar a renormalização do valor correspondente ao intervalo de decodificação (*Range*) e o valor do deslocamento dentro do intervalo (*Value* ou *Offset*). Esta execução precisa ser realizada em um intervalo curto, pois serve de entrada para a próxima máquina de decodificação aritmética.

A implementação neste trabalho é baseada na técnica de renormalização apresentada em Zheng et al. (2007). Esta versão mais simples é construída a partir de um conjunto de *shifters* com deslocamentos incrementais. Os resultados apropriados são selecionados através de multiplexadores. As seleções dos multiplexadores são definidas pelo valor do intervalo de decodificação a ser renormalizado (*Range*). A renormalização só ocorre se os 2 *bits* mais significativos forem iguais a zero, caso contrário a escolha é realizada a partir de uma Tabela pré-definida (256 posições) indexada pelos 8 *bits* restantes. A mesma escolha de renormalização do intervalo é aplicada ao *Value* (ou *Offset*). A precisão utilizada é de 10 *bits* e o deslocamento do novo valor do *Value* é complementado com *bits* do *bitstream* nas posições menos significativas. O *bitstream* é representado por dois *buffers* de 8 *bits* cada, correspondendo aos dois *bytes* mais atuais. A Figura 4.10 é a representação dos componentes internos do módulo.

Este módulo está presente no módulo de decodificação concatenada do tipo A.

4.1.5 DBL_DECODER

Este é o módulo composto pela concatenação de três máquinas de decodificação aritmética (tipo A), possibilitando a decodificação de até três *bins* por ciclos de *clock*, sendo que no máximo dois desses *bins* serão considerados válidos. Sua interface está representada na Figura 4.11 e a descrição das entradas e saídas nas Tabelas 4.10 e 4.11.

Entradas	
Porta	**Descrição**
range_in	Valor do *Range* a ser utilizado na decodificação.
value_in	Valor do *Value* (ou *Offset*) a ser utilizado na decodificação.
buffer1	*Byte* atual a ser consumido do *bitstream*.
buffer2	Próximo *Byte* a ser consumido do *bitstream*.
bits_to_go	Índice para o *bit* a ser consumido.
Saídas	
new_range	Valor atualizado do *Range* após a operação de decodificação.
new_value	Valor atualizado do *Value* (ou *Offset*) após a operação de decodificação.
new_bits_to_go	Índice atualizado do próximo bit a ser consumido do *bitstream*.
new_buffer	Valor atualizado do *buffer*, com o *byte* atual do *bitstream*.
incr_buffer	Valor de incremento no ponteiro de consumo do *bitstream*.

Tabela 4.9: Portas de entrada e saída do módulo de fast_renorm.

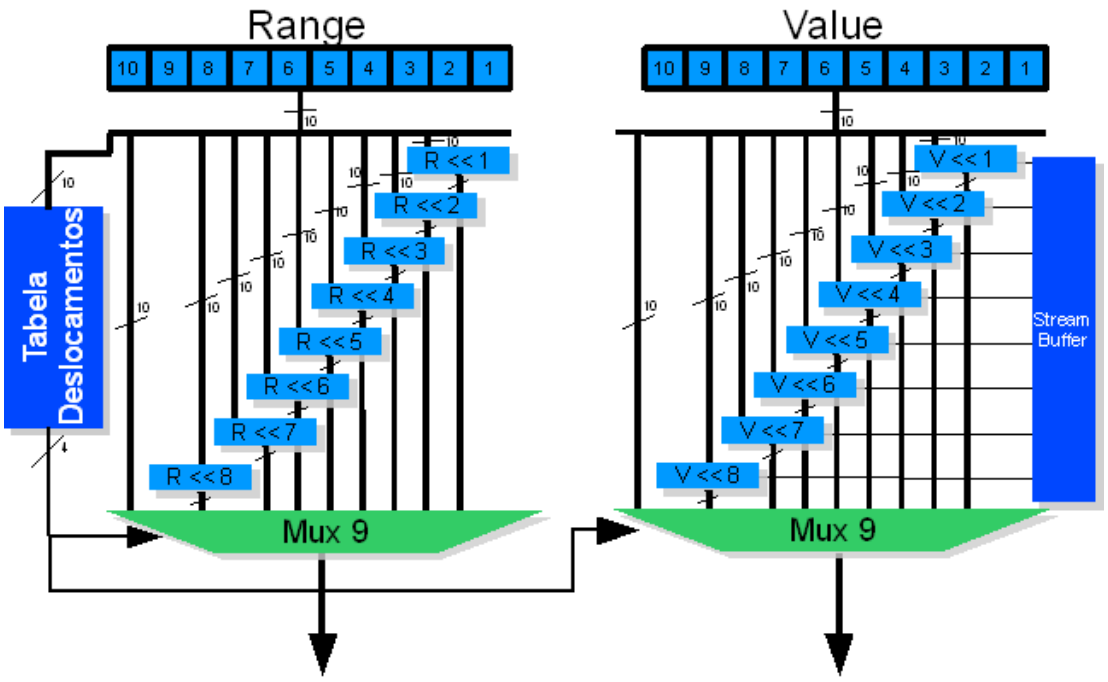

Figura 4.10: Detalhamento interno do módulo fast_renorm.

Entradas	
Porta	**Descrição**
active_in	Sinal para ativar a decodificação do módulo.
context_data	Informação provinda do contexto selecionado para a decodificação do primeiro *bin* regular. Composto de 1 *byte*, onde 1 *bit* é o valor de MPS, 6 *bits* são referentes ao estado de probabilidade da máquina de decodificação e mais 1 *bit* não utilizado.
context2_data	Informação provinda do contexto selecionado para a decodificação do segundo *bin* regular. Composto de 1 *byte*, onde 1 *bit* é o valor de MPS, 6 *bits* são referentes ao estado de probabilidade da máquina de decodificação e mais 1 *bit* não utilizado.
addr_context	Endereço do primeiro contexto a ser utilizado na decodificação.
addr_context2	Endereço do segundo contexto a ser utilizado na decodificação.
condition	Sinal com valor de referência para a decodificação do primeiro *bin* regular. Determina qual será a seleção do segundo *bin* decodificado, baseado no tipo definido pela entrada *decode_type*.
decode_type	Determina qual será a saída do segundo *bin* decodificado. A instrução é formada por 4 *bits*. Os dois *bits* menos significativos determinam a saída do segundo *bin* caso o valor do primeiro *bin* decodificado seja igual ao valor de *condition*. Em outro caso, os outros dois *bits* determinam a saída. Os valores possíveis estão apresentados na Tabela 4.12.
range_in	Valor do *Range* a ser utilizado na decodificação.
value_in	Valor do *Value* (ou *Offset*) a ser utilizado na decodificação.
bits_to_go	Índice para o *bit* a ser consumido.
buffer1	*Byte* atual a ser consumido do *bitstream*.
buffer2	Próximo *Byte* a ser consumido do *bitstream*.
buffer3	*Byte* seguinte ao buffer2 a ser consumido do *bitstream*.

Tabela 4.10: Portas de entrada do módulo de dbl_decoder.

Saídas	
ready	Sinal que informa a conclusão da decodificação.
bins	Valor dos *bins* decodificados. O *bit* na posição menos significativa representa o valor do *bin* determinado pelo primeiro decodificador aritmético do tipo regular. O segundo *bit* representa o valor do *bin* determinado pelo decodificador definido pelas entradas *condition* e *decode_type*.
new_range	Valor atualizado do *Range* após a operação de decodificação.
new_value	Valor atualizado do *Value* (ou *Offset*) após a operação de decodificação.
new_bits_to_go	Índice atualizado do próximo bit a ser consumido do *bitstream*.
new_buffer	Valor atualizado do *buffer*, com o *byte* atual do *bitstream*.
incr_buffer	Valor de incremento no ponteiro de consumo do *bitstream*.
new_context_data	Conteúdo atualizado para o primeiro contexto.
write_context_data	Sinal para realização de atualização do primeiro contexto.
new_addr_context	Endereço do primeiro contexto.
new_context2_data	Conteúdo atualizado para o segundo contexto.
write_context_data	Sinal para realização de atualização do segundo contexto.
new_addr_context	Endereço do segundo contexto.
double_decoding	Sinal que informa se ocorreu a decodificação de dois *bins* válidos.

Tabela 4.11: Portas de saída do módulo de dbl_decoder.

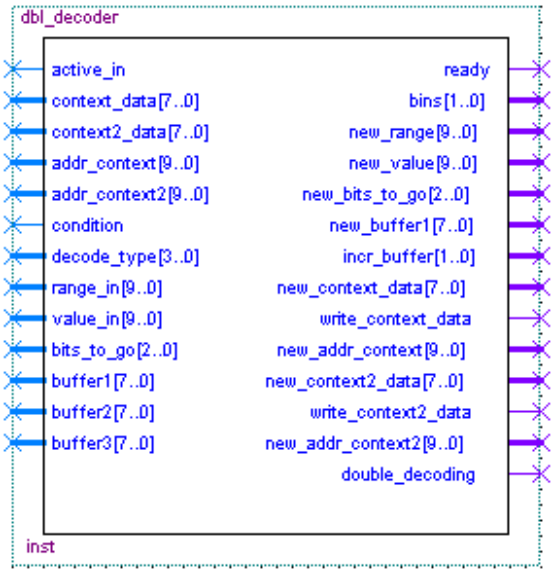

Figura 4.11: Interface do módulo dbl_decoder.

decode_type	**Configuração de decodificação**
00	Decodificação simples. Segundo *bin* definido para zero.
01	Decodificação dupla. Segundo *bin* determinado pelo segundo decodificador aritmético regular.
10	Decodificação dupla. Segundo *bin* determinado pelo decodificador aritmético para o final de slice.
11	Decodificação simples. Segundo *bin* definido para zero.

Tabela 4.12: Configurações de decodificação do módulo dbl_decoder.

O módulo dbl_decoder produz sempre pelo menos um *bin* válido do tipo regular e dois outros *bins* passíveis de validação, um do tipo regular e outro do tipo final. Este *bin* do tipo final é utilizado na decodificação do sufixo de mb_type para slices P e B. A instância deste módulo utilizado na decodificação dos elementos UEGk não necessita da decodificação dos *bins* do tipo final.

A concatenação das três máquina de decodificação necessita de elementos intermediários que farão a preparação e a seleção dos dados produzidos e consumidos por cada máquina de decodificação.

Os valores atualizados de *Range* e *Value* produzidos pelo primeiro decodificador do tipo regular são renormalizados por um módulo fast_renorm. Estes valores renormalizados são passados paralelamente para o módulo de decodificação de símbolo final e outro decodificador do tipo regular. Posteriormente os valores de *Range* e *Value* desses módulos passam por outros módulos fast_renorm e são direcionados para um multiplexador que fará a seleção das saídas do módulo dbl_decoder.

Para os módulos de decodificação do tipo regular existem as entradas e saídas referentes aos contextos. O dado do primeiro contexto é utilizado diretamente pelo primeiro decodificador do tipo regular. O segundo decodificador do tipo regular é alimentado a partir de um multiplexador que possui duas entradas: O valor atualizado do primeiro contexto e os dados do segundo contexto. Uma avaliação nos endereços de contextos determina a seleção do multiplexador. Os valores atualizados dos contextos são direcionados para outro multiplexador que fará a seleção das saídas do módulo dbl_decoder. Quando dois contextos diferentes são utilizados, ambos são atualizados ao final da decodificação. Caso o contexto seja o mesmo para ambos os decodificadores do tipo regular, apenas a saída referente ao segundo contexto será atualizada.

Internamente ao módulo dbl_decoder existem 3 tipos de ROMs que armazenam as entradas pré-definidas necessárias para a decodificação aritmética do tipo regular. Essas ROMs são duplicadas de forma a existir uma instância para cada módulo biari_decoder. As descrições das ROMs estão apresentadas na Tabela 4.13.

As saídas dos módulo dbl_decoder são determinadas por duas entradas: *condition* e decode_type. O valor de *condition* determina qual é o valor esperado para o *bin* decodificado pelo primeiro módulo biari_decoder. O valor de decode_type determina qual será a seleção da saída nos dois casos possíveis para a avaliação do primeiro *bin* decodificado. A entrada decode_type é dividida em duas partes. Os dois *bits* menos significativos determinam a seleção da saída para a condição do *bin* possuir valor igual ao definido por *condition*, os dois *bits* mais significativos determinam a saída para a outra condição. As seleções possíveis estão apresentadas na Tabela 4.12.

A Figura 4.12 apresenta uma visão simplificada da organização interna do módulo dbl_decoder.

Tipo de ROM	Descrição
rLPS_ROM	Contém a Tabela com os valores pré-calculados do *Range* para o símbolo menos provável. Memória de 8 *bits* de endereçamento e palavras de 8 *bits*.
nextSateLPS_ROM	Contém a Tabela com o próximo estado de probabilidade caso o símbolo menos provável ocorra. Memória de 6 *bits* de endereçamento e palavras de 6 *bits*.
nextSateMPS_ROM	Contém a Tabela com o próximo estado de probabilidade caso o símbolo mais provável ocorra. Memória de 6 *bits* de endereçamento e palavras de 6 *bits*.

Tabela 4.13: Memórias ROMs internas ao módulo dbl_decoder.

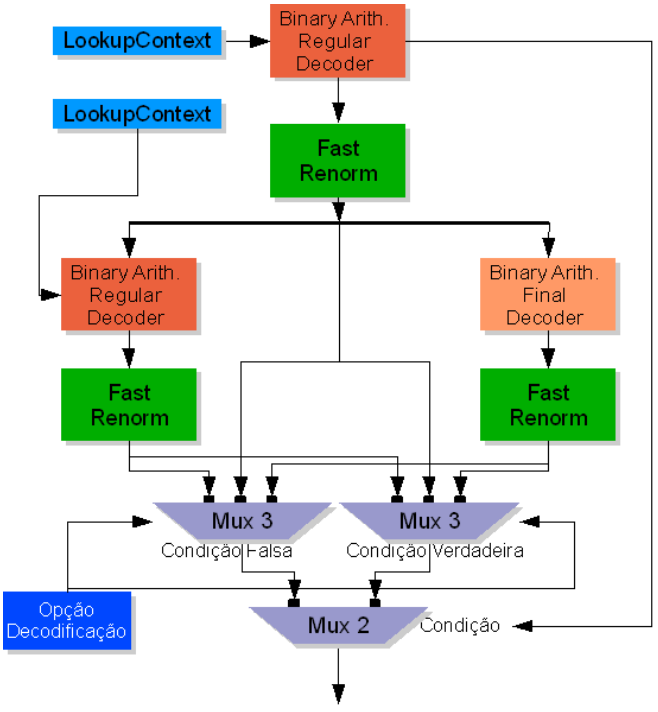

Figura 4.12: Detalhamento interno do módulo dbl_decoder.

4.1.6 DBL_EQ_DECODER

Este é o módulo composto pela concatenação de duas máquinas de decodificação aritmética de *bins* do tipo *bypass* (tipo B), possibilitando a decodificação de até dois *bins* válidos por ciclo de *clock*. Sua interface está representada na Figura 4.13 e a descrição das entradas e saídas nas Tabelas 4.14 e 4.15.

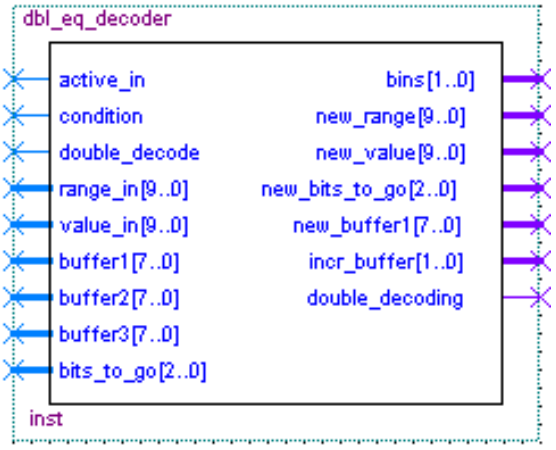

Figura 4.13: Interface do módulo dbl_eq_decoder.

Entradas	
Porta	**Descrição**
active_in	Sinal para ativar a decodificação do módulo.
condition	Sinal com valor de referência para a decodificação do primeiro *bin* do tipo *bypass*. Determina se a decodificação do segundo *bin* é válida.
range_in	Valor do *Range* a ser utilizado na decodificação.
value_in	Valor do *Value* (ou *Offset*) a ser utilizado na decodificação.
buffer1	*Byte* atual a ser consumido do *bitstream*.
buffer2	Próximo *Byte* a ser consumido do *bitstream*.
buffer3	*Byte* seguinte ao buffer2 a ser consumido do *bitstream*.
bits_to_go	Índice para o *bit* a ser consumido.

Tabela 4.14: Portas de entrada do módulo de dbl_eq_decoder.

O módulo dbl_eq_decoder é semelhante ao módulo dbl_decoder. As diferenças estão na utilização de máquinas de decodificação aritméticas para

Saídas	
bins	Valor dos *bins* decodificados. O *bit* na posição menos significativa representa o valor do *bin* determinado pelo primeiro decodificador aritmético do tipo *bypass*. O segundo *bit* representa o valor do *bin* decodificado pelo segundo decodificador aritmético do tipo *bypass*.
new_range	Valor atualizado do *Range* após a operação de decodificação.
new_value	Valor atualizado do *Value* (ou *Offset*) após a operação de decodificação.
new_bits_to_go	Índice atualizado do próximo *bit* a ser consumido do *bitstream*.
new_buffer1	Valor atualizado do *buffer*, com o *byte* atual do *bitstream*.
incr_buffer	Valor de incremento no ponteiro de consumo do *bitstream*.
double_decoding	Sinal que informa se ocorreu a decodificação de dois *bins* válidos.

Tabela 4.15: Portas de saída do módulo de dbl_eq_decoder.

processamento de *bins* do tipo *bypass*, a ausência de módulos de renormalização, a ausência de informações de contexto e de tipo de decodificação. O módulo produz sempre pelo menos um *bin* válido do tipo *bypass* e mais um *bin* do mesmo tipo passível de validação.

As saídas da primeira máquina de decodificação são as entradas da segunda máquina. Apenas as saídas das duas máquinas são utilizadas como entradas de um multiplexador para determinar a saída do módulo dbl_eq_decoder. A seleção desse multiplexador é definido pelo valor da entrada *condition*. Se o valor definido for igual a 0, a segunda decodificação será válida independentemente do valor decodificado para o primeiro *bin*. Se *condition* for igual a 1, a segunda decodificação somente será válida se o primeiro *bin* decodificado for igual a 1.

A Figura 4.14 apresenta uma visão simplificada da organização interna do módulo.

4.1.7 Módulos para cálculo de índice de contexto

A decodificação aritmética para *bins* regulares necessita dos dados armazenados em contextos. A decodificação de cada *bin* necessita que o índice do contexto apropriado seja calculado previamente para que os dados sejam recuperados e operados. Para permitir a decodificação dupla executada pelos módulos dbl_decoder é necessário executar o cálculo para o segundo *bin* a ser decodificado de forma especulativa.

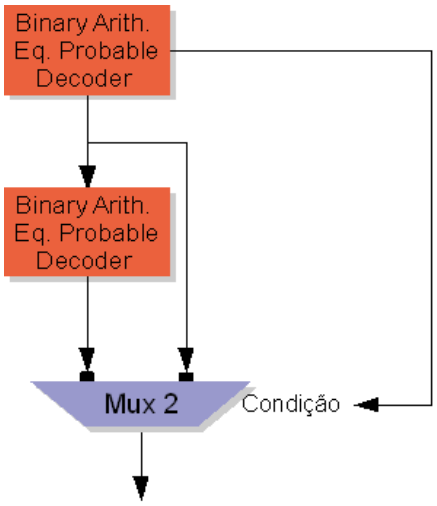

Figura 4.14: Detalhamento interno do módulo dbl_eq_decoder.

Optou-se por utilizar um conjunto de módulos especializados no cálculo dos incrementos dos índices, todos operando através de lógica combinacional baseados em multiplexadores, somadores e subtratores. Cada módulo fornece a seqüência de dois incrementos de índices de um ou mais elementos sintáticos e/ou seu prefixos e sufixos, conforme Tabela representada pela Figura 4.15. Esses incrementos, são relativos ao *bin* atual em decodificação e o seguinte. Tais valores devem ser adicionados a uma base (*ctxOffset*) específica para cada elemento sintático para então se obter o índice completo. O modelo de contextos e a seqüência de índices foram baseados no software de referência JM 10.2, que possui um conjunto de 863 contextos.

As informações dos índice de base estão armazenados em uma memória ROM de 5 *bits* de endereçamento e palavra de 10 *bits* (ctxOffset_rom) endereçados pelo tipo de elemento sintático em processo de decodificação. Alguns elementos sintáticos possuem um índice base variável relativo a parâmetros previamente decodificados. O índice base para esses elementos são calculados a partir de módulos específicos que consultam sub-Tabelas pré-definidas de valores.

Os módulos que calculam os incrementos de índices operam sobre valores de elementos sintáticos previamente decodificados. Esses valores devem ser recuperados e passados para o módulo principal de decodificação (cabac_decoder) por uma entidade externa, como o próprio *parser* do H.264 / AVC. Esses valores são passados através da entrada img_inputs, obedecendo a ordem apresentada na Tabela 4.4.

Os módulos para cálculo dos incrementos dos índices são divididos em dois grupos. O primeiro é formado por todos os módulos apresentados com exceção do módulo decod_ctx_significance_coeff_unary_expg_mod. O segundo é formado pelo módulo decod_ctx_significance_coeff_unary_expg_mod e uma instância do módulo decod_ctx_mb_mvd_mod.

O primeiro grupo é utilizado pela máquina de controle principal para decodificar a maioria dos elementos sintáticos, o segundo grupo é utilizado

Prefixo e/ou sufixo dos elemento(s) sintático(s)		Módulo	
1	mb_field_decoding_flag	decod_ctx_mb_field_decod_flag_mod	1
2	mvdl0	decod_ctx_mb_mvd_mod	2
3	mvdl1		
4	sub_mb_type para slices P e SP	decod_ctx_sub_mb_type_mod	3
5	sub_mb_type para slices B	decod_ctx_sub_mb_type_bslice_mod	4
6	mb_skip_flag	decod_ctx_mb_skip_flag_mod	5
7	transform_size_8x8_flag	decod_ctx_transf_size_8x8_flag_mod	6
8	mb_type para slices I	decod_ctx_mb_type_islice_mod	7
9	mb_type para slices SI	decod_ctx_mb_type_sislice_mod	8
10	mb_type para slices B	decod_ctx_mb_type_bslice_mod	9
11	mb_type para slices P	decod_ctx_mb_type_pslice_mod	10
12	sufixo de mb_type para slices P e B	decod_ctx_mb_type_suffix_pbslice_mod	11
13	intra_chroma_pred_mode	decod_ctx_mb_intra_chroma_pred_mode_mod	12
14	prev_intra4x4_pred_mode_flag	decod_ctx_intra_pred_mode_mod	13
15	prev_intra8x8_pred_mode_flag		
16	rem_intra4x4_pred_mode		
17	rem_intra8x8_pred_mode		
18	ref_idx_l0	decod_ctx_ref_idx_mod	14
19	ref_idx_l1		
20	mb_qp_delta	decod_ctx_mb_qp_delta_mod	15
21	significant_coeff_flag (frame)	decod_ctx_significance_map_frame_nfield_mod	16
22	significant_coeff_flag (field)	decod_ctx_significance_map_nframe_field_mod	17
23	last_significant_coeff_flag (frame e field)	decod_ctx_last_significance_map_mod	18
24	prefixo de coeff_abs_level_minus1	decod_ctx_significance_coeff_mod	19
25	sufixo de coeff_abs_level_minus1	decod_ctx_significance_coeff_unary_expg_mod	20
26	prefixo de coded_block_pattern	decod_ctx_coded_block_pattern_prefix_mod	21
27	sufixo de coded_block_pattern	decod_ctx_coded_block_pattern_suffix_mod	22
28	coded_block_flag	decod_ctx_cbp_flag_not_luma8x8_mod	23

Figura 4.15: Tabela - Elemento sintático e respectivos módulos para índice de contextos

por uma máquina de controle secundária especializada na decodificação dos elementos sináticos mvd_l0, mvd_l1 e coeff_abs_level_minus1.

Cada grupo de módulos possui uma unidade composta por multiplexadores que concentram todos os incrementos dos índices de contextos e os índices de base. O tipo de elemento sintático em processo de decodificação seleciona a base e o incremento específico, os quais são somados e direcionados para a saída. Essa configuração, permite a indexação de dois contextos por vez por grupo de contextos.

4.1.8 Unidades de controle

A coordenação das unidades operativas e do processo de decodificação do módulo decodificador CABAC (cabac_decoder) é realizada a partir de um conjunto de três máquinas de estados organizadas de forma hierárquica (M1, M2 e M3). A primeira máquina de estados (M1) é a principal e coordena a decodificação de todos os elementos sintáticos sob responsabilidade do decodificador CABAC. A segunda máquina de estados (M2) está sob o comando da primeira e é especializada na decodificação dos prefixos de três tipos de elementos sintáticos:

1. mvd_l0;
2. mvd_l1;
3. coeff_abs_level_minus1.

A terceira máquina (M3) está sob o comando da segunda e é especializada na decodificação dos sufixos dos mesmos três elementos sintáticos da segunda máquina. As máquinas M2 e M3 são, em resumo, responsáveis pela decodificação dos elementos sintáticos com binarização do tipo *Unary* concatenada com *Exp-Golomb* de ordem k (UEGk).

A operação de decodificação trabalha em períodos de dois ciclos de *clock*, onde cada ciclo é identificado por um valor de fase (0 ou 1). Na fase 0, as máquinas de decodificação aritmética operam e registram os valores que serão atualizados e avaliados. A fase 1 é a fase de avaliação das saídas das máquinas de decodificação aritmética, recuperação de entradas necessárias e atualização dos estados das máquinas de controle.

A máquina de estados principal M1 possui oito estados válidos. Na fase 0, a operação da máquina de estados controla os sinais de ativação e alguns parâmetros do módulo de decodificação aritmética do tipo A, além de outros dois módulos individuais de decodificação de *bins* do tipo *bypass* (para decodificação de sinal dos elementos sintáticos com binarização UEGk) e *bins* do elemento *terminate*. Na fase 1 a máquina de estados carrega e processa os resultados das máquinas de decodificação aritmética acionadas na fase 0. M1 também atualiza o estado, os sinais de controle das outras duas máquinas (M2 e M3), realiza a anti-binarização da sequência de *bins* e atualiza as saídas do decodificador CABAC (cabac_decoder). Estes sinais são definidos pelo valor do estado atual e dos *bins* recentemente decodificados.

A máquina M2 executa durante a fase 1 e é acionada a partir de sinais coordenados por M1. A máquina M3 também executa durante a fase 1, mas sua ativação é realizada por M2. A máquina M2 controla e avalia os resultados do segundo módulo de decodificação aritmético do tipo A enquanto que a máquina M3 é responsável pelo módulo de decodificação aritmético do tipo B.

O controle da máquina de estados principal (M1) é programada através de microcódigo armazenados em duas memórias ROM:

1. **nextState_symbol_rom**: Armazena a seqüência de estados para a máquina M1 e os valores intermediários e finais para o elemento sintático em processo de decodificação. Essas instruções correspondem ao processo de anti-binarização da seqüência de *bins* decodificados.

2. **otherSignals_rom**: Armazena os sinais de ativação das diversas máquinas de decodificação aritmética, da máquina de estados M2 e os valores de *condition* e decode_type, para o módulo dbl_decoder controlado pel máquina M1.

O controle das duas outras máquinas de estados (M2 e M3) está armazenado internamente na lógica dessas unidades de controle.

Uma versão simplificada das máquinas de estado M1, M2 e M3 está apresentada na Figura 4.16.

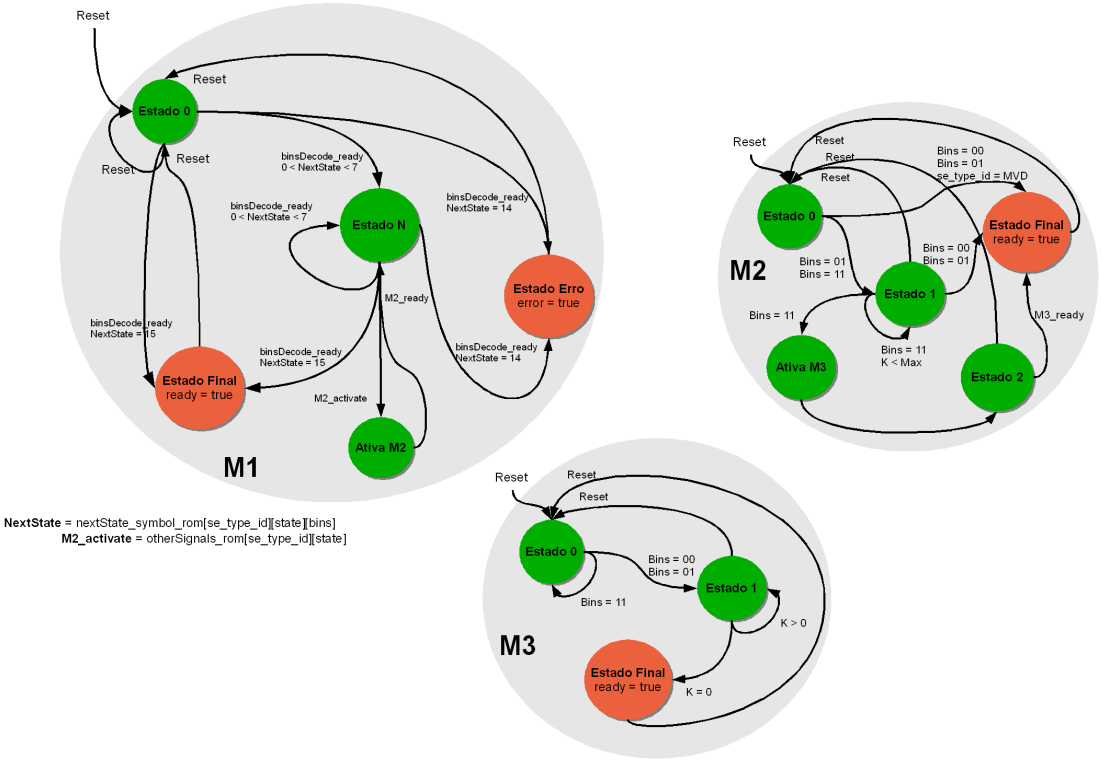

Figura 4.16: Visão simplificada das máquinas de estados M1, M2 e M3.

Além das máquina de estados, a arquitetura possui diversos outros subcomponentes que realizam controles secundários de seleção e registro que

são baseados em estruturas simples, formadas por multiplexadores e registradores.

Entre elas, podemos destacar o controle de atualização dos registradores do ambiente de decodificação. Os seguintes dados:

- *Range*;
- *Value*;
- buffer1;
- incr_decode_stream_length;
- bits_to_go.

São necessários para a manutenção do estado de execução das máquinas de decodificação aritmética. Esses dados são armazenados em registradores coletivos, que são acessados e atualizados individualmente por cada uma das máquinas. O controle de escrita é realizado por uma unidade especializada que atualiza tais registradores durante o ciclo de fase 1, selecionando a entrada apropriada através da avaliação dos sinais de ativação específicos.

Nas sessões seguintes serão detalhadas as informações referentes as duas memórias internas e o processo de execução do decodificador CABAC.

4.1.9 Memórias de contextos

A arquitetura utiliza duas memórias para armazenamento dos dados relativos aos contextos de decodificação. Foi utilizado, como referência, o conjunto de contextos definido pela JM 10.2, que define 863 contextos.

As memórias possuem endereçamento de 10 bits e palavras de 8 *bits*. Onde cada palavra possui a informação sobre o estado de probabilidade do contexto (6 *bits*) e o valor do símbolo mais provável do contexto (MPS) (1 *bit*). O *bit* restante é desprezado.

A primeira memória armazena o conjunto de contextos necessários para todos os elementos sintáticos decodificáveis pelo CABAC, com exceção dos prefixos e sufixos dos elementos sintáticos:

- mvd_l0;
- mvd_l1;
- coeff_abs_level_minus1.

Apesar de possuir capacidade para 1024 palavras, a segunda memória armazena de forma válida apenas 45 contextos referentes as exceções da primeira memória. A utilização de duas memórias de mesma capacidade visa a simplificação do processo de carga de contextos, mas que pode ser facilmente revertida para memórias heterogêneas.

Ambas as memórias utilizadas possuem *dual port*, com leitura e escrita síncrona com um *clock* distinto do *clock* das máquinas de decodificação. Esse *clock* será designado como *mem_clk* no restante do texto. Tais características vem da necessidade de prover a informação de no máximo dois contextos distintos para o módulo de decodificação duplo. O *clock* distinto é utilizado para permitir a leitura e escrita em um intervalo inferior ao ciclo de *clock* do decodificador CABAC.

4.2 Execução

A operação do módulo cabac_decoder é semelhante a operação das funções de decodificação de elementos sintáticos do software de referência JM 10.2. O solicitante deve informar o tipo de elemento sintático a ser decodificado (Tabela 4.16) e os parâmetros necessários a decodificação. Também é necessário a disponibilização dos três *bytes* mais atuais do *bitstream*. O cabac_decoder atualizará a cada ciclo o incremento necessário para a posição atual no *bitstream*. Esse incremento deve ser somado ao índice atual e os *bytes* correspondentes atualizados. Ao final, o decodificador CABAC retornará o valor, ou valores, do elemento sintático decodificado, com precisão de 16 *bits* (complemento de dois) e os valores atualizados dos registradores do ambiente de decodificação.

Tipo de elemento sintático	**Tipo de elemento cabac_decoder**	**ID**
mb_field_decoding_flag	MB_FIELD_DECODING_FLAG_ID	0
mvdl0	MVD_ID	1
mvdl1	MVD_ID	1
sub_mb_type para slices P e SP	SUB_MB_TYPE_P_SP_ID	2
sub_mb_type para slices B	SUB_MB_TYPE_B_ID	3
mb_skip_flag para slices B	MB_SKIP_FLAG_B_ID	4
mb_skip_flag para slices P e SP	MB_SKIP_FLAG_P_SP_ID	5
transform_size_8x8_flag	TR_SIZE_8X8_FLAG_ID	6
mb_type para slices I	MB_TYPE_I_ID	7
mb_type para slices SI	MB_TYPE_SI_ID	8
mb_type para slices B	MB_TYPE_B_ID	9
mb_type para slices P	MB_TYPE_P_ID	10
sufixo de mb_type para slices P e B	MB_TYPE_SUFFIX_PB_ID	11
intra_chroma_pred_mode	CI_PREDMODE_ID	22
prev_intra4x4_pred_mode_flag	INTRA_PRED_MODE_ID	12
prev_intra8x8_pred_mode_flag	INTRA_PRED_MODE_ID	12
rem_intra4x4_pred_mode	INTRA_PRED_MODE_ID	12
rem_intra8x8_pred_mode	INTRA_PRED_MODE_ID	12
ref_idx_l0	MVD_REF_IDX_ID	13
ref_idx_l1	MVD_REF_IDX_ID	13
mb_qp_delta	DQUANT_ID	14
significant_coeff_flag (frame)	MAP_SIG_ID	17
significant_coeff_flag (field)	MAP_SIG_ID	17
last_significant_coeff_flag (frame e field)	MAP_SIG_ID	17
prefixo de coeff_abs_level_minus1	COEF_ABS_ID	18
sufixo de coeff_abs_level_minus1	COEF_ABS_ID	18
prefixo de coded_block_pattern	CBP_PREFFIX_ID	19
sufixo de coded_block_pattern	CBP_SUFFIX_ID	20
coded_block_flag	CBP_BLOCK_BIT_ID	23
end_of_slice_flag	END_OF_SLICE_FLAG_ID	24

Tabela 4.16: Identificadores dos elementos sintáticos decodificáveis pelo módulo cabac_decoder.

O ciclo de operação ocorre na seguinte ordem:

1. Acionamento do sinal *reset*;
2. Atribuição das entradas correspondentes ao elemento sintático a ser decodificado;
3. Desativação do sinal de reset e o acionamento do sinal de ativação;
4. Atualização da posição do *bitstream* a partir do incremento no ponteiro informado pelo módulo.
5. Monitoramento do sinal *ready* ou *error*;
6. Desativação do sinal de ativação;
7. Recuperação dos valores decodificados ou tratamento do erro ocorrido.

Alguns elementos sintáticos necessitam de operação especializadas conforme descrito nos trechos seguintes.

- As decodificações dos elementos significant_coeff_flag e last_significant_coeff_flag são realizadas a partir de um único acionamento do módulo para o elemento referente ao mapa de significância (MAP_SIG). O módulo cabac_decoder aciona o sinal de *coeff_reset* para zerar uma memória externa, que servirá para o armazenamento do mapa. O conjunto de sinais é decodificado e disponibilizado através das portas coeff1 e coeff2, a cada dois ciclos de *clock*. Esses valores precisam ser armazenados na memória externa específica. O valor de coeff_index funciona como indexador para a posição de coeff1, enquanto que a posição de coeff2 é determinada por coeff_index + 1. Ao final o módulo indica através do sinal *ready* que todo o mapa de significância foi decodificado.
- Os elementos sintáticos do tipo coeff_abs_level_minus1 devem ser decodificados através de chamadas individuais para cada elemento da matriz que possuir o *flag* correspondente de significância ativo.
- O elemento sintático coded_block_pattern é decodificado através de até duas chamadas que devem ser coordenadas pelo solicitante. A decodificação é realizada para o prefixo e em caso da imagem possuir informação de chroma, é necessário fazer uma solicitação de decodificação do sufixo. Ao final, os dois valores devem ser somados para obter o valor correto para o elemento.

Entre as atividades executadas pelo decodificador CABAC proposto, não estão inclusas as atividades relacionadas com a inicialização e carregamento dos contextos e a inicialização dos registradores *Range* e *Value*.

Este capítulo apresentou uma descrição detalhada da arquitetura desenvolvida neste trabalho. O decodificador CABAC, projetado para *hardware*, possui otimizações que permitem a decodificação simultânea de até dois *bins* válidos em apenas um ciclo.

Capítulo 5

Metodologia e Resultados

5.1 Metodologia

O trabalho iniciou com o estudo de publicações (livro, artigos e outras fontes) que descrevem o funcionamento da codificação H.264 / AVC, do CABAC, do codificador aritmético Q-Coder e de propostas de arquiteturas otimizadas para a decodificação CABAC.

Após obter conhecimento sobre o funcionamento do CABAC foi possível realizar uma avaliação mais prática do funcionamento através da análise do software de referência para o padrão H.264 / AVC (software JM versão 10.2 da ITU). Através deste software foi possível esclarecer dúvidas sobre a norma.

A partir das propostas de arquiteturas CABAC otimizadas e da análise do processo de decodificação, foi possível definir o primeiro esboço da arquitetura proposta. A arquitetura se baseia na utilização de módulos de decodificação formados por concatenações de máquinas de decodificação binária m-*coder*, permitindo a decodificação de mais de um *bin* das strings de *bits* decodificáveis em um mesmo ciclo. Por ser um processo essencialmente seqüencial, onde a decodificação de cada *bin* depende da decodificação do *bin* antecessor, é necessário elaborar mecanismos que provêem o suporte necessário para o fornecimento dos dados de entrada e que tratem e avaliem os dados das saídas. Devido a complexidade envolvida na construção e operação desses mecanismos opta-se por reduzir o escopo de otimização apenas para um subconjunto de elementos decodificáveis. Esse subconjunto é composto pelos elementos mais freqüentes e/ou que possuem maior comprimento de *bins*. A arquitetura proposta por esse trabalho busca realizar a decodificação dupla para todos os elementos decodificáveis, baseado na definição de elementos operativos e de controle que possuem poucas diferenças na complexidade de operação para o conjunto completo de elementos sintáticos.

A metodologia utilizada se baseia em prototipação. O software de referência JM foi recodificado parcialmente de forma manual. Foram realizadas alterações nas funções do decodificador CABAC, visando obter uma reestruturação no fluxo de operação semelhante ao modelo de arquitetura

proposta. A recodificação buscou aproximar o código de *software* da descrição de *hardware*, provida por linguagens de descrição de *hardware* de alto nível, tais como VHDL e SystemC.

A versão alterada do software JM foi então testada, utilizando a decodificação de quatro seqüências de vídeos de referência:

- *Foreman*;
- *Tennis*;
- *Sunflower* e
- *Rushhour*.

Todas as seqüências foram codificados no formato H.264 / AVC através do codificador do software de referência. As seqüências foreman e tennis foram codificadas utilizando o perfil *High* no nível 2.0 e slices do tipo B. A seqüência sunflower foi codificada utilizando o perfil *Main* nível 4.0 e slices do tipo B e a seqüência rushhour utilizou o perfil *Main* nível 4.0 e slices do tipo P. Maiores informações sobre as seqüências estão apresentadas na Tabela 5.1. Os testes buscavam apenas a validação funcional da JM alterada. Por estar diretamente acoplado ao restante do *software* de referência, o teste é bastante ágil e permite correções mais rápidas no funcionamento básico da arquitetura proposta.

Nome	**Resolução / fps**	***Frames***	**Perfil**	**Nível**	***Slices***	**Período entre *frames* I**	**Quantização**
Foreman	352x288 / 30fps	299	*High*	2.0	B	0*	I: 28, P: 28, B: 30
Tennis	352x240 / 30fps	149	*High*	2.0	B	0*	I: 28, P: 28, B: 30
Sunflower	1920x1080 / 30fps	119	*Main*	4.0	B	3	I: 28, P: 28, B: 30
Rushhour	1920x1080 / 25fps	149	*Main*	4.0	P	3	I: 28, P: 28, B: 30

Tabela 5.1: Informações sobre a codificação das seqüências de referência.

0* - Apenas o primeiro frame do tipo I.

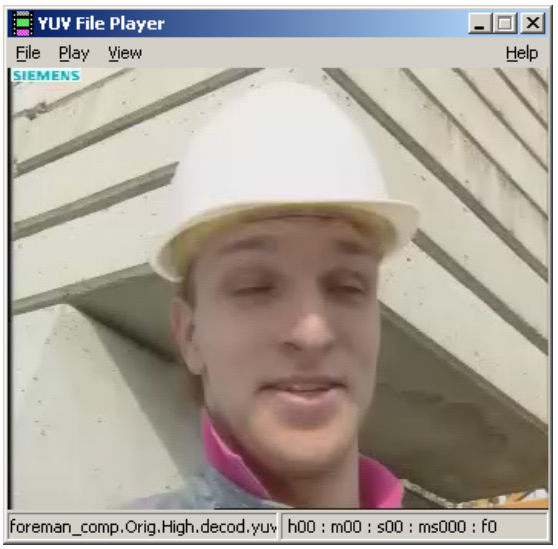

Figura 5.1: *Frame* da sequência Foreman.

Figura 5.2: *Frame* da sequência Tennis.

Figura 5.3: *Frame* da sequência Sunflower.

Figura 5.4: *Frame* da sequência Rushhour.

Após a validação da JM alterada, foi iniciado o desenvolvimento do decodificador em uma linguagem mais próxima e com maior suporte para a descrição de hardware. Dessa forma, foi gerado, a partir das funções modificadas da JM, uma descrição em SystemC do decodificador CABAC. A descrição utilizando esse *framework* para C++, permitiu uma modelagem mais semelhante ao *hardware*, com suporte a especifidades, tais como os ciclos de *clock*, sinais e comportamento de circuitos combinacionais. Avaliação realizada pela ferramenta CLOC v. 1.08 (Danial (2009)) relata que a descrição em SystemC ocupou 6085 linhas de código e *headers*, enquanto que o módulo de teste ocupou 615 linhas de código (excluindo-se em ambos, comentários e linhas em branco). Essa versão já não pôde ficar acoplada diretamente ao restante do *software* de referência e precisou ser testada através da injeção de sinais de entradas e comparação de sinais de saídas com informações capturadas durante o processo de decodificação da JM 10.2, conforme ilustrado na Figura 5.5. O registro dessas informações foi realizado em 7 arquivos binários. O conjunto foi dividido da seguinte forma:

1. Entradas do módulo cabac_decoder (signals_dec.bin);
2. Saídas do módulo cabac_decoder (sifnals_dec_out.bin);
3. Contextos pré-decodificação (contexts_dec.bin);
4. Contextos pós-decodificação (contexts_dec_out.bin);
5. Coeficientes pré-decodificação (coeffs_dec.bin);
6. Coeficientes pós-decodificação (coeffs_dec_out.bin);
7. Marca de iniciação de contextos (init_contexts_dec.bin);

A seqüência da função principal é: A carga dos arquivos de entrada; Atualização dos contextos e coeficientes e execução da chamada ao módulo do decodificador CABAC. Após a decodificação, a função compara os contextos, coeficientes e saídas do módulo com as respectivas informações lidas

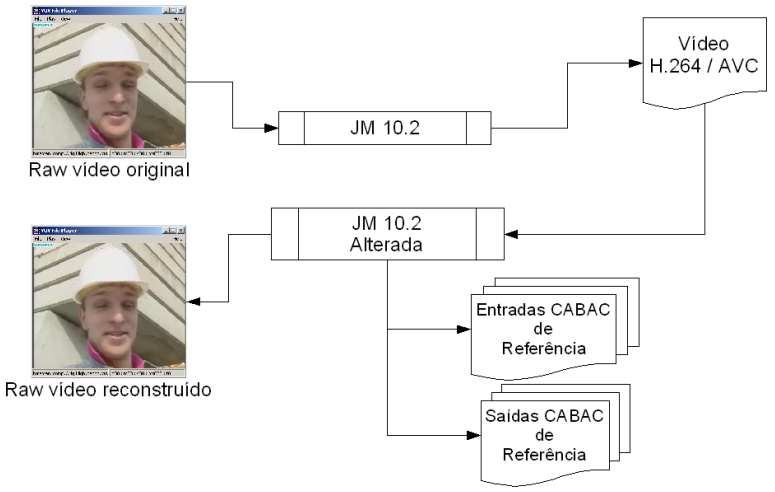

Figura 5.5: Processo para gerar arquivos com os dados de referência para testes.

dos arquivos, conforme ilustrado na Figura 5.6. Os erros encontrados eram confrontados através da comparação da execução passo-a-passo do módulo SystemC contra a execução da versão alterada da JM.

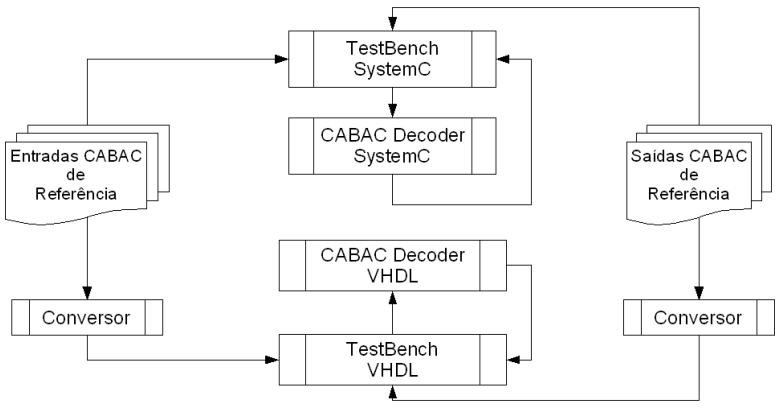

Figura 5.6: Processo de verificação das descrições SystemC e VHDL.

Através da versão em SystemC foi possível eliminar as principais falhas funcionais do modelo da arquitetura e determinar uma prévia da relação de ciclos / *bins* decodificados que a proposta poderia prover.

Ao adquirir certa estabilidade de funcionamento, o modelo SystemC foi re-codificado manualmente em VHDL, uma linguagem de alto nível para descrição de *hardware*. A nova codificação corresponde a versão a ser implantada em *hardware*. Através dessa versão é possível avaliar os tempos reais de decodificação e a freqüência máxima dos *clocks*. Os testes funcionais se tornam mais complexos pela impossibilidade de utilizar diretamente as informações de teste das etapas anteriores.

A versão em VHDL precisou ser validada através de uma entidade de *testbench*, também desenvolvida em VHDL. A unidade de *testbench* engloba

o cabac_decoder, injeta os sinais de entrada e avalia os sinais de saída. O sinais de entrada e saída são adquiridos das informações extraídas durante o processo de decodificação pelo decodificador alterado da JM. Tais dados são previamente processados a partir dos arquivos binários para um formato texto legível pelo VHDL. O módulo de *testbench* também realiza as atividades de carga inicial de contextos e dos registradores do ambiente de decodificação, recupera os valores decodificados do mapa de significância e armazena-os em um *array* local. Os sinais de saída são comparados com as saídas de referência e os erros são registrados em um arquivo de *log*.

Avaliação realizada pela ferramenta CLOC v. 1.08 (Danial (2009)) relata que a descrição em VHDL ocupou 6736 linhas de código, enquanto que o módulo de teste ocupou 733 linhas de código (excluindo-se em ambos, comentários e linhas em branco).

Os testes da versão em VHDL foram executados utilizando a ferramenta ModelSim ALTERA WEB EDITION 6.3g_p1 Rev. 2008.08 de Agosto de 2008.

5.2 Resultados da síntese

A descrição em VHDL foi compilada através da ferramenta Quartus II Version 9.0 Build 184 04/29/2009 SP 1 SJ Web Edition, que gerou o resultado apresentado na Tabela 5.2.

Quartus II Version 9.0 Build 184 04/29/2009 SP 1 SJ Web Edition	
Nome da entidade *Top-level*	cabac_decoder
Família	Stratix II GX
Dispositivo	EP2SGX30DF780C3
Utilização lógica ALUTs combinacionais Registradores lógicos dedicados	17 % 3,472 / 27,104 (13 %) 439 / 27,104 (2 %)
Total de registradores	439
Total de pinos	235 / 406 (58 %)
Total de pinos virtuais	0
Total de *bits* de blocos de memória	16,384 / 1,369,728 (1 %)

Tabela 5.2: Informações sobre a compilação do módulo cabac_decoder.

O módulo utiliza de cerca de 3.472 ALUTs (*Adaptive Lookup-Tables*) e 16.384 *bits* de blocos de memória de um FPGA Stratix II GX da Altera. Além de 17% dos recursos lógicos e 1% dos recursos de memória do dispositivo. Incluindo as duas memórias RAM para os contextos e as diversas memórias ROM.

Para análise de tempo, foi utilizado a ferramenta Altera TimeQuest Timing Analysis, a qual apontou para o dispositivo, freqüências máximas de 75,57 MHz para o clk e 500 MHz para mem_clk.

5.3 Avaliação da arquitetura

O módulo projetado permite a decodificação de todos os elementos sintáticos decodificáveis pelas funções de decodificação CABAC do software de referência JM 10.2. Estão inclusos os elementos sintáticos necessários para decodificação de *slices* do tipo I, P, B, SP e SI (*Slices* do tipo SP e SI não foram testados funcionalmente).

A partir da arquitetura desenvolvida e da programação do controle do decodificador é possível analisar quais seriam as taxas de ciclos/*bin* para cada elemento decodificável, no pior e melhor caso. A Tabela 5.3 apresenta essa análise.

Tipo de elemento cabac_decoder	**ID**	**Pior relação Ciclos/bin**	**Melhor relação Ciclos/bin**
MB_FIELD_DECODING_FLAG_ID	0	3	3
MVD_ID	1	3	1,44
SUB_MB_TYPE_P_SP_ID	2	3	1,67
SUB_MB_TYPE_B_ID	3	3	1,17
MB_SKIP_FLAG_B_ID	4	3	3
MB_SKIP_FLAG_P_SP_ID	5	3	3
TR_SIZE_8X8_FLAG_ID	6	3	3
MB_TYPE_I_ID	7	3	1,4
MB_TYPE_SI_ID	8	3	3
MB_TYPE_B_ID	9	3	1,17
MB_TYPE_P_ID	10	2,5	2,33
MB_TYPE_SUFFIX_PB_ID	11	3	1,4
INTRA_PRED_MODE_ID	12	3	1,25
MVD_REF_IDX_ID	13	3	1,25
DQUANT_ID	14	3	1,25
SIG_COEF_ID	15	NA	NA
LAST_SIG_COEF_ID	16	NA	NA
MAP_SIG_ID	17	3	1,5
COEF_ABS_ID	18	3	1,35
CBP_PREFFIX_ID	19	2,25	1,25
CBP_SUFFIX_ID	20	3	1,5
CBP_CHROMA_ID	21	NA	NA
CI_PREDMODE_ID	22	3	1,5
CBP_BLOCK_BIT_ID	23	3	3
END_OF_SLICE_FLAG_ID	24	3	3

Tabela 5.3: Taxas de ciclos/*bin* para o pior e melhor caso da decodificação.

A contagem leva em consideração apenas os ciclos necessários para o decodificador concluir sua operação e não inclui os ciclos necessários de início de *slice* ou de recuperação de parâmetros de entrada, executados por entidades externas ao módulo.

A relação de ciclos/bin geralmente é pior para os elementos sintáticos representados por apenas um *bin*. Essa mesma relação tende a diminuir quando o elemento sintático é formado por diversos *bins*.

Os resultados obtidos comprovam uma arquitetura que possibilita a decodificação dos *bins* a uma taxa de 3 ciclos por *bin* decodificado no pior caso e 1,84 ciclos/*bin* na média dos melhores casos.

A partir das freqüências máximas determinadas, foi possível fazer uma avaliação do processo de decodificação das seqüências de vídeos de referência. Para a avaliação, determinou-se que a carga dos contextos, iniciados para cada slice, seria realizada em 432 ciclos do *clock* das memórias

(mem_clk). A cada ciclo, dois registros de cada uma das duas memórias seriam atualizados. No caso dos testes funcionais, essa carga foi de responsabilidade do módulo de *testbench*. A Tabela 5.4 apresenta a avaliação realizada. As freqüências utilizadas foram: *clk*: 72,46 MHz e *mem_clk*: 434,78 MHz.

Seqüência	***Frames***	**FPS**	**Número de iniciações de contextos**	***Bins* decodificados**	**Mbins / segundo**	**Tempo total para decodificação dos *bins***	**Taxa bruta Ciclos/*bin***	**Taxa líquida Ciclos/*bin***
Foreman	299	30	299	3.602.571	0,4	0,106 seg	2,13	2,12
Tennis	149	30	149	2.917.779	0,6	0,083 seg	2,08	2,07
Sunflower	119	30	119	23.152.747	5,8	0,690 seg	2,16	2,16
Rushhour	149	25	149	43.344.701	7,3	1,271 seg	2,13	2,13

Tabela 5.4: Avaliação da decodificação das seqüências dos vídeos de referência.

Taxa bruta: Avaliação com a inclusão dos ciclos gastos com a carga dos contextos iniciados;
Taxa líquida: Avaliação sem a inclusão dos ciclos gastos com a carga dos contextos iniciados;

Baseado na avaliação anterior, podemos fazer uma estimativa do desempenho do módulo cabac_decoder para decodificação de seqüências genéricas. As freqüências de *clock* adotadas serão: clk: 75MHz e mem_clk: 450MHz.

Para essa estimativa, assume-se que o processo externo que realiza a preparação dos parâmetros de entrada utilizaria cerca de 7 ciclos em média para cada elemento sintático. Esse valor é a aproximação da média de parâmetros necessários por elemento sintático, que é cerca de 6,73. A partir da maior taxa de elementos sintáticos por segundo (cerca 3,5 Milhões/s) processados para a seqüência Rushhour, define-se então a taxa de 4 Milhões de elementos sintáticos por segundo para a estimativa. Assim chegamos a taxa de 28 Milhões de ciclos por segundo para a preparação dos parâmetros de entrada. Adotando-se então que a freqüência de execução do módulo externo será o dobro da freqüência do cabac_decoder, ou seja 150 MHz, teremos, para o processo de decodificação de *bins* cerca de 81% do tempo total.

As avaliações da relação máxima e compressão máxima utilizam como base a taxa máxima do *bitstream* para o perfil 4.0, que é de 20 Mbits/segundo, com exceção do caso médio para seqüências CIF, que utiliza a taxa para o perfil 2.0, que é de 2 Mbits/segundo. A Tabela 5.5 apresenta a avaliação.

Situação	**% tempo disponível**	**Taxa de ciclos / *bin***	**Taxa máxima de *Mbins* / segundo**	**Relação máxima de *bins/bit***	**Compressão entrópica máxima do *bitstream* suportada**
Pior caso	81	3	20,25 Mbins/s	1,01	1,23%
Melhor caso	81	1,84	33 Mbins/s	1,65	39,42%
Caso médio 1080p	81	2,15	28,26 Mbins/s	1,41	29,22%
Caso médio CIF	81	2,1	28,93 Mbins/s	14,46	93,09%

Tabela 5.5: Estimativa de desempenho para seqüências de vídeo genéricas.

O melhor caso é a média das melhores taxas de ciclos/*bin* de cada elemento sintático;
Caso médio 1080p: Média das taxa de ciclos/*bin* das seqüências sunflower e rushhour, relatadas anteriormente;
Caso médio CIF: Média das taxa de ciclos/*bin* das seqüências foreman e tennis, relatadas anteriormente;

Baseado no desempenho do caso médio para seqüências 1080p, seria possível utilizar o decodificador CABAC desenvolvido na decodificação de vídeos 1080p em tempo real, com limitação para taxas de compressão de *bins* para *bit* superiores a 29,22%, quando a taxa do *bitstream* alcança o limite máximo de 20 Mbits/s.

5.4 Comparação da proposta

A comparação da arquitetura aqui proposta com os trabalhos de referência apresentados no capítulo 3 não pode ser feita de forma direta, pois cada autor pode utilizar uma métrica diferente para avaliar o desempenho. Além disso, os trabalhos implementam decodificadores com níveis distintos de funcionalidades.

Para tentar fazer uma comparação, vamos optar pela métrica de quantidade de bins decodificados por segundo, que é uma informação apresentada por quase todos os trabalhos, apesar da própria contagem ser realizada de formas diferentes. A Tabela 5.6 apresenta tais informações.

Trabalho		**bins / ciclo**	**Freqüência máxima**	**Mbins / segundo**	**Observações**
1	Yu e He (2005)	1 a 3 bins/ciclo	~149 MHz	~149 a ~447	Taxas mínima e máxima
2	Zheng et al. (2007)	~0,25 bins/ciclo	~100 MHz	~25	Taxa média
3	Chen e Lin (2007)	1,24 bins/ciclo	137 MHz	169	Taxa média
4	Yi e Park (2007)	~0,25 bins/ciclo	225 MHz	~56	Taxa média
5	Xu et al. (2007)	Não Informado	~80 MHz	-	Não é possível calcular
6	Zhang et al. (2007)	~16 bins/ciclo	45 MHz	~720	Taxa máxima
7	**Arquitetura proposta**	**~0,47 x 81% = ~0,38 bins/ciclo**	**75 MHz**	**~28,5**	**Taxa média 1080p**

Tabela 5.6: Comparação de desempenho entre as propostas de decodificadores CABAC.

Como podemos observar, a arquitetura proposta aqui, possui desempenho inferior a maioria das arquiteturas de referência, mas a taxa para o caso médio 1080p é superior a taxa apresentada pela proposta 2 (Zheng et

al. (2007)). Vale lembrar que no caso do nosso trabalho, essa taxa é obtida através de estimativas que incluem o tempo para iniciação de contextos e registradores de ambiente, preparação e acionamento do decodificador. A proposta 5 (Xu et al. (2007)) não traz informações de taxas de decodificação.

Também é preciso considerar que grande parte dos trabalhos consideram parte do escopo do decodificador uma ou as duas dessas tarefas:

- Realizar o controle da seqüência de elementos sintáticos, seja para o *Slice*, *Macrobloco* ou partição;
- Armazenar os elementos sintáticos e recuperá-los como parâmetros de uma nova decodificação;

Essas tarefas são ao mesmo tempo problema e solução. Problema, pois são atividades extras que tornam o decodificador mais complexo e conseqüentemente afetam a freqüência máxima final. Solução, pois permite a preparação prévia da estrutura de decodificação para os elementos seguintes de decodificação, poupando ciclos de iniciação.

A arquitetura aqui proposta, considera que o controle da seqüência e a manipulação dos elementos sintáticos devem ser responsabilidades de uma entidade externa, como por exemplo, o próprio *parser* do H.264 / AVC, visto que a norma não especifica que tais tarefas sejam atividades do decodificador CABAC. A percepção é que um módulo externo, com tais tarefas, possa executá-las de maneira mais eficiente e com uma limitação de freqüência máxima de operação superior ao do decodificador CABAC.

Este capítulo apresentou a metodologia utilizada no desenvolvimento e validação da arquitetura. Apresenta também o resultado da síntese em *hardware* e a avaliação do desempenho alcançado.

Capítulo 6

Conclusão

A proposta do trabalho deste mestrado era projetar a arquitetura de um decodificador CABAC com suporte aos diversos perfis e *slices*, com desempenho suficiente para prover suporte a decodificação em tempo real de vídeos H.264 / AVC com resolução 1920x1080p - 30 fps, perfil *Main* nível 4.0.

Conforme apresentado nessa dissertação, foram realizadas atividade de estudo do problema, análise de soluções e trabalhos relacionados, desenvolvimento de software de apoio, desenvolvimento da arquitetura no *framework* SystemC, descrição da mesma arquitetura em VHDL, validação funcional da arquitetura, análise do desempenho e comparação de resultados.

Os resultados demonstram que a arquitetura utiliza cerca de 17% da capacidade lógica e 1% da capacidade de memória dedicada de um dispositivo FPGA Stratix II GX EP2SGX30DF780C3. O sistema pode operar com *clocks* de freqüência máxima de 75MHz para o decodificador e 500MHz para as memórias RAM utilizadas.

A partir das informações levantadas, é possível estimar que o desempenho da arquitetura permite a decodificação com taxas máximas que variam de 20 a 33 Mbins/segundo, o que a princípio seria suficiente para a decodificação de vídeos no formato e resolução alvo.

Podemos destacar algumas característica adotadas para a arquitetura:

1. Decodificador formado por concatenação de duas máquinas de decodificação aritmética para *bins* do tipo regular;
2. Decodificador formado por concatenação de duas máquinas de decodificação aritmética para *bins* do tipo *bypass*;
3. Renormalizador de um ciclo, baseado em deslocamentos pré-definidos em ROM;
4. Divisão do controle de decodificação em três máquinas de estados;
5. Controle especializado em decodificação de elementos no formato UEGk;
6. Otimização do controle para realizar decodificação de dois *bins* por ciclo abrangendo todos os elementos sintáticos multi-*bins*;

7. Utilização de duas memórias RAM dual-port para contextos;

8. Suporte a decodificação dos elementos sintáticos para slices I, P e B.

Apesar de possuir um desempenho inferior a maioria das referências apresentadas, a arquitetura alternativa desenvolvida provou que o decodificador CABAC, com otimização para todos os elementos sintáticos decodificáveis, apresenta desempenho suficiente para o propósito original, mesmo sintetizado, sem otimizações, para dispositivo FPGA. O decodificador foi testado para pelo menos dois perfis *main* e *high*.

A área ocupada pela arquitetura e a interface de comunicação simples permite sua integração com outros elementos que constituirão um decodificador de vídeos no formato H.264 / AVC. O qual ainda pode, posteriormente, ser integrado a um sistema em silício mais complexo.

6.1 Trabalhos Futuros

O trabalho aqui desenvolvido possui diversos pontos que podem ser reavaliados, complementados e alterados. Os seguintes foram identificados durante a execução:

1. É necessário implementar a iniciação dos contextos e dos registradores de ambiente de decodificação (*Range, Value ou Offset*). A implementação pode ser realizada através de um módulo independente do decodificador, que facilite a integração a memória de contextos e tire proveito da alta freqüência de execução;

2. A segunda instância do módulo duplo de decodificação (dbl_decoder) pode ser alterado para remover a máquina de decodificação de *bin* do tipo *terminate* (biari_final_decoder) interno, visto que o mesmo é desnecessário para o processo de decodificação UEGk;

3. Pode-se considerar a substituição da máquina de decodificação de *bin* do tipo *terminate* (biari_final_decoder) do módulo duplo de decodificação (dbl_decoder) por mais uma máquina de decodificação de *bin* do tipo regular (biari_decoder) e aumentar a possibilidade de decodificação de dois *bins* regulares em um ciclo;

4. Incluir mais uma máquina de decodificação de *bins* do tipo *bypass* concatenada ao módulo de decodificação dupla de *bins* do tipo *bypass*, visando a decodificação do *bin* de sinal dos elementos do tipo UEGk;

5. Alterar a estrutura de contextos para reduzir o tamanho do conjunto para a quantidade especificada na norma. Essa alteração irá reduzir o tempo necessário para iniciação dos contextos;

6. Realizar testes de decodificação para seqüências de vídeo entrelaçado, vídeos codificados para utilizarem *slices* SI e SP e demais elementos

sintáticos não exercitados. Buscando validar a estrutura atualmente implementada;

7. Realizar a integração do decodificador CABAC com a entidade que fará o controle da seqüência de decodificação dos elementos sintáticos, possibilitando uma melhor avaliação do desempenho real do decodificador;

8. Integrar o decodificador CABAC a um decodificador H.264 / AVC.

Bibliografia

ACHARIA, T.; TSAI, P.-S. *JPEG2000 standard for image compression: concepts, algorithms and VLSI architectures*. [S.l.]: John Wiley and Sons, 2005. 185 p.

BARNI, M. *Document and image compression, Volumes 978-3553*. [S.l.]: CRC Press, 2006. 120 p.

CHEN, J.-W.; CHANG, C.-R.; LIN, Y.-L. A hardware accelerator for context-based adaptive binary arithmetic decoding in h.264/avc. *IEEE International Symposium on Circuits and Systems, 2005. ISCAS 2005*, v. 5, p. 4525–4528, Maio 2005. Disponível em: <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1465638>. Acesso em: 02 jul. 2008.

CHEN, J.-W.; LIN, Y.-L. A high-performance hardwired cabac decoder. *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2007*, v. 2, p. II–37, II–40, Abril 2007. Disponível em: <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4217339>. Acesso em: 02 jul. 2008.

DANIAL, A. *CLOC - Count Lines Of Code. Ver. 1.08*. 2009. Software open source para contagem de linhas de código. Disponível em: <http://cloc.sourceforge.net/>. Acesso em: 08 dez. 2009.

FLORDAL, O.; WU, D.; LIU, D. Accelerating cabac encoding for multi-standard media with configurability. Artigo sobre Accelerating CABAC Encoding for Multi-standard Media with Configurability. 2006.

H.264/AVC SOFTWARE COORDINATION. *H.264/AVC JM Reference Software*. 2009. Disponível em: <http://iphome.hhi.de/suehring/tml/>. Acesso em: 08 dez. 2009.

IAIN, R. E. G. *H.264 and MPEG-4 Video Compression, Video coding for Next-generation Multimedia*. [S.l.]: John Wiley & Sons Ltd., 2003. 159–223 p.

IEEE COMPUTER SOCIETY. *IEEE Std 1666 - Standard SystemC Language*. [S.l.], 2005. Manual de Referência. Disponível em: <http://standards.ieee.org/getieee/1666/download/1666-2005.pdf>. Acesso em: 08 dez. 2009.

ITU. *ITU-T Recommendation H.264*: Advanced video coding for generic audiovisual services. [S.l.], 2005.

JR., G. G. L. An introduction to arithmetic coding. *IBM J. RES. DEVELOP*, v. 28, n. 2, p. 135–149, Março 1984.

JR., G. G. L.; RISSANEN, J. Compression of black-white images with arithmetic coding. *IEEE Trans. Commun. COM-29*, p. 858–867, Junho 1981.

MACKAY, D. J. *Information Theory, Inference, and Learning Algorithms*. [S.l.]: Cambridge University Press, 2004. 110–131 p.

MARPE, D.; KIRCHHOFFER, H.; MARTEN, G. Fast renormalization for h.264 / mpeg-4-avc arithmetic coding. *14th European Signal Processing Conference - EUSIPCO 2006*, Setembro 2006.

MARPE, D.; SCHWARZ, H.; WIEGAND, T. Context-based adaptive arithmetic coding in the h.264/avc video compression standard. *IEEE Transactions on Circuits and Systems for Video Technology*, v. 13, p. 620–636, Julho 2003.

MOFFAT, A.; BELL, T. C.; WITTEN, I. H. Lossless compression for text and images. *International Journal of High Speed Electronics and Systems*, v. 8, p. 179–231, 1995. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.4360>. Acesso em: 08 dez. 2009.

OSORIO, R. R.; BRUGUERA, J. D. Arithmetic coding architecture for h.264/avc cabac compression system. Proceedings of the EUROMICRO Systems on Digital System Design (DSD-04). 2004.

SIWERT, S. Ibm soc drawers articles - the resource view. resource allocation can determine system architecture. Outubro 2005. Disponível em: <http://www.ibm.com/developerworks/power/library/pa-soc1>. Acesso em: 08 dez. 2009.

SULLIVAN, G. J.; TOPIWALA, P.; LUTHRA, A. The h.264/avc advanced video coding standard: Overview and introduction to the fidelity range extensions. Presented at the SPIE Conference on Applications of Digital Image Processing XXVII. Special Session on Advances in the New Emerging Standard: H.264/AVC. Agosto 2004.

WIEGAND, T. et al. Overview of the h.264/avc video coding standard. *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, v. 13, n. 7, p. 560–576, Julho 2003.

XU, M.-H. et al. Optimizing design and fpga implementation for cabac decoder. *International Symposium on High Density packaging and Microsystem Integration*, p. 1–5, Junho 2007. Disponível

em: <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4283645>. Acesso em: 02 jul. 2008.

YI, Y.; PARK, I.-C. High-speed h.264/avc cabac decoding. *IEEE Transactions on Circuits and Systems for Video Technology*, v. 17, p. 490–494, Abril 2007. Disponível em: <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4162539>. Acesso em: 02 jul. 2008.

YU, W.; HE, Y. A high performance cabac decoding architecture. *IEEE Transactions on Consumer Electronics*, v. 51, p. 1352–1359, Novembro 2005. Disponível em: <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1561867>. Acesso em: 02 jul. 2008.

ZHANG, P. et al. High-performance cabac engine for h.264/avc high definition real-time decoding. *International Conference on Consumer Electronics, ICCE 2007*, p. 1–2, Janeiro 2007. Disponível em: <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4146003>. Acesso em: 02 jul. 2008.

ZHENG, Y. et al. A time and storage optimized hardware design for context-based adaptive binary arithmetic decoding in h.264/avc. *IEEE International Conference on Multimedia and Expo*, p. 1567–1570, Julho 2007. Disponível em: <http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4284963>. Acesso em: 02 jul. 2008.