UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
INSTITUTO DE FÍSICA
INSTITUTO DE INFORMÁTICA
INSTITUTO DE QUÍMICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MICROELETRÔNICA

CRISTIANO CARAFINI THIELE

Desenvolvimento da Arquitetura Integrada dos Codificadores de Entropia Adaptativos ao Contexto CAVLC e CABAC do padrão H.264/AVC

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Microeletrônica

Prof. Dr. Sergio Bampi
Orientador

Porto Alegre, julho de 2012.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Thiele, Cristiano Carafini
Desenvolvimento da Arquitetura dos Codificadores de Entropia Adaptativos CAVLC e CABAC do padrão H.264/AVC - 2012
70f.:il.
Orientador: Sergio Bampi
Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Microeletrônica. Porto Alegre, BR – RS, 2012.
1. Compressão de Vídeo 2. H.264/AVC 3. Entropia 4. Arquitetura de Hardware. 5. FPGA I. Bampi, Sergio. II. Mestrado em Microeletrônica.

AGRADECIMENTOS

Aos colegas de laboratório por todo o apoio, compartilhamento de conhecimento e momentos de descontração proporcionados. Aos grandes Cláudio M. Diniz, Bruno B. Vizzotto, André M. Martins, Bruno Zatt, Leandro Max, Fabio Walter, Vagner Rosa, Felipe Sampaio, Daniel Palomino, Leonardo B. Soares, Kleber H. Stangherlin, Cauane Silva, Franco Valdez, Vinícius Baldo, Felipe Dalcin e Eduarda Monteiro.

Aos professores do INF e da Física pelo conhecimento transmitido e a experiência dividida. Ao professor Sergio Bampi pelo apoio, confiança, conselho e orientação nesta jornada de conhecimento e aprendizado.

Aos meus pais por todo apoio e compreensão.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AC	*Alternate Current*
BRAM	*Block Random Access Memory*
BITSTREAM	Fluxo de bits
CABAC	*Context-Based Adaptive Binary Arithmetic Coding*
CAVLC	*Context-Based Adaptive Variable Length Coding*
CIF	*Common Intermediate Format*
DC	*Direct Current*
DCT	*Discrete Cosine Transform*
DRAM	*Dynamic Random Access Memory*
ES	Elemento Sintático
EXPG	*Exponential Golomb*
FIFO	*First In First Out*
FPGA	*Field Programmable Gate Array*
FPS	*Frames per second*
HDTV	*High Definition Digital Television*
IEC	*International Electrotechnical Commission*
IEEE	*Institute of Electric and Electronics Engineers*
INTER	*Inter Prediction*
INTRA	*Intra Prediction*
ISO	*International Organization for Standardization*
ITU-T	*International Telecommunication Union - Telecommunication*
JVT	*Joint Video Team*
MB	Macrobloco
MC	*Motion Compensation*
ME	*Motion Estimation*
MPEG	*Moving Picture Experts Group*
MSB	*Most Significant Bit*

MPS	*Most Probable Symbol*
NAL	*Network Adaptation Layer*
PC	*Personal Computer*
PSNR	*Peak Signal-to-Noise Ratio*
Q	*Quantization*
Q^{-1}	*Inverse Quantization*
QCIF	*Quarter Common Intermediate Format*
QP	*Quantization Parameter*
RAM	*Random Access Memory*
RGB	*Red, Green, Blue*
SBTVD	Sistema Brasileiro de Televisão Digital
SP	*Switching P*
SI	*Switching I*
T	*Transform*
T^{-1}	*Inverse Transform*
UFRGS	Universidade Federal do Rio Grande do Sul
VCEG	*Video Coding Experts Group*
VCL	*Video Coding Layer*
VGA	*Video Graphics Array*
VHDL	*VHSIC Hardware Description Language*
VLC	*Variable Length Coding*
YCbCr	*Luminance, Chrominance Blue, Chrominance Red*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Um codificador de entropia é responsável pela representação simbólica de dados de forma a representá-los com um menor número de bits. O H.264/AVC possui três codificadores de entropia: o Exponencial Golomb, o CAVLC que é o codificador de menor complexidade porém com um *throughput* maior de dados e o CABAC, com maior complexidade e com uma maior capacidade de compressão. A complexidade do codificador de entropia e a dependência dos dados sequenciais no bitstream original são os principais desafios para atender os requisitos de desempenho para compressão em tempo real. Por isso o desenvolvimento destas arquiteturas em hardware dedicado se faz necessário.

Neste contexto, esta dissertação descreve os algoritmos que fazem parte da entropia do padrão H.264/AVC e as arquiteturas para estes codificadores entrópicos (*Exponential Golomb*, CAVLC e CABAC), além de uma arquitetura de hardware dedicada que integra todos estes a um montador final que atende às especificações da norma H.264/AVC. As arquiteturas foram escritas em VHDL e sintetizadas para dispositivos integrados FPGA. Em um dispositivo Virtex-5, este codificador de entropia completo suporta codificação de vídeos no nível 4.2 do padrão H.264/AVC (Full HD a 60 quadros por segundo). Esta arquitetura é a que apresenta o melhor desempenho de processamento dentre os melhores trabalhos relacionados, além de ser um codificador com todas as alternativas de codificação de entropia requeridas pela norma implementadas em um mesmo módulo.

Palavras-Chave: Compressão de Vídeo, H.264/AVC, Entropia, EXPG, CAVLC, CABAC, Vídeo de Alta Definição.

Integrated Architecture Development of CAVLC and CABAC Context-Adaptive Entropy Encoders for H.264/AVC

ABSTRACT

An entropy encoder is responsible for the symbolic representation of a data stream so that the final representation contains less bits than the original. The H.264/AVC has three entropy coding schemes: the Exponential Golomb, the CAVLC encoder, that is less complex but with a higher data throughput, and the CABAC that is more complex while allowing for higher compression capability. The complexity of the entropy encoding and data dependencies on the original bitstream are the main challenges to meet the performance requirements for real-time compression. The development of these architectures in dedicated hardware is therefore necessary for high performance encoders.

In this context, this work describes the algorithms that are part of the entropy encoders of the H.264/AVC standard, and the corresponding entropy coding architectures (Exponential Golomb, CAVLC and CABAC), plus a dedicated hardware architecture that integrates all of these encoders to a final bitstream assembler that is compliant to the aforementioned standard. The architectures were written in VHDL and synthesized into FPGA devices. In a Virtex-5 device, this full entropy encoder supports video encoding at level 4.2 of the H.264/AVC standard (Full HD at 60 frames per second). The developed architecture performs best among the most recent related architectures published, and has the unique feature of an encoder that implements in the same module all the alternative entropy encoders present in this standard for video compression.

Keywords: Video Coding, H.264/AVC, Entropy, EXPG, CAVLC, CABAC, High Definition Video.

1 INTRODUÇÃO

Com a disseminação de vídeos digitais em alta definição nos mais diferentes meios, tais quais, computadores, sistemas de televisão, dispositivos portáteis entre outros, torna-se imprescindível o estudo e o desenvolvimento de técnicas de compressão de vídeo que sejam eficientes em termos de armazenamento e transmissão de dados. A massificação dos vídeos digitais tornou-se uma realidade a partir do desenvolvimento dos padrões de compressão de vídeo. Entre estes padrões de compressão e codificação de vídeos digitais existentes, o padrão H.264/AVC é o que está em evidência atualmente. O H.264/AVC foi definido pelo grupo JVT (*Joint Video Team*), uma união entre VCEG (*Video Coding Expert Group*) da ITU-T (*International Telecommunication Union – Telecommunication Standard Sector*) e MPEG (*Moving Picture Expert Group*) da ISO (*International Organization for Standardization*) e IEC (*International Electrotechnical Commission*). A primeira versão do padrão H.264/AVC tinha o objetivo de se tornar um padrão mais eficiente que os já existentes H.263 da VCEG e o MPEG-4 do MPEG. Estudos apresentaram que o H.264/AVC alcançou reduções de bits de 39% em relação ao MPEG-4, e de 49% em relação ao H.263 (WIEGAND, 2003). Estes resultados foram alcançados ao custo de uma alta complexidade computacional. A utilização do H.264/AVC para comprimir vídeo abriu a possibilidade de escolher entre alternativas como: utilizar uma resolução mais elevada, mantendo inalterada a largura de banda necessária para a transmissão/armazenamento; ou reduzir a largura de banda necessária para a transmissão e armazenamento, com a mesma resolução.

O H.264/AVC conta com três codificadores de entropia. O *Exponential Golomb* (EXPG) que é um codificador de códigos de tamanho variável utilizado exclusivamente na codificação dedados da configuração do vídeo. A codificação de comprimento variável adaptativo ao contexto - CAVLC (*Context Adaptive Variable Length Coding*) trata exclusivamente dos dados de imagem. A codificação binária aritmética adaptativa ao contexto - CABAC (*Context Adaptive Binary Arithmetic Coding*) trata de processar alguns dados de configuração e também de dados da imagem, podendo assim substituir o CAVLC.

A complexidade de um codificador de vídeo é decorrente do conjunto de ferramentas (algoritmos) utilizado para: analisar a sequência de vídeo, identificar a relevância e redundância dos dados analisados e, representar estes dados de uma maneira mais reduzida. Esta complexidade pode exigir do hardware uma taxa extremamente elevada de processamento, principalmente se estiver tratando de aplicações de vídeo em tempo real, com altas taxas de resolução. Soluções em software executadas sobre um processador de propósito geral podem ser insuficientes no tratamento destas sequências de vídeo, e multiprocessadores tem um grande consumo de energia tornando inviável o seu uso em dispositivos portáteis, por isto, o

desenvolvimento de arquiteturas dedicadas de hardware para a codificação de vídeo se faz extremamente necessária.

Considerando a complexidade, potência, desempenho e a eficiência como os maiores motivos no desenvolvimento de arquiteturas dedicadas, a aplicação de técnicas para otimizar o sistema como paralelismo e *pipeline* acabam sendo empregadas no codificador de vídeo. Existe, no entanto, uma parte do codificador onde a dependência de dados limita estas técnicas, causando um gargalo. Este desafio ocorre no Bloco de Entropia onde os dados estão correlacionados, chegando em alguns casos ter relação bit a bit, e onde no fim devem gerar uma única sequência de bits. Este problema se torna mais desafiador se utilizarem vídeos em mais alta resolução, que demandam maior processamento e ainda mais para aplicações em tempo real.

A principal contribuição deste trabalho foi o desenvolvimento de arquiteturas de codificadores de entropia EXPG e CAVLC , de um montador final e a integração destas com os módulos que tratam do CABAC. A integração das mesmas requereu o desenvolvimento de arquitetura de controle nova, de forma a compor assim um bloco de hardware para a entropia completa para o codificador H.264/AVC. Todo este bloco é capaz de prover uma codificação de vídeos até o nível 4.2 do padrão H.264/AVC (Full HD a 60 quadros por segundo). Este trabalho está inserido dentro da rede H.264 SBDTV que visa o desenvolvimento de um codificador completo dentro dos requisitos da televisão digital brasileira. Este codificador de entropia está desenvolvido sobre uma plataforma FPGA (*Field Programmable Gate Array*) da mesma forma que outros blocos do codificador completo.

Neste trabalho é apresentada uma visão geral do padrão H.264/AVC, com uma visão mais detalhada sobre o Bloco de Entropia, apresentando soluções arquiteturais para o EXPG, CAVLC, Montador Final e a integração completa com uma arquitetura do CABAC desenvolvida por (MARTINS, 2011) e (ROSA, 2010). O trabalho está organizado da seguinte forma: no capítulo 2 será realizada uma revisão sucinta sobre os conceitos básicos necessários para a compressão de vídeo digital; no capítulo 3 é apresentado o padrão H.264/AVC, com uma abordagem sobre a implementação de cada um dos módulos; no capítulo 4 estão descritos os algoritmos entrópicos do codificador H.264/AVC; o capítulo 5 aborda as arquiteturas desenvolvidas no Bloco de Entropia; o capítulo 6 aborda os resultados de área e desempenho das arquiteturas implementadas e a revisão bibliográfica de trabalhos sobre codificadores desenvolvidos para FPGA, com críticas sobre as soluções encontradas e comparação de resultados; e por fim, as Conclusões desta dissertação.

2 CONCEITOS DE VÍDEO DIGITAL E COMPRESSÃO DE DADOS

Neste capítulo serão apresentados os principais conceitos relacionados ao processo de geração e compressão de um vídeo digital. Estes conhecimentos se tornam relevantes para um melhor entendimento de como as ferramentas relacionadas à compressão de dados atuam na codificação de um vídeo digital.

2.1 Imagem e Vídeo Digital

A imagem digital é representada por elementos de figura, mais conhecidos por pixels, agrupados em matrizes bidimensionais. A resolução de uma imagem é a quantidade de pixels existentes na matriz utilizada para representar esta imagem. Existem diversas resoluções de vídeos e a quantidade de pixels necessária ou a melhor resolução para representar uma imagem depende da aplicação que se deseja utilizar. Por exemplo, uma imagem em *Full High Definition* (Full-HD) tem uma resolução de 1920 pixels na horizontal por 1080 pixels na vertical.

O pixel pode assumir qualquer cor. Para isso ele deve ser representado por uma combinação de diferentes componentes que dependem do espaço de cores utilizado. Um espaço de cores é um modelo matemático que descreve uma fórmula para cada componente do pixel para determinar a sua cor. No espaço de cores RGB, um pixel é dividido em três componentes distintas o vermelho(R), verde(G) e azul(B). Cada componente geralmente varia de 0 até 255, considerando 8 bits de representação por amostra. Quando todas estas são igual a zero tem-se o preto e quando todas são 255 tem-se o branco. O RGB é utilizado nas telas de televisores e monitores coloridos. No entanto o espaço de cores preferencial para a codificação de vídeo digital é o YCbCr, onde as três componentes utilizadas são a luminância (Y), que define a intensidade luminosa ou o brilho, crominância azul (Cb) e crominância vermelha (Cr). Com este espaço de cores pode-se explorar uma compressão das informações de crominâncias conhecida como subamostragem.

O vídeo digital é composto por um conjunto de imagens que são chamadas de quadros. Estas imagens são reproduzidas em uma sequência, a uma determinada frequência, normalmente 25, 30 ou 60 quadros por segundo, causando a percepção de movimento. Quanto maior o número destas imagens amostradas no tempo, mais suaves serão os movimentos em uma cena. Dependendo da resolução do vídeo, a quantidade de dados manipulados por unidade de tempo pode ser muito alta. Por exemplo, para um

vídeo de resolução Full HD (1920 x 1080pixels) a uma taxa de 30 quadros por segundo, a quantidade de dados manipulada seria entorno de 62,2 megapixels por segundo. Se cada pixel é formado de 3 componentes que são representadas em 8 bits a quantidade de informação manipulada passa para 1,5 gigabits por segundo de vídeo.

O vídeo digital sem compressão, como apresentado no exemplo anterior, acaba utilizando uma grande quantidade de bits de memória para ser armazenado e de elevadas taxas de transmissão para que possa ser transmitido. Estes vídeos, porém, possuem uma característica intrínseca de extrema importância que possibilita a compressão de dados: a Redundância. Desta maneira dentro de um vídeo digital existe uma grande quantidade de dados que se repetem e que manipulados poderiam ser descartados, possibilitando a representação de um vídeo com uma quantidade muito menor de bits do que a original.

2.2 Redundâncias

Nos vídeos digitais crus, existe muita informação repetida. Estas informações repetidas podem ser áreas com mesmas características dentro de um mesmo quadro, ou regiões similares de um quadro para outro ou de uma vista para outra no caso de multivistas. Aproveitando-se disto os codificadores de vídeo exploram o que se chama de redundâncias. Existem cinco tipos diferentes de redundâncias de dados que podem ser exploradas na compressão de vídeo: a entropia, redundância psicovisual, redundância espacial, redundância temporal e a redundância de disparidade.

2.2.1 Redundância Psicovisual

O olho humano é menos sensível às informações de cores contidas nas imagens (GONZALEZ, 2003). Então os padrões de compressão de imagens e vídeos podem explorar esta característica psicovisual humana para aumentar a eficiência de codificação através da redução da taxa de amostragem dos componentes de crominância em relação aos componentes de luminância, em uma operação chamada de subamostragem de cores. Por isso, a compressão de vídeos é aplicada no espaço de cores do tipo luminância e crominância, como o YCbCr (GHANBARI, 2010). A Figura 2.1 apresenta as subamostragens mais comuns encontradas em imagem e vídeo.

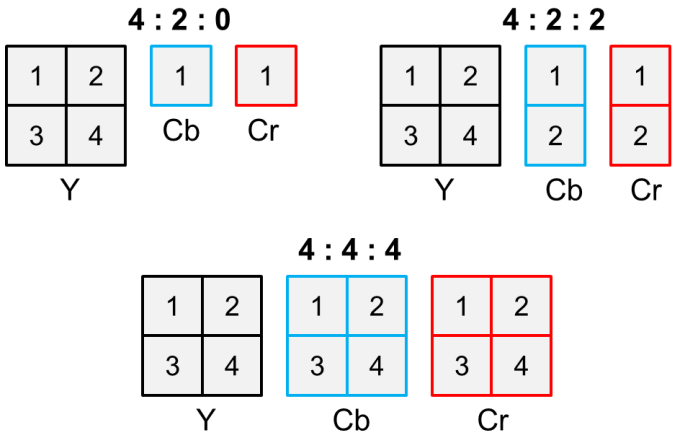

Figura 2.1: Formatos de Subamostragem

Na subamostragem de componentes os formatos mais comuns são: 4:4:4, onde não existe subamostragem de cores; 4:2:2 onde para cada 4 amostras de luminância existem 2 amostras para cada crominância; e 4:2:0 onde para cada 4 amostras de luminância existe 1 amostra para cada crominância. Um vídeo no formato 4:2:0, por exemplo, irá utilizar para representar 4 pixels, 6 amostras (4 de Luminância e 2 de Crominâncias) ao passo que um vídeo sem subamostragem utilizará 12 amostras (4 para cada uma das componentes), o que significa uma redução de 50% no tamanho de todo o vídeo.

A subamostragem de cores diminui significativamente a quantidade de dados que representam um vídeo, mas gerando perdas, uma vez que parte da informação de cor da imagem é simplesmente descartada.

2.2.2 Redundância Espacial

A Redundância Espacial ou Intra-quado é a correlação entre os *pixels* vizinhos de um mesmo quadro. Os *pixels* vizinhos ao longo de uma linha ou coluna variam gradativamente de intensidade, possuindo valores muitos similares entre si, com exceção das regiões de borda. Desta maneira, os compressores exploram esta redundância que gera pouco acréscimo de informação visual para cada ponto em relação aos vizinhos através de funções de autocorrelação (GHANBARI, 2010).

2.2.3 Redundância Temporal

A Redundância Temporal é fruto da correlação existente entre os *pixels* de quadros temporalmente próximos em um vídeo. Vários blocos de *pixels* simplesmente permanecem iguais de um quadro para outro em uma sequência de vídeo, como por exemplo, um fundo ou um objeto que permaneceu parado de um quadro para outro. Outras correlações se dão por *pixels* que podem apresentam uma pequena variação de valores em função da iluminação, ou por bloco de *pixels* que simplesmente tenha se deslocado de um quadro para o outro, como por exemplo, em um movimento de um objeto em uma cena. A exploração da redundância temporal é capaz de gerar elevadas taxas de compressão por parte dos codificadores de vídeo. Esta redundância é conhecida como a Interquadro (GHANBARI, 2010).

2.2.4 Redundância de Disparidade

A Redundância de Disparidade é explorada na codificação de múltiplas vistas em vídeos de mais de uma vista e que se encontram próximas. Vistas próximas podem compartilhar de dados similares. Com isto é possível explorar quadros de regiões correlacionadas entre as diferentes vistas de um vídeo, a partir de um quadro atual ou dos seus quadros de referência (MERKLE, 2007).

2.2.5 Entropia

A entropia em teoria da informação e comunicações trata da representação simbólica de dados. Ela representa uma medida da quantidade média de dado transmitido por símbolo (SHI, 2007). De modo genérico, a entropia é aplicada em qualquer tipo de dado, e com o vídeo contendo normalmente um conjunto grande de dados, os codificadores buscam uma representação simbólica mais eficiente. Os codificadores de vídeo representam o maior número de dados com o menor número de símbolos, mas geralmente, não tratam diretamente de um pixel ou suas componentes, mas de dados já pré-codificados. Por exemplo: dados que acontecem com maior frequência são representados por símbolos de menor tamanho. Este tipo de exploração é uma compressão sem perdas, ou seja, a informação comprimida é integralmente recuperada.

3 O PADRÃO H.264/AVC

Este capítulo apresenta a sintaxe do padrão e a organização do H.264/AVC. É abordada também a definição de importantes elementos sintáticos que controlam ou fazem parte da codificação de entropia. E por fim a organização em blocos das principais técnicas/ferramentas de codificação utilizadas no padrão, de uma maneira sucinta para uma melhor contextualização de todo o sistema antes de falar especificamente da codificação de entropia.

O padrão H.264/AVC teve seu lançamento em 2003 e contínuas atualizações até 2010. Na norma foram explícitos todos os formatos de vídeo que podem vir a ser suportados, a sintaxe e a semântica dos dados produzidos para o processo de decodificação do vídeo no H.264/AVC. Nesta norma nenhuma arquitetura do processo de codificação é descrito ou pré-determinado. As únicas informações descritas na norma que relacionam o codificador no nível de arquitetura são os tamanhos dos dados e a ordem destes elementos no fluxo de bits de saída que devem ser os mesmos de entrada para o decodificador.

3.1 Sintaxe do H.264/AVC

A sintaxe do H.264/AVC especifica a estruturação de dados de configuração e de imagem codificada para representar o vídeo digital. Esta sintaxe é hierárquica, tratando desde o nível mais alto, que é de dados de configuração referentes à sequência do vídeo, passando por dados de configuração do quadro, chegando finalmente a estrutura de macroblocos e blocos de pixels. A organização hierárquica possui como camada de topo a *Network Abstraction Layer* (NAL). As unidades NAL podem ser de:

- Configurações de parâmetros de uma sequência de vídeo (SPS);

- Configurações de parâmetros de um quadro do vídeo (PPS);

- Configurações e Dados codificados do vídeo (VCL).

Em uma unidade NAL SPS são definidos: o Perfil, o Nível, a resolução do vídeo e o número máximo de quadros de referência utilizados. Em uma NAL PPS são definidos: tipo de codificação de entropia utilizada, quantidade de quadros de referência ativos, parâmetros iniciais de quantização. Cada unidade PPS mantém um vínculo com uma unidade SPS. Em NAL VCL, também conhecida por *slice*, encontram-se algumas configurações restritas ao *slice*, e os dados codificados de um vídeo agrupados em macroblocos (MB). A Figura 3.1 apresenta a hierarquia de dados da NAL, focando na VCL com suas subdivisões até os blocos de pixels do vídeo.

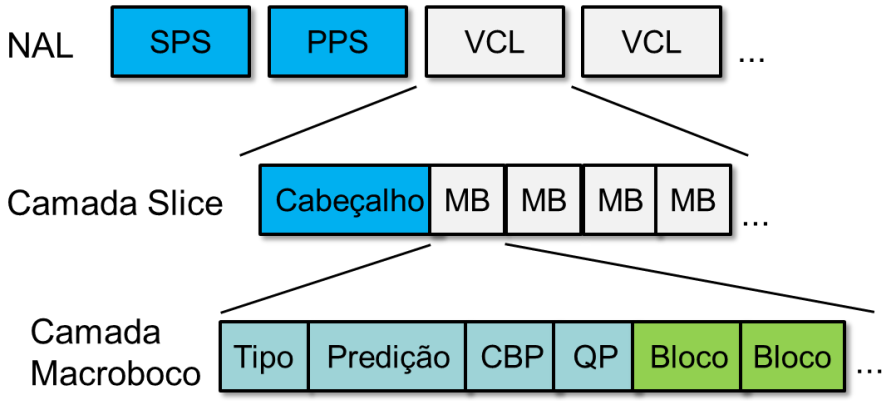

Figura 3.1: Estrutura sintática do padrão H.264/AVC

Os elementos sintáticos codificados de SPS, PPS e VCL na sua grande maioria tem tamanho variável de bits e dependência do elemento previamente codificado, o que dificulta a aceleração do processo de montagem. A seguir alguns dos dados/elementos serão descritos para um melhor entendimento de alguns aspectos deste padrão de vídeo.

3.1.1 Quadro de Referência, Slice e Macrobloco

No codificador, cada quadro de um vídeo codificado pode ser reutilizado para a exploração temporal. Os quadros são identificados numericamente dentro de uma lista. Estes quadros codificados anteriormente são chamados de quadros de referência para a predição do próximo quadro. No H.264/AVC os quadros de referência podem ser localizados temporalmente anteriormente ou posteriormente ao quadro atual e são separados em diferentes listas.

Na exploração das redundâncias espaço-temporais, os preditores realizam suas comparações com os quadros de referência ou com outras regiões similares dentro do mesmo quadro. Estes quadros são divididos em fatias (*slices*). Um quadro pode ser representado por um ou mais *slices*, mas as operações de compressão não são realizadas no nível de *slice*. Estes *slices* são divididos ainda em conjuntos menores de pixels chamados de Macroblocos (MB). Todo MB é composto por 16 blocos de luminância, e em função da subamostragem 4:2:0, também possui 4 blocos de crominância azul e 4 blocos de crominância vermelha. O *slice* delimita que tipo de informação de redundância pode ser explorado pela predição dos MBs que o compõem. Há 3 tipos principais de slices: I, P e B.

Nos *slices* I só podem ocorrer macroblocos do tipo I, que são codificados usando a codificação intraquadro, o que significa que haverá predição somente com as amostras do s*lice* atual. A codificação pode acontecer sobre macroblocos completos ou para cada bloco. Nos *slices* P podem ocorrer macroblocos do tipo I e do tipo P. Macroblocos do tipo P são codificados usando a codificação inter-quadro, ou seja, com referência a quadros anteriormente codificados. Um macrobloco codificado no modo inter-quadros pode ser ainda dividido em partições de macroblocos menores como: 16x8, 8x16 , 8x8, 8x4, 4x8 e 4x4. Nos *slices* do tipo B podem aparecer macroblocos do tipo I e do tipo B. Macroblocos do tipo B são codificados usando uma codificação inter-quadros que permite a predição de um ou dois quadros de referência, que se encontram antes ou depois temporalmente do quadro atual dentro da sequência de vídeo (RICHARDSON, 2010).

3.1.2 Perfil e Nível

O padrão H.264/AVC por tratar-se de um codificador genérico define sete diferentes perfis: *Baseline*, *Main*, *Extended*, *High*, *High 10*, *High 4:2:2 e H.4:4:4*. Os perfis com suas principais características são apresentados resumidamente na Figura 3.2. Cada perfil criado tem por função atender uma demanda ou tipo de aplicação e determina as ferramentas e particularidades que serão usadas para a codificação de um vídeo. Para os perfis *Baseline*, *Main e Extended*, as amostras dos componentes de cor tem 8 bits de tamanho e a relação de subamostragem de cores é fixada em 4:2:0. Os codificadores desenvolvidos neste trabalho estão inseridos e atendem aos perfis *Baseline* e *Main* do codificador H.264/AVC.

O perfil *Baseline* é direcionado a aplicações como videotelefonia, videoconferência e vídeo sem fio. O perfil *Baseline* suporta codificação intra e inter (usando somente *slices* I e P) e uma codificação de entropia com o CAVLC.

O perfil *Main* é focado na transmissão de televisão e no armazenamento de vídeo. O perfil *Main* inclui o suporte para vídeo entrelaçado, o suporte à codificação inter quadros utilizando *slices* do tipo I, P e B, utilização da predição ponderada e o suporte à codificação de entropia CAVLC e CABAC.

O perfil *Extended* é focado em aplicações de *streaming* de vídeo e não suporta CABAC, mas agrega modos e ferramentas para habilitar uma troca eficiente entre *bitstreams* codificados (através de *slices* do tipo SP e SI).

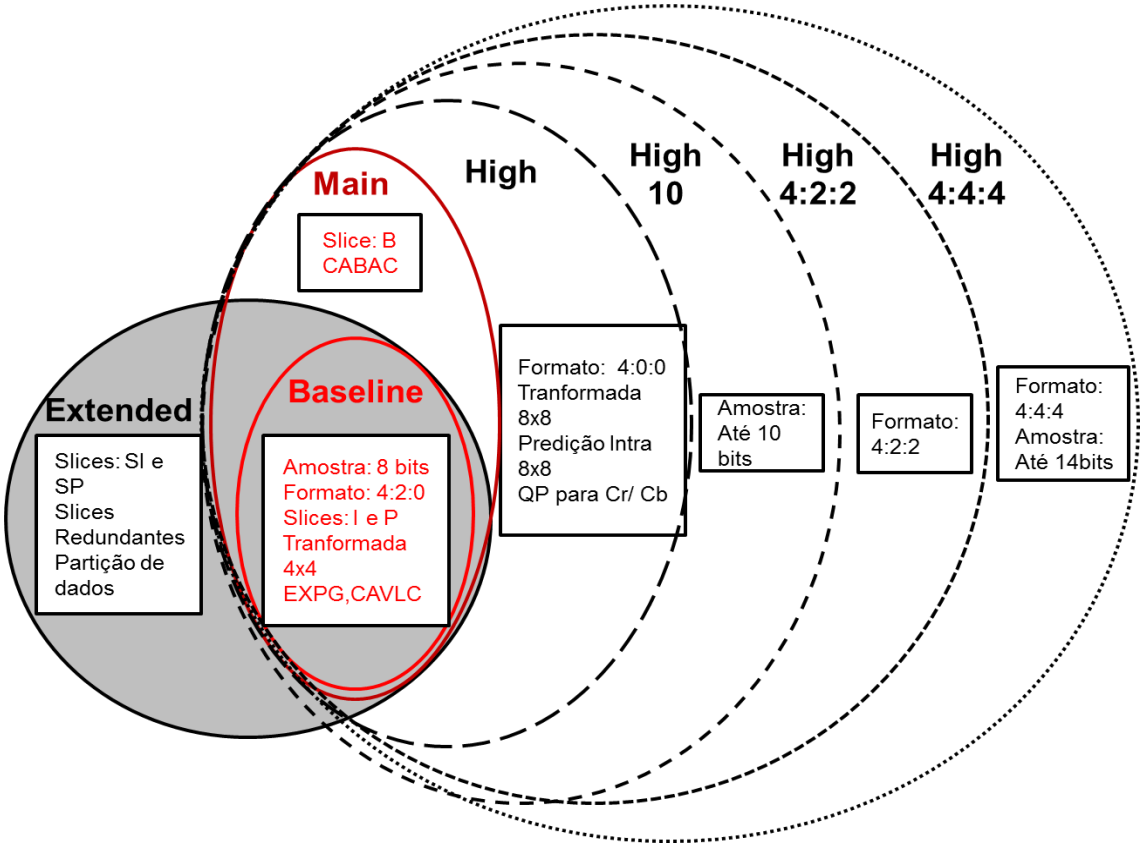

Figura 3.2: Os principais perfis definidos pelo padrão H.264/AVC.

Os perfis *High* são dedicados para aplicações que exigem maior qualidade na imagem. Estes perfis têm em comum o suporte a um tamanho de bloco adaptativo para a transformada (4x4 ou 8x8), matrizes de quantização baseadas em percepção e uma representação sem perdas de regiões específicas do conteúdo do vídeo (SULLIVAN, 2004). No mais os perfis *High* extrapolam os demais perfis em relação ao tamanho em bits das amostras que podem vir a ser de até 14 bits e/ou alterando a relação de subamostragem de cores dos 4:2:0 para 4:2:2 ou 4:4:4 (ITU-T, 2010) aumentando a qualidade de cores e brilho no vídeo .

Tabela 3.1 – Relação entre Níveis, Resoluções e Taxas de amostragem

Nível	Resolução	Quadros por segundo
1.1	QCIF (176x144)	30
1.3, 2	CIF (352x288)	30
3	525 SD (720x480)	30
3.1	720p HD (1280x720)	30
4, 4.1	1080p HD (1920x1080)	30
4.2	1080p HD (1920x1080)	60
5.1	4Kx2K (4096x2048)	30

Os Níveis têm por função delimitar as taxas mínimas e máximas para processamento do fluxo codificação/decodificação. Ele associa assim para cada perfil, a taxa de operação necessária para atender a determinada resolução de vídeo e taxas de amostragem de quadros. A Tabela 3.1 relaciona os níveis em algumas resoluções e taxas. Os níveis variam de 1 até 5.1. Como exemplo, na decodificação de um vídeo 1080p HD a uma taxa de trinta quadros por segundo, é preciso que a aplicação atenda as especificações a partir do Nível 4 que é o foco deste trabalho.

3.2 A estrutura do codificador H.264/AVC

O codificador H.264/AVC realiza todas as suas operações tendo como base o macrobloco. Deste modo o processamento de um vídeo digital começa quando um MB passa pelo estágio de predição. Neste estágio buscam-se as redundâncias temporais e espaciais em relação aos macroblocos de referência. Como saída desta etapa tem-se os resíduos que nada mais são que as diferenças em relação ao quadro de referência. O próximo estágio aplica-se às transformadas e realiza-se a quantização destes resíduos. Na saída obtém-se um macrobloco cujos blocos apresentam características particulares.

A partir deste ponto, o macrobloco percorre dois caminhos diferentes. No primeiro caminho o macrobloco entra num *loop* onde é quantizado inversamente, transformado

inversamente e passado por um filtro redutor de blocagem. Isto é realizado para que o codificador tenha um quadro reconstruído que sirva de referência para as predições e que seja idêntico ao existente no decodificador. No segundo caminho o macrobloco é codificado bloco a bloco pelas ferramentas de entropia. Após explorar a redundância entrópica, as NALs são devidamente montadas com os cabeçalhos, informações de configuração e resíduos com os dados codificados de imagem. A Figura 3.3 apresenta um diagrama em blocos do codificador H.264/AVC completo, brevemente descrito anteriormente.

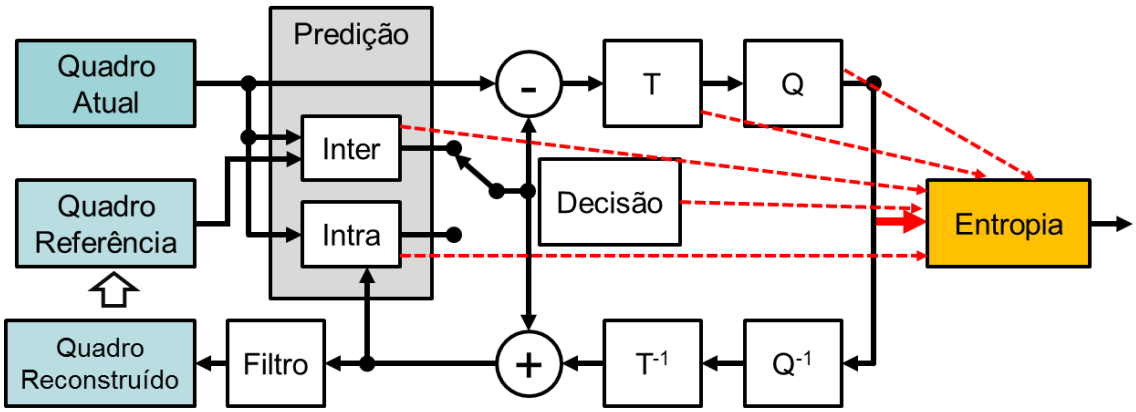

Figura 3.3: Diagrama em blocos do codificador H.264/AVC.

Os blocos do codificador são brevemente detalhados nas próximas seções, de acordo com o fluxo de dados no H.264/AVC. Como o foco deste trabalho trata do estudo do Bloco de entropia, este será descrito detalhadamente no capítulo 4.

3.2.1 Blocos de Predição e Decisão

É no bloco de predição do codificador H.264/AVC onde ocorrem os maiores ganhos de compressão de dados em relação aos outros codificadores. É o bloco que apresenta a maior complexidade computacional dentro do codificador (RICHARDSON, 2010). O H.264/AVC ao processar um MB pode optar pela predição Intra ou Inter.

A predição Intra busca similaridades com as amostras dos blocos vizinhos já codificados do quadro atual. Esta predição se destaca em vídeos com áreas homogêneas onde o bloco predito é idêntico a seus vizinhos. Esta predição trabalha com tamanhos de bloco 4x4, 8x8 e 16x16. Nas predições Intra 4x4 e 8x8 existem 9 modos: Vertical, Horizontal, DC, Diagonal Inferior Esquerda, Diagonal Inferior Direita, Vertical Direita, Horizontal Inferior, Vertical Esquerda e Horizontal Superior. Para blocos 16x16 existem 4 modos: Vertical, Horizontal, DC e Plana.

A predição Inter é formada por dois grandes blocos a Estimação de Movimento (ME) e a Compensação de Movimento (MC). A ME e MC podem usar além de macroblocos, partições de macrobloco e partições de submacroblocos quando realizarem a predição. A ME busca nos quadros de referência o bloco de maior similaridade com o bloco a ser predito. Quando encontrado a ME calcula a diferença espacial deste bloco em relação ao predito, gerando um vetor de movimento e também o índice do quadro de referência do bloco utilizado. Estes dados são enviados para

entropia e para a MC. A MC com as informações da ME busca no quadro de referência aquele bloco, calcula a diferença do bloco a ser predito e o bloco da referência e gera o bloco de resíduos que vai ser processado. Este tipo de predição é utilizado em áreas de movimento no vídeo. Dependendo do perfil a ser utilizado, algumas opções de predição podem não estar disponibilizadas.

O bloco de Decisão tem por objetivo determinar qual o melhor modo de predição a ser utilizado, seja para ter a maior compressão de dados ou para manter o fluxo de dados mais constante possível num canal. Na literatura são apresentados diversos algoritmos de decisão, com diferentes níveis de complexidade (CORRÊA, 2010). A decisão do modo de predição, apesar de não normatizada, tem como base técnicas de otimização taxa-distorção (WIEGAND, 2003). A otimização taxa-distorção é um processo que relaciona a taxa de bits da saída de um codificador de vídeo e o grau de distorção do vídeo no decodificado. Uma má decisão na escolha da predição resulta numa compressão menos eficiente.

3.2.2 Transformadas Diretas (T) e Quantização Direta (Q)

O bloco das Transformadas tem por função converter o bloco de resíduos proveniente da predição em uma representação equivalente no domínio das frequências. No H.264/AVC para a transformação no domínio das frequências, são usadas aproximações inteiras da Transformada Discreta do Cosseno (DCT) e Hadamard. Estas transformadas são realizadas sobre blocos de dimensão 4x4 e 2x2. A DCT é utilizada para tratar os blocos 4x4, Luma AC e Chroma AC. Nos macroblocos em que é feita a predição INTRA 16x16, é aplicada uma Hadamard 4x4 sobre os coeficientes DC dos dezesseis blocos 4x4 de luminância gerando o bloco -1, enquanto que para os blocos de crominância, é aplicada uma transformada Hadamard 2x2 sobre os coeficientes DC das crominâncias para gerar os blocos 2x2 números 16 e 17, relativos ao croma azul e ao croma vermelho respectivamente. Na Figura 3.4 são mostrados todos os blocos de amostras.

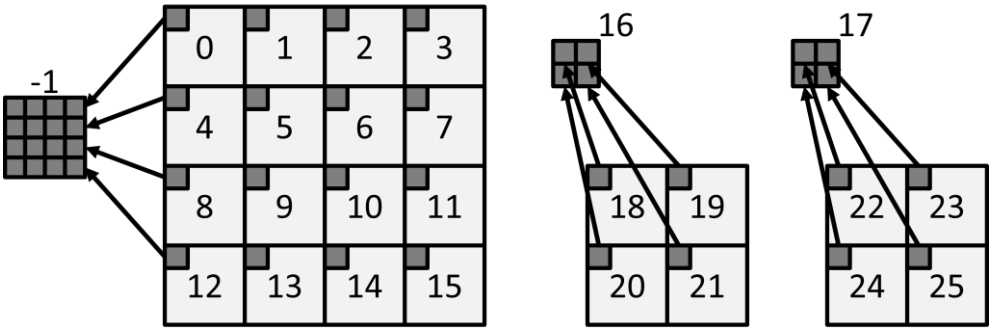

Figura 3.4: Blocos de coeficientes com tamanho 4x4 e 2x2 para aplicação das transformadas do padrão H.264/AVC.

No bloco da quantização as entradas são os coeficientes da DCT ou Hadamard. É aplicado sobre estes coeficientes um fator de quantização conhecido como QP. O QP no H.264/AVC varia de 0 a 51, onde valores mais altos resultam em menor número de bits para representar o vídeo e consequentemente na perda da qualidade da imagem. O QP também serve de ferramenta para controlar a relação taxa-distorção do vídeo no

H.264/AVC. A quantização é a ferramenta do H.264/AVC que introduz perdas na qualidade da imagem do vídeo. Após a quantização o bloco é ordenado em duplo Z, como mostrado na Figura 3.5. Este ordenamento normalmente posiciona as amostras de maior valor absoluto primeiro e de valores zero no final para facilitar o processo de entropia.

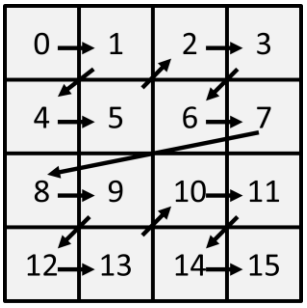

Figura 3.5: Ordem Duplo Z dos Coeficientes.

3.2.3 Quantização Inversa (Q^{-1}) e Transformada Inversa (T^{-1})

No bloco Q^{-1} ocorre o *rescaling*, onde se realiza a correção de escala para o cálculo das transformadas inversas, utilizando sempre o mesmo QP da quantização direta. Após esta quantização inversa, no bloco de T^{-1} é aplicada a DCT inversa para a decodificação dos coeficientes. Para a transformação inversa dos blocos DC são utilizadas as transformadas Hadamard 2x2 e Hadamard 4x4 diretamente sobre os coeficientes provenientes da etapa de quantização, antes mesmo da etapa de quantização inversa para otimizar processo (RICHARDSON, 2010).

3.2.4 Filtro Redutor de Blocagem

O filtro redutor de efeitos de bloco é usado para suavizar o efeito de blocos do quadro reconstruído que será usado como quadro de referência para a predição inter. A capacidade de suavização do filtro depende do parâmetro de quantização utilizado. No H.264/AVC o filtro redutor de blocagem é adaptativo, conseguindo "distinguir" entre um contorno ou fronteira real, que não deve ser filtrada, e uma distorção gerada por um elevado parâmetro de quantização. Ou seja, o filtro de deblocagem deve ser aplicado nos casos em que há uma descontinuidade entre os blocos que é grande o suficiente para ser visto e é pequeno o suficiente para não caracterizar um contorno ou fronteira real da imagem. A operação de filtragem afeta até três amostras de cada lado na fronteira entre os blocos. Ele proporciona um aumento significativo da qualidade do vídeo reconstruído.

3.2.5 Codificação de Entropia

No padrão H.264/AVC a codificação de entropia é a responsável pela exploração das redundâncias simbólicas encontradas nos blocos de resíduos após a transformação e quantização e pela codificação de elementos sintáticos de controle e configuração do decodificador. Estes elementos de controle e configuração são produzidos nos blocos de Predição, de Transformada e Quantização e sinalizam todas as opções que estes blocos realizaram para a codificação da imagem. No H.264/AVC existem três métodos para codificação de entropia, sendo eles: Exponencial Golomb (EXPG), Codificação de Comprimento Variável Adaptativo ao Contexto (CAVLC- *Context-Adaptive Variable*

Length Coding) e a Codificação Aritmética Binária Adaptativa ao Contexto(CABAC - *Context-Adaptive Binary Arithmetic Coding*).

O codificador Exponencial Golomb (EXPG) está presente em todos os perfis do H.264. O EXPG é dedicado exclusivamente para o tratamento das informações de controle e configuração para o decodificador de vídeo. Ele codifica unidades NAL de sequência de vídeo (SPS) e de quadros de imagem (PPS) além do cabeçalho do *slice* e alguns dados de predição e quantização do MB. O Codificador por Código de Comprimento Variável Adaptativo ao Contexto (CAVLC) também está presente em todos os perfis do padrão e é considerado o codificador de resíduos de baixa complexidade do H.264/AVC. O CAVLC trata exclusivamente dos dados provenientes da quantização, ou seja, os macroblocos que representam a imagem. O Codificador de Aritmética Binária Adaptativa ao Contexto (CABAC) está presente a partir do perfil *Main* do padrão é a ferramenta de entropia mais complexa do H.264/AVC. O CABAC processa os elementos sintáticos de controle e configuração na camada de macrobloco e os coeficientes residuais provenientes da quantização. A Figura 3.6 mostra a organização do Bloco de Entropia conforme padrão H.264/AVC.

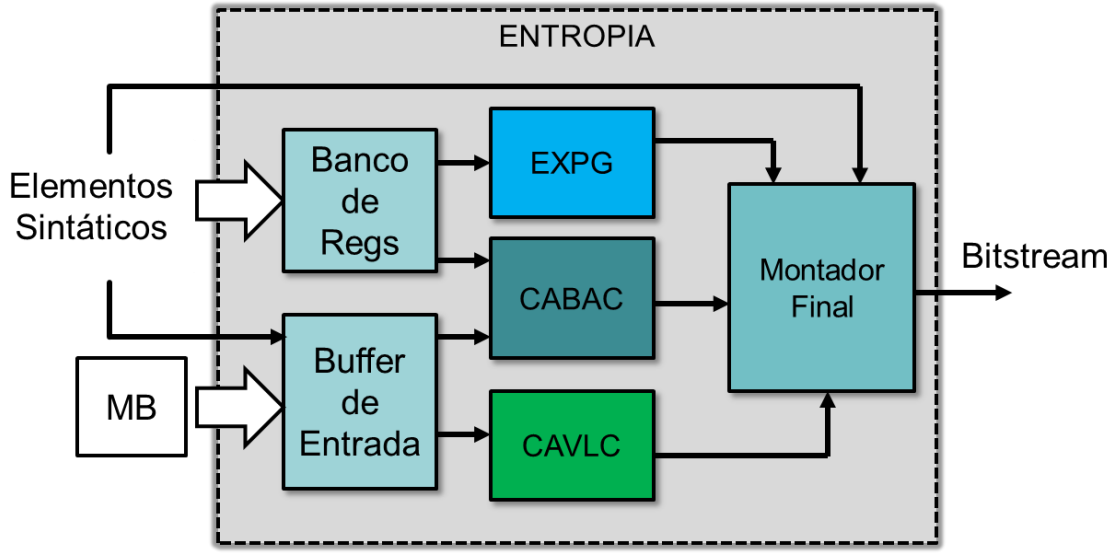

Figura 3.6: Diagrama em blocos da Entropia.

Os coeficientes residuais dos blocos podem ser codificados com o CAVLC ou CABAC. A seleção do codificador entrópico a ser empregado depende do perfil utilizado. Se os dois métodos estão disponíveis em determinado perfil, a decisão pelo uso de um ou outro dependerá do codificador. No padrão H.264/AVC a principal inovação introduzida na codificação de entropia está na técnica de codificação adaptativa baseada em contextos, que é aplicada tanto no CAVLC quanto no CABAC. Com esta técnica os diferentes elementos sintáticos são codificados baseados nos elementos sintáticos codificados em passos anteriores (RICHARDSON, 2010).

4 ALGORITMOS DE CODIFICAÇÃO DE ENTROPIA

Este capítulo apresenta os algoritmos e métodos utilizados para a codificação simbólica dos dados. Como mostrado anteriormente, o codificador H.264/AVC conta com três ferramentas para a codificação entrópica dos dados processados: EXPG, CAVLC e CABAC. As próximas seções apresentam os algoritmos e os métodos para estes compressores, e estão baseados na norma do H.264/AVC (ITU-T, 2009).

4.1 Exponencial Golomb

A Codificação Exponencial Golomb é uma codificação binária de tamanho variável baseada em um padrão regular. Considerando a probabilidade simbólica dos dados, os símbolos de ocorrências mais frequentes são representados por códigos binários menores ao passo que símbolos menos frequentes acabam sendo associados a códigos binários mais extensos. O dado codificado pelo Exponencial Golomb apresenta a seguinte estrutura:

$$\boldsymbol{Código} = [\boldsymbol{Prefixo\ de\ Zeros}][\mathbf{1}][\mathbf{INFO}] \tag{4.1}$$

No algoritmo do Exponencial Golomb o elemento que virá a ser codificado deve ser um número inteiro sem sinal, e é chamado de *CodeNum*. O *Prefixo de Zeros* indicam quantos bits de dado de *INFO* após o "1" são válidos. O *Prefixo de Zeros* é derivado de *CodeNum.* Abaixo é mostrado o cálculo da variável Z que indica o número de zeros no *Prefixo de Zeros*:

$$\boldsymbol{Z = log_2(Code_{Num} + 1)} \tag{4.2}$$

E em (4.3), *INFO* é calculado a partir de *CodeNum*e Z.

$$\boldsymbol{INFO = CodeNum + 1 - 2^Z} \tag{4.3}$$

Na codificação de vídeo os elementos sintáticos são de diferentes tipos. Os elementos cujos valores são inteiros e sem sinal são classificados como **ue** (*unsigned element*) e são diretamente relacionados ao *CodeNum*. Os elementos sintáticos que podem apresentar valores negativos são do tipo **se** (*signed element*). Para estes

elementos, é feito um rearranjo, associando cada valor a um determinado *CodeNum*, de acordo com equacionamento em (4.4).

$$\mathbf{CodeNum} = \begin{cases} \mathbf{2|x|} & , \ \mathbf{x < 0} \\ \mathbf{0} & , \ \mathbf{x = 0} \\ \mathbf{2x - 1} & , \ \mathbf{x > 0} \end{cases} \tag{4.4}$$

Há elementos sintáticos que podem ter seus valores fracionados na codificação e são tratados como **te** (*truncated element*). Se o valor do elemento é maior que 1, ele é codificado como um **ue,** se este elemento pode assumir um valor máximo de 1 então é codificado como o inverso de seu valor. O elemento *Coded Block Pattern* (CBP) na codificação H.264/AVC é processado de maneira peculiar e é classificado como **me** (*mapped element*). O valor de CBP é mapeado para tabelas de acordo com o tipo de predição do bloco, e então recebe o valor de *CodeNum* para ser comprimido. Estas tabelas são encontradas na norma (LEE, 2008 e ITU-T, 2009).

4.2 CAVLC

O algoritmo de codificação do CAVLC pode ser dividido em duas grandes etapas. Na primeira deve-se fazer a pré-codificação que consiste na leitura dos dados em uma determinada ordem para a extração de alguns parâmetros. Na segunda etapa, os parâmetros são processados para gerar os elementos sintáticos resultantes da codificação. Este processamento consiste de cálculos e consulta de tabelas com códigos de comprimento variável. Estas tabelas foram elaboradas estatisticamente, assim os parâmetros mais comuns e frequentes da codificação assumem códigos de comprimento menor. Os elementos sintáticos resultantes do CAVLC são denominados: *Coeff_token*, *Trailing_ones_sign_flag*, *Levels*, *Total_zeros* e *Run_before* (ITU-T, 2009). Uma terceira etapa que deve ser considerada, se a arquitetura do CAVLC for levada em conta, é o processo de montagem final dos elementos sintáticos para gerar um *bitstream*.

Para iniciar o processo de codificação no CAVLC é necessário fazer uma leitura em Ziguezague das amostras existentes dentro de um bloco com coeficientes já transformados e quantizados. O CAVLC deve estar preparado para codificar 3 tipos de tamanhos de bloco. Um bloco pode ter 16 amostras, quando este for um bloco de Luma 4x4 ou Luma DC 16x16, 15 amostras, quando for Luma AC 16x16 ou Croma AC, e 4 amostras, quando este for um Croma DC.

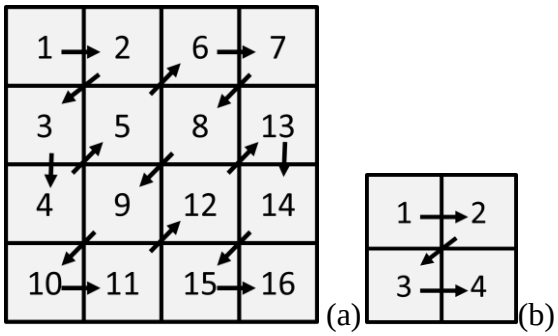

Figura 4.1: Ordem de Leitura dos dados em Zigue-zague

Na Figura 4.1 é apresentada a ordem Ziguezague de leitura de dados dentro do bloco. Em (a) é mostrado a ordem para blocos de 16 e 15 elementos. Quando for um bloco de quinze elementos simplesmente ignora-se a primeira amostra. Em (b) é apresentada a leitura para quatro elementos.

No fim do processo de leitura, como dito anteriormente, parâmetros são extraídos para a codificação do bloco. Estes parâmetros são:

- *TotalCoeff*: indica o número total de elementos não zeros dentro do bloco.
- *Trailing Ones* (T1s): considerando a ordem inversa de leitura Ziguezague, são contados até três elementos ±1 dentro do bloco. Caso algum elemento não zero e diferente de ±1 é lido, a contagem de *Trailing Ones* se encerra. Qualquer ±1 excedente ao terceiro ou lido posteriormente a um elemento não zero diferente de ±1 é considerado um *CoeffLevel*.
- *CoeffLevel:* considerando a ordem inversa de leitura ziguezague, armazena todos os elementos não zero e que não são *Trailing Ones.*
- *Total Zeros:* considerando a ordem inversa de leitura ziguezague, é o número total de zeros existentes antes do primeiro elemento não zero.
- *Run*: considerando a ordem inversa de leitura ziguezague, armazena o número de zeros entre cada par de coeficientes não zero, até o último coeficiente não zero.

Outro parâmetro também importante na codificação do CAVLC, mas que não é obtido na leitura de elementos é o *nC*. O *nC* é um parâmetro que relaciona o número de coeficientes não zeros da vizinhança (bloco superior e bloco à esquerda) para a codificação do bloco atual. Quando os dois blocos estão presentes o *nC* é a média destes, quando somente um deles está disponível *nC* é igual ao valor do vizinho presente e quando a vizinhança não existe *nC* é igual a zero.

Com os parâmetros anteriormente descritos já definidos, pode-se fazer a compressão propriamente dita e obterem-se assim os elementos sintáticos *Coeff_token*, *Trailing Ones Sign*, *Levels*, *Total Zeros* e *Run Before*. Para a codificação de todos os elementos sintáticos são utilizadas tabelas com códigos de tamanho variável. Os elementos sintáticos *Level* e *Run Before* são processados cada qual de maneira iterativa, sendo estes os que despendem de mais esforço para serem codificados.

- *Coeff_token*: É o elemento sintático relacionado com o total de amostras não zeros dentro do bloco. Os parâmetros necessários para a codificação deste elemento são *TotalCoeff* , *Trailing Ones* e *nC*. Baseado nestes parâmetros se realiza a consulta às tabelas existentes na norma do padrão. Pode-se observar que existe uma tabela específica para tratar os blocos de Luma, Croma AC, e Croma DC.

- *Trailing Ones Sign*: Este elemento sintático como o próprio nome indica, determina o sinal para cada um dos *Trailing Ones* existentes naquele bloco. Se um *Trailing One* é +1 ele é codificado como 0 e se o *Trailing One* é igual -1 é codificado como 1. Se não houver *Trailing One* no bloco este elemento não é codificado.

- *Level*: O elemento sintático *Level* está diretamente ligado aos elementos não zeros que não são *Trailing Ones*. O cálculo do *Level* é baseado nos parâmetros *TotalCoeff*, *Trailing Ones* e de todos os *CoeffLevels*. Se não houver *CoeffLevel* no bloco este elemento não é codificado. Todo *Level* é composto de um prefixo e um sufixo. O prefixo é composto por <zeros_prefixo 1>, como mostra a Tabela 4.1.

Tabela 4.1 – Codificação do Prefixo do *Level*

Zeros_Prefixo	Valor Codificado
0	1
1	01
2	001
3	0001
4	0000 1
5	0000 01
6	0000 001
...	...

O cálculo do sufixo e do zeros_prefixo é mostrado no pseudocódigo a seguir. Uma variável "sinal" é usada para o cálculo, quando *CoeffLevel* tem um valor positivo o sinal = 0 e quando negativo sinal = 1.

Passo1:

Se *TotalCoeff* > 10 e *Trailing Ones* < 3, Tamanho_sufixo = 1

Senão, Tamanho_sufixo= 0.

Vai para passo 2.

Passo2:

Se tamanho_sufixo = 0 e (TotalCoeff ≤ 3 ou Trailing Ones< 3),

|CoeffLevel| = |CoeffLevel| - 1

Se |CoeffLevel| < 8,

Tamanho_sufixo = 0.

zeros_prefixo = 2 x (|CoeffLevel|-1) + sinal.

Se |CoeffLevel| < 16,

Tamanho_sufixo = 4.

Sufixo = (|CoeffLevel| - 8) concatenado com sinal.

Zeros_prefixo = 14.

Se |CoeffLevel| > 15,

Vx = |CoeffLevel| - 16.

Tamanho_sufixo = 12 + (Vx >>11).

Sufixo = Vx concatenado com sinal.

Zeros_prefixo = 15 + 2 x (Vx >>11).

Vai para Passo 6.

Senão, vai para Passo 3.

Passo 3:

Se tamanho_sufixo = 1,

|CoeffLevel| = |CoeffLevel| - 1

Vai para Passo 4.

Passo 4:

Se |CoeffLevel| - 1 > 15 x 2tamanho_sufixo – 1,

Vx = (|CoeffLevel| - 1) - 15 x 2tamanho_sufixo – 1.

Tamanho_sufixo = 12 + (Vx >>11).

Sufixo = Vx concatenado com sinal.

Zeros_prefixo = 15 + 2 x (Vx >>11).

Vai para Passo 6.

Senão, vai para Passo 5.

Passo 5:

Se tamanho_sufixo > 1,

sufixo = a tamanho_sufixo bits menos significativos de (|CoeffLevel| - 1) concatenado com sinal.

zeros_prefix = ao valor dos bits mais significativos de (|CoeffLevel| - 1) remanecentes.

Vai para Passo 6.

Passo 6:

Neste passo é determinado tam_sufixo para o cálculo do próximo Level. Se CoeffLevel é o primeiro CoeffLevel a ser codificado e CoeffLevel > 3, o tamanho_sufixo = 2. Senão baseado no valor absoluto de CoeffLevel, determina-se prox_tam_sufixo, conforme mostrado na Tabela 4.2.

Se prox_tam_sufixo > tamanho_sufixo já existente,

O tamanho_sufixo é incrementado para o próximo Level.

Se prox_tam_sufixo < tamanho_sufixo já existente,

O tamanho_sufixo continua o mesmo para o próximo Level.

Se ainda houver CoeffLevel para ser codificado, volte para o Passo 4, senão FIM.

A tabela a seguir apresenta os valores do próximo tamanho de sufixo baseado no valor de CoeffLevel, para o cálculo do Passo 6 do pseudo-código para se obter o *Level*.

Tabela 4.2 – Tabela para sufixo do próximo Level

CoeffLevel	prox_tam_sufixo
0	0
1 - 3	1
4 - 6	2
7 - 12	3
13 - 24	4
25 - 48	5
> 48	6

- *Total Zeros*: Este elemento está relacionado com o número total de zeros existentes antes do primeiro elemento não zero. Utilizando os parâmetros *TotalCoeff* e Total Zeros é realizada uma consulta em tabela presente na norma do padrão. Da mesma forma que o *Coeff_token*, também existe uma tabela particular para tratar o bloco de Croma DC. Caso não existam Total Zeros este elemento não é codificado.

- *Run Before*: O Elemento *Run Before* está relacionado com a distribuição de zeros entre o primeiro e último elementos não zeros do bloco. Para o cálculo de cada Run Before são utilizados os parâmetros Total Zeros e *Run* e a variável *Zeros_Left*. A variável *Zeros_Left* é calculada a partir dos parâmetros anteriores e juntamente com o parâmetro *Run* serve de indexador para a tabela existente na norma e obter-se um *Run before*. No primeiro momento *Zeros_Left* assume o valor de Total Zeros, e a cada iteração o valor de *Run* é deduzida de *Zeros_Left*, até acabarem-se todos os *Runs* ou *Zeros_Left* for igual a zero. Caso não existam Total Zeros este elemento não é codificado.

4.3 CABAC

O algoritmo do CABAC trata da codificação de elementos sintáticos no nível de macrobloco, ou seja, todos os elementos sintáticos pertencentes ao *slice*, fora o seu cabeçalho que é codificado pelo EXPG. Disponível para os perfis *Main* e *Highs* o CABAC tem um desempenho de compressão em média 7% mais eficiente se comparado ao CAVLC (MARPE, 2003). O processo de leitura do bloco de resíduos ocorre da mesma maneira que o CAVLC, porém os elementos sintáticos produzidos para representar os blocos são:

- *coded_block_flag*: indica se o bloco possui elementos diferentes de zero. Caso este elemento sintático for igual a ‘1’ os demais elementos são codificados, se ‘0’ a codificação do bloco termina.

- *significant_coeff_flag*: identifica todos os elementos sintáticos na ordem de varredura, quando ‘0’ a amostra é zero, quando ‘1’ o elemento é uma amostra não zero.

- *last_significant_coeff_flag*: identifica se é a última amostra não zero no bloco. Este elemento sempre precede um *significant_coeff_flag*= ‘1’. Se a amostra não é a última do bloco, este elemento é igual a ‘0’, se for a última este elemento é ‘1’.

- *coeff_abs_level_minus1*: é o valor absoluto menos um de uma amostra não zero.

- *coeff_sign_flag*: determina o sinal de uma amostra não zero. Quando positivo ‘0’, quando é um valor negativo, ‘1’. Este elemento sempre precede um *coeff_abs_level_minus1*.

Com todos os elementos sintáticos determinados, pode-se realizar o método de codificação de compressão do CABAC que se divide em três etapas: a Binarização, a Escolha do Modelo de Contexto e a Codificação Aritmética.

4.3.1 Binarização

A binarização é a etapa em que todos os elementos sintáticos são mapeados para códigos binários conhecidos por *binstring.* O CABAC apresenta quatro métodos básicos de binarização: Unário (U), Unário Truncado (TU), Tamanho Fixo (FL) e Exponencial Golomb Parametrizável (EGk). O método unário converte um número inteiro sem sinal numa sequência de ‘1’s acrescido de um ‘0’ ao final, com tantos ‘1’s quanto for o valor do número. O método Unário Truncado funciona da mesma maneira que o método Unário, mas limitado por uma variável ‘c’ que determina o tamanho máximo do *binstring.* Quando o valor unário extrapola o valor de ‘c’ a sequência binária acaba sendo representada somente por ‘1’s. O método Tamanho fixo representa um dado valor unário de uma maneira binária com tamanho limitado pela variável ‘L’. No método Exponencial Golomb Parametrizável a variável ‘k’ determina o tamanho inicial de bits do sufixo, o prefixo sempre começará com 1 quando houver um sufixo e aumentará quando o sufixo também aumentar o tamanho em bits de sua representação. O prefixo é composto de uma sequência de ‘1’s mais um ‘0’ que separa o prefixo do sufixo. A Tabela 4.3 apresenta exemplos de binarização utilizando os métodos descritos anteriormente.

Tabela 4.3 – Exemplo dos Métodos de Binarização

Elemento Sintático	Unário	Truncado Unário, c=4	Tamanho Fixo, L=4	Exponencial Golomb Parametrizável k=2
0	0	0	0000	0
1	10	10	0001	1 0 00
2	110	110	0010	1 0 01
3	1110	1110	0011	1 0 10
4	1111 0	1111	0100	1 0 11
5	1111 10	1111	0101	11 0 000

Alguns elementos sintáticos possuem uma binarização específica. Os elementos *mb_type* e s*ub_mb_type* são binarizados utilizando um conjunto de tabelas presentes na norma ITU-T(2005). O elemento CBP tem seu prefixo de 4 bits binarizado por FL e o sufixo composto de 2 bits, quando presente, por TU. O elemento *mb_qp_delta* é um dos poucos elementos que pode assumir um valor negativo. É tratado inicialmente da mesma maneira que um *CodeNum* de um elemento **se** do EXPG para depois ser

binarizado com o método U. Os vetores de movimento e os *coeff_abs_level_minus1* do bloco dos resíduos são binarizados por Concatenação de Unário Truncado com Exponencial Golomb Parametrizável (UEGk). Nestes casos o elemento sintático começa a ser codificado pelo método Unário Truncado. Quando ocorre a extrapolação da representação do valor, começa-se a binarizar pelo Exponencial Golomb Parametrizável e concatenam-se seus valores (MARPE, 2003; ITU-T, 2010; MARTINS, 2011). A Tabela 4.4 apresenta os elementos sintáticos que são codificados pelo CABAC. Ela também mostra o método de binarização com os valores das variáveis definidas, assim como o tamanho possível para os *binstrings* de cada elemento sintático.

Tabela 4.4 – Métodos de Binarização e tamanho de *bins* dos Elementos Sintáticos

Elemento Sintático	Largura do Binstring		Método de Binarização
	Prefixo	Sufixo	
mb_skip_flag	1		FL (L=1)
mb_field_decoding_flag	1		FL (L=1)
transform_size_8x8_flag	1		FL (L=1)
mb_type	1-6	0-7	Tabela
sub_mb_type	1-6		Tabela
coded_block_pattern	4	0-2	FL (L=4) e TU (c=2)
mb_qp_delta	1-26		U (caso especial)
end_of_slice	1		FL (L=1)
ref_idx_l0	1-5		U
ref_idx_l1	1-5		U
mvd_l0	1-18	0-9	UEGk(c=9, K=3)
mvd_l1	1-18	0-9	UEGk(c=9, K=3)
intra_chroma_pred_mode	1-2		TU (c=2)
prev_intra4x4_pred_mode	1		FL (L=1)
rem_intra4x4_pred_mode	3		FL (L=3)
prev_intra8x8_pred_mode	1		FL (L=1)
rem_intra8x8_pred_mode	3		FL (L=3)
coded_block_flag	1		FL (L=1)
significant_coeff_flag	1		FL (L=1)
last_significant_coeff_flag	1		FL (L=1)
coeff_abs_level_minus1	1-25	0-13	UEGk(c=14, K=0)
coeff_sign_flag	1		FL (L=1)

4.3.2 Escolha do Modelo de Contexto

O Modelo de Contexto é um modelo probabilístico com as estatísticas de frequência simbólica para cada *bin* binarizado. Este modelo é utilizado pela codificação aritmética

para gerar o *bitstream*. Para cada *bin* é selecionado um modelo probabilístico que possui a informação do símbolo menos provável (LPS) de ocorrer, 0 ou 1, e um índice de probabilidade (σ) que determina o tamanho da probabilidade de LPS ocorrer dentro do intervalo. Dependendo do perfil utilizado podem haver até 1024 modelos de contexto para os *bins* regulares. Para estes *bins* é utilizada a codificação aritmética regular. Existem *bins* que não usam modelagem de contextos. Estes *bins* estão associados aos elementos sintáticos *mvd_l0, mvd_l1, coeff_abs_level_minus1* e *coeff_sign_flag*. Para estes *bins* é usada a codificação aritmética *bypass*.

Para a escolha do Modelo de Contexto de um símbolo deve-se determinar o seu índice (*ctxIdx*). Em (4.4) é apresentado o equacionamento para determinar-se *ctxIdx*.

$$ctxIdx = ctxOffset + ctxInc + ctxCat \quad (4.4)$$

A variável *ctxOffset* é diretamente relacionada ao tipo de elemento sintático que o símbolo pertence. Ela seleciona o início do intervalo de contextos daquele elemento sintático. A variável *ctxInc* seleciona dentre os diferentes contextos de um elemento sintático, baseado em fatores como:

- O tipo do elemento sintático e índice do *bin*;

- Valores dos elementos sintáticos dos macroblocos vizinhos (superior e a esquerda);

- Os elementos sintáticos e *bins* já codificados;

- A posição do coeficiente da amostra no bloco de resíduos;

- Informações do *coeff_abs_level_minus1* no bloco de resíduos.

A variável *ctxCat* só é incluída no cálculo de *ctxIdx* para determinar os contextos relacionados aos elementos sintáticos do bloco de resíduos. Funcionalmente o *ctxCat* atribui os melhores contextos para os elementos sintáticos de acordo com o tipo do bloco. Ele categoriza o tipo do bloco que está sendo codificado, conforme tamanho: 4x4, 8x8 ou 16x16, se luminância ou crominância, e se AC ou DC. Uma descrição bem definida sobre a seleção de contextos dos elementos sintáticos pode ser consultada na norma (ITU-T, 2009) e (MARTINS , 2011).

4.3.3 Codificação Aritmética

O codificador aritmético é o módulo responsável pela geração do *bitstream* a partir da leitura dos *bins* e seus respectivos modelos de contexto quando existirem. Ele possui três modos distintos de operação: regular, *bypass* e *terminate*.

O Modo regular trata daqueles *bins* que possuem um modelo de contexto. Maiores detalhes a respeito desta codificação aritmética podem ser encontradas em (ROSA, 2010) e (SALOMON, 2010). O algoritmo da codificação aritmética realiza o seguinte procedimento resumidamente:

- Inicia-se lendo o modelo de contexto. O contexto determina o valor do símbolo menos provável (LPS), consequentemente o do mais provável (MPS), e a estatística que define o tamanho da faixa de cada símbolo. Lê-se assim o valor do LPS e determina-se

um comprimento de faixa correspondente dentro do intervalo de possibilidades para LPS e MPS.

- Lê-se o *bin*, para cada *bin* existem duas possibilidades: 0 ou 1, e seleciona-se a faixa do símbolo com o mesmo valor do bin. Esta faixa torna-se o novo intervalo de probabilidades. Toda vez que um intervalo é selecionado gera-se um bit na saída. Se o intervalo de probabilidades for menor que 256 a codificação aritmética se encerra senão volta-se para a leitura de um novo modelo de contexto.

O Modo *bypass* trabalha com os *bins* equiprováveis, ou seja, cujas probabilidades de ocorrência são iguais. Neste modo não ocorre consulta a um contexto nem o processo de normalização do intervalo como no modo regular .Somente são atualizados os limites superior e inferior do intervalo de probabilidades e gerado o bit de saída, simplificando a codificação.

O Modo *terminate* trata da codificação do elemento final de um macrobloco que possui um contexto de código específico. Neste modo há uma probabilidade fixa, onde não ocorre mais leitura de *bins*.

5 ARQUITETURAS PARA O BLOCO DE ENTROPIA

Este capítulo trata da descrição de todas as arquiteturas desenvolvidas dentro do bloco de entropia. São apresentadas soluções arquiteturais para os módulos do EXPG, CAVLC, CABAC, Buffer de Entrada e Montador Final. Estruturas de controle também foram desenvolvidas para a perfeita sincronização e acoplamento do processo de codificação. Dentre os módulos presentes, a arquitetura do CABAC não foi desenvolvida durante o mestrado sendo utilizadas as soluções já propostas por Martins (2011) e Rosa (2010). Tais soluções sofreram algumas modificações para integração entre si e com os demais módulos do bloco e por isto serão descritas a seguir.

5.1 Exponencial Golomb

A arquitetura do EXPG desenvolvida apresenta três entradas. A entrada "Início" recebe o sinal que dispara o processo de codificação. O sinal "Modo" indica o tipo do elemento se (ue), (se), (te) ou (me), além de selecionar o tipo de tabela a ser consultada na codificação do elemento sintático *Coded Block Pattern*. A entrada "ES" recebe o valor do elemento sintático em questão, sendo esta uma entrada de 8 bits. A arquitetura apresenta uma saída de 8 bits para o dado codificado, uma saída de 5 bits com o tamanho de bits válidos para o dado além de uma sinalização que indica quando este código está pronto. A Figura 5.1 apresenta um diagrama da arquitetura do EXPG.

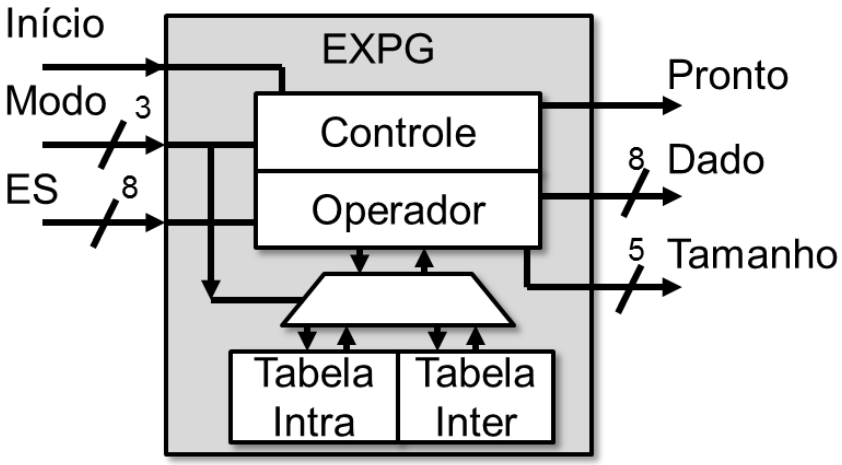

Figura 5.1: Diagrama de bloco do módulo EXPG

A necessidade de processar muitos elementos sintáticos diferentes exige que este codificador de entropia realize todo processo algorítmico no menor número de ciclos sem afetar de maneira drástica a frequência de operação. Assim, de uma primeira arquitetura que codificava um elemento sintático em três ciclos após o disparo do codificador, chegou-se a uma arquitetura que codifica em dois ciclos de relógio após a sinalização de início. Uma vez acionado o "Início" o dado de ES é codificado de acordo com o "Modo" existente. Caso o "Modo" for (me) haverá a consulta às tabelas. As tabelas variam com o tipo de predição escolhida na codificação do macrobloco. A codificação de qualquer elemento sintático, independente do "Modo", consome dois ciclos. No próximo ciclo o sinal de "Pronto" é acionado indicando que o elemento codificado e o seu tamanho já estão disponíveis na saída.

5.2 Gerenciador Exponencial Golomb

A arquitetura do EXPG está inserida dentro de uma estrutura maior, que gerencia e abastece o codificador com os elementos sintáticos de acordo com o tipo de unidade NAL a ser montada. Após um estudo da norma ITU-T(2005), da norma brasileira ABNT (2007) e das características intrínsecas dos módulos do codificador H.264/AVC já desenvolvidos no grupo de codificação de vídeo da UFRGS, uma lista com todos os elementos sintáticos relevantes foi criada.

Todos estes elementos sintáticos que devem ser processados pelo EXPG possuem entradas distintas para este módulo gerenciador. Por se tratar de elementos sintáticos de tamanhos diferentes, utilizou-se uma FSM para a ordenação destes elementos. Esta estratégia permite também o controle do armazenamento dos dados de saída do EXPG. A Figura 5.2 apresenta um diagrama deste Gerenciador.

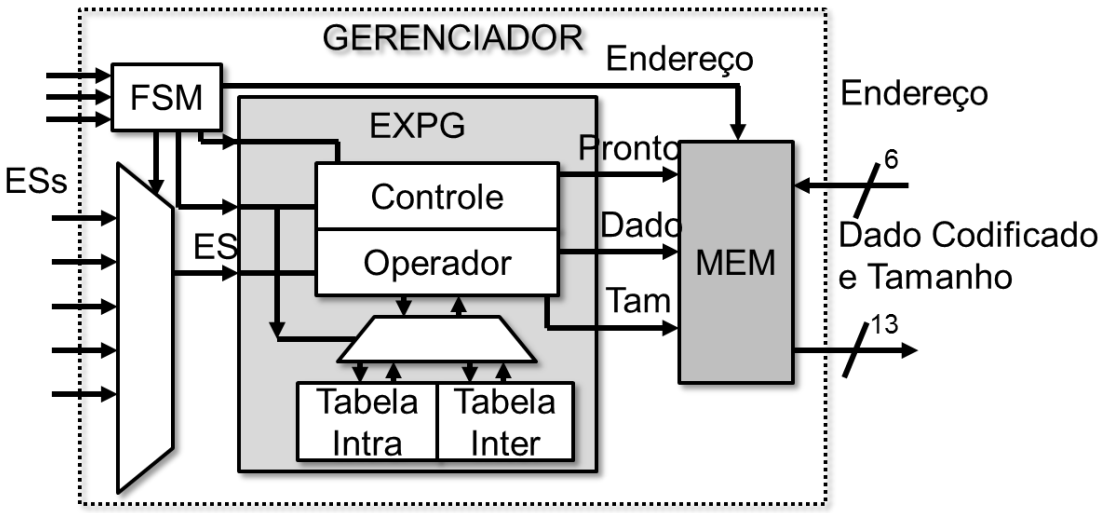

Figura 5.2: Diagrama de bloco do módulo Gerenciador

Para cada unidade NAL (SPS, PPS ou *Slice*) ele inicia a leitura dos elementos sintáticos existentes, associando o devido "Modo" a cada um deles e acionando o EXPG. Ao fim da operação do EXPG, os dados de saída são armazenados em uma memória. A utilização de uma memória na saída se fez necessária para a sincronização

com o Montador Final. O dado e seu tamanho são armazenados na mesma posição de memória, assim esta possui um tamanho de palavra de 13 bits. O sinal de Pronto do EXPG é usado como sinal habilitador de escrita. O endereçamento desta memória é realizado pelo gerenciador. A FSM também sinaliza ao Montador Final quando os elementos sintáticos de uma NAL estão codificados para o processo de montagem do *bitstream*.

A verificação deste módulo foi realizada em conjunto com o EXPG. O *testbench* desenvolvido gera valores válidos para os elementos sintáticos e aciona o Gerenciador para que este inicie o processo. Alguns elementos sintáticos foram testados forçando valores específicos. Os resultados gerados e armazenados são comparados paralelamente com os valores obtidos previamente na codificação do software de referência.

5.3 Buffer de Entrada

Conforme projeto do codificador H.264/AVC e a arquitetura dos blocos de Transformada e Quantização desenvolvida por DINIZ (2009), a saída de dados para a entropia é um bloco inteiro de amostras. Isto significa que o tamanho da saída é de 256 bits (16 amostras de 16 bits). Dentro do contexto da arquitetura proposta por Diniz (2009), um macrobloco inteiro quantizado deve ser armazenado em memória antes de ser processado pelos codificadores de entropia. Isto se deve porque ele calcula e armazena inicialmente o macrobloco 4x4. Depois processa o 16x16 e então determina em uma análise prévia, o melhor modo de predição em relação à taxa de bits e distorção. Se for o 16x16 reescreve o novo macrobloco, senão mantém na memória e por fim sinaliza para o buffer que o dado está pronto. O Buffer de Entrada é esta estrutura de memória responsável pelo armazenamento de um macrobloco quantizado e tem por finalidade disponibilizar este para as ferramentas de entropia CAVLC e CABAC. Conforme mostrado anteriormente, um macrobloco possui:

- 16 blocos de Luma, cada um com 16 amostras de 8 bits; ou 1 bloco de luma DC com 16 amostras e outros 16 blocos de luma AC contendo 15 amostras;

- 2 blocos de Croma DC (um de cada crominância), cada com 4 amostras de 8 bits;

- 8 blocos de Croma AC (quatro de cada crominância), cada com 15 amostras de 8 bits.

O buffer desenvolvido por (Ramos, 2010) apresenta duas estruturas de memória capazes de armazenar dois macroblocos e um controle para a leitura/escrita intercalada destas, a fim de armazenar um macrobloco caso o primeiro não ainda não for processado. Cada estrutura de memória possui 27 palavras, uma para cada bloco dentro do macrobloco, com tamanho de palavra de 256 bits.

Após estudos e experimentos realizados, a utilização de um buffer com uma única estrutura de memória para o armazenamento de um único macrobloco se mostrou suficiente, porque o processamento da entropia é capaz de consumir este macrobloco quantizado antes de o bloco de quantização calcular um próximo. Para evitar um possível conflito de dados foi desenvolvido ainda um sinal de ocupado para o bloco de

quantização indicando uma pequena espera para escrita de um novo bloco de macrobloco na memória.

A arquitetura do Buffer de Entrada proposta neste trabalho tem capacidade para armazenar um macrobloco inteiro, ou seja, 27 blocos de 256 bits. Com a seguinte disposição:

- A posição “0” é dedicada ao bloco de Luma DC;

- As posições de 1 a 16 são para os blocos de Luma ou Luma AC;

- A posição 17 é para o bloco de Croma DC Azul;

- A posição 18 é para o bloco de Croma DC Vermelho;

- As posições de 19 a 22 são para os blocos Croma AC Azul;

- As posições de 23 a 26 são para os blocos Croma AC Vermelho.

O buffer conta com mecanismos de controle para escrita e leitura dos blocos a fim de facilitar a escrita por parte do Bloco de Transformada e Quantização e leitura dos resíduos pelos codificadores de entropia. Durante a escrita a ferramenta de quantização só necessita aciona wr_buffer para escrever na memória o bloco contido na porta *input_data*. O endereçamento inicial do buffer é controlado pelo sinal de intra_mode que determina a posição do primeiro bloco do Macrobloco. Este sinal indica quando o bloco é 4x4 ou 16x16. Quando for 16x16, a posição “0” da memória é utilizada para armazenar o bloco de LUMA DC, quando 4x4 o primeiro bloco que é de Luma é armazenado na posição “1”.

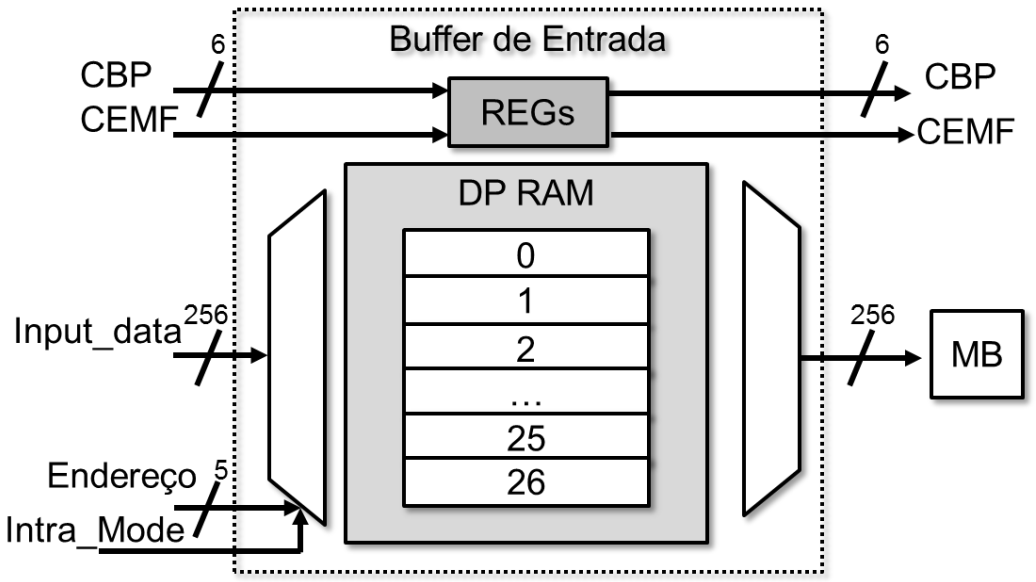

Figura 5.3: Diagrama em Bloco do Buffer de Entrada

Este Buffer é usado por ambos os codificadores CAVLC e CABAC. Além disso, os elementos sintáticos *Coding_Entropy_Mode* (CEMF) e *Coded_Block_Pattern* (CBP) são armazenados e necessários para a leitura do buffer. A Figura 5.3 apresenta um diagrama da FSM reponsável pela leitura de blocos. O elemento *Coding_Entropy_Mode* controla e aciona a respectiva ferramenta de codificação, quando ‘0’, sinaliza o uso do

CAVLC e quando ‘1’, o CABAC. O Bloco inteiro é disponibilizado para ambos os codificadores. O CBP é composto por 6 bits. Cada um dos 4 primeiros bits indica uma região de 4 blocos de Luma do Macrobloco e os 2 últimos indicam os blocos de Croma do Macrobloco que possuem dados quantizados diferentes de zero. Para o CAVLC existe a possibilidade de o buffer só enviar blocos com dados relevantes e sinalizar ao CAVLC de um bloco zerado. Como o CBP é usado internamente pelo CABAC, todos os blocos são enviados para que o controle interno do CABAC trate destes dados e funcione corretamente.

5.4 Codificação por Código de Comprimento Variável Adaptativo ao Contexto - CAVLC

Na codificação de entropia do H.264/AVC, o CAVLC é a solução de baixa complexidade. O CAVLC busca explorar as características particulares dos blocos de elementos quantizados, entre elas podemos destacar: o bloco quantizado possuir uma matriz esparsa (muitos elementos zerados), no bloco o nível DC e os componentes de baixa frequência (valores com maior magnitude) concentram-se no canto superior esquerdo e os componentes de mais alta frequência (baixa magnitude) se distribuem pela diagonal oposta, para a direita e para baixo. Além disso, explora-se também a similaridade entre os blocos vizinhos a fim de adaptar-se ao “contexto”.

A arquitetura do codificador foi dividida em três estágios de macropipeline como nos trabalhos de (Silva, 2007) e (Ramos, 2010). Após estudo e pesquisa por outras soluções arquiteturais tentou-se aplicar as técnicas propostas de Rahman e Badawy (2007) e Albanese e Licciardo (2010) que reduziram de maneira significativa a área do CAVLC, através das técnicas de *Table Split* e *Arithmetic Table Elimination* (ATE). Uma visão geral da arquitetura é apresentada na Figura 5.4.

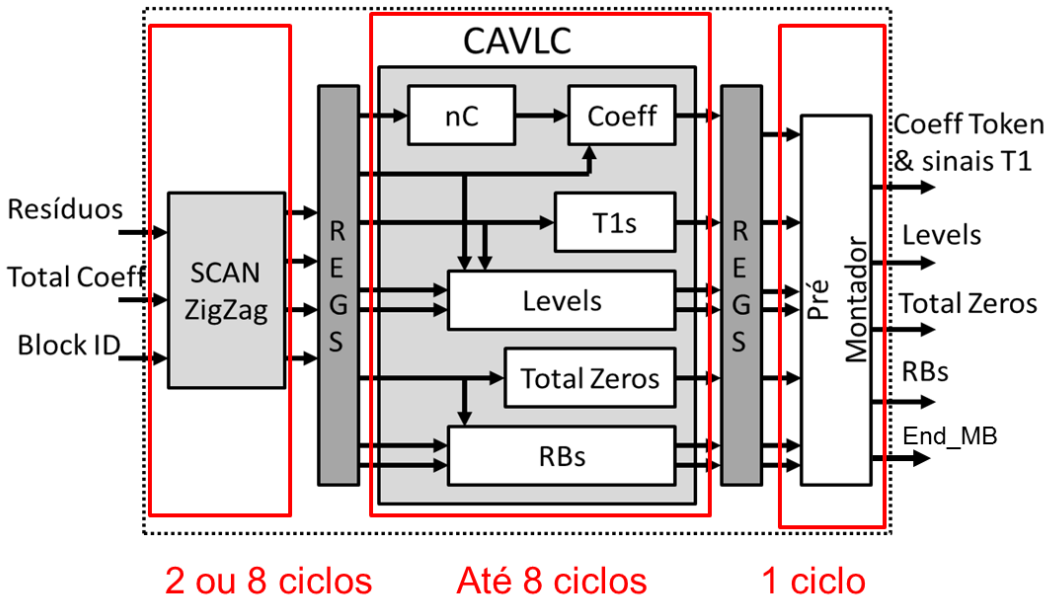

Figura 5.4: Diagrama em blocos da arquitetura do CAVLC

O Bloco *Scan ZigZag* faz a leitura de 2 amostras do bloco por ciclo. Assim para blocos com 16 e 15 amostras a leitura ocorre em 8 ciclos ao passo que para 4 amostras leva 2 ciclos. Nesta leitura Tot_coeff indica o número máximo de amostras daquele bloco para controlar os ciclos de leitura. Os sinais de *Block ID* e *end_MB* são somente armazenados sincronizados com os elementos extraídos da leitura, para a próxima etapa de codificação. Os elementos extraídos da leitura são: *TotalCoeff*, número de T1s, sinais de T1s, número de *Levels*, valores dos *Levels*, números de *Total_Zeros*, Número de RBs, valores de RBs e seus respectivos *Zeros_ Left*.

A primeira barreira temporal é formada por registradores para todos os dados provenientes da leitura do bloco. Um controle de *handshake* dela com o Bloco de Leitura e o Bloco de Tabelas sem a utilização de um controle geral baseado em FSM garante o perfeito funcionamento do macropipeline do CAVLC. No Bloco de Tabelas ocorre a codificação dos elementos sintáticos onde cada elemento sintático possui um bloco específico para sua codificação.

No Bloco nC é armazenado o *TotalCoeff* deste bloco para serem utilizados pelos seus vizinhos e cálculo do nC deste bloco. O tamanho da memória dedicada é baseado no número máximo de macroblocos em uma linha de quadro *Full HD*. Para endereçar a memória um contador de macroblocos foi utilizado e é delimitado por um parâmetro global de número máximo de macroblocos em uma linha do quadro. Cada posição de memória armazena o total de elementos não-zero de um Macrobloco (4 blocos inferiores de Luma, 2 blocos inferiores de croma azul e 2 blocos inferiores de croma vermelho), para serem utilizados pelos blocos superiores do macrobloco inferior a este. O conjunto de registradores armazena o *TotalCoeff* de um bloco para outro dentro do mesmo macrobloco e também armazena os *TotalCoeffs* para os blocos do próximo macrobloco à direita. O *Block ID* é utilizado como referência para determinar as vizinhanças do bloco em questão para o cálculo de nC. Este bloco consome 1 ciclo de relógio.

O Bloco Coeff é responsável pelo elemento sintático *Coeff_Token*. Baseado nos elementos *TotalCoeff*, nC e número de T1s. Estas tabelas foram suprimidas para cada tipo de nC a fim de minimizar o consumo de recursos. Para nC igual a 0 e 1 é utilizada uma tabela de 64 palavras de 11 bits - 6 bits com o valor de *Coeff_Token* e 5 bits para o tamanho. Para nC igual a 2 e 3 e de 4 à 7 usam-se tabelas de 64 palavras de 10 bits - 6 bits com o valor de *Coeff_Token* e 4 bits para o tamanho. Para nC igual -1, uma tabela de 14 palavras de 10 bits - 6 bits com o valor de *Coeff_Token* e 4 bits para o tamanho. Para nC igual -2, uma tabela de 30 palavras de 10 bits - 6 bits com valor de *Coeff_Token* e 4 bits para o tamanho. Quando nC igual a 8 é realizada uma lógica com o valor de *TotalCoeff – 1* - com tamanho de 4 bits - concatenado ao número de T1s - com tamanho de 2 bits - gerando um código de tamanho fixo de 6 bits. Este bloco consome 1 ciclo por ser apenas uma consulta a tabela e mesmo dependendo do valor do nC, este não se torna o caminho crítico deste estágio.

O Bloco T1s é somente uma pequena lógica para reordenar e determinar os *T1_sign*. Este bloco consome 1 ciclo não sendo determinante no tempo de codificação deste estágio do macropipeline.

O Bloco Levels realiza a codificação de dois *Levels* por ciclo. Para determinar o primeiro índice são levados em conta o *TotalCoeff* e número de T1s. Para que os dois *Levels* possam ser codificados ao mesmo tempo são realizadas as codificações para

todos os índices possíveis, através de um conjunto de máscaras deixando assim somente o cálculo do índice para definir o *Level* codificado. Um segundo índice para o próximo *Level* é derivado do cálculo do primeiro índice. Após a definição destes índices, o primeiro *Level* e segundo *Level* são codificados simultaneamente. Este bloco se manteve inalterado de Ramos (2010). Este bloco pode ter um consumo de até 9 ciclos caso todos os resíduos do bloco sejam *Levels*.

O Bloco Total Zeros recebe *TotalCoeff* e o número de *Total Zeros* para gerar o elemento sintático Total_zeros. Neste bloco foi aplicada a técnica para reduzir o tamanho da tabela. Deste modo, as tabelas têm palavras de 7 bits (3 bits de código e 4 bits indicando o tamanho). Este bloco por ser uma simples consulta à tabela, leva 2 ciclos para codificar seu elemento sintático.

O Bloco RBs processa 2 *Run_Before* por ciclo. Com o *Zeros_Left* de cada *Run_Before* já determinado durante o processo de leitura do bloco de resíduos, cada *Run_before* pode ser codificado independentemente. Sua tabela com resultados foi substituída por lógica para a economia de recursos do FPGA. Este bloco pode consumir até 8 ciclos no pior caso onde 14 *Run_before* precisam ser codificados.

A segunda barreira temporal armazena todos os elementos sintáticos codificados e seus respectivos tamanhos para a montagem. O pré-montador dos dados codificados foi desenvolvido visando uma taxa de processamento alta o suficiente para não aumentar o número de ciclos entre estágios do macropipeline do CAVLC. Assim *coeff_token* e *Trailling_ones* são montados juntos num ciclo, dois *Levels* são montados por ciclo e dois *run_befores* são montados juntos.

A verificação deste modo foi realizada através de um *testbench* contemplando também o Módulo Buffer de Entrada. Macroblocos quantizados foram amostrados na entrada para o processo entrópico e os resultados comparados com os resultados obtidos com o software de referência. Foi realizado também uma simulação considerando o pior caso (bloco de 16 amostras, todas sendo *Levels*), depois do *pipeline* cheio, um bloco destes consome 9 ciclos para ser processado.

5.5 Codificação por Código Aritmético Binário Adaptativo ao Contexto - CABAC

O CABAC utilizado neste codificador de entropia é formado pelas soluções arquiteturais já desenvolvidas na Universidade Federal do Rio Grande do Sul por Martins(2011) e Silva(2010). A organização do CABAC é dividida em dois grandes módulos de hardware: o Módulo do Modelo de Contexto Binário (BCM) e o Módulo do Codificador Aritmético Binário (BAE), como mostrado na Figura 5.5.O BCM está dividido em três macropipelines. No Bloco de Mapa de Significância, o bloco transformado quantizado contendo as amostras de resíduos é processado em paralelo para a geração dos elementos sintáticos. Os elementos sintáticos não residuais possuem entradas individuais no Bloco Gerador e Ordenador de ESs. Lá todos os elementos sintáticos são montados ordenadamente para a binarização.

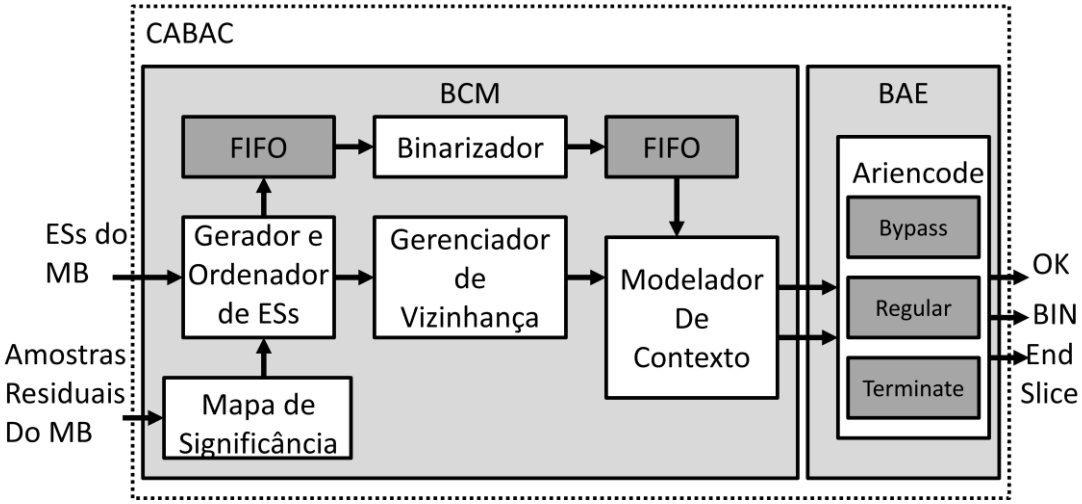

Figura 5.5: Diagrama em blocos da arquitetura do CABAC

Na binarização os ESs são convertidos em códigos binários. A arquitetura do binarizador escolhida para esta implementação do CABAC é a *Context-Aware*, por ser a mais eficiente na relação processamento por área ocupada. O esquema desta arquitetura pode ser observado na Figura 5.6.

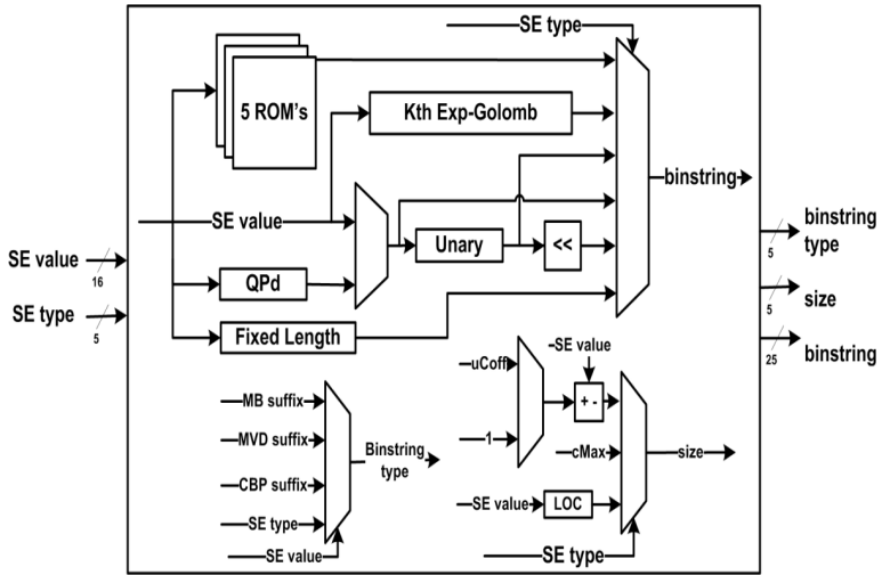

Figura 5.6: Esquema do Binarizador (Martins, 2011)

As 5 ROMs apresentadas são usadas na conversão de ESs de construção compostas como o *mb_type*. O Módulo Kth Exp-Golomb é usado para codificar o sufixo de ESs do tipo UEGk. O módulo QPd converte os ESs com sinal. O módulo Unary faz a codificação dos ESs do tipo Unário, Truncado Unário (usado em conjunto com o registrador de deslocamento) e prefixo do UEGk. O Bloco Fixed length é usado para os ESs do tipo tamanho fixo. O Bloco LOC determina os tamanhos dos *binstrings* gerados pelo ESs do tipo Unário. A binarização de ESs compostos por prefixo e sufixo demoram 2 ciclos ao passo que os demais ESs levam apenas um ciclo.

No Modelador de Contexto os contextos são calculados e, além disso, os *bins* são sincronizados com os respectivos contextos para a codificação aritmética. Neste módulo são identificados os *bins* do tipo *bypass* para o devido tratamento. Foi introduzido um controle nesta arquitetura que trava a FSM de controle do Modelador caso o BAE esteja ocupado.

No Bloco de Codificação Aritmética (BAE) o fluxo de dados é limitado pela sua característica sequencial de processamento *bin* a *bin*, onde o valor do bin a ser processado depende do valor do seu valor anterior. Apesar das diferentes estratégias para a implementação arquitetural da mesma, foi definida para a utilização deste codificador entrópico a solução sequencial. Esta solução tem capacidade para processar os macroblocos com o uso de pouca área de circuitaria, mostrando-se a mais eficiente considerando desempenho por área dentre as apresentadas por (Rosa, 2010). O codificador é baseado num *pipeline* de 4 estágios de execução. A arquitetura deste codificador aritmético é composto por implementações para tratar de *bins* regulares e *bypass* sob um mesmo controle para a simplificação do hardware. O Bloco de Codificação Aritmética com arquitetura de renormalização sequencial é apresentado na Figura 5.7.

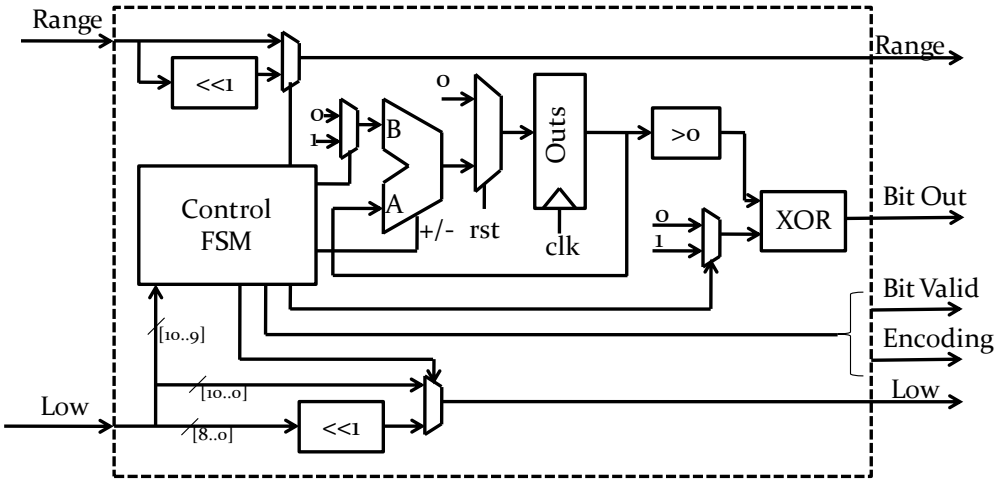

Figura 5.7: Esquema da arquitetura sequencial para renormalização (Rosa, 2010)

Esta arquitetura está limitada a um passo de renormalização por ciclo. Pode ser necessário até seis passos de renormalização para que registrador RANGE assuma um valor maior que 255. Quanto aos bits *outstanding* que podem vir a ser produzidos, está arquitetura possui um contador de 8 bits (até 256 bits) para o seu registro para que quando for decidido um bit válido, os *outstanding* também serem decididos. Maiores detalhes desta arquitetura podem ser consultados em (Rosa, 2010). No BAE foi criado um sinal de *end_slice* no modo Terminate para indicar ao Montador a finalização da NAL.

A simulação do CABAC foi realizada conjuntamente com o Buffer de Entrada. O *testbench* alimentava com MBs quantizados e os *bins* foram comparados com os obtidos no software de referência. O desempenho da arquitetura ficou limitada ao desempenho do BAE que obteve um desempenho de 0,67 *bin*/ciclo, mesmo que o processo de binarização garanta 1 *bin*/ciclo na saída do BCM. Este desempenho é capaz de atender uma codificação Full HD de vídeo.

5.6 Montador Final

O Montador Final tem como função concatenar em ordem todos os elementos sintáticos já processados pelos codificadores de entropia em uma única sequência de bits. Basicamente ele é formado por três máquinas de estados e um conjunto de registradores de deslocamento, como mostra a Figura 5.8.

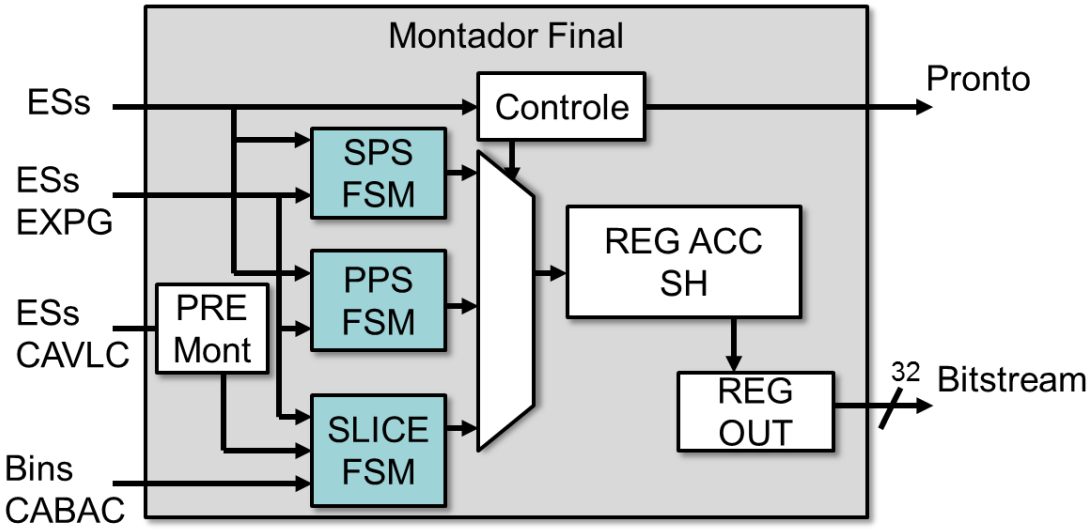

Figura 5.8: Diagrama em blocos da arquitetura do Montador Final

Cada uma das FSMs é responsável pelo ordenamento de um tipo de unidade NAL. E cada uma tem um sinal de *start* independente. Quando uma delas é acionada o controle direciona o endereçamento da memória do EXPG e seus dados de saída ficam disponíveis para o conjunto de registradores de deslocamento para a concatenação dos dados até a NAL ser totalmente montada. As SPS FSM e PPS FSM utilizam dados com valores pré-fixados, dados que não passam por uma codificação entrópica e elementos sintáticos provenientes da codificação do EXPG. Determinados elementos sintáticos que através de estudos das normas e das limitações impostas pelos módulos já desenvolvidos no projeto do codificador H.264/AVC tiveram seus valores fixados para uma simplificação do processo de montagem e assim, reduzindo estados e agrupando alguns elementos. Este montador contempla somente a funcionalidade para os perfis *Baseline* e *Main*. A SPS FSM possui 10 estados de montagem. A PPS FSM possui 7 estados de montagem. O anexo A apresenta uma lista com todos os elementos sintáticos tratados.

O fluxograma da SLICE FSM é mostrado na Figura 5.9. Ela possui 18 estados para a montagem do cabeçalho de *Slice*, 8 estados para o cabeçalho de macrobloco, 5 estados para a manipulação dos elementos sintáticos do CAVLC e 2 estados para os *bins* do CABAC.

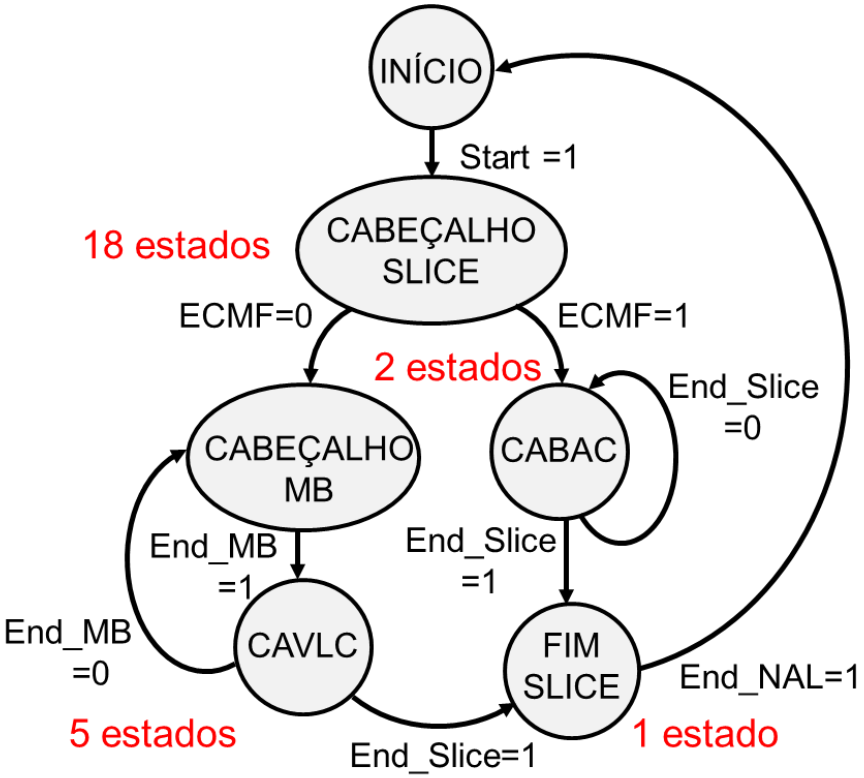

Figura 5.9: Fluxograma da SLICE FSM

O sinal de *entropy_code_mode_flag* (ECMF) determina de qual codificador os dados serão montados na NAL. O processo entra em um laço onde cabeçalhos de MB, os blocos e macroblocos são montados. Se o montador receber por parte dos codificadores o sinal de *end_slice* ocorre o fechamento da NAL de *Slice* com a devida sinalização como apresentado em (ITU- T, 2010). E o montador entra em um estado de espera por um novo sinal de início. O Conjunto de registradores pode receber dados de até 32 bits e a cada 16 bits montados ocorre o carregamento da saída e a sua sinalização de dado válido na saída.

A verificação deste módulo foi realizada em conjunto com os outros módulos da entropia. Foram realizadas simulações para montagem de cada tipo de NAL. Foram definidos valores de elementos sintáticos que são gerados por outros módulos do codificador para a validação das NALs de SPS e PPS e comparadas com *traces* extraídos do software de referência. Foram validadas as montagem dos dados da NAL de VLC provenientes dos codificadores entrópicos Exponecial Golomb, CAVLC e do CABAC.

6 RESULTADOS E ANÁLISE DAS ARQUITETURAS

Neste capítulo são mostrados os resultados obtidos neste trabalho na integração das arquiteturas dos codificadores de entropia desenvolvidos, em termos de área e de desempenho em FPGAs. A seguir são apresentadas as arquiteturas de codificação de entropia do H.264/AVC mais relevantes em FPGA encontradas em trabalhos da literatura juntamente com uma comparação com a deste trabalho.

6.1 Resultados Obtidos

A arquitetura proposta para o Codificador de Entropia foi desenvolvida utilizando a linguagem de descrição de hardware VHDL. Esta descrição de hardware foi direcionada para ser implementada em uma plataforma FPGA. A ferramenta de síntese utilizada foi o ISE Project Navigator Versão 10.1. A etapa de síntese foi realizada com a configuração padrão da ferramenta que é *Timing Driven*. O FPGA escolhido foi o Virtex-5 xc5vlx110T-3-FF1136 da Xilinx (Xilinx, 2012). Dentre as estruturas dedicadas existentes neste FPGA, foram utilizadas o DSP48Es, um multiplicador acumulador (utilizado no CABAC), e a BRAM, uma memória dedicada de 36Kbits.

Os resultados de síntesede toda a arquitetura desenvolvida no dispositivo Virtex5 empregam 4304 Slice registers, 6868 Slice LUTs e 7 BRAMs atingindo uma frequência de operação de 185 MHz. A Tabela 6.1 resume os resultados obtidos na síntese individual de cada módulo da arquitetura desenvolvida para o codificador de entropia.

Tabela 6.1 – Resultados de Síntese Lógica dos Módulos para Virtex5

Módulo	Virtex-5		
	Slice Register	Slice LUT	Freq. (MHz)
Gerenciador + EXPG	33	134	309
CAVLC	2367	2536	204
CABAC	1373	3311	219
Buffer de Entrada	316	417	322
Montador Final	173	579	298

As simulações para medir o desempenho do codificador foram realizadas utilizando a ferramenta Modelsim SE 6.3F. A implementação do CAVLC apresenta resultados de desempenho utilizando a métrica de ciclos/MB. Em um quadro com resolução 1920x1080 existem 8160 MBs. Operando em uma taxa de 30 fps, desta forma serão 244.800 MBs por segundo. Nas simulações foram utilizadas as sequências pós quantização de vídeo Akyio, Foreman e Mobile extraídos do software de referência para servir de dados de entrada para a simulação dos módulos. Neste caso as sequências apresentaram respectivamente em média 216, 227 e 248 ciclos por macrobloco desconsiderando o CBP que simplificaria o processo de codificação e diminuiria o número de ciclos consumidos. Considerando 250 ciclos por macrobloco, verificou-se que utilizando uma frequência de 125MHz é possível codificar um vídeo de 1920x1080 a uma taxa de 60fps, atendendo ao nível 4.2 da norma.

Foi realizada uma análise de pior caso, onde todos os blocos de todos os MBs do quadro exijam o máximo de tempo nos estágios de pipeline. Baseado no funcionamento da arquitetura do CAVLC temos que o tempo do primeiro estágio é de 8 ciclos, do segundo estágio pode ser de até 8 ciclos e na etapa de montagem exatamente 10 ciclos. Uma vez que o pipeline está cheio, o tempo dos estágios do pipeline é ditado pelo tempo de montagem, que é de 10 ciclos. Assim para os 26 blocos de um MB, temos 257 ciclos para que todos os blocos entrem no CAVLC mais 20 ciclos para o seu esvaziamento. Deste modo, o tempo total para o pior caso de codificação do CAVLC é de 277 ciclos. Para este caso ainda é possível processar um vídeo 1920x1080p a 60fps, utilizando uma frequência de operação de 140MHz.

No CABAC a métrica de desempenho utilizada é *bins*/ciclo. Como visto anteriormente o processo de binarização possui uma taxa de 0,83 *binstring*/ciclo, e o Modelador de Contexto garante 1 bin/ciclo, porém o CABAC inteiro fica limitado pelo BAE que codifica a uma taxa de 0,677 bin/ciclo. Durante as simulações o desempenho do CABAC foi de ~330 ciclos por MB. Desta forma, utilizando o CABAC como ferramenta de codificação para o processamento de vídeos 1920x1080 a 60fps, é necessário uma frequência de 163 MHz.

6.2 Trabalhos Relacionados

Os trabalhos que possuem arquiteturas de entropia para comparação com as arquiteturas desenvolvidas aqui são apresentados a seguir. É interessante notar que são poucos os trabalhos direcionados para soluções em FPGA. Muitos artigos apresentam no seu conteúdo validação em FPGA, mas os dados apresentados não permitem uma análise precisa, como por exemplo, mostrar o número de LUTs ocupada e não citar o dispositivo utilizado. Assim, os trabalhos aqui apresentados são aqueles que apresentaram resultados relevantes em termos de área e desempenho.

6.2.1 Trabalhos de Rahman

Os dois trabalhos apresentados por este autor focam em arquiteturas para a redução de área, implementando somente o CAVLC. Ambas as arquiteturas foram sintetizadas para um FPGA da família Virtex II, da Xilinx.

Nestas arquiteturas o CAVLC é divido em 3 estágios. Na primeira parte é realizada a leitura de uma amostra por vez e alguns parâmetros são pré-codificados. A segunda etapa realiza a consulta das tabelas e a codificação de *Levels* e *RBs.* Nesta etapa o autor

propõe uma simplificação nas tabelas de *Coeff_token*, *Zeros_left* e *RB* e aplica a técnica de *Arithmetic Table Elimination*, que consiste na substituição das tabelas somente por cálculos, para obter-se o prefixo e sufixo do *Level*. Desta maneira, ele busca obter a melhor economia em área. Pode-se observar também que ele não considera o armazenamento dos coeficientes dos vizinhos para o cálculo do *nC*, sendo que estes são buscados de uma memória externa ao CAVLC. Na última etapa realiza-se a montagem do *bitstream*.

Na primeira arquitetura proposta (RAHMAN, 2006), o desempenho atingido foi de 30 quadros CIF (352x288) por segundo, com uma frequência máxima de 60 MHz. A área ocupada foi de 1533 LUTs. Na arquitetura do segundo trabalho (RAHMAN, 2007), focada em redução de área, o desempenho atingido foi também de 30 quadros CIF (352x288) por segundo, mas com uma frequência máxima de 52,8 MHz. A área ocupada foi de 925 LUTs.

6.2.2 Trabalho de Silva (2007)

No artigo (SILVA, 2007) encontra-se o desenvolvimento de um codificador de entropia com EXPG e CAVLC. Esta arquitetura foca no desempenho para atender a demanda de alta definição e foi sintetizada para um FPGA Virtex II da Xilinx.

Este CAVLC apresenta um pipeline multiciclo de 3 estágios. No primeiro estágio há a leitura de uma amostra por vez e pré-tratamento dos elementos sintáticos. No segundo estágio são realizadas as consultas a tabelas e cálculos. Neste estágio o artigo mostra que as tabelas para os elementos *nC*, *TotalZeros* e RB são implementadas em ROMs num total de 5, mas não são descriminados seus tamanhos. E no terceiro estágio é realizada a montagem do *bitstream*, elemento a elemento.

Esta arquitetura processa uma amostra por ciclo, o que em termos de ciclos por MBs é igual a 384, Tem um desempenho para atender 30 quadros Full HD (1920 x 1080) em uma frequência máxima de 117,62 MHz. A área consumida para esta arquitetura foi de 3500 células lógicas.

6.2.3 Trabalho de Pastuszak (2008)

No artigo (PASTUSZAK, 2008) o autor apresenta um codificador de entropia completo com EXPG, CAVLC e CABAC. Esta arquitetura busca um compartilhamento de entradas, memórias e saídas entre o CAVLC e CABAC. Neste trabalho ainda o autor apresenta diferentes versões na arquitetura do codificador aritmético e modelador de contexto do CABAC. Este autor apresenta os resultados para suas arquiteturas separadamente o que permite uma comparação com os demais.

Ele propõe um CAVLC com três estágios de pipeline multiciclo. No primeiro estágio são lidas todas as amostras, uma a uma, e são extraídos parâmetros como *TotalCoeff*, TotalZeros, T1s e um mapa de significância que indica que elemento deve ser processado no próximo estágio. Neste primeiro estágio também é determinado o *nC* para o próximo estágio. No segundo estágio cada elemento é codificado e algumas memórias foram implementadas usando M4K (memórias dedicadas), e no terceiro os elementos sintáticos são montados.

O CABAC é dividido em quatro estágios de pipeline. No primeiro há a leitura, identificação dos elementos sintáticos e armazenamento dos dados na memória de vizinhança. No segundo estágio ocorre a binarização. No terceiro estágio ocorre a codificação aritmética e por fim a montagem e encapsulamento da NAL. As diferentes

versões apresentadas de CABAC, num total de cinco, apresentam diferenças no número de canais de codificação, introduzindo até 3 modos *by_pass*, e na arquitetura do binarizador. Não fica claro qual arquitetura do CABAC foi utilizada na composição CAVLC+CABAC que apresenta valores de *throughput* e área.

Esta arquitetura foi sintetizada para um FPGA Stratix II da Altera, ocupando uma área de 3917 células lógicas. Ela suporta 30 quadros HD (720x576) com uma frequência máxima de 125 MHz.

6.2.4 Trabalhos de Albanese (2010a, 2010b)

Nestes trabalhos o autor focou em desenvolver arquiteturas de CAVLC que atendam a alta definição e a economia de área. O FPGA utilizado na síntese foi da família Spartan3. Ambas as arquiteturas apresentam dois estágios pipeline, uma de pré-codificação e outra de codificação e montagem. Na pré-codificação existem dois módulos de leitura acelerando este estágio. Na pré-codificação também já é calculado o índice para os *Levels*, permitindo o cálculo de dois *Levels* ao mesmo tempo no estágio de codificação. O cálculo de *nC* já é realizado na pré-codificação. A memória com dados de vizinhança e a estrutura responsável por seu endereçamento, para o cálculo de *nC* não é definida dentro deste projeto.

No estágio de codificação da primeira arquitetura se aplicam as técnicas de simplificação das tabelas de *Coeff_token*, *Zeros_left* e *RB*, e *Arithmetic Table Elimination* para os *Levels*. Já na segunda arquitetura é utilizada *Arithmetic Table Elimination* para quase todas as tabelas de elementos sintáticos substituindo-as por cálculos. No montador nenhuma arquitetura especial é descrita. de modo que as acelerações propostas para os estágios anteriores o transformam no gargalo da compressão.

As arquiteturas propostas atingem o desempenho de 30 quadros Full HD (1920 x 1080) por segundo, com uma frequência máxima de ~63 MHz. A área ocupada na primeira foi de 3447 LUTs (ALBANESE, 2010a). Na arquitetura do segundo trabalho a área ocupada foi de 2200 LUTs (ALBANESE, 2010b).

6.2.5 Trabalho de Ramos (2010)

No artigo (RAMOS, 2010) é apresentada uma arquitetura de CAVLC visando alto desempenho. O CAVLC neste projeto apresenta resultados para um FPGA Virtex5 da Xilinx.

Este trabalho apresenta uma arquitetura com três estágios de *pipeline*: Pré-codificação, codificação e montagem. O autor considera a leitura de uma amostra do bloco por ciclo como um ponto crítico no projeto e propõe uma arquitetura que lê dois elementos por ciclo no estágio de pré-codificação.

Na etapa de codificação cada elemento sintático é calculado paralelamente, propõe-se também uma codificação de dois *Levels* e dois *Runs Before* para aceleração do processo de codificação. No estágio de montagem a arquitetura realiza a montagem d dois elementos por vez. Um problema detectado é que com estas duplicações de hardware para acelerar a compressão, resultaram em uma grande aumento de área. O CAVLC desta arquitetura contempla a memória de dados da vizinhança dentro de sua arquitetura. Esta arquitetura tem um desempenho que atinge 115 quadros Full HD (1920 x 1080) em uma frequência máxima de 180 MHz. A área consumida para esta arquitetura foi de 10791 LUTs, além de 39 BRAMs.

6.3 Comparações com Trabalhos Relacionados

Para uma comparação justa com as arquiteturas apresentadas na literatura buscou-se realizar sínteses com as mesmas famílias de dispositivos apresentadas, assim foram usados resultados de síntese de Spartan3 para comparação com Albanese (2010a,2010b) e de Virtex2 para comparações com Rahman(2006,2007). Os trabalhos de Silva (2007) e Pastuszak(2008) terão seu resultados normalizados de acordo com Altera (2012) que considera uma célula lógica da Stratix2 equivalente a 1,3 LUTs da Virtex4, esta também equivalente a uma LUT da família Virtex2. A

Tabela 6.2 apresenta os resultados de síntese lógica para as famílias Spartan3 e Virtex2.

Tabela 6.2 – Resultados de Síntese Lógica para Spartan3 e Virtex2

Módulo	Spartan3		Virtex2	
	LUTs	F(MHz)	LUTs	F(MHz)
Gerenciador + EXPG	269	154,1	253	202,3
CAVLC	4185	86,6	4090	118,6
CABAC	4995	96,2	5068	120,7
Buffer de Entrada	1877	151,7	1484	188,3
Montador Final	786	142,2	777	183,2

Os trabalhos de Rahman mostraram arquiteturas para CAVLC focadas na menor área possível, 2,6x e 4,4x menor que a área do CAVLC deste trabalho. No entanto, a limitação de desempenho (30 quadros CIF) que elas possuem, mesmo que fossem sintetizadas para uma plataforma Virtex5 impossibilitariam que elas suportassem quadros 1920x1080 a 30fps.

O trabalho de Silva (2007) apresenta uma arquitetura de EXPG e CAVLC para Stratix2. A área de 3500 células lógicas normalizada equivaleria a 4550 LUTs. Ocupando uma área 1,12x menor que o Gerenciador + EXPG, CAVLC e Montador final, o trabalho do autor precisa de 100MHz para atender 1920x1080 a 30 fps ao passo que a arquitetura proposta é capaz de processar 1920x1080 a 30 fps a uma frequência de 65 MHz.

O trabalho de Pastuszak (2008) apresenta 5 diferentes soluções para o CABAC. A solução CABAC + CAVLC apresentada possui uma área normalizada de 5092 LUTs, sendo 1,84x menor que Gerenciador + EXPG +CAVLC + CABAC deste trabalho. No modo CAVLC ocorre o processamento de 1920x1080 30fps em uma frequência de ~100MHz. O autor não deixa claro que a arquitetura do CABAC em FPGA suporta Full HD quando opera nesta mesma frequência.

A Tabela 6.3 apresenta os resultados resumidos de área, frequência, resolução suportada, arquitetura implementada e o dispositivo utilizado por cada um dos autores analisados anteriormente.

Tabela 6.3 – Resultados comparados com a Literatura

Autor	FPGA	Area (LUT)	Freq (MHz)	FPS Resolução	Arquitetura
Este trabalho	Virtex 5	6868	165	60 Full HD	MB Buffer + EXPG + CAVLC + CABAC + Montador Final
	Virtex II	11672	118,6	30 Full HD	
	Spartan3	12112	86,6	30 Full HD	
Rahman (2006)	Virtex II	1533	60	30 CIF	CAVLC
Rahman (2007)	Virtex II	925	52,8	30 CIF	CAVLC
Silva (2007)	Stratix II*	4550*	117,6	30 Full HD	EXPG + CAVLC
Pastuszak (2008)	Stratix II*	5092*	125	30 HD	Entropia completa*
Albanese (2010a)	Spartan3	3447	63	30 Full HD	CAVLC
Albanese (2010b)	Spartan3	2200	63	30 Full HD	CAVLC
Ramos (2010)	Virtex5	10791	180	60 Full HD	MB Buffer + CAVLC

Os trabalhos de Albanese mostram arquiteturas de CAVLC com economia de área e que atendem alta definição. O primeiro trabalho (Albanese,2010a) possui uma área 1,18x menor que o CAVLC proposto e 1,85x menor se considerado o segundo trabalho (Albanese,2010b). Nestes trabalhos não são considerados as memórias com os dados de vizinhança, nem uma estrutura de endereçamento o que é um problema visto que o acesso e cálculo de endereçamento não é trivial, sendo que nada é tratado a respeito. Outro aspecto não abordado por ambos é a arquitetura do montador. Mesmo que Albanese apresente uma aceleração de hardware nos momentos de pré-codificação e codificação, ela pode ser inútil se a montagem tornar-se o gargalo da entropia. Considerando que estas arquiteturas fossem implementadas em uma Virtex5, dada a limitação da frequência a 63MHz na Spartan3, supõem-se que estas não seriam capazes de processar 1280x1080 a 60fps.

O trabalho de Ramos (2010) mostra o melhor desempenho dentre as arquiteturas comparadas, processando até 115 quadros Full HD, entretanto com um custo muito alto em área, ocupando 10791 LUTs e também utilizando 39 BRAMs. A arquitetura do CAVLC deste trabalho é 4,25x menor que o de Ramos (2010) e utiliza somente 2 BRAMs e o codificador entrópico inteiro é 1,5x menor usando 9 BRAMs, possuindo desempenhos muito similares de compressão.

7 CONCLUSÃO

Este trabalho apresentou uma descrição geral do padrão H.264/AVC com enfoque no estudo, desenvolvimento e integração das arquiteturas dedicadas dos módulos de Entropia. Foram apresentados os algoritmos entrópicos Exponencial Golomb, Código de Tamanho Variável Adaptativo ao Contexto (CAVLC) e Código Aritmético Binário Adaptativo ao Contexto (CABAC), assim como o desenvolvimento e a integração deles em hardware. As arquiteturas foram desenvolvidas e descritas em linguagem VHDL, visando a seguir a síntese em FPGA.

A principal contribuição deste trabalho foi o desenvolvimento de arquiteturas de codificadores de entropia e de um montador final e a integração destas com os módulos que tratam do CABAC. A integração das mesmas requereu o desenvolvimento de arquitetura de controle nova, de forma a compor assim um bloco de hardware para a entropia completa para o codificador H.264/AVC.

As arquiteturas para o EXPG e CAVLC implementadas nesta dissertação estão voltadas para um alto desempenho. As opções selecionadas para o CABAC na incorporação do bloco de entropia foram as mais eficientes de acordo com os próprios autores. O Montador Final é capaz de ordenar elementos sintáticos restringidos pela norma brasileira para os perfis *baseline* e *main* do codificador H.264/AVC. Os resultados obtidos nas simulações provam que o bloco entrópico é capaz de suportar uma compressão de vídeos de 1080p à taxa de 60 quadros por segundo, o equivalente ao nível 4.2 da norma. Sendo que o CAVLC, demanda no pior dos casos, um processamento de 277 ciclos por MB, tendo um poder de processamento igual a Silva (2007), Pastuszak (2008), com uma frequência de operação menor. Os resultados de síntese de todo o bloco mostram que este ocupa uma área de 4304 Slice registers, 6868 Slice LUTs do dispositivo Virtex5 que é menor que a arquitetura do CAVLC apresentada por Ramos (2010) e possui a mesma capacidade de processamento de MBs.

Os trabalhos futuros relacionados com esta dissertação que ainda podem ser alvo de estudos: a integração deste Módulo de Entropia aos demais módulos do codificador, com simulação e avaliação dos resultados de área, energia e desempenho para um codificador completo de vídeo. O estudo e desenvolvimento de uma nova integração entre os codificadores entrópicos considerando o compartilhamento de estruturas e analisando o impacto destas mudanças nos resultados de desempenho e área. Por fim o desenvolvimento de arquiteturas alternativas para a aceleração dos codificadores entrópicos (CAVLC e CABAC), visando o tratamento da dependência de dados com o menor impacto em área, para prover suporte a níveis mais elevados do padrão H.264/AVC ou outras extensões que exigem um *throughput* maior como o SVC e o MVC.

REFERÊNCIAS

ABNT NBR 15602-1. **Televisão digital terrestre — Codificação de vídeo, áudio e multiplexação**. Parte 1: Codificação de vídeo. 2008. Disponível em: http://www.dtv.org.br/download/pt-br/ABNTNBR15602-1_2007Vc_2008.pdf

ALBANESE, L. F.; LICCIARDO, G. D. **High Speed CAVLC Encoder Suitable for Field Programmable Platforms**. International Conference on Signals and Electronic Systems (ICSES), p. 327-330, Sep 2010a.

ALBANESE, L. F.; LICCIARDO, G. D. **An Area Reduced Design of the Context-Adaptive Variable-Length Encoder Suitable for Embedded Systems**. I/V Communications and Mobile Network / 5th International Symposium on Visual Computing (ISVC), p.1-4, Sep-Oct 2010b.

ALTERA, **Stratix II vs. Virtex-4 Density Comparison,** Disponível em: http://www.altera.com/literature/wp/wpstxiixlnx.pdf. Acessado em Jan/2012.

CORRÊA, G. R. **Estudo e Desenvolvimento de Heurísticas e Arquiteturas de Hardware para Decisão Rápida do Modo de Codificação de Bloco para o Padrão H.264/AVC**. 2010. 95 f. Dissertação (Mestrado em Computação) – Instituto de Informática, UFRGS, Porto Alegre.

DINIZ, C. M. **Arquitetura de Hardware Dedicada para a Predição Intra-Quadro em Codificadores do Padrão H.264/AVC de Compressão de Vídeo.** 2009. 96 f. Dissertação (Mestrado em Computação) – Instituto de Informática, UFRGS, Porto Alegre.

GHANBARI, M. **Standard Codecs: Image Compression to Advanced Video Coding**. 3rd ed. United Kingdom: The Institution of Electrical Engineers, 2011.

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

ITU-T H.264 | ISO/IEC 14496-10:2009 Recommendation. **Advanced Video Coding for generic audio-visual services**, March 2009.

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

MARPE. et al. **Context-Based Adaptative Binary Arithmetic Coding in the H.264/AVC Video Compression Standard**. IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, 2003.

MARTINS, A. L. M. **Projeto da Arquitetura de Hardware para Binarização e Modelagem de Contextos para o CABAC do Padrão de Compressão de Vídeo H.264/AVC**. 2011. 95 f. Dissertação (Mestrado em Computação) – Instituto de Informática, UFRGS, Porto Alegre.

MERKLE, P.; SMOLIC, A.; MÜLLER K. **Efficient Prediction Structures for Multiview Video Coding.** IEEE Transactions on Circuits and Systems for Video Technology, VOL. 17, NO.11, NOV. 2007.

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

RAHMAN, C.A.; BADAWY, W. **CAVLC Encoder Design for Real-Time Mobile Video Applications**. IEEE Trans. on Circuits and Systems II - Expr. Briefs, vol. 54, pp. 873-877, 2007.

RAHMAN, C.A.; BADAWY, W. **An Area Efficient Real-time CAVLC IP-Block towards the H.264/AVC Encoder**. IEEE Workshop on Signal Processing Systems (SIPS), pp. 368-371, 2006.

RAMOS, F. L. L. **Arquitetura para o Algoritmo CAVLC de Codificação de Entropia segundo o Padrão H.264/AVC.** 2010. 99 f. Dissertação (Mestrado em Computação) – Instituto de Informática, UFRGS, Porto Alegre.

RICHARDSON, I. **The H.264/AVC Advanced Video Compression Standard**. 2nd ed. Chichester: John Wiley and Sons, 2010.

ROSA, V. S. **Arquiteturas de Hardware Dedicadas para Codificadores de Vídeo H.264 – Filtragem de Efeitos de Bloco e Codificação Aritmética Binária Adaptativa a Contexto**. 2010. 171 f. Tese (Doutorado em Computação) – Instituto de Informática, UFRGS, Porto Alegre.

SALOMON, D; GIOVANNI, M. **Handbook of Data Compression**.5th ed. Springer, 2010.

SHI, Y. Q.; SUN, H. **Image and Video Compression for Multimedia Engineering.**2nd ed. CRC Press, 2007.

SILVA, T.; VORTMANN, J.; AGOSTINI L.; BAMPI S.; SUSIN A. **FPGA Based Design of CAVLC and Exp-Golomb Coders for H.264/AVC Baseline Entropy Coding.** III Southern Conference on Programmable Logic **(**SPL), 2007.

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

XILINX. **Xilinx: Virtex-5 FPGA User Guide**. Disponível em: <http://www.xilinx.com/support/documentation/user_guides/ug190.pdf>. Acesso em: Mar. 2012.

ANEXO A

ESPECIFICAÇÃO DO BITSTREAM PARA O CODIFICADOR H.264 HDL – PROJETO REDE H.264 SBTVD

Este documento visa guiar todas as simplificações utilizadas para implementação dos codificadores compatíveis com o padrão H.264/AVC no escopo do projeto FINEP REDE H.264 SBTVD em desenvolvimento na UFRGS até 2012. Ele foi criado com o objetivo de definir os elementos sintáticos presentes no bitstream gerado pelas 3 etapas de desenvolvimento do codificador a ser desenvolvido em VHDL, como definidas na reunião geral do projeto realizada em Junho de 2009, em Porto Alegre, RS. Tendo definidas todas as ferramentas de codificação (*coding features*) compatíveis com o padrão (seções 1 a 3), todos elementos sintáticos a serem suportados foram listados na tabela da seção 4. Para alguns elementos sintáticos que possuem valores constantes, a tabela possui uma sugestão de valor. A seção 5 apresenta comentários que justificam a escolha do número de bits de entrada e o valor sugerido para alguns dos elementos sintáticos em questão. Este documento pode ser editado por qualquer desenvolvedor da equipe do Codificador HDL neste projeto.

Histórico do documento

Data	**Revisão**	**Descrição**	**Responsável na UFRGS**
23/08/2010	1.0	Revisão inicial	Cláudio Diniz
21/10/2010	1.1	Versão definitiva pré-reunião de 22/10/2010. Correção de alguns erros e revisão do texto.	Cláudio Diniz, Cristiano Thiele
02/09/2011	1.2	Restrições de alguns elementos sintáticos (decisões de projeto).	Cláudio Diniz, Cristiano Thiele

1. Primeira versão do codificador (básico)

- Espaço de cores YCbCr, com subamostragem 4:2:0 e 8 bits por amostra;
- Predição intra-quadro: suporta Luma_16x16, Luma_4x4 e Chroma_8x8 (não suporta I_PCM);

- Nenhum tipo de predição inter-quadros é utilizada (somente quadros I).
- Transformadas DCT 4x4, Hadamard 4x4 e Hadamard 2x2, diretas e inversas (sem suporte a transformadas 8x8);
- Quantizações diretas e inversas;
- Gera *bitstream* compatível com o Perfil *Main*, Nível 4.0 (HD1080);
- Modo de entropia simplificado (*entropy_coding_mode_flag* = 0): códigos de comprimento fixo e variável, códigos Exp-Golomb, CAVLC;
- Não inclui *Deblocking Filter* (*deblocking_filter_control_present_flag* = 0)
- Suporte a vídeo progressivo somente (não inclui suporte a vídeo entrelaçado, ou MBAFF – *Macroblock Adaptive Frame-Field*);
- Utiliza um *slice* por quadro;

2. Segunda versão do codificador

- Espaço de cores YCbCr, com sub-amostragem 4:2:0 e 8 bits por amostra;
- Predição intra-quadro: suporta Luma_16x16, Luma_4x4 e Chroma_8x8 (não suporta I_PCM);
- Predição inter-quadros: suporte a quadros P somente. Quarter-pixel. Tamanho de bloco fixo.
- Transformadas DCT 4x4, Hadamard 4x4 e Hadamard 2x2, diretas e inversas (sem suporte a transformadas 8x8);
- Quantizações diretas e inversas;
- Gera *bitstream* compatível com o Perfil *Main*, Nível 4.0 (HD1080);
- Modo de entropia simplificado (*entropy_coding_mode_flag* = 0): códigos de comprimento fixo e variável, códigos Exp-Golomb, CAVLC;
- Inclui Deblocking Filter (*deblocking_filter_control_present_flag* = 1)
- Suporte a vídeo progressivo somente (não inclui suporte a vídeo entrelaçado, ou MBAFF – *Macroblock Adaptive Frame-Field*);
- Utiliza um *slice* por quadro;

3. Terceira versão do codificador

- Espaço de cores YCbCr, com sub-amostragem 4:2:0 e 8 bits por amostra;
- Predição intra-quadro: suporta Luma_16x16, Luma_4x4 e Chroma_8x8 (não suporta I_PCM);
- Predição inter-quadros: suporte a quadros P. *Quarter-pixel*. Tamanho de bloco fixo.
- Transformadas DCT 4x4, Hadamard 4x4 e Hadamard 2x2, diretas e inversas,
- Quantizações diretas e inversas;

- Gera *bitstream* compatível com o Perfil *High*, Nível 4.0 (HD1080);
- Modo de entropia avançado (*entropy_coding_mode_flag* = 1): códigos de comprimento fixo, CABAC;
- Inclui *Deblocking Filter* (*deblocking_filter_control_present_flag* = 1)
- Suporte a vídeo progressivo somente (não inclui suporte a vídeo entrelaçado, ou MBAFF – *Macroblock Adaptive Frame-Field*);
- Utiliza um *slice* por quadro;

4. Tabela de especificação do bitstream

A tabela abaixo detalha a especificação do bitstream, contendo todos os elementos sintáticos que definem a sintaxe de bitstream compatível com o padrão H.264/AVC de acordo com a seção 7 da recomendação H.264 da ITU-T (03/2005) e da norma brasileira de televisão digital ABNT NBR 15602-1:2007. Cada elemento pertence a uma camada do bitstream, que é apontada na segunda coluna da tabela, como segue:

NAL - *Network Abstraction Layer*

SPS – *Sequence Parameter Set*

PPS – *Picture Parameter Set*

VUI – *Video Usability Information*

HRD – *Hypotetical Reference Decoder*

SEI – *Suplemental Enhancement Information*

SEIP - *Suplemental Enhancement Information Message*

SH – *Slice Header*

RPLR – *Reference Picture List Reordering*

PWT – *Prediction Weight Table*

DRPM – *Decoded Reference Picture Marking*

SD – *Slice Data*

MB – *Macroblock Layer*

SUBMB – *Sub-macroblock Layer*

RES – *Residual Layer*

A terceira coluna especifica o número de bits de entrada necessários a cada elemento sintático para definir a entrada do módulo “montador” do bitstream. Este número de bits de entrada nem sempre é igual ao número de bits de saída, devido a mapeamentos e aos métodos de codificação de entropia aplicados.

Nas últimas colunas, existe uma sugestão de valores a serem usados para cada elemento sintático (os valores estão em decimal), devido ao fato de que diversos elementos sintáticos serão fixados em um único valor, para simplificação da implementação do codificador. É importante comentar que o conjunto de ferramentas do codificador não é normatizado, mas somente o bitstream deve ser compatível com a norma H.264 e ABNT. Existem alguns casos especiais:

n : não existe um valor constante, seu valor é estipulado pelos demais módulos do codificador para o codificador de entropia/montador.

Não existe: existem casos em que alguns elementos sintáticos não aparecem no bitstream, pois sua presença está condicionada ao valor de outros elementos sintáticos, que neste caso pode possui valor fixo. Por exemplo: se *entropy_coding_mode_flag* = 0 (na primeira versão do codificador), o *Slice Header* (SH) não contém o elemento sintático *cabac_init_idc*.

Este documento deve ser utilizado juntamente com a norma ITU e a norma ABNT:

H.264: *Advanced video coding for generic audiovisual services* (03/2005). http://www.itu.int/rec/T-REC-H.264/en.

ABNT NBR 15602-1. Televisão digital terrestre — Codificação de vídeo, áudio e multiplexação. Parte 1: Codificação de vídeo. http://www.dtv.org.br/download/pt-br/ABNTNBR15602-1_2007Vc_2008.pdf

Elemento sintático	**Camada**	**Bits de entrada**	**Valores sugeridos por versão do codificador (em decimal)**		
			Versão 1	**Versão 2**	**Versão 3**
forbidden_zero_bit	NAL	1	0	0	0
nal_ref_idc	NAL	2	1	0 ou 1	0 ou 1
nal_unit_type	NAL	5	1, 5, 7, 8	1, 5, 7, 8	1, 5, 7, 8
rbsp_byte[i]	NAL	8	n		
emulation_prevention_three_byte	NAL	8	0x03		
profile_idc	SPS	1	0 (77)	0 (77)	1 (100)
constraint_set0_flag	SPS	1	0	0	0
constraint_set1_flag	SPS	1	1	1	0
constraint_set2_flag	SPS	1	0	0	0
constraint_set3_flag	SPS	1	0	0	0
reserved_zero_4bits	SPS	4	0	0	0
level_idc	SPS	8	40	40	40
seq_parameter_set_id	SPS	5	0	0	0
chroma_format_idc	SPS	2	Não existe	Não existe	1
residual_colour_transform_flag	SPS	1	Não existe		
bit_depth_luma_minus8	SPS	3	Não existe	Não existe	0
bit_depth_chroma_minus8	SPS	3	Não existe	Não existe	0
qpprime_y_zero_transform_bypass_flag	SPS	1	Não existe	Não existe	0
seq_scaling_matrix_present_flag	SPS	1	Não existe	Não existe	0
log2_max_frame_num_minus4	SPS	2	3	3	3
pic_order_cnt_type	SPS	2	0	0	0
log2_max_pic_order_cnt_lsb_minus4	SPS	4	8	8	8

num_ref_frames	SPS	3	0	1	1
gaps_in_frame_num_value_allowed_flag	SPS	1	0	0	0
pic_width_in_mbs_minus1	SPS	7	n	n	n
pic_height_in_map_units_minus1	SPS	7	n	n	n
frame_mbs_only_flag	SPS	1	1	1	1
direct_8x8_inference_flag	SPS	1	1	1	1
frame_cropping_flag	SPS	1	0	0	1
frame_crop_left_offset	SPS	8	Não existe	Não existe	0
frame_crop_right_offset	SPS	8	Não existe	Não existe	0
frame_crop_top_offset	SPS	8	Não existe	Não existe	0
frame_crop_bottom_offset	SPS	8	Não existe	Não existe	8
vui_parameters_present_flag	SPS	1	0	0	1
aspect_ratio_info_present_flag	VUI	1	Não existe	Não existe	1
aspect_ratio_idc	VUI	8	Não existe	Não existe	1
overscan_info_present_flag	VUI	1	Não existe	Não existe	0
video_signal_type_present_flag	VUI	1	Não existe	Não existe	1
video_format	VUI	3	Não existe	Não existe	2
video_full_range_flag	VUI	1	Não existe	Não existe	0
colour_description_present_flag	VUI	1	Não existe	Não existe	1
colour_primaries	VUI	8	Não existe	Não existe	1
transfer_characteristics	VUI	8	Não existe	Não existe	1
matrix_coefficients	VUI	8	Não existe	Não existe	1
chroma_loc_info_present_flag	VUI	1	Não existe	Não existe	1
chroma_sample_loc_type_top_field	VUI	1	Não existe	Não existe	0
chroma_sample_loc_type_bottom_field	VUI	1	Não existe	Não existe	0
timing_info_present_flag	VUI	1	Não existe	Não existe	1
num_units_in_tick	VUI	32	Não existe	Não existe	1 001
time_scale	VUI	32	Não existe	Não existe	30 000
fixed_frame_rate_flag	VUI	1	Não existe	Não existe	0
nal_hrd_parameters_present_flag	VUI	1	Não existe	Não existe	0
vcl_hrd_parameters_present_flag	VUI	1	Não existe	Não existe	0
pic_struct_present_flag	VUI	1	Não existe	Não existe	0
bitstream_restriction_flag	VUI	1	Não existe	Não existe	0
rbsp_stop_one_bit	SPS	1	1		
rbsp_alignment_zero_bit	SPS	1	0		
pic_parameter_set_id	PPS	5	0	0	n
seq_parameter_set_id	PPS	5	0	0	0
entropy_coding_mode_flag	PPS	1	0	0	1

pic_order_present_flag	PPS	1	0	0	0
num_slice_groups_minus1	PPS	1	0	0	0
num_ref_idx_l0_active_minus1	PPS	2	0	0	0
num_ref_idx_l1_active_minus1	PPS	2	0	0	0
weighted_pred_flag	PPS	1	0	0	0
weighted_bipred_idc	PPS	2	0	0	0
pic_init_qp_minus26	PPS	5	n	n	n
pic_init_qs_minus26	PPS	5	n	n	n
chroma_qp_index_offset	PPS	5	0	0	0
deblocking_filter_control_present_flag	PPS	1	1	1	1
constrained_intra_pred_flag	PPS	1	0	0	0
redundant_pic_cnt_present_flag	PPS	1	0	0	0
transform_8x8_mode_flag	PPS	1	0	0	0
pic_scaling_matrix_present_flag	PPS	1	0	0	0
second_chroma_qp_index_offset	PPS	5	Não existe	Não existe	0
rbsp_stop_one_bit	PPS	1	1	1	1
rbsp_alignment_zero_bit	PPS	1	0	0	0
ff_byte	SEI	8	Não existe	Não existe	255
last_payload_type_byte	SEI	8	Não existe	Não existe	n
last_payload_size_byte	SEI	8	Não existe	Não existe	n
pic_struct	SEIM	4	Não existe	Não existe	n
clock_timestamp_flag[i]	SEIM	1	Não existe	Não existe	n
ct_type	SEIM	2	Não existe	Não existe	n
nuit_field_based_flag	SEIM	1	Não existe	Não existe	n
counting_type	SEIM	5	Não existe	Não existe	n
full_timestamp_flag	SEIM	1	Não existe	Não existe	n
discontinuity_flag	SEIM	1	Não existe	Não existe	n
cnt_dropped_flag	SEIM	1	Não existe	Não existe	n
n_frames	SEIM	8	Não existe	Não existe	n
seconds_value	SEIM	6	Não existe	Não existe	n
minutes_value	SEIM	6	Não existe	Não existe	n
hours_value	SEIM	5	Não existe	Não existe	n
seconds_flag	SEIM	1	Não existe	Não existe	n
minutes_flag	SEIM	1	Não existe	Não existe	n
hours_flag	SEIM	1	Não existe	Não existe	n
time_offset	SEIM	24	Não existe	Não existe	n
bit_equal_to_one	SEIP	1	Não existe	Não existe	1
bit_equal_to_zero	SEIP	1	Não existe	Não existe	0

first_mb_in_slice	SH	1	0	0	0
slice_type	SH	4	n	n	n
pic_parameter_set_id	SH	5	0	0	n
frame_num	SH	7	n	n	n
idr_pic_id	SH	1	0	0	0
pic_order_cnt_lsb	SH	12	n	n	n
num_ref_idx_active_override_flag	RPLR	1	Não existe	0	n
num_ref_idx_l0_active_minus1	RPLR	2	Não existe	Não existe	1
num_ref_idx_l1_active_minus1	RPLR	2	Não existe	Não existe	1
ref_pic_list_reordering_flag_l0	RPLR	1	Não existe	0	0
no_output_of_prior_pics_flag	DRPM	1	0	0	0
long_term_reference_flag	DRPM	1	0	0	0
adaptive_ref_pic_marking_mode_flag	DRPM	1	0	0	0
cabac_init_idc (CABAC)	SH	2	Não existe	Não existe	n
slice_qp_delta	SH	6	n	n	n
disable_deblocking_filter_idc	SH	2	1	0	0
slice_alpha_c0_offset_div2	SH	4	Não existe	n	n
slice_beta_offset_div2	SH	4	Não existe	n	n
cabac_alignment_one_bit (CABAC)	SD	1	Não existe		1
mb_skip_run (Exp-Golomb)	SD	13	Não existe	n	Não existe
mb_skip_flag(CABAC)	SD		Não existe		n
end_of_slice_flag (CABAC)	SD	1	Não existe		n
mb_type	MB	5	n	n	n
prev_intra_4x4_pred_mode_flag [i]	MB	1	n	n	n
rem_intra_4x4_pred_mode [i]	MB	3	n	n	n
intra_chroma_pred_mode	MB	2	n	n	n
coded_block_pattern	MB	6	n	n	n
mb_qp_delta	MB	6	n	n	n
ref_idx_l0	MB	6	Não existe	n	n
mvd_l0	MB	10	Não existe	n	n
coeff_token	RES	variável	n	n	n
trailing_ones_flag	RES	variável	n	n	n
level_prefix	RES	variável	< 15	< 15	< 15
level_suffix	RES	variável	n	n	n
total_zeros	RES	variável	n	n	n
run_before	RES	variável	n	n	n
coded_block_flag (CABAC)	RES	1	n	n	n
significant_coeff_flag (CABAC)	RES	1	n	n	n

last_significant_coeff_flag (CABAC)	RES	1	n	n	n
coeff_abs_level_minus1 (CABAC)	RES	16	n	n	n
coeff_sign_flag[i] (CABAC)	RES	1	n	n	n

5. Comentários sobre a escolha de valores constantes para alguns elementos sintáticos

- Existe uma restrição no intervalo máximo de pontos de acesso aleatório (RAP – *Random Access Point*) de 5s, definido pela ABNT. Isto influi no tamanho do GOP (*Group of Pictures*), que é o definido pelo conjunto de quadros P e B entre dois quadros IDR na ordem de exibição. Já que o valor de *frame_num* é colocado em 0 a cada frame IDR, e o maior valor de frame_num permitido é 150, colocamos a restrição em *frame_num* < 128, ou seja, frame_num é representado com 7 bits. Logo, o valor de *log2_max_frame_num_minus4*, que controla o número de bits de *frame_num* é de 3 (decimal), ou seja, pode ser representado com 2 bits de entrada.
- *num_ref_frames* deve ficar entre 0 e *MaxDpbSize*, onde *MaxDpbSize* é igual a: Min(1024 * MaxDPB / (*PicWidthInMbs* * *FrameHeightInMbs* * 384), 16) onde MaxDPB para o level 4.0 é igual a 12288.0 (Table A-1 anexo A). Faendo o cálculo para o vídeo HD1080 (1920 x 1088)
  PicWidthInMbs = 120
  FrameHeightInMbs = 68
  Assim: Min (1024 * 12288 / (120 * 68 * 384) , 16)
  Min (4 , 16) = 4 quadros de referência
- A existência dos elementos *mb_adaptive_frame_field_flag*, *field_pic_flag* e *bottom_pic_flag* e seus valores dependem do suporte, ou não, ao vídeo entrelaçado (no caso de *frame_mbs_only_flag* = 0).
- *max_long_term_frame_idx_plus1* pode ter o valor do máximo número de quadros de referência, ou seja, 4 quadros, de acordo com o level 4.
- *mb_skip_run* pode ter o valor de, no máximo, o número de macroblocos em um quadro, ou seja, 8192 (pelo level 4.0), ou 2^{13}.
- *disable_deblocking_filter_idc* = 1 desabilita o *deblocking filter*, então este valor é usado na primeira versão do codificador. Um valor diferente deste habilita o *deblocking filter*, então deixa-se parametrizado para as próximas duas versões do codificador. Deste modo, *slice_alpha_c0_offset_div2* e *slice_beta_offset_div2* podem ser usados somente nas versões 2 e 3.
- A presença de uma mensagem SEI depende da aplicação e não é determinada pela recomendação H.264. Verificar, então, se a aplicação requer a transmissão da mensagem SEI. As mensagens SEI que devem ser obrigatoriamente tratadas pelo DECODIFICADOR, segundo a norma ABNT, são as mensagens ***pic_timing(payloadSize)*** e ***pan_scan_rect(payloadSize)***.