UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ALONSO AYMONE DE ALMEIDA SCHMIDT

INTEGRAÇÃO DO CABAD AO DECODIFICADOR DE VÍDEO H.264/AVC PARA O SBTVD

Porto Alegre
2011

ALONSO AYMONE DE ALMEIDA SCHMIDT

INTEGRAÇÃO DO CABAD AO DECODIFICADOR DE VÍDEO H.264/AVC PARA O SBTVD

Projeto de Diplomação apresentado ao Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul como parte dos requisitos para a obtenção do título de Engenheiro Eletricista.

ORIENTADOR: Prof. Dr. Altamiro Amadeu Susin

CO-ORIENTADOR: Dr. André Borin Soares

Porto Alegre
2011

ALONSO AYMONE DE ALMEIDA SCHMIDT

INTEGRAÇÃO DO CABAD AO DECODIFICADOR DE VÍDEO H.264/AVC PARA O SBTVD

Este Projeto foi julgado adequado para a obtenção dos créditos da Disciplina Projeto de Diplomação do Departamento de Engenharia Elétrica e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____
Prof. Dr. Altamiro Amadeu Susin, UFRGS
Doutor pelo Institut National Polytechnique de Grenoble, França

Banca Examinadora:

Prof. Dr. Altamiro Amadeu Susin, UFRGS
Doutor pelo Institut National Polytechnique de Grenoble, França

Prof. Dr. Marcelo Götz, UFRGS
Doutor pela Universität Paderborn, Alemanha

Prof. Dr. Tiago Roberto Balen, Unilasalle
Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil

Chefe do DELET: _____
Prof. Dr. Altamiro Amadeu Susin

Porto Alegre, julho de 2011.

DEDICATÓRIA

Dedico este trabalho ao meu pai, por ter me inspirado entusiasmo por tecnologia e motivação para buscar oportunidades longe de casa, e à minha mãe por sempre estar afetuosamente a meu favor.

AGRADECIMENTOS

Agradeço primeiramente o meu orientador Prof. Dr. Altamiro Susin pela oportunidade concedida de iniciação científica dentro do projeto extenso Rede H.264, no qual se encontra tantos desafios e problemas instigantes para resolver.

Agradeço a minha família, pela educação em casa e por todo o suporte que me possibilitou ingressar na UFRGS e cursar Engenharia Elétrica. Sem a base familiar e todo o apoio dado, não teria sido possível me empenhar com dedicação para cumprir todos os meus objetivos. Em especial, agradeço a minha vó Circe Aymone por emprestar o apartamento que possui em Porto Alegre e ajudar-me a cobrir gastos para manter-me.

Agradeço os meus amigos, tanto os que conheci na minha cidade natal na fronteira, quanto na praia de Xangri-lá. Agradeço também os amigos que fiz no decorrer do curso de Engenharia Elétrica pelos trabalhos em grupos, trocas de ideias e parcerias na resolução de problemas. Agradeço amigos de outros cursos, por boas festas para descontrair e companhia. Todos meus amigos foram muito importantes pelo prestígio e por acreditarem na minha competência.

Agradeço os meus colegas e ex-colegas de laboratório: Márlon que me indicou o LaPSI, Dr. Marcelo Negreiros, Dra. Letícia Guimarães, M.Sc. Alexsandro Bonatto, M.Sc. Fábio Pereira, Dr. Gilberto Marchioro, M.Sc. Gustavo Ilha, Vinicius Souza, Bruno Freitas, Guilherme Chagas, Fábio Vidor, Marcos Carvalho, Jeffrei Moreira, Dierles Éneas e os demais com quem convivi. Em especial agradeço aqui o meu co-orientador Dr. André Borin por me guiar na realização do meu trabalho, pelas revisões e ainda por todas as orientações durante minhas atividades de iniciação científica. Tive troca de idéias e experiências com as pessoas com quem convivi no laboratório, o que culminou em aquisição de competências profissionais para desenvolver o projeto de diplomação.

Eu agradeço a Universidade Federal do Rio Grande do Sul, por todos os serviços e benefícios que oferece, tanto no âmbito da Educação de nível superior de qualidade quanto pelas atividades culturais. Agradeço os professores do Departamento de Engenharia Elétrica, Instituto de Matemática, Instituto de Física, Instituto de Informática e outros. Agradeço também os demais funcionários.

RESUMO

Este trabalho apresenta a integração do CABAD (*Context-based Adaptive Binary Arithmetic Decoder*) ao decodificador de vídeo H.264/AVC em *hardware* para o Sistema Brasileiro de Televisão Digital. Na arquitetura do *hardware*, o CABAD é um submódulo do *parser*. Este é responsável por controlar o fluxo dos elementos sintáticos que devem ser decodificados para fornecer parâmetros que demais módulos do decodificador vídeo necessitam para atuar na reconstrução das imagens. Estes módulos são o MC (compensação de movimento), o *Intra* (predição espacial), a IQIT (transformada e quantização inversas) entre outros. A CABAC (*Context-based Adaptive Binary Arithmetic Coding*) comprime dados ao combinar codificação aritmética binária com um modelamento de contextos baseado em valores de elementos sintáticos próximos previamente codificados e em estatísticas locais, otimizando assim estimativas de probabilidades. O algoritmo do CABAD envolve essencialmente os processos de seleção de contextos, decodificação aritmética binária e de-binarização para obtenção de valores de elementos sintáticos. O módulo-alvo de integração exige um vetor de sinais de entrada para realizar os processos do algoritmo e um controle externo para sua inicialização, habilitação e recuperação de valores decodificados. Estas necessidades promovem o desenvolvimento de um gerenciador de fluxo de *bits*, um gerenciador de elementos sintáticos, um construtor de blocos de coeficiente de resíduo e um controle sincronizador. Após integração dos módulos desenvolvidos, o *parser* é verificado funcionalmente com fluxo de *bits* de vídeo gerado com um codificador em *software* de referência, comparando-se os valores esperados com os sinais obtidos através de simulação. Síntese do *parser* com a integração do CABAD para FPGA atinge frequência máxima de 56 MHz, sendo que para 50 MHz é verificado que a taxa de processamento de dados é suficiente para processar vídeo HD 720p.

Palavras-chave: H.264/AVC, CABAD, VHDL, FPGA, SBTVD.

ABSTRACT

This work presents the integration of CABAD (Context-based Adaptive Binary Arithmetic Decoder) into the H.264/AVC hardware video decoder for the Brazilian Digital Television System (SBTVD). In the hardware architecture, CABAD is a sub-module of the parser. The latter is responsible for controlling the flow of the syntactic elements that must be decoded to supply parameters needed by the other video decoder modules in order to reconstruct the images. Those modules are the MC (Motion Compensation), the Intra (spatial prediction), the IQIT (Inverse Quantization and Inverse Transform) among others. CABAC (Context-based Adaptive Binary Arithmetic Coding) compress data by combining binary arithmetic coding with context modeling based on syntactic elements values nearby that were previously decoded and on local statistics, thus optimizing probability estimation. The algorithm of CABAD comprehends essentially the processes of context selection, binary arithmetic decoding and de-binarization. The target-module for integration requires a vector of input signals to accomplish the processes of the algorithm and external control for its initialization, enabling and for retrieving decoded values. Those requisites promotes the development of a bit-stream handler, a syntactic elements manager, a residual coefficients block builder and a synchronizing control. After integration of the developed modules, the parser's operation is verified with a bit-stream of video generated by a reference software encoder, comparing expected values with signals obtained through simulation. FPGA synthesis of the parser after the integration of CABAD reaches maximum frequency of 56 MHz, and at 50 MHz it is verified that the data throughput is sufficient to process HD 720p video.

Keywords: H.264/AVC, CABAD, VHDL, FPGA, SBTVD.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ASIC	*Application Specific Integrated Circuit*
AVC	*Advanced Video Coding*
CABAC	*Context-based Adaptive Binary Arithmetic Coding*
CABAD	*Context-based Adaptive Binary Arithmetic Decoder*
CAD	*Computer Aided Design*
CAVLC	*Context-based Adaptive Variable Lenght Coding*
CAVLD	*Context-based Adaptive Variable Lenght Decoder*
DELET	Departamento de Engenharia Elétrica
DRAM	*Dynamic Random Acess Memory*
DCT	*Discrete Cosine Transform*
FIFO	*First In, First Out*
FPGA	*Field Programmable Gate Array*
HD	*High Definition*
ITU	*International Telecommunication Union*
LaPSI	Laboratório de Processamento de Sinais e Imagens
LPS	*Least Probable Symbol*
LUT	*Look-up Table*
MBAFF	*Macroblock-adaptive frame-field*
MPS	*Least Probable Symbol*
NAL	*Network Abstraction Layer*
PIXEL	*Picture element* (Elemento de imagem)
RGB	*Red-Green-Blue*
SBTVD	Sistema Brasileiro de Televisão Digital
SEI	*Supplemental enhancement information*
SRAM	*Static Random Access Memory*
UFRGS	Universidade Federal do Rio Grande do Sul

VHDL *VHSIC hardware description language*

VHSIC *Very High Speed Integrated Circuits*

SUMÁRIO

1 INTRODUÇÃO

Codificação de entropia é comummente utilizada para comprimir dados em diversas aplicações bem conhecidas tais como arquivos e pastas, imagens, áudio e vídeo em computadores pessoais e dispositivos portáteis.

Existem diversas formas de codificação de entropia, usadas para diferentes aplicações. Porém todas elas aproveitam-se da distribuição de probabilidades de ocorrência não-uniforme de símbolos, associando de forma direta ou indireta códigos curtos a símbolos mais prováveis e códigos compridos a códigos menos prováveis. Assim, sendo boa a estimativa de probabilidade há uma compressão ao converter símbolos a códigos curtos.

Este projeto está relacionado à codificação de entropia CABAC, que é empregada em vídeos do padrão de compressão de vídeo H.264/AVC adotado pelo Sistema Brasileiro de Televisão Digital (SBTVD). A meta do projeto é integrar o CABAD ao decodificador de vídeo H.264/AVC desenvolvido em *hardware* para o SBTVD.

Uma contextualização do projeto é feita no segundo capítulo, tratando desde o Sistema Brasileiro de Televisão Digital de uma forma geral até os objetivos do trabalho. Para isso é apresentado um projeto maior do qual este faz parte, a Rede H.264 e o padrão de codificação de vídeo H.264/AVC.

Um enfoque especial para o CABAD é dado no terceiro capítulo, pois trata-se do principal objeto de estudo para realizar o projeto. É exposto do que é especificado em norma, as informações mais relevantes para realização da integração e sua verificação funcional.

No quarto capítulo, a metodologia para realizar o trabalho seguida é descrita. As normas que foram utilizadas são apresentadas e discutidas, apontando no final quais são os requisitos de desempenho impostos pelo SBTVD. A seguir são apresentadas as ferramentas que viabilizaram todo o desenvolvimento necessário de *hardware*.

No quinto capítulo são apresentados módulos adotados para integração, bem como o desenvolvimento de novos módulos para suprir necessidades envolvidas. Considerações de projeto e escolhas são justificadas. Neste capítulo também são exibidos e explicados diagramas em blocos de arquiteturas explicitando interfaces, máquinas de estados finitos, parte operativas e o comportamento dos módulos.

No sexto capítulo há uma compilação dos resultados obtidos no projeto de diplomação, incluindo dados de síntese em FPGA tais como área ocupada e frequência máxima de operação. Também é feita uma verificação de desempenho através da taxa de processamento de dados pelo CABAD integrado ao *parser*.

Finalmente, nas conclusões é feita uma análise dos resultados obtidos, verificação das metas alcançadas e de uma forma geral qual foi a contribuição do projeto.

2 CONTEXTUALIZAÇÃO

Neste capítulo é apresentado o contexto e o projeto no qual o trabalho se situa. Os objetivos do SBTVD e suas principais características são apresentadas na primeira seção. A seguir é apresentado o projeto Rede H264, que tem o propósito de tratar a codificação dos sinais-fontes de áudio e vídeo do SBTVD. O padrão de compressão de vídeo H.264 é descrito, do ponto de vista do decodificador.

Na última seção (2.4), os objetivos do projeto são expostos, indicando o que é realizado dentro do contexto do projeto maior de um decodificador de vídeo H.264/AVC.

2.1 SBTVD

O SBTVD (Sistema Brasileiro de Televisão Digital) já é adotado em vários países da América Latina além do Brasil. Foi estabelecido que em 2016 a transmissão analógica deverá ser encerrada no Brasil, e todas emissoras devem migrar para o formato digital, devido as várias vantagens que o novo sistema oferece. A operação comercial começou em 2 de Dezembro de 2007, em São Paulo.

O novo padrão tem as seguintes características:

- Imagens de alta definição, com resolução de até 1920x1080 *pixels*.
- Áudio de alta qualidade, sistema 5.1.
- Robustez a interferências. Imagens sem chuvisco ou distorções.
- Recepção *Full-seg* na resolução *Full HD* e recepção *One-seg* para dispositivos portáteis nas resoluções SQVGA, QVGA e CIF.
- Interatividade, prevendo um canal de retorno ao usuário.

- Multiprogramação, podendo ser exibido até 3 programações diferentes em um só canal.

2.2 Rede H.264

A Rede H.264 [5] é um projeto realizado em cooperação entre a Universidade Federal do Rio Grande do Sul e outras universidades brasileiras, além do CEITEC, com objetivo de desenvolver produtos para tratar da codificação dos sinais-fonte para o SBTVD e propor melhorias no padrão H.264/AVC.

O codificador e o decodificador de vídeo H.264/AVC são desenvolvidos na UFRGS, em equipes separadas. Três equipes que desenvolvem o codificador situam-se na informática, enquanto a equipe que desenvolve o decodificador de vídeo em VHDL encontra-se no Departamento de Engenharia Elétrica da Escola de Engenharia.

2.3 H.264/AVC

O H.264/AVC foi escolhido para ser o padrão de codificação de vídeo do SBTVD, baseando-se na norma ITU [1] que o especifica tecnicamente. A norma brasileira [6] impõe algumas limitações e especificações ao uso do H.264/AVC no que diz respeito à resolução máxima, taxa de *bits*, conjunto de ferramentas que pode ser usado, etc.

Como toda técnica de compressão de vídeo, o padrão H.264/AVC busca explorar redundâncias espaciais e temporais presentes nas imagens, diminuindo significantemente a quantidade de informação necessária para representar um vídeo. O seu ganho em compressão, quando relacionado a padrões de compressão que o antecederam, é resultado de diversas novas técnicas introduzidas, cada uma contribuindo isoladamente com uma pequena parcela na taxa de compressão total.

Na figura 1 é mostrado um diagrama de blocos simplificado do decodificador de vídeo H.264/AVC. Um fluxo de *bits* de vídeo codificado é recebido na entrada do decodificador para ser processado pelo *parser*, extraindo elementos sintáticos que são entradas para os blocos de predição (inter e intra) e os blocos de quantização inversa e transformada inversa. Estes elementos sintáticos são obtidos no *parser* através do emprego de seu módulo mais expressivo, o decodificador de entropia. Codificação de entropia é usada para comprimir qualquer tipo de dados associando códigos mais curtos para símbolos de

ocorrência mais provável.

A partir dos elementos sintáticos obtidos pelo *parser*, é escolhido o tipo de predição, *inter* ou *intra*. Usando-se os dois tipos de predição, uma imagem é reconstruída da melhor forma possível, processando-se uma quantidade de informação muito menor do que a necessária para transmitir os valores de cada uma de suas amostras. Porém, como a predição é imperfeita, faz-se necessário adicionar dados de resíduos aos dados preditos. Estes dados são obtidos pelo decodificador de entropia na forma de coeficientes de uma transformada, devendo passar pelos processos de quantização inversa e transformada inversa.

Após uma imagem inteira ser reconstruída através da predição com adição de resíduo, um filtro redutor de efeitos de borda suaviza as fronteiras dos blocos da imagem que são decodificadas separadamente. As imagens decodificadas são armazenadas para exibição e para servir de referência para a predição *inter*.

Maiores detalhes sobre o padrão H.264/AVC são apresentados nas próximas subseções.

2.3.1 Estrutura e divisão do vídeo e das imagens

Imagens digitais são representadas num plano bidimensional por uma matriz de amostras ou *pixels* que quantificam valores de cor e brilho, em um dado espaço de cores. O exemplo mais conhecido de espaço de cores é o RGB, em que cada ponto da imagem é representado por valores numéricos de vermelho, verde e azul. A quantidade de amostras presente na matriz determina a resolução da imagem.

Para a representação de vídeo digital, as imagens são exibidas sequencialmente a uma determinada taxa, tipicamente 30 quadros ou 60 campos por segundo na transmissão *Full-seg*, compondo uma sequência de vídeo. As imagens numa sequência de vídeo podem ter o formato de quadros completos ou de campos. Campos consistem na metade da informação de um quadro, contendo somente as linhas pares (campo de cima) ou as somente as linhas ímpares do quadro completo (campo de baixo). Numa sequência de vídeo entrelaçado os campos de cima e de baixo são exibidos de forma alternada.

2.3.1.1 Espaço de cores no H.264/AVC

Para representar uma imagem digital no padrão H.264/AVC, é utilizado um espaço de cores mais apropriado para codificação de vídeo. Em vez do sistema RGB, as imagens

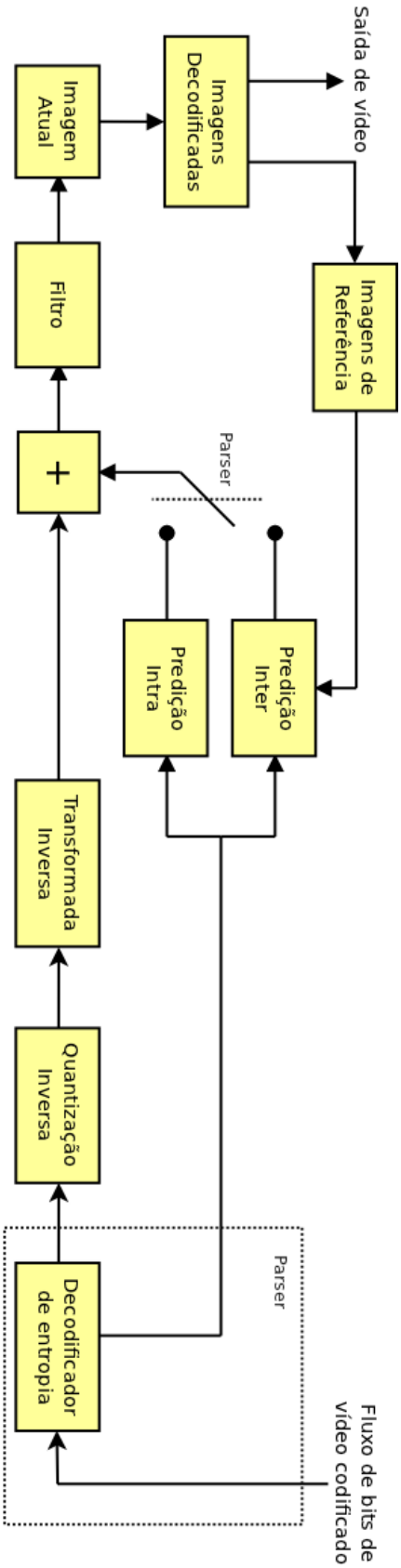

Figura 1: Diagrama de blocos simplificado do decodificador de vídeo H.264/AVC

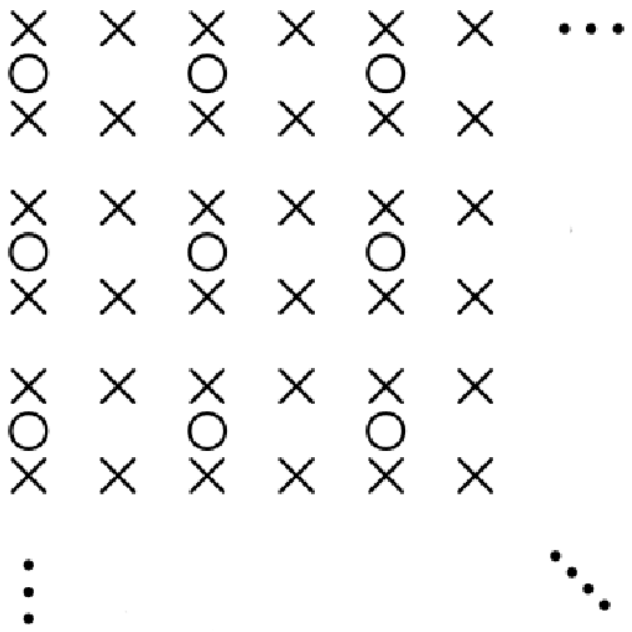

Figura 2: Espaço de cores YUV 4:2:0 (subamostrado) [1]

digitais são representadas por matrizes de amostras de luminância, crominância azul e crominância vermelha, compondo o sistema YUV. Desta forma, correlações na imagem podem ser melhor exploradas.

Outra vantagem deste espaço de cores é que a percepção humana é mais sensível ao brilho. Assim, é possível sub-amostrar as componentes de crominância em relação à luminância. O padrão H.264/AVC explora esta característica, usando o sistema YUV 4:2:0, conforme representado na figura 2. As matrizes de crominância tem a metade do número de amostras tanto na horizontal quanto na vertical, em relação à matriz de luminância. Assim, a cada quatro amostras de luminância correspondem uma amostra de crominância azul e uma amostra de crominância vermelha, totalizando seis amostras para compro quatro *pixels*. No sistema RGB, são necessárias doze amostras (quatro de cada cor) para compor quatro pixels. Desta forma, o sistema YUV 4:2:0 proporciona uma redução de metade dos dados, sem perdas significativas na qualidade das imagens.

2.3.1.2 Slices

Um *slice* é uma porção da imagem codificada de forma completamente independente em relação a outras. Ou seja, *slices* podem ser codificados paralelamente. Eles são compostos por macroblocos: regiões de 16x16 amostras de luminância e 8x8 amostras de cada crominância associadas. Os *slices* estão contidos em NALs, e incluem um cabeça-

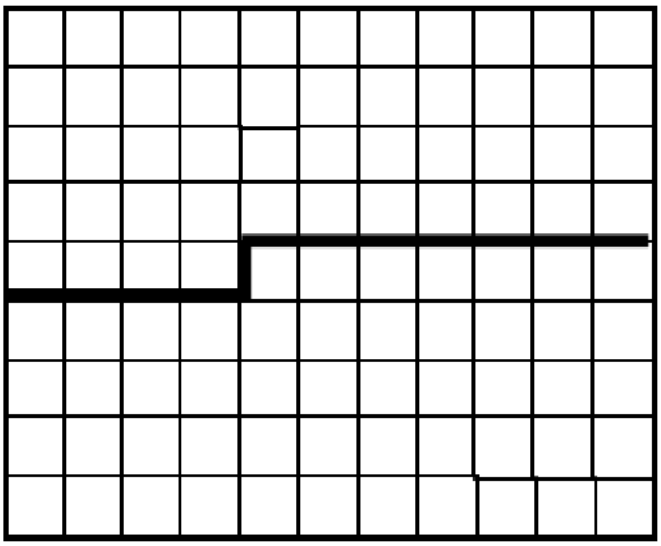

Figura 3: Divisão de *slices* em *raster-scan*

lho (*slice_header*) com parâmetros de sua codificação e uma porção de dados (*slice_data*) que contém os macroblocos codificados em sequência.

No padrão H.264/AVC quadros ou campos podem ser divididos em um ou mais *slices*, em diversos arranjos. Porém a norma brasileira [6] restringe a codificação de *slices* somente ao arranjo *raster-scan*, em que os macroblocos são dispostos de maneira contígua, da esquerda para a direita até a borda da imagem para continuar uma linha de macroblocos abaixo, na borda esquerda da imagem. A figura 3 mostra um exemplo de imagem com dois *slices*, onde os quadrados menores representam os macroblocos e a borda mais espessa no meio da imagem representa a divisória dos *slices*.

Os *slices* podem ser do tipo P, B ou I. Nos *slices* do tipo P pode haver tanto predição *inter* com uma imagem de referência do passado quanto predição *intra*. Nos *slices* do tipo B pode haver predição *inter* com até duas imagens anteriormente decodificadas (do futuro ou do passado na ordem de exibição) quanto predição *intra*. Em *slices* I, somente pode haver predição *intra*.

O tipo de predição empregado é obtido a partir do tipo de macrobloco, contido no elemento sintático *mb_type*.

2.3.1.3 MBAFF

O uso de *MBAFF* (*Macroblock-adaptive frame-field* é uma possibilidade inovadora na codificação de sequências de vídeo no padrão H.264/AVC. Ele determina que os macroblocos dos *slices* são dispostos em pares compostos por um macrobloco de cima e

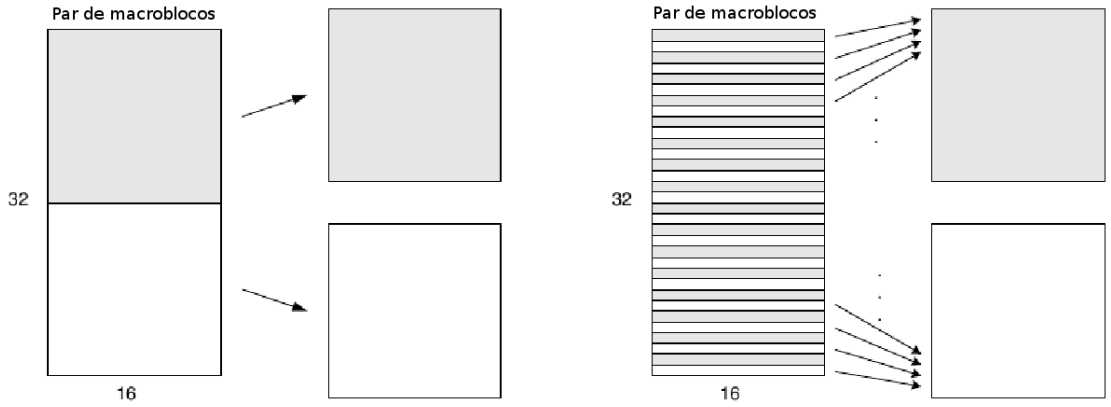

Figura 4: Codificação de pares de macroblocos em campos ou quadros [2]

um macrobloco de baixo, que são processados em sequência, para depois ser tratado o próximo par. Os pares de macroblocos podem ser codificados como macroblocos-quadro ou macroblocos-campo, conforme é mostrado na figura 4. Sua recomposição, no quadro adaptativo que o contém, é feita concatenando-se os macroblocos de pares codificados como quadro, e entrelaçando-se os macroblocos de pares codificados como campo.

Esta técnica proporciona a vantagem codificar melhor vídeos com regiões nas imagens ora de alto movimento como campos, ora de pouco movimento como quadros. A informação de codificação como quadro ou campo está contida no elemento sintático *mb_field_decoding_flag* que é presente para cada par de macroblocos.

2.3.2 Predição temporal

A predição temporal (*inter*) consiste na cópia de regiões de imagens anteriormente decodificadas para a região da imagem sendo atualmente decodificada. Pode-se usar uma ou duas imagens de referência para a predição. No segundo caso é feita uma média ponderada das amostras. A diferença de posição entre a região sendo atualmente predita e a região da imagem de referência é dada por um vetor de movimento, com precisão de um quarto de *pixel*.

Quando a predição *inter* deve ocorrer no *slice*, o macrobloco sendo atualmente predito é do tipo P ou B. Macroblocos desse tipo podem ser particionados para que a compensação de movimento seja feita em blocos de tamanho variável, conforme as possibilidades mostradas na figura 5. Um macrobloco pode ser particionado conforme as possibilidades abaixo:

- nenhum particionamento

- duas partições de 8x16 amostras
- duas partições de 16x8 amostras
- quatro partições de 8x8 amostras – submacroblocos

Os submacroblocos (partições 8x8), por sua vez, podem ser particionados seguindo o mesmo padrão:

- nenhum particionamento
- duas partições de 4x8 amostras
- duas partições de 8x4 amostras
- quatro partições de 4x4 amostras – blocos

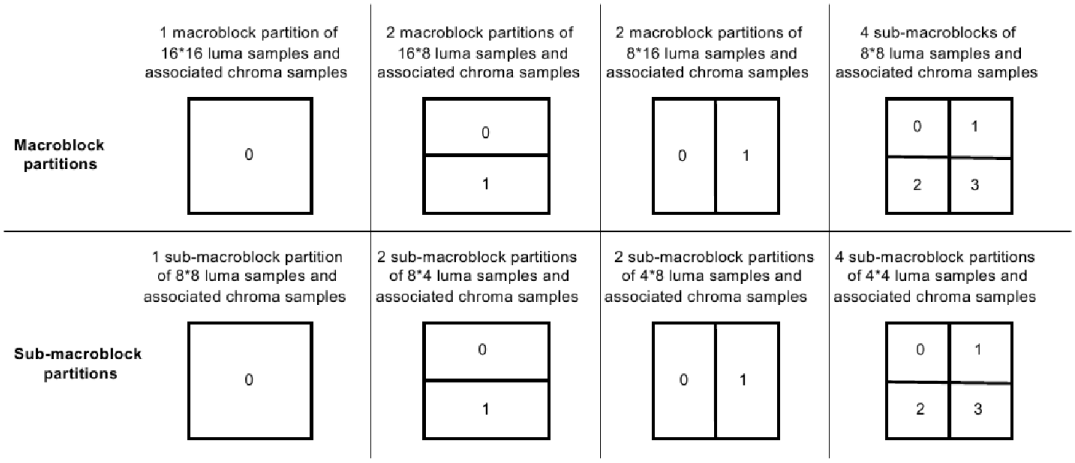

Figura 5: Particionamento para a predição inter [1]

O particionamento dos macroblocos é obtido a partir da identificação do tipo de macrobloco P ou B em tabelas, sendo 4 entradas possíveis para o P e 23 possíveis para o B. Se a identificação for P_8x8 ou B_8x8, a seguir deve ser obtido o particionamento do submacrobloco no elemento sintático *sub_mb_type*. Na identificação dos macroblocos e submacroblocos do tipo B está associada também o modo de predição, que pode ser:

Pred_L0 Partição predita com uma imagem da lista de imagens de referência 0

Pred_L1 Partição predita com uma imagem da lista de imagens de referência 1

BiPred Partição predita com duas imagens de referência, uma da lista 0, outra da lista 1.

Os índices nas listas das imagens de referência usadas para a predição estão presentes codificadas no fluxo de *bits* quando necessárias, no elemento sintático *ref_idx*. A lista 0 começa com a imagem do passado mais recente e segue até a mais longínqua. É a única lista presente em macroblocos do tipo P. A lista 1 começa com imagens do futuro na ordem de exibição, da mais próxima a atual até a mais distante no tempo de exibição.

Os vetores de movimento são preditos a partir de partições vizinhas, basicamente fazendo-se a mediana de cada componente (x e y) de três vetores de movimento que se encontram nas partições A, B e C representadas na figura 6. Quando a partição C está indisponível, a D é usada para substituir.

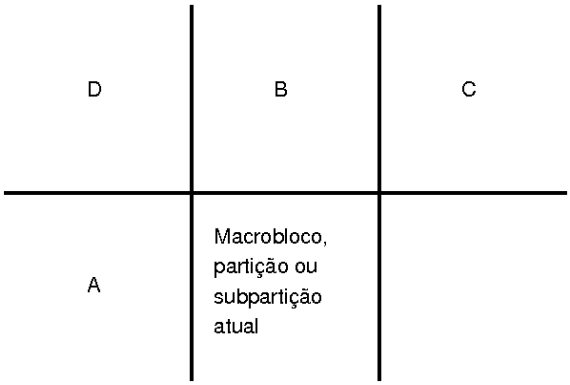

Figura 6: Partições vizinhas para predição de vetores de movimento

Após a predição dos vetores, excetuando-se casos de predição direta ou *skip*, um vetor de movimento diferencial é somado para obter o valor final do vetor de movimento. O vetor de movimento diferencial está presente codificado no fluxo *bits*, pelo elemento sintático *mvd_L0* ou *mvd_L1*.

2.3.3 Predição espacial

A predição espacial (*intra*) é empregada para explorar redundâncias dentro do próprio *slice*, com uma cópia direcional ou média DC de amostras vizinhas da região sendo decodificada. Para as amostras de luminância, a predição é realizada uniformemente por todo o macrobloco, com um dos quatros modos de predição Intra 16x16 representados na figura 7, ou a predição é realizada à nível de bloco 4x4, com um dos nove modos de predição Intra 4x4 representados na figura 8. Com o uso de bloco de transformada de tamanho 8x8 (ver seção 2.3.4), o tamanho de bloco é 8x8 em vez de 4x4. A predição das

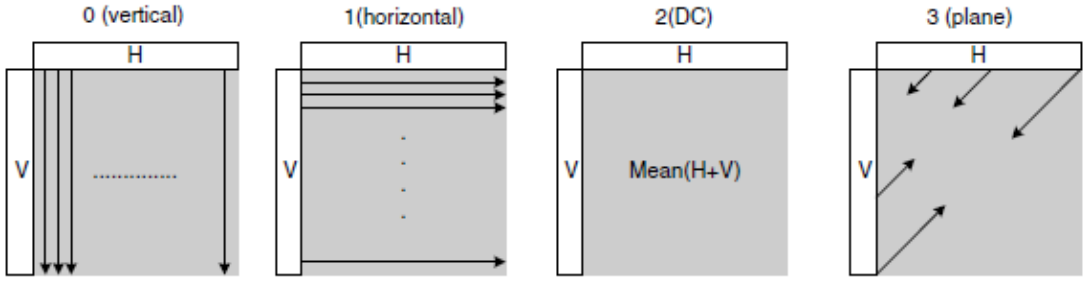

Figura 7: Representação dos modos de predição Intra 16x16 [2]

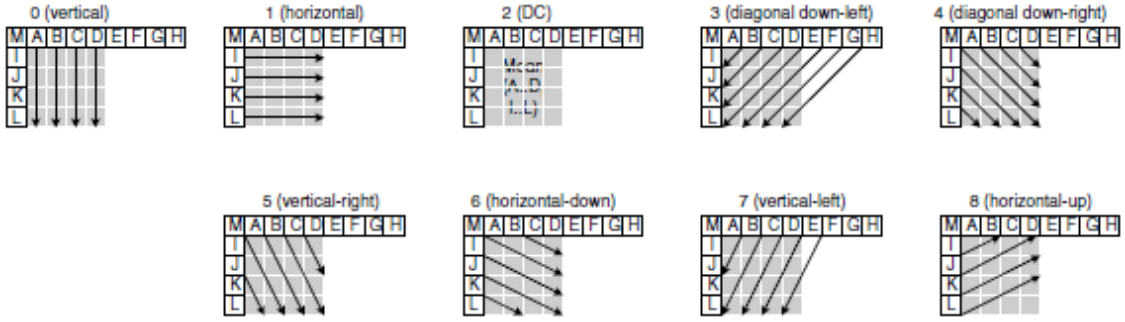

Figura 8: Representação dos modos de predição Intra 4x4 [2]

amostras de crominância é feita separadamente usando um dos quatros modos de predição Intra 16x16, que pode ser diferente do modo usado para a luminância.

O modo de predição de amostras de luminância é obtido a partir da identificação do tipo de macrobloco I. No total há 26 possibilidades, sendo 24 delas Intra 16x16 que associam um dos seus 4 modos de predição a um dos 6 padrões de codificação de resíduo mostrados na tabela 1. Os modos de predição 4x4 são obtidos através de elementos sintáticos separadamente, após decodificar o tipo de macrobloco I NxN (N = 4, ou 8 se o tamanho do bloco de transformada é 8x8). A última possibilidade, macrobloco I_PCM, é uma ocorrência anômala, em que todos os valores das amostras de luminância e crominância do macrobloco são enviados diretamente sem codificação. Neste caso, nenhuma predição ou adição de resíduo ocorre. Em qualquer caso exceto I_PCM, o modo de predição de amostras de crominância é obtido separadamente através do elemento sintático *intra_chroma_pred_mode*.

2.3.4 Quantização e Transformada

Dois processos são empregados para a codificação do resíduo: a transformada e a quantização.

Tabela 1: Codificação de resíduo para macroblocos Intra 16x16

Codificação de resíduo	Luminância	Crominância DC	Crominância AC
1	Ausente	Ausente	Ausente
2	Ausente	Presente	Ausente
3	Ausente	Presente	Presente
4	Presente	Ausente	Ausente
5	Presente	Presente	Ausente
6	Presente	Presente	Presente

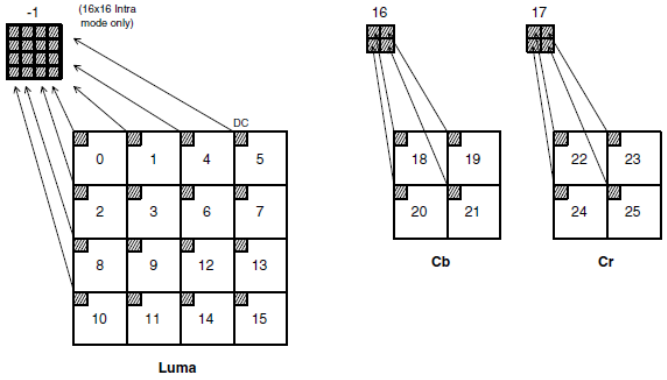

Figura 9: Tranformada DC [2]

A transformada é utilizada para passar os resíduos para o domínio da frequência, sendo uma adaptação da DCT (*Discrete Cosine Transform*), aproximada para não envolver multiplicações de ponto flutuante [2]. Ela é realizada em blocos de tamanho 4x4 para amostras de luminância e crominância ou blocos de tamanho 8x8 somente para amostras de luminância no caso em que o elementos sintático *transform_size_8x8_flag* estabelece o seu uso. Amostras de luminância, quando é usada a predição Intra 16x16, e amostras de crominância geram dois blocos de coeficientes distintos no processo de transformada, o AC e o DC. Os blocos DC são o agrupamento dos coeficientes DC de cada bloco 4x4 de amostras que constitui o macrobloco, conforme é representado na figura 9. Os blocos AC correspondem aos coeficientes de cada bloco 4x4 ou 8x8. Quando há transformada DC, o total de coeficientes AC é 15, e quando não há, o total de coeficientes é 16 ou 64 (blocos 8x8).

A determinação do tamanho do bloco de transformada DC se dá a partir do número de blocos de amostras 4x4 contidos no macroblocos. Num macrobloco há 16 blocos 4x4 de amostras de luminância e 4 blocos 4x4 de amostras de crominância, pois esta é subamostrada. Portanto blocos DC de luminância são de 16 coeficientes, enquanto que blocos DC de crominância são de 4 coeficientes.

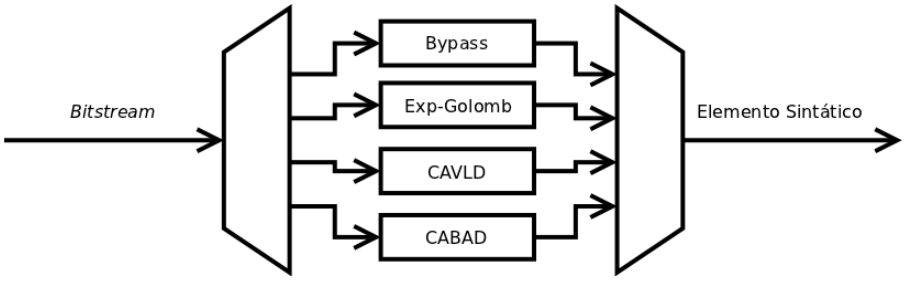

Figura 10: Obtenção de elementos sintáticos no H.264/AVC

Após o processo da transformada, os coeficientes são quantizados de acordo com um determinado parâmetro de quantização dado inicialmente para todo o *slice* mas que pode variar de macrobloco a macrobloco. A quantização é uma forma de diminuir a quantidade de informação necessária para representar o resíduo ao diminuir a precisão dos coeficientes, eliminando o que é menos significativo. Trata-se portanto de um método de compressão com perdas.

Os coeficientes, finalmente quantizados, são comprimidos pelo codificador de entropia CABAC ou CAVLC, que aproveitam a distribuição não-uniforme de sua probabilidades de ocorrência para diminuir ainda mais a quantidade de informação a ser transmitida.

2.3.5 Codificação de entropia

A codificação de entropia é empregada para comprimir dados de qualquer tipo, associando a símbolos códigos de comprimento proporcional à probabilidade de ocorrência. Assim, havendo uma distribuição não-uniforme na função densidade de probabilidade, esta característica pode ser explorada a fim de atingir compressão de dados através da correspondência de símbolos mais prováveis a códigos mais curtos, e símbolos menos prováveis a códigos mais compridos.

No padrão H.264/AVC, com poucas exceções, os dados, na forma de elementos sintáticos, são comprimidos com uma codificação de entropia. Conforme está ilustrado na figura 10, o decodificador obtém os elementos sintáticos do fluxo de *bits* fazendo uma leitura direta (*bypass*) ou aplicando uma das três codificações de entropia utilizadas no padrão: *Exponential Golomb*, CAVLC ou CABAC.

As principais codificações são CAVLC e CABAC, elas não podem ser utilizadas numa mesma sequência de imagens. A escolha da codificação é feita através da leitura de um parâmetro global, antes de decodificar as imagens (ver seção 2.3.6.2). A CAVLC é empregada apenas para coeficientes de resíduo, enquanto que a CABAC é empregada para

os elementos sintáticos da camada de dados do *slice*, incluindo resíduos.

A leitura direta é feita para todo *flag*, exceto quando é empregada a CABAC. Para poucos elementos sintáticos na camada de dados do *slice* quando a codificação principal é CAVLC e para alguns parâmetros globais a leitura direta também é feita.

Exp-Golomb (Exponencial Golomb) é empregada para a maioria dos elementos sintáticos da camada de dados do *slice* quando a codificação principal escolhida é CAVLC, e para parâmetros globais, independente da codificação principal.

2.3.5.1 Exp-golomb

A codificação *Exp-Golomb* associa a números inteiros códigos de comprimento variável proporcional ao valor que representam. Códigos *exp-golomb* de ordem K consistem de uma sequência de $N - K - 1$ '0's, seguidas por um '1' de parada e mais $N - 1$ bits.

Para codificar um número não negativo num código *exp-Golomb* de ordem K, pode ser usado o seguinte método [7] :

1. Tomar o valor em binário com exceção dos últimos K dígitos e somar um a ele. Guardar este valor.

2. Contar o número de *bits* do valor guardado e subtrair um, e escrever esse número final de '0's precedentes ao valor guardado.

3. Escrever os últimos K dígitos em binário.

Para fim de extração de elementos sintáticos a partir do fluxo de *bits*, é sempre utilizado *exp-Golomb* de ordem $K = 0$. A codificação é eficiente para valores de baixa magnitude ao mesmo tempo em que não impõe nenhum limite de comprimento.

2.3.5.2 CAVLC

A CAVLC (*Context-based Adaptive Variable Lenght Coding*) emprega tabelas cujas entradas são valores de elementos sintáticos para construção de blocos de coeficientes de resíduo, e as saídas códigos de comprimento variável. Tabelas diferentes são escolhidas adaptativamente, de acordo com o número de coeficientes de transformada não-nulos em blocos vizinhos.

2.3.5.3 CABAC

A CABAC (*Context-based Adaptive Binary Arithmetic Coding*), em vez de códigos de comprimento variável diretamente, emprega aritmética binária para explorar probabilidades de ocorrência de símbolos. Portanto, os valores de elementos devem ser binarizados, de forma a possibilitar codificação de decisões binárias (*bins*). Variáveis de contexto armazenam as probabilidades de ocorrência dos *bins* e são atualizadas, adaptando-se a estatísticas locais, a cada codificação de decisão binária. Para todos os tipos de elementos sintáticos, são indexados, no total 460 variáveis de contexto. Elas são selecionadas através de um modelamento de contextos, que considera codificações de *bins* anteriormente codificados e valores de elementos sintáticos vizinhos, explorando correlação na imagem.

Mais detalhes da CABAC, sob o ponto de vista do decodificador são apresentados no capítulo 3.

2.3.6 NALUs

Para fins de transmissão, os dados codificados são empacotados em camadas de abstração de rede (*Network Abstraction Layers*). As unidades de NAL (NALU) são identificadas pelo *parser* por um código de início de NALU e um padrão de término. Contém um cabeçalho que informa seu tipo e se é referência para outras NALUs. As NALUs são codificadas separadamente e podem conter um conjunto de parâmetros ou um *slice* codificado.

Nas seções subsequentes são apresentados possíveis tipos de NALUs.

2.3.6.1 SPS

O SPS (*Sequence Parameter Set* contém parâmetros globais da sequência de vídeo, dentre os quais destacam-se :

- Perfil de codificação
- Nível de codificação
- Espaço de cores: YUV 4:2:0 ou YUV 4:0:0 (tons de cinza)
- Número máximo de imagens de referência
- Largura da imagem em números de macroblocos

- Altura da imagem em números de macroblocos
- Uso de vídeo entrelaçado
- Uso de MBAFF

2.3.6.2 PPS

O PPS (*Picture Parameter Set* contém parâmetros globais da imagem de vídeo. Ele deve fazer referência a um SPS previamente (de)codificado. Entre os seus parâmetros destacam-se:

- Principal método de codificação de entropia utilizado
- Número de imagens de referência ativas na lista 0
- Número de imagens de referência ativas na lista 1
- Uso e tipo de predição *inter* ponderada
- Parâmetro de quantização inicial para as imagens
- Uso de transformada com bloco de tamanho 8x8

2.3.6.3 Slice

O *slice* é o principal tipo de NALU, e contém a maior quantidade de informação. Ele é composto por um cabeçalho e uma parte de dados. No cabeçalho do *slice*, encontra-se parâmetros relacionados à sua codificação, como os citados abaixo:

- Primeiro macrobloco no *slice*
- Tipo de *slice*
- Estrutura do *slice*: quadro ou campo (de cima ou de baixo)
- Novo número de imagens de referência ativas na lista 0
- Novo número de imagens de referência ativas na lista 1
- Parâmetros de reordenamento de listas de referências
- Parâmetros de marcação das imagens como referência

- Tabela de pesos para a predição *inter* ponderada
- Índice de inicialização para a CABAC
- Parâmetro de quantização para o *slice* (diferença do inicial em PPS)

A parte de dados do *slice* implementa um laço para codificação de todos os macroblocos, com uma sequência de elementos sintáticos codificados condicionalmente. Dentro da camada de macrobloco há laços internos para codificação de elementos sintáticos que ocorrem mais de uma vez.

2.3.6.4 Outros tipos

Os três tipos de NALU destacados nas seções anteriores são suficientes para compor uma sequência de vídeo completa para codificação. Porém, há outros tipos que podem ser usados de forma suplementar:

SEI Contém mensagens para auxiliar na decodificação, exibição e outros propósitos

Acess unit delimiter Delimitador de unidades de acesso

End of sequence Indicador de final de sequência

End of stream Indicador de final do fluxo

Filler Data Dados de enchimento de pacotes

2.3.7 Perfis e níveis

A norma do padrão H.264/AVC especifica diferentes perfis de codificação de vídeo, nos quais são definidos conjuntos de ferramentas que podem ser utilizados na codificação de vídeo e o nível máximo. A norma da ABNT permite que no SBTVD sejam usados os perfis *baseline*, *main* e *high*, até o nível 4.0. Vídeos *one-seg* podem utilizar apenas o perfil *baseline* com nível até 1.3, enquanto que os outros perfis só podem ser utilizados com vídeos *full-seg*.

A figura 11 ilustra o conjunto de ferramentas presente em cada nos três perfis do H.264/AVC adotados pela ABNT. Nota-se que os perfis *baseline*, *main* e *high* estão em ordem crescente de complexidade. A CABAC só pode ser utilizado a partir do perfil *main*.

No perfil *high* e nível 4.0, destaca-se:

- Predição *inter* com até duas imagens de referência, ponderada
- CABAC
- Vídeo entrelaçado
- MBAFF (*Macroblock Adaptive Frame-field Flag*)
- Transformada com blocos de tamanho 8x8.
- Resolução máxima de 1920x1080 amostras de luminância
- Taxa de exibição de 30 quadros por segundo
- Taxa de *bits* na entrada de 20 Mbps
- Memória para armazenar 4 quadros de referência na resolução máxima

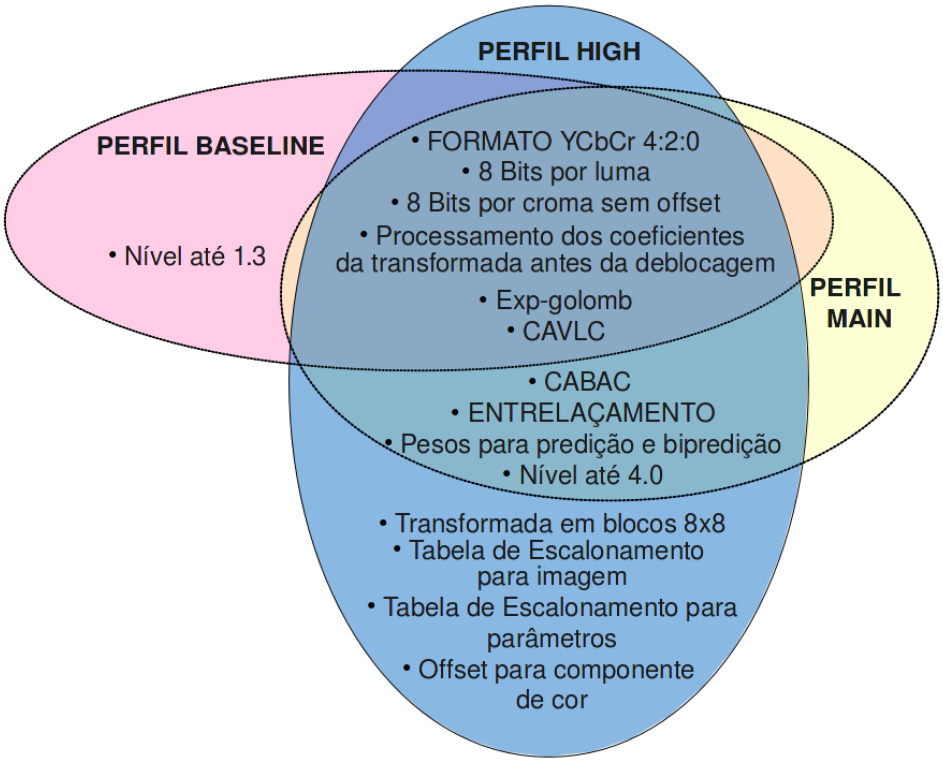

Figura 11: Perfis e ferramentas de acordo com normas ITU e ABNT

2.4 Objetivos deste trabalho

Tendo em vista o panorama do padrão de codificação de vídeo H.264/AVC e o trabalho realizado na equipe do LaPSI, este trabalho tem como objetivo avançar o desenvolvimento do decodificador de vídeo em VHDL para que ele seja capaz de decodificar vídeos até o perfil *High*, concentrando-se no *parser* e principalmente na integração do decodificador de entropia CABAC, definindo interfaces entre eles. As contribuições deste trabalho tornam possível a decodificação de todos os elementos sintáticos que são entradas dos módulos preditores e dos módulos de obtenção do resíduo a partir de coeficientes. A inclusão do CABAD ao projeto é a necessidade mais urgente para esta finalidade, pois é o decodificador de entropia utilizado por padrão nas transmissões *Full-seg*.

O projeto também prevê como objetivo a verificação de desempenho para conformidade com a norma brasileira através da avaliação das frequências máximas dos módulos constituintes e do sistema como um todo, e a taxa de processamento de dados. Assim deve-se determinar a resolução de vídeo que o decodificador é capaz de processar.

3 CABAD

O decodificador de entropia CABAC, denominado por simplicidade como CABAD (*Context-based Adaptive Binary Arithmetic Decoder*) é usado em vídeos de alta definição, nos perfis *main* e *high*, uma vez que a CABAC é mais eficiente que a CAVLC, atingindo maiores taxas de compressão de dados. Esta vantagem é possível devido à sua maior complexidade.

O fluxograma exibido na figura 12 apresenta uma visão geral dos processos envolvidos no funcionamento do CABAD. Antes da decodificação do primeiro elemento sintático de um *slice*, as variáveis de contextos devem ser inicializadas. Depois, entra-se num laço de decodificação de decisões binárias (*bins*) que se repete concatenando uma cadeia de *bins* até que se obtenha uma associação válida a um possível valor de elemento sintático. Neste laço de decodificação, é feita para cada *bin* a seleção de um contexto, que contém as probabilidades de ocorrência dos valores '0' e '1'. A decodificação dos *bins* é feita por uma máquina aritmética binária, e o contexto utilizado é atualizado adaptando-se às estatísticas locais com uma transição de estado de probabilidade.

Nas seções deste capítulos são expostos os principais aspectos do CABAD, esclarecendo cada uma das etapas mostradas no fluxograma.

3.1 Decodificação aritmética binária

A decodificação aritmética binária baseia-se em subdivisão de intervalos. Um intervalo inicial é dividido em dois subintervalos, de tamanhos proporcionais às probabilidades de ocorrência do símbolo mais provável e do símbolo menos provável respectivamente. Um dos dois subintervalos é escolhido fazendo-se uma comparação dos seus limites com o código presente no fluxo de *bits*, decodificando um *bin*. O subintervalo escolhido torna-

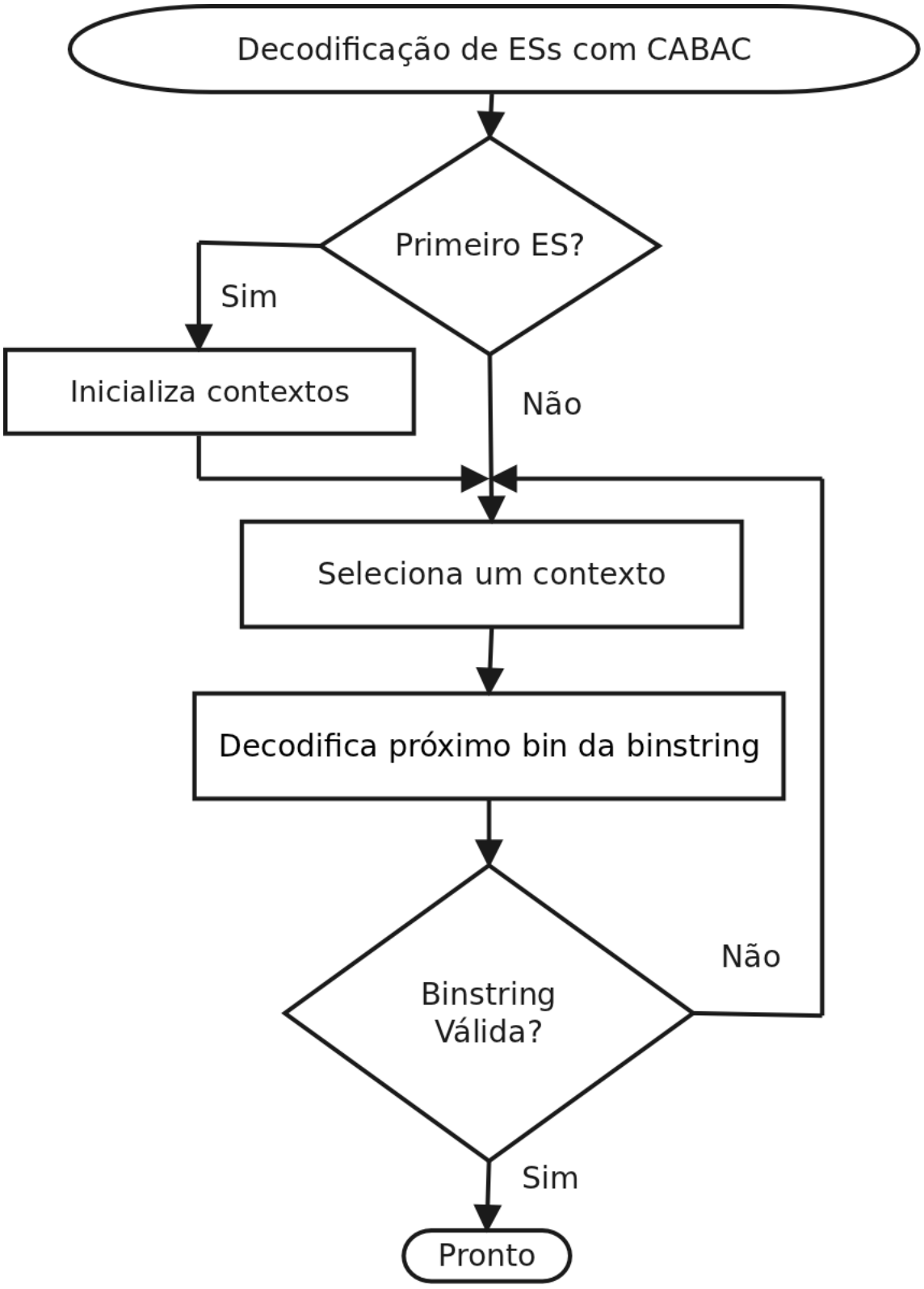

Figura 12: Diagrama de blocos geral do CABAD

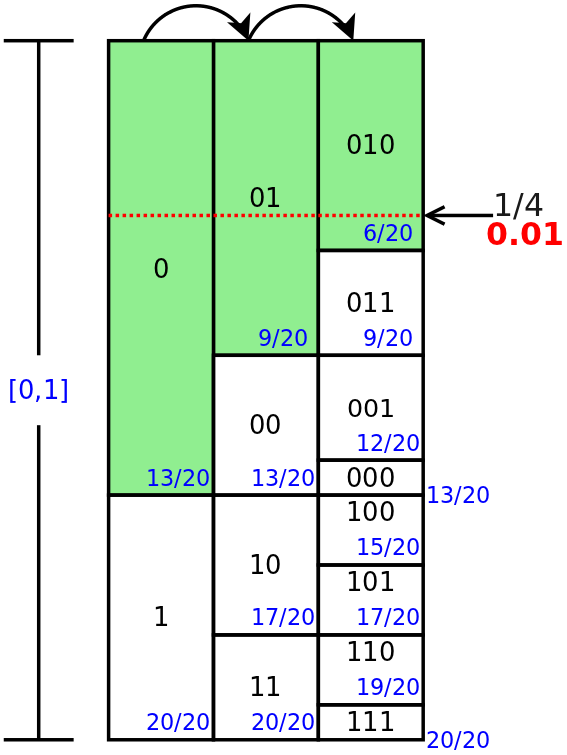

Figura 13: Exemplo de decodificação aritmética binária

se o novo intervalo inicial e o processo é repetido para obter novos *bins*.

A figura 13 ilustra o conceito de decodificação aritmética binária com um exemplo em que o código do fluxo de *bits* é 0,01 em binário (1/4 em decimal), podendo ser representado por apenas 2 *bits*. A decodificação dos *bins* começa comparando-se o valor do código aos intervalos iniciais correspondentes ao símbolo mais provável 0, de 0 a 13/20, e ao símbolo menos provável 1, de 13/20 a 20/20. Como 1/4 encontra-se entre 0 e 13/20, o símbolo mais provável 0 em destaque na figura é decodificado. Na sequência, divide-se o intervalo de 0 a 13/20 nos intervalo correspondente ao símbolo mais provável 1, de 0 a 9/20 e no intervalo de símbolo menos provável 0 de 9/20 a 13/20. 1/4 encontra-se entre 0 e 9/20, portanto o símbolos mais provável 1 é decodificado, formando a sequência de *bins* 01. Finalmente, ao comparar-se o código aos limites dos intervalos da terceira decodificação (6/20 e 9/20), obtém-se o terceiro *bin* da sequência, 0.

Neste exemplo, foi possível decodificar uma sequência de 3 *bins* com 2 *bits*, demonstrando como uma compressão está presente quando há uma boa estimativa de probabilidades. Nota-se que no exemplo, os valores do símbolo mais provável e a proporcionalidade dos intervalos podem se alterar a cada decodificação, o que caracteriza uma decodificação aritmética adaptativa.

Na CABAC, o intervalo de comparação tem um tamanho determinado pela variável *codIRange* de 9 *bits*, que é mantido sempre que começa a decodificação de um *bin* entre os valores 256 e 510 através de um processo de renormalização. O código do fluxo de *bits* é lido para a variável *codIOffset* também de 9 *bits*, a qual se lê mais um *bit* menos significativo toda vez que é feita uma renormalização.

Para determinar os tamanhos dos intervalos de acordo com a probabilidade de ocorrência é usada uma LUT com valores aproximados em vez de multiplicações. As entradas da LUT são o índice *qCodIRangeIdx* que pode variar de 0 a 3, e o índice do estado de probabilidade *pStateIdx* que pode variar de 0 a 63. A saída da LUT é a variável *codIRangeLPS*, um número de 2 a 128 ou 240 (dependendo de *qCodIRangeIdx*), que representa o tamanho do intervalo correspondente ao símbolo menos provável.

O índice *qCodIRangeIdx* é composto pelos *bits* 6 e 7 da variável *codIRange*, ou seja os dois *bits* mais significativos com exceção do primeiro que é igual a 1 devido à renormalização. Este índice representa portanto o tamanho do intervalo *codIRangeLPS* apontando para uma de quatro possíveis faixas de intervalo igualmente distribuídas entre 256 a 511.

O índice *pStateIdx* indica um dos 64 estados de probabilidade possíveis que representam a probabilidade de ocorrência do símbolos menos provável, dispostos numa progressão geométrica de 0,5 a 0,01875 [3]. O quociente da progressão geométrica é dado pela equação abaixo:

$$\alpha = \frac{0{,}01875}{0{,}5}^{\frac{1}{63}}$$

A LUT para *codIRangeLPS* contém portanto aproximações numéricas para multiplicações previamente realizadas, considerando os 64 estados de probabilidades e as 4 faixas de tamanho para *codIRange* indicadas por *qCodIRangeIdx*.

A figura 14 ilustra o fluxograma para decodificação de *bins*. Após obter o valor o valor de *codIRangeLPS*, este é usado para decrementar *codIRange*, tornando-o do tamanho do intervalo de símbolo mais provável. Após feita esta operação de subtração, compara-se *codIOffset* diretamente com codIRange: se menor, o símbolo mais provável é decodificado (*binVal = valMPS*); se maior, o símbolo menos provável é decodificado (*binVal = 1 - valMPS*), *codIOffset* é ajustado para o valor dentro do intervalo correspondente ao símbolo menos provável e *codIRange* é feito do tamanho do do intervalo de símbolo menos provável. Após, o estado de probabilidade é atualizado conforme a decodificação (MPS ou LPS) e as variáveis *codIRange* e *codIOffset* são renormalizadas.

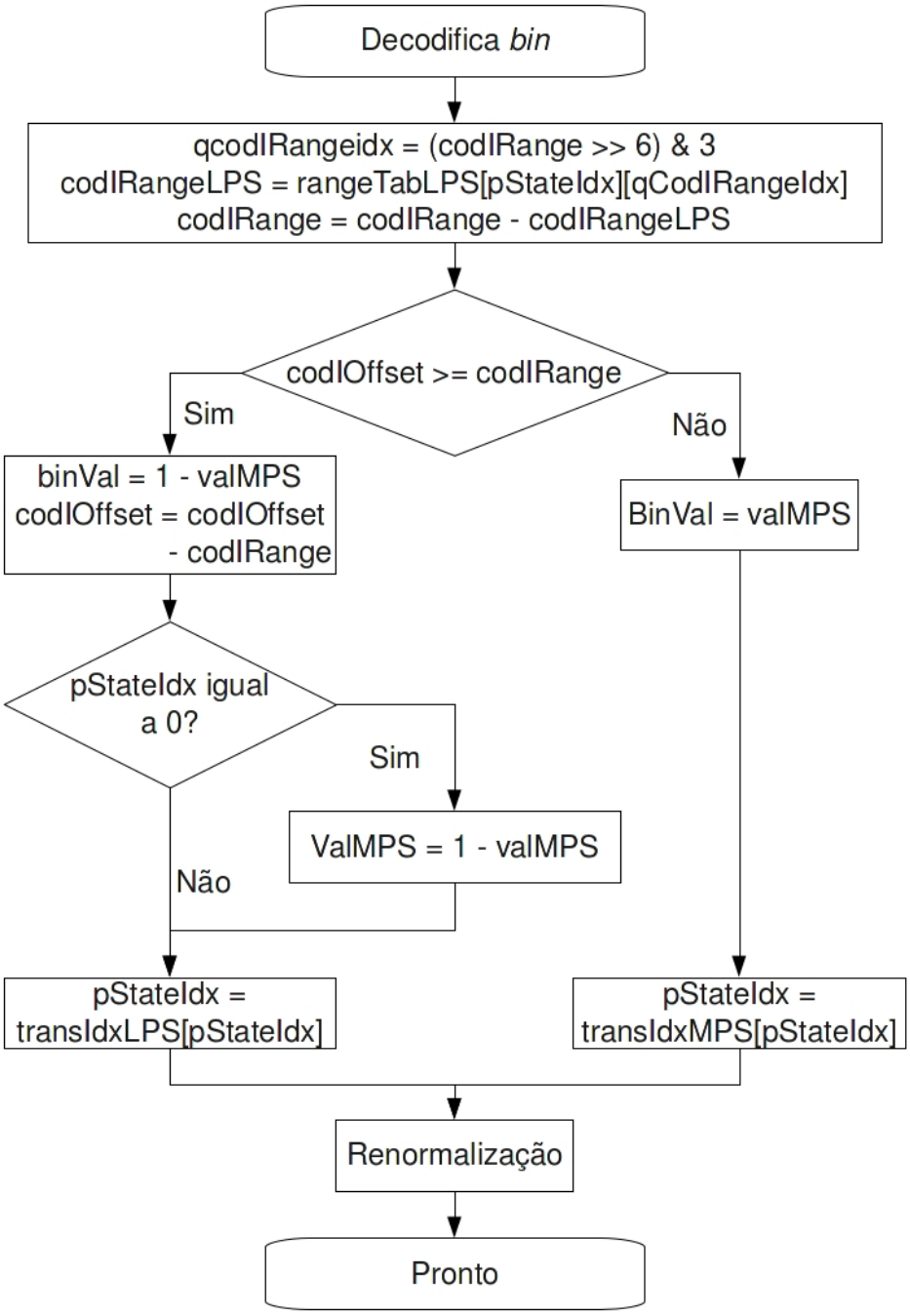

Figura 14: Decodificação de *bins* no CABAD

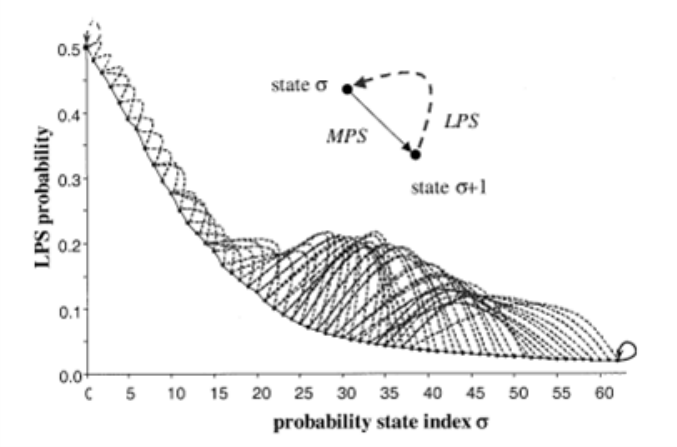

Figura 15: Transição de estado de probabilidade [3]

A transição de estados de probabilidades atualiza a variável *pStateIdx*. Conforme está representado pela figura 15, que representa o estado de probabilidade no eixo das abscissas e a probabilidade do símbolo menos provável (LPS) no eixo das ordenadas, quando ocorre decodificação de um símbolo mais provável (MPS) a probabilidade do símbolo menos provável diminui, caso contrário ela aumenta. Adicionalmente, no estado de probabilidade 0, em que a probabilidade de ocorrência do símbolo menos provável é 0,5, se ela ocorre, o valor do símbolo mais provável (MPS) é invertido conforme mostrado na figura 14.

A renormalização ocorre como mostrado na figura 16. Se o valor da variável *codIRange* é menor que 256, ele é dobrado, e lê-se um *bit* menos significativo do fluxo de *bits* para a variável *codIOffset*. O processo é repetido se *codIRange* continua menor que 256, caso contrário, é conclui-se a renormalização.

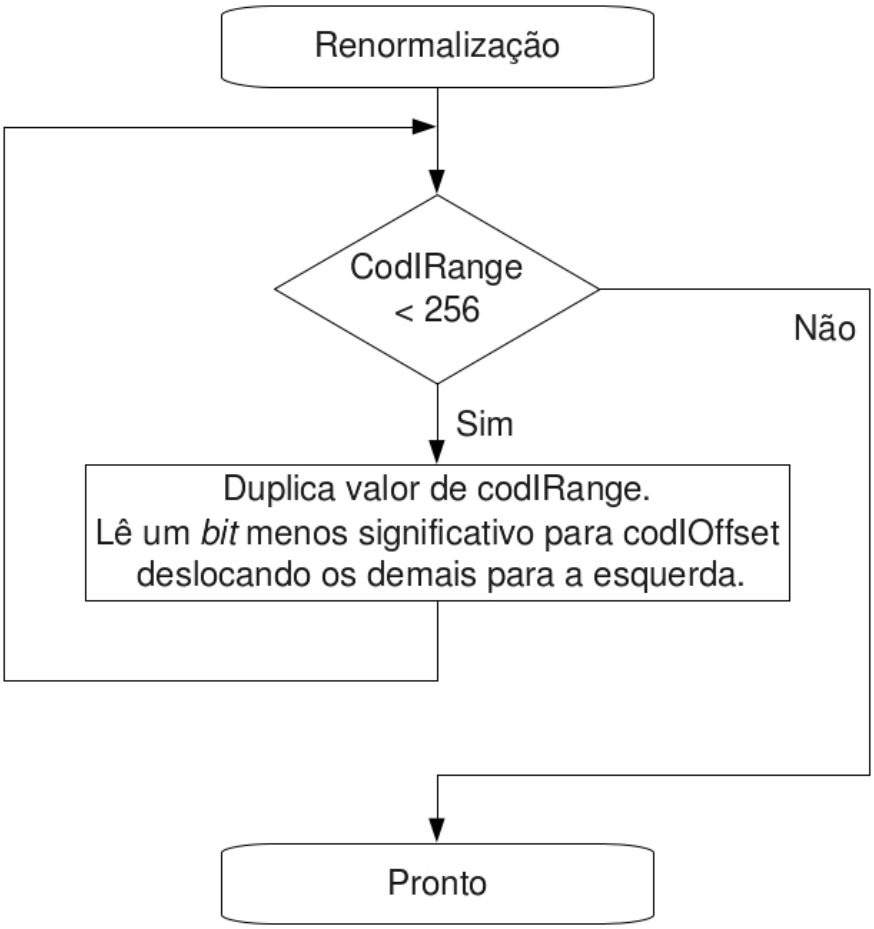

Figura 16: Renormalização das variáveis *codIRange* e *codIOffset*

3.2 De-binarização

A de-binarização converte sequência de *bins* em valor de elemento sintático, conforme um dado tipo de binarização. Emprega-se uma forma de binarização diferente a cada tipo de elemento sintático conforme é apresentado nas próximas seções.

3.2.1 Binarização unária

Este tipo de binarização consiste em associar uma sequência de N '1's seguida por um '0' de parada a um valor numérico N. Na versão truncada, o valor numérico N é limitado e assim que o número de '1's chega a este valor não é necessário o '0' de parada. Na tabela 2 é dado um exemplo de binarização unária truncada em N = 6.

Tabela 2: Binarização unária truncada a N = 6

Valor do elemento sintático	Cadeia de *bins*					
0	0					
1	1	0				
2	1	1	0			
3	1	1	1	0		
4	1	1	1	1	0	
5	1	1	1	1	1	0
6	1	1	1	1	1	1
Índice do *bin*	0	1	2	3	4	5

A binarização unária é empregada para índices de referência (*ref_idx*) que indica imagem utilizada para compensação de movimento, para a diferença no parâmetro de quantização de um macrobloco (*mb_qp_delta*) de forma mapeada para poder representar inteiros. A binarização unária truncada é empregada para o modo de predição *intra* para amostras de crominância (*intra_chroma_pred_mode*) com truncamento no terceiro *bin*.

3.2.2 Binarização unária concatenada a Exp-Golomb de ordem k

Este tipo de binarização estabelece um prefixo do qual se extrai um valor como na binarização unária truncada e um sufixo do qual se extrai um valor seguindo a regra de associação de valores a códigos por Exp-Golomb de ordem k, como apresentado na seção 2.3.5.1. Os valores extraídos do prefixo e sufixo são somados para obter o valor do elemento sintático. Se o elemento sintático for um inteiro, um bit adicional do sufixo indica o sinal.

Para vetores diferenciais de movimento (*mvd*) é empregado este tipo de binarização

com truncamento na parte unária de 9 *bins* e ordem 3 para o Exp-Golomb. Para o valor absoluto de coeficientes de resíduo (*coeff_abs_level*) também é empregado este tipo de binarização, com truncamento na parte unária de 14 *bins* e ordem 0 para o Exp-Golomb.

3.2.3 Binarização de tamanho fixo

A binarização de tamanho fixo é a mais simples de todas. Ela estabelece que uma *binstring* tem associação com valor de elemento sintático quando já tem um número determinado de *bins*. O valor de elemento sintático é obtido fazendo-se leitura direta da *binstring*.

Esta binarização é empregada para o modo de predição *intra* 4x4 e para todos os *flags*.

3.2.4 Binarização para tipo de macrobloco e submacrobloco

Para os elementos sintáticos tipo de macrobloco (*mb_type*) e tipo de submacrobloco (*sub_mb_type*), utiliza-se um esquema de árvore estritamente binária. Cada *bin* da *binstring* determina o nó que deve escolhido desde o primeiro nó partindo da raiz até uma folha, quando há uma associação de valor de elemento sintático.

Há uma árvore binária para cada um dos tipos de macrobloco e submacrobloco:

- Tipo de macrobloco em *slices* P
- Tipo de macrobloco em *slices* B
- Tipo de macrobloco em *slices* I
- Tipo de submacrobloco em *slices* P
- Tipo de submacrobloco em *slices* B

Nos *slices* P e B, há um prefixo que determina decodificação de macrobloco I. Neste caso é usado na sequência a mesma árvore empregada em *slices* I (ver seção 2.3.1.2).

Na figura 17 é mostrada a binarização do tipo árvore binária para tipos de macroblocos em *slices* P. Nas folhas da árvore binária encontram-se os valores de elemento sintático correspondentes.

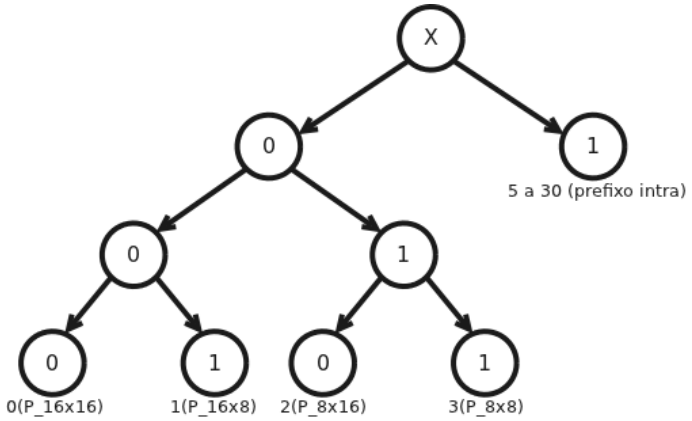

Figura 17: Binarização para tipo de macrobloco em *slices* P

3.3 Contextos de probabilidades

Valores de elementos sintáticos podem ser decodificados com o uso de diferentes contextos de probabilidade. Cada tipo de elemento sintático emprega um diferente conjunto de contextos, e a seleção neste conjunto é feita à nível de decisão binária (*bin*), com base em valores de elementos sintáticos anteriormente decodificados na vizinhança dentro do próprio *slice* ou *bins* anteriormente decodificados, possibilitando exploração de correlação.

O contexto de probabilidade é representado por uma variável de 7 *bits* indexada por *ctxIdx*. Esta variável é composta por 6 *bits* da variável *pStateIdx* que representa a probabilidade do símbolo menos provável (LPS) e 1 *bit* que informa o valor do símbolo mais provável (MPS).

3.3.1 Inicialização dos contextos

Na norma da versão do H.264/AVC adotada pelo SBTVD [1] são especificados 460 diferentes contextos de probabilidade. Os contextos devem ser inicializados com probabilidades baseadas no tipo de *slice* e seu parâmetro de quantização, logo antes do início de sua decodificação.

Os valores de *pStateIdx* e *valMPS* para cada índice de contexto são obtidos através das operações da figura 18. *Clip3(a,b,c)* é uma função que mantém o valor de *c* entre *a* e *b* verificando condições de saturação. *m* e *n* são inteiros obtidos de uma tabela de quatro selecionáveis. A seleção se dá de acordo com o tipo de *slice* e com um índice inicializador de 0 a 2 para *slices* P e B.

```
1. preCtxState = Clip3(1, 126, ((m * Clip3(0, 51, SliceQP))
                 >> 4) + n);
2. Se preCtxState <= 63 {
    pStateIdx = 63 - preCtxState;
    valMPS = 0;
} caso contrário {
    pStateIdx = preCtxState - 64;
    valMPS = 1;
}
```

Figura 18: Operações para inicialização dos contextos

3.3.2 Seleção de contexto para elementos sintáticos de resíduo

Dos 460 contextos presentes no H.264/AVC, 375 são usados para resíduo. Os índices de contexto para decodificação de elementos sintáticos são obtidos de acordo com a fórmula:

$$ctxIdx = ctxIdxOffset + ctxIdxBlockCatOffset + ctxIdxInc$$

ctxIdxOffset é fixo para o tipo de elemento sintático num dado tipo de estrutura (quadro ou campo). *ctxIdxBlockCatOffset* é fixo para o tipo de bloco de coeficientes que é decodificado: luminância AC, luminância DC, luminância com 16 coeficientes, crominância AC, crominância DC, luminância com 64 coeficientes (ver seção 2.3.4). *ctxIdxInc* é obtido para cada *bin* decodificado, conforme descrito a seguir.

3.3.2.1 ctxIdxInc para coded_block_flag

O elemento sintático *coded_block_flag* determina se um bloco de coeficientes é codificado ou contém somente coeficientes nulos e portanto isto não é necessário. Em blocos de coeficientes de tamanho 8x8 o elemento sintático *coded_block_flag* é inferido como '1' sem passar pelo decodificador de entropia.

ctxIdxInc para a decodificação de *coded_block_flag* é obtido de acordo com a fórmula abaixo:

$$ctxIdxInc = condA + 2 \times condB$$

condA é uma condição para o bloco vizinho da esquerda e *condB* uma condição para o bloco vizinho acima.

condA e *condB* são obtidos a partir das verificações:

```
Se é o primeiro bin a decodificar = 0 {
    Se numDecodAbsLevelGt1 é diferente de 0,
        ctxIdxInc = 0;
    caso contrário,
        ctxIdxInc = Min(4, 1 + numDecodAbsLevelEq1);
} caso contrário (não é o primeiro bin a decodificar) {
    Se o bloco é de coeficientes DC Chroma,
        ctxIdxInc = 5 + Min(3, numDecodAbsLevelGt1);
    caso contrário,
    ctxIdxInc = 5 + Min(4, numDecodAbsLevelGt1);
}
```

Figura 19: *ctxidxInc* para o elemento sintático *coeff_abs_level*

1. Se o bloco vizinho em questão não está disponível e o modo de predição do macrobloco atual é *Inter*, ou o bloco vizinho está disponível e seu *coded_block_flag* é igual a 0, a condição é 0.

2. Se o bloco vizinho em questão não está disponível e o modo de predição do macrobloco atual é *Intra*, ou o bloco vizinho está contido num macrobloco do tipo I_PCM, ou o bloco vizinho tem seu *coded_block_flag* é igual a 1, a condição é 1.

3.3.2.2 ctxIdxInc para significant_coeff_flag e last_significant_coeff_flag

Os elementos sintáticos *significant_coeff_flag* e *last_significant_coeff_flag* são usados para criar um mapa de significâncias para o bloco de coeficientes de resíduo que é decodificado, e seu *ctxIdxInc* da mesma forma.

Em blocos de coeficientes de tamanho 8x8, *ctxIdxInc* é obtido de uma tabela que tem como entrada o tipo de codificação do macrobloco (quadro ou campo) e a posição do coeficiente no bloco que o contém (índice).

Em outros blocos de coeficientes, *ctxIdxInc* é igual ao índice que determina a posição do coeficiente no bloco que o contém.

3.3.2.3 ctxIdxInc para coeff_abs_level

O elemento sintático *coeff_abs_level* indica o valor absoluto de um coeficiente, e é decodificado, um a um, após ser obtido todo o mapa de significâncias para o bloco de coeficientes.A seleção do índice de contexto se dá de acordo com o pseudo-código mostrado na figura 19.

numDecodAbsLevelGt1 é o número de coeficientes com módulo maior que 1 decodificados para o bloco atual. numDecodAbsLevelEq1 é o número de coeficientes com módulo igual a 1 decodificados para o bloco atual. *Min(a,b)* é uma função retorna o mínimo entre *a* e *b*.

3.3.3 Seleção de contexto para demais elementos sintáticos

Para os demais elementos sintáticos, o contexto é determinado de acordo com a fórmula

$$ctxIdx = ctxIdxOffset + ctxIdxInc$$

ctxIdxOffset é fixo para o tipo de elemento sintático e *ctxIdxInc* é o incremento obtido para cada *bin*. Este incremento é obtido de tabela que tem como entrada o tipo de elemento sintático que é decodificado e o *bin* que é decodificado. Conforme os *bins* são decodificados o incremento é maior. Para a maioria dos elementos sintáticos, *ctxIdxInc* para o primeiro *bin* é obtido dependendo de valores de elementos sintáticos vizinhos no *slice*.

3.3.3.1 ctxIdxInc para mb_skip_flag

Este elemento sintático indica se o macrobloco é do tipo *skip* e, portanto não se adiciona diferença de vetor de movimento ao predito nem resíduo. *ctxIdxInc* é obtido de acordo com a fórmula abaixo:

$$ctxIdxInc = condA + condB$$

condA é uma condição para o macrobloco vizinho da esquerda e *condB* é uma condição para o macrobloco vizinho acima. Elas são determinados a partir das verificações:

1. Se o macrobloco vizinho em questão não está disponível ou *mb_skip_flag* para ele é igual a 1, a condição é 0.

2. Caso contrário (se o macrobloco vizinho está disponível e *mb_skip_flag* para ele é 0), a condição é 1.

3.3.3.2 ctxIdxInc para mb_field_decoding_flag

ctxIdxInc para a decodificação de *mb_field_decoding_flag* é obtido a partir da fórmula:

$$ctxIdxInc = condA + 2 \times condB$$

condA é a condição para o par de macroblocos à esquerda e *condB* é a condição para o par de macroblocos acima. Elas são determinadas a partir das verificações:

1. Se o par de macroblocos vizinho não está disponível ou está estruturado como par de macroblocos como quadros, a condição é 0.

2. Caso contrário, a condição é 1.

3.3.3.3 ctxIdxInc para mb_type, intra_chroma_pred_mode e transform_8x8_flag

ctxIdxInc para a decodificação destes elementos é obtido a partir da fórmula:

$$ctxIdxInc = condA + 2 \times condB$$

condA e *condB* são condições para os macroblocos vizinhos à esquerda e acima, respectivamente. Elas são determinadas a partir das verificações:

1. Se o macrobloco vizinho está indisponível ou valor do elemento sintático do macrobloco vizinho é 0 (do tipo que está sendo decodificado), a condição é 0.

2. Caso contrário, a condição é 1.

3.3.3.4 ctxIdxInc para coded_block_pattern

Este elemento sintático é composto por duas partes, uma de 4 *bins* para luminância e outra de 2 *bins* para crominância. Cada uma destas partes (o prefixo e o sufixo) tem diferentes *ctxIdxOffset*.

No prefixo, cada um dos *bins* indica a presença de resíduo para um submacrobloco (8x8 amostras). O incremento é obtido a partir da fórmula:

$$ctxIdxInc = condA + 2 \times condB$$

condA e *condB* são condições para o submacrobloco à esquerda e acima, respectivamente. Elas são obtidas a partir das verificações:

1. Se o submacrobloco à esquerda não está disponível, ou seu tipo é *I_PCM* ou ele tem resíduo codificado, a condição é 0.

2. Caso contrário, a condição é 1.

No sufixo, é extraído, através do processo de de-binarização unária truncada, o valor *CodedBlockPatternChroma*. Se o valor é 0, não há coeficientes de crominância resíduo não-nulos; se o valor é 1, há coeficientes DC não-nulos; se o valor é 2 há coeficientes DC não-nulos e coeficientes AC não-nulos.

ctxIdxInc para a decodificação do primeiro *bin* do sufixo é obtido a partir da fórmula

$$ctxIdxInc = condA + 2 \times condB$$

condA e *condB* são as condições para o macrobloco vizinho à esquerda e acima, respectivamente. Elas são obtidas a partir das seguintes verificações:

1. Se o macrobloco vizinho não está disponível ou não tem coeficientes DC codificados, a condição é 0

2. Caso contrário a condição é 1.

ctxIdxInc para a decodificação do segundo *bin* do sufixo é obtido a partir da fórmula

$$ctxIdxInc = condA + 2 \times condB + 4$$

condA e *condB* são as condições para o macrobloco vizinho à esquerda e acima, respectivamente. Elas são obtidas a partir das seguintes verificações:

1. Se o macrobloco vizinho não está disponível ou não tem coeficientes AC codificados, a condição é 0

2. Caso contrário a condição é 1.

3.3.3.5 ctxIdxInc para ref_idx_l0 e ref_idx_l1

ctxIdxInc para a decodificação de *ref_idx_l0* e *ref_idx_l1* é obtido a partir da fórmula:

$$ctxIdxInc = condA + 2 \times condB$$

condA e *condB* são as condições para a partição de submacrobloco à esquerda ou acima, respectivamente. Considerando *ref_idx_lx ref_idx_l0* ou *ref_idx_l1*, conforme qual está sendo decodificado, as condições são obtidas a partir das verificações:

1. Se a partição vizinha está indisponível, ou não tem *ref_idx_lx* codificado ou *ref_idx_lx* para a partição vizinha é nulo, a condição é 0.

2. Caso contrário, a condição é 1.

Em quadros com MBAFF (ver seção 2.3.1.3), algumas vezes é necessário fazer correção temporal para referenciar *ref_idx_lx*: se o macrobloco atual é codificado como quadro e o macrobloco ao qual pertence a partição vizinha é codificado como campo, *ref_idx_lx* é dividido por 2. Assim, se seu valor original é menor que 2, ele é considerado nulo.

3.3.3.6 ctxIdxInc para mvd_l0 e mvd_l1

Seja *mvd_lx* igual a *mvd_l0* ou *mvd_l1*, conforme qual está sendo decodificado, *ctxIdxInc* é obtido a partir das comparações:

- Se (*absMvdCompA* + *absMvdCompB*) é menor que 3, *ctxIdxInc* é igual a 0.

- Se (*absMvdCompA* + *absMvdCompB*) é maior que 32, *ctxIdxInc* é igual a 2.

- Se (*absMvdCompA* + *absMvdCompB*) é maior ou igual a 3 e menor ou igual a 32, *ctxIdxInc* é igual a 1.

absMvdCompA e *absMvdCompB* são os valores absolutos da componente (x ou y) sendo atualmente decodificada do vetor de movimento da menor partição à esquerda ou acima da partição atual, respectivamente.

Em quadros com MBAFF (ver seção 2.3.1.3), pode ser necessário fazer correção temporal para referenciar *absMvdCompA* e *absMvdCompB* quando se decodifica a componente y, conforme abaixo:

- Se o macrobloco atual é estruturado como quadro e o macrobloco ao qual pertence a partição vizinha é do tipo campo, a componente tem seu valor duplicado.

- Caso contrário, se o macrobloco atual é estruturado como campo e o macrobloco ao qual pertence a partição vizinha é do tipo quadro, a componente tem seu valor dividido por 2.

- Em qualquer outro caso, a componente mantém o seu valor.

4 METODOLOGIA

O desenvolvimento de um decodificador de vídeo H.264/AVC abriga um alto grau de complexidade. Nos capítulos anteriores o H.264/AVC em geral e o CABAD isoladamente foram apresentados com simplificações. A especificação rigorosa e completa do padrão é extensa. Assim, faz-se necessário seguir uma metodologia para promover a sua implementação em *hardware*.

O decodificador H.264/AVC é desenvolvido de forma modular e incremental por uma equipe. Cada bloco que o compõe é elaborado isoladamente por um projetista, com exceção de que sua interface com demais módulos deve ser prevista e estabelecida num acordo.

Muitas vezes, seguindo uma prática comum para sistemas digitais complexos, o projeto parte de um modelo previamente desenvolvido em *software*, tal como em linguagem C ou *Matlab*. Para se desenvolver o modelo e o módulo em *hardware*, deve-se estudar a norma que o descreve tecnicamente e fazer um levantamento de suas especificações funcionais e de desempenho.

Durante o projeto, os módulos desenvolvidos são validados isoladamente através de verificações funcionais, aplicando sinais a suas entradas e comparando-se as saídas geradas com valores esperados. Os dados de referência são obtidos de um *software* que pode ser seu próprio modelo.

Sínteses para FPGA são feitas também para verificar o correto funcionamento do sistema digital num protótipo e para gerar resultados de área, desempenho.

Após o desenvolvimento com validação de forma isolada, segue a etapa de integração de módulos. Da mesma forma é feita validação e síntese em FPGA do sistema em conjunto.

Nas próximas seções são apresentadas as normas que foram consultadas para o desen-

volvimento do trabalho, o fluxo de projeto e as ferramentas essenciais e auxiliares para desenvolvimento.

4.1 Normas

As normas técnicas estudadas para realização deste trabalho foram a norma internacional que especifica o padrão H.264/AVC e a norma brasileira que trata da codificação de vídeo para o SBTVD. Nas próximas seções são apresentados resumos do que trata cada norma, observações importantes e requisitos de desempenho que são estabelecidos por elas.

4.1.1 ITU-T

A norma ITU-T [1] especifica tecnicamente todos os processos envolvidos na decodificação de vídeo H.264/AVC. Porém ela não determina como implementá-los, qualquer decodificador que trate um fluxo de *bits* gerando saídas idênticas está em conformidade. A norma também não especifica como é realizada a codificação.

Ela é utilizada como referência para estabelecer as especificações funcionais que os módulos CABAD e *parser* devem atender, definindo os seus comportamentos.

4.1.2 ABNT

A norma ABNT de codificação de vídeo [6] adota a norma ITU-T como base, definindo o H.264/AVC como padrão para o SBTVD. Adicionalmente, são aplicadas restrições a transmissão de conteúdo a portáteis e televisores fixos, especificando para cada alvo principalmente:

- Formatos (entrelaçado ou progressivo), resoluções e taxas de exibição permitidas
- Os perfis e os níveis de codificação que podem ser utilizados
- Restrições de parâmetros de configuração

4.1.3 Requisitos de desempenho

A norma ITU-T em conjunto com a norma ABNT definem o desempenho necessário para o projeto do decodificador de vídeo H.264/AVC. Este trabalho considera os limites máximos para transmissões *Full-seg*, que se dá no nível 4 de codificação. A tabela 3

sumariza as especificações de interesse para a verificação do desempenho do CABAD integrado ao *parser*.

Tabela 3: Limites de interesse do nível 4 de codificação

Taxa máxima de processamento de macroblocos (MB/s)	245.760
Tamanho máximo do quadro em unidades de macroblocos (MBs)	8.192
Taxa de *bits* do vídeo codificado (Mbps)	20

Atendendo condição de pior caso, o decodificador deve atingir os limites máximos. A taxa máxima de processamento de macroblocos está diretamente relacionada ao tamanho máximo do quadro em unidades de macroblocos por um fator de 30, que é taxa típica de quadros por segundo.

O tempo de decodificação de todos os elementos sintáticos de um macrobloco pelo CABAD é muito variável, pois depende da quantidade de *bins* associadas a estes elementos sintáticos e da taxa de processamento de *bins* por segundo que o CABAD atinge. Quando muitos dados estão contidos num fluxo de *bits* para corrigir os resultados de predição (principalmente coeficientes de resíduo e vetores diferenciais de movimento), a quantidade de *bins* por consequência é elevada. É o que normalmente ocorre no primeiro quadro decodificado sem referência a imagens anteriores, no qual somente pode-se explorar redundâncias espaciais na própria imagem. É necessário processar uma quantidade muito grande de macroblocos além do primeiro quadro para chegar a uma relação de *bins* por macrobloco global do vídeo e verificar o desempenho do CABAD. Esta medida portanto não é a mais adequada.

A taxa de consumo de *bits* na entrada do *parser*, por sua vez, pode ser verificada. Quando se usa o CABAD, esta taxa depende da relação de *bins* por *bit*, sendo a taxa de *bins* por segundo mais constante e um critério de desempenho para sua arquitetura. A última medida é portanto a mais importante para verificação de desempenho para garantir a conformidade com as normas.

4.2 Ferramentas

Projetos de sistemas digitais de grande porte exigem o uso de ferramentas de CAD como meio para realizá-los. Nesta seção são expostos a linguagem de descrição de *hardware* VHDL e os *softwares* que foram empregados para modelamento, simulação comportamental, síntese lógica e validação.

4.2.1 VHDL

VHDL (*VHSIC hardware description language*) é uma linguagem que pode ser utilizada para descrever à nível comportamental ou estrutural o modelo lógico de um circuito digital. Ela permite lidar com o paralelismo inerente em projeto de *hardware*, com a descrição de sistemas concorrentes. Um arquivo VHDL representa um módulo com uma parte de 'entidade' que define sua interface e outra de 'arquitetura' que descreve seu comportamento. Os módulos podem ser sintetizados lógica e fisicamente, dependendo da etapa no projeto. No caso de síntese física, os alvos podem ser FPGA ou ASIC.

VHDL também pode ser usado para gerar *testbenches* não sintetizáveis, mas que são utilizados para executar uma simulação comportamental de um módulo que instancia com formas de onda. O *testbench* pode executar funções de leitura e escritas de arquivos, viabilizando comparações de dados para automatizar o processo de validação.

4.2.2 Ferramentas de simulação

A simulação comportamental foi feita com a combinação das ferramentas GHDL e gtkWave, ambos *softwares* livres. A primeira ferramenta é usada para analisar os arquivos módulos VHDL, elaborar o projeto e executar um *testbench* salvando um arquivo de forma de onda. A forma de onda é aberta com a segunda ferramenta de forma a possibilitar a visualização dos sinais.

4.2.3 Programas de referência

Um grupo de programadores da ITU-T desenvolveu o *software* JM [8] para ser utilizado como referência para a validação de codificadores e decodificadores de vídeo no formato H.264/AVC. O *software* além de possibilitar a codificação de vídeos com extensas possiblidades de configuração, possui código aberto. Todavia, a estrutura do código do *software* JM é bastante intrincada, inviabilizando-o como modelo para *hardware*. Por esse motivo, foi criado o decodificador PRH.264 que no momento da publicação deste trabalho era capaz de decodificar vídeos no formato progressivo até o perfil *main* [9]. Este *software* é desenvolvido em linguagem C com organização modular e uma interface gráfica implementada em C++ conforme mostrado na figura 20.

Na interface do PRH.264 podem ser vistas opções para salvar em formato de texto dados intermediários do processo de decodificação de vídeo. Estes dados são utilizados

na validação dos módulos através de *testbenches* que envolvam comparações e contagem de erros.

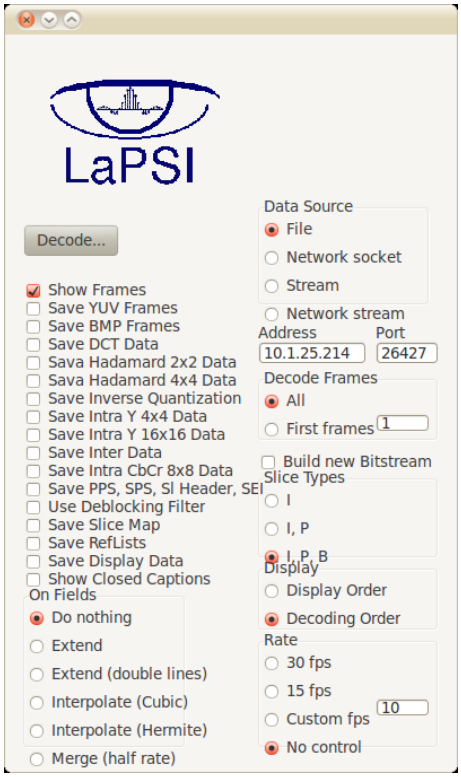

Figura 20: Interface do PRH.264

4.2.4 Placa de prototipação e ferramenta de síntese

O decodificador de vídeo H.264/AVC, antes e após a integração, tem seus módulos VHDL sintetizados para o FPGA da Xilinx Virtex-5 XC5VLX110T [4]. Quando o decodificador com seus módulos integrados é sintetizado e o FPGA é programado com fluxo de *bits*, tem-se um protótipo do qual pode-se demonstrar o funcionamento utilizando uma saída de vídeo. Na figura 21 é mostrada a placa. O *software* utilizado para fazer a síntese e gerar o fluxo de *bits* é o *Xilinx ISE* 10.1.

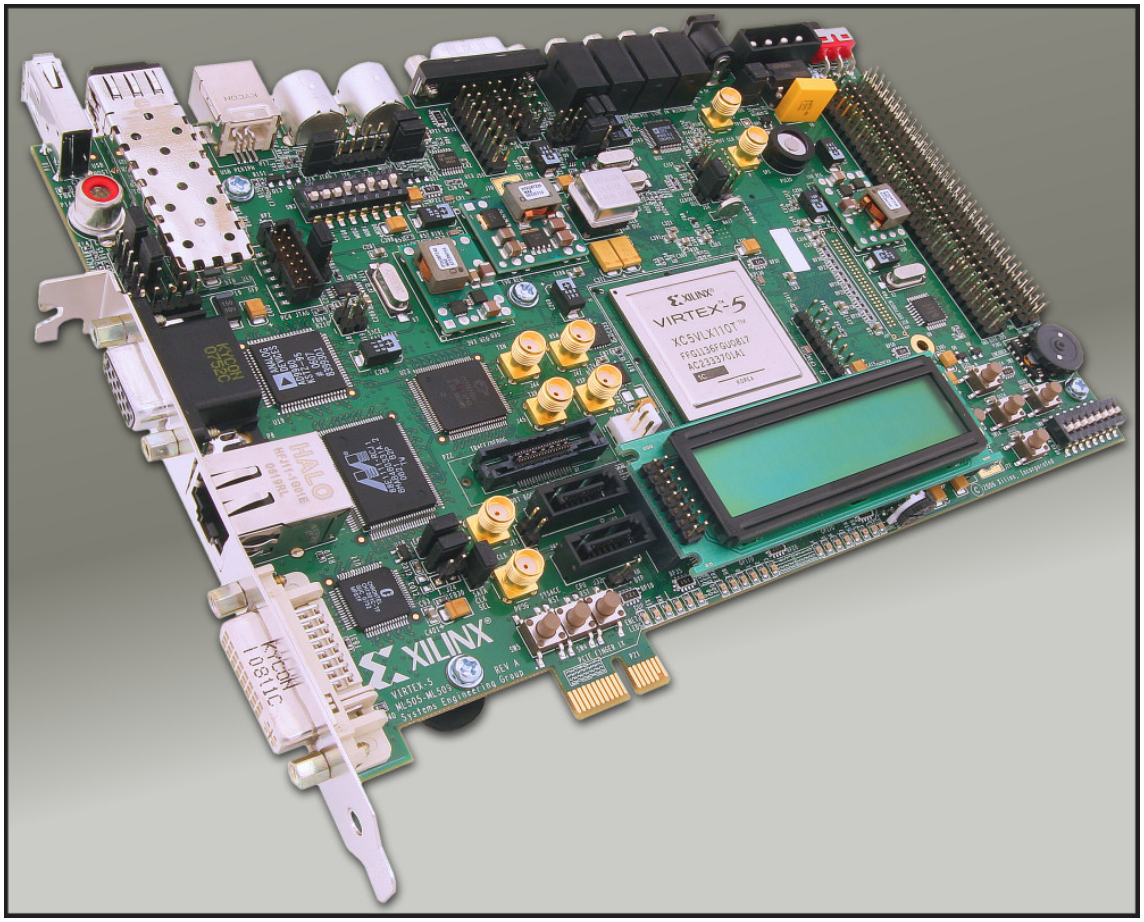

Figura 21: Placa Xilinx Virtex-5 XC5VLX110T [4]

5 INTEGRAÇÃO E DESENVOLVIMENTO DO CABAD

Este projeto delimita-se no *parser*, dentro do qual estão presentes os decodificadores de entropia do decodificador de vídeo H.264/AVC. Na integração buscou-se aproveitar módulos anteriormente desenvolvidos que se relacionam ao CABAD. O principal deles é responsável por realizar as funções mais essenciais de decodificação aritmética binária, de-binarização e seleção do contexto, denominado neste trabalho como núcleo do CABAD.

Na seção seguinte são discutidos os módulos de integração adotados, modificações necessárias e interface. A seguir nas próximas seções são tratados os módulos que foram necessários desenvolver para suprir necessidades da integração e no fim do capítulo uma verificação de funcionamento através de simulação.

5.1 Módulos para integração

Os módulos para integração são desenvolvidos na própria equipe que desenvolve o decodificador H.264/AVC ou em outras equipes do projeto Rede H.264, em especial a partir de projetos de diplomação e dissertações de mestrado anteriores feitos. Para a integração do CABAD foram considerados o módulo do *parser* e arquiteturas para o CABAD.

5.1.1 *Parser*

O *parser* do decodificador H.264/AVC [10] foi desenvolvido de forma a prever a futura integração de demais módulos. Ele fornece suporte completo para a decodificação de elementos sintáticos presentes até o nível *high*.

Sua arquitetura é composta por um módulo de controle geral que implementa uma máquina de estados finita para, conforme o tipo de NAL detectada na entrada, coordenar

a habilitação da decodificação de elementos sintáticos pelos módulos:

- Decodificador SPS
- Decodificador PPS
- Decodificador *slice header*
- Decodificador *slice data*

Os dados da NAL são injetados numa FIFO e desta passam para o módulo *barrel shifter*, que fornece dados do fluxo de *bits* aos módulos listados acima e aos decodificadores de entropia. Através de uma entrada de *bits* consumidos, o ponteiro no fluxo de *bits* é deslocado em um ciclo de relógio.

Na figura 22 é exibida a arquitetura do *parser*, incluindo a integração do CABAD. Este faz interface direta com o módulo decodificador *slice data*, recebendo deste sinais de comando. Os dados retornados são os valores dos elementos sintáticos decodificados, os coeficientes de resíduo e o número de *bits* consumidos.

Os sinais de comando para o CABAD são dados na tabela 4. Para solicitar uma decodificação de elemento sintático o decodificador *slice data* deve habilitar o CABAD, informar o tipo de elemento sintático e mais outras informações necessárias.

Tabela 4: Sinais de comando ao CABAD enviados pelo decodificador *slice data*

Nome do sinal	**Objetivo**	**Num. de *bits***
enable	habilitação	1
ack	reconhecimento da decodificação feita pelo CABAD	1
se_type_id	tipo de elemento sintático	5
chroma	0 - luminância, 1 - crominância	1
loop_idx	índice de laço de decodificação	5
mb_part_idx	índice de partição de macrobloco	3
lx	índice de lista de referência (0 ou 1)	1
comp_idx	índice de componente de vetor de movimento (x ou y)	1

Após decodificação de elemento sintático, o CABAD envia um sinal de 'pronto'. Neste momento se o decodificador *slice data* estiver pronto para recuperar o valor decodificado, o sinal de reconhecimento (*ack*) está em alto.

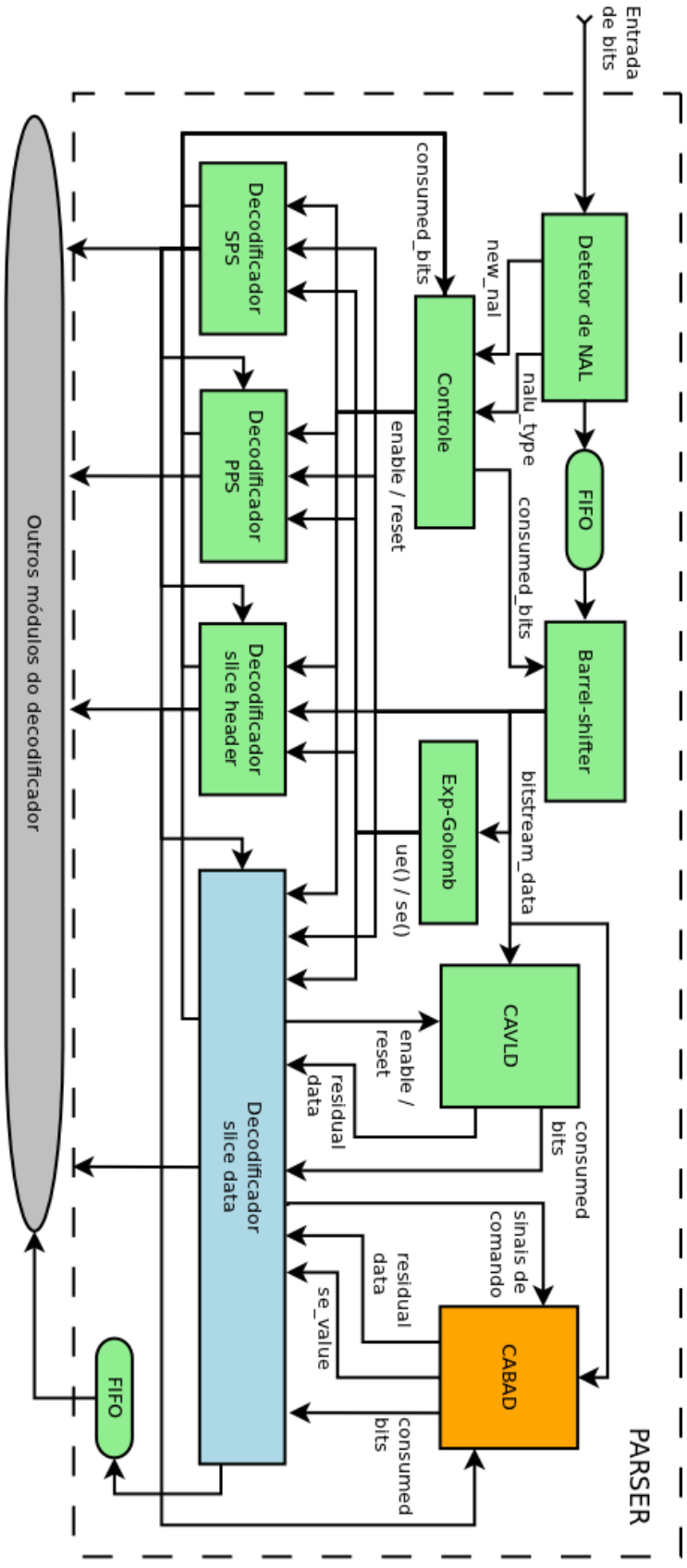

Figura 22: Arquitetura do *parser*

5.1.2 Núcleo do CABAD

Para este trabalho foram considerados duas alternativas de decodificadores de entropia CABAC para integração, conforme descrito abaixo. Ambos trabalhos foram desenvolvidos em dissertações de mestrado e careciam de desenvolvimento adicional para serem completos.

5.1.2.1 Trabalho de Deprá, 2009

O decodificador de entropia CABAC, ou CABAD, proposto por Deprá [11], é uma arquitetura concebida com decisões baseadas num extensivo estudo estatístico com diversas sequências de vídeo codificadas com o JM [8], variando-se resolução e parâmetro de quantização. O resultado deste estudo reuniu informações tais como os tipos de codificações aritméticas binárias mais frequentes (*regular*, *bypass*, ou *terminate*), o número mínimo, máximo e médio de *bins* produzidos por tipo de elemento sintático, entre outros dados. Tendo em vista as informações obtidas, Deprá determinou o número de instâncias de cada tipo de máquina de codificação aritmética binária para obter ganhos de desempenho de forma eficiente. Segundo o autor, o CABAD é um gargalo no processo de decodificação de vídeo, e deve ser explorado para atingir desempenho exigido para decodificar vídeo de alta resolução em tempo real.

Porém, o decodificador de Deprá, não apresenta suporte completo a macroblocos do tipo P e B nem suporte a vídeos entrelaçados ou com o uso de MBAFF, exigindo ainda extenso desenvolvimento nas parte de seleção de modelos de contextos e de-binarização. O trabalho inclui porém inicialização das variáveis de contexto e um controle da sequência de elementos sintáticos suportados pela arquitetura a serem decodificados. Este controle já é implementado no *parser* adotado de forma a suportar todos os elementos sintáticos até o perfil *High*, tanto quando o decodificador de entropia usado é o CAVLD quando o CABAD.

5.1.2.2 Trabalho de Carvalho, 2009

O trabalho de Carvalho [12] propõe uma arquitetura com múltiplas máquinas aritméticas binárias assim como o de Deprá. Em alguns casos, é possível decodificar dois *bins* em um ciclo.

Tendo em vista o curto tempo para integração do CABAD e a compatibilidade com o

parser, esta arquitetura foi adotada, pois apresenta um maior suporte aos elementos sintáticos do padrão H.264/AVC, tendo apenas não sido validada com vídeos entrelaçados. A concepção de Carvalho ao contrário da de Deprá, não inclui o controle que já se encontra no *parser*, que determina a sequencia dos elementos sintáticos a serem decodificados, facilitando a integração do decodificador como um módulo escravo.

Para a decodificação de cada elemento sintático, Carvalho definiu o seguinte ciclo de operação para seu decodificador:

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

Para decodificação de coeficientes de resíduo, a operação é diferente. Quando um bloco de coeficientes é codificado, faz-se solicitação do mapa de significâncias para este bloco. O decodificador de Carvalho aciona o sinal de *coeff_reset* para zerar uma memória externa, onde será armazenado o mapa. Após, o mapa de significâncias é decodificado e disponibilizado através das portas *coeff1* e *coeff2*. O valor da porta *coeff_index* fornece a posição no mapa de significâncias do valor das porta *coeff1*, enquanto que a posição de *coeff2* é indexada por *coeff_index* + 1.

No projeto de integração, o o decodificador de Carvalho é denominado como núcleo do CABAD. Ele não contempla:

- Inicialização das memórias de contexto
- Inicialização do intervalo de decodificação *Range* e do código do fluxo de *bits Offset*
- Gerenciamento direto do fluxo de *bits*

- Armazenamento de elementos sintáticos anteriormente decodificados e fornecimento de valores vizinhos
- Construção completa do bloco de coeficientes de resíduo

5.1.3 Inicializador de contextos

O inicializador de contextos para integração foi tomado do trabalho de Deprá [11], fazendo-se pequenas alterações. A arquitetura do inicializador emprega tabelas para os valores de m e n, e divide as operações aritméticas necessárias (ver seção 3.3.1) em quatros estágios de *pipeline* que são executados para os 460 contextos definidos na ITU. São necessários, portanto, 464 ciclos para o estágio de inicialização de contextos.

5.2 Módulos desenvolvidos

Neste trabalho, o CABAD é composto pelo seu núcleo acrescido de outros módulos que foram necessários ser desenvolvidos e integrados além do inicializador de contextos. A figura 23 ilustra simplificadamente os blocos que compõem a sua arquitetura bem como seus sinais de interface. Os blocos em azul foram adotados para integração e tiveram pequenas modificações. Os blocos em laranja foram desenvolvidos por completo. À esquerda tem-se os sinais de entradas para o CABAD e à direita os sinais de saída.

A integração do CABAD ao *parser* exigiu o desenvolvimento de módulos porque havia necessidades na arquitetura de Carvalho e na sua integração que não são contempladas (ver seção 5.1.2.2). Um módulo de controle geral foi desenvolvido para coordenar as etapas de inicialização do CABAD, o ciclo de operação definido por Carvalho e a sincronização com o *parser*. Ele é descrito nas próximas seções, assim como o gerenciador de fluxo de *bits*, o gerenciador de elementos sintáticos e o construtor de blocos de coeficientes de resíduo.

5.2.1 Controlador e sincronizador

O módulo foi desenvolvido para sincronizar as solicitações de elementos sintáticos pelo *parser*, na camada de dados do *slice* com as etapas de inicialização dos contextos, inicialização de códigos e intervalos, e decodificação com o CABAD. Denominado controle do CABAD, ele implementa a máquina de estados representada na figura 24, podendo resetar e habilitar cada um dos módulos da integração.

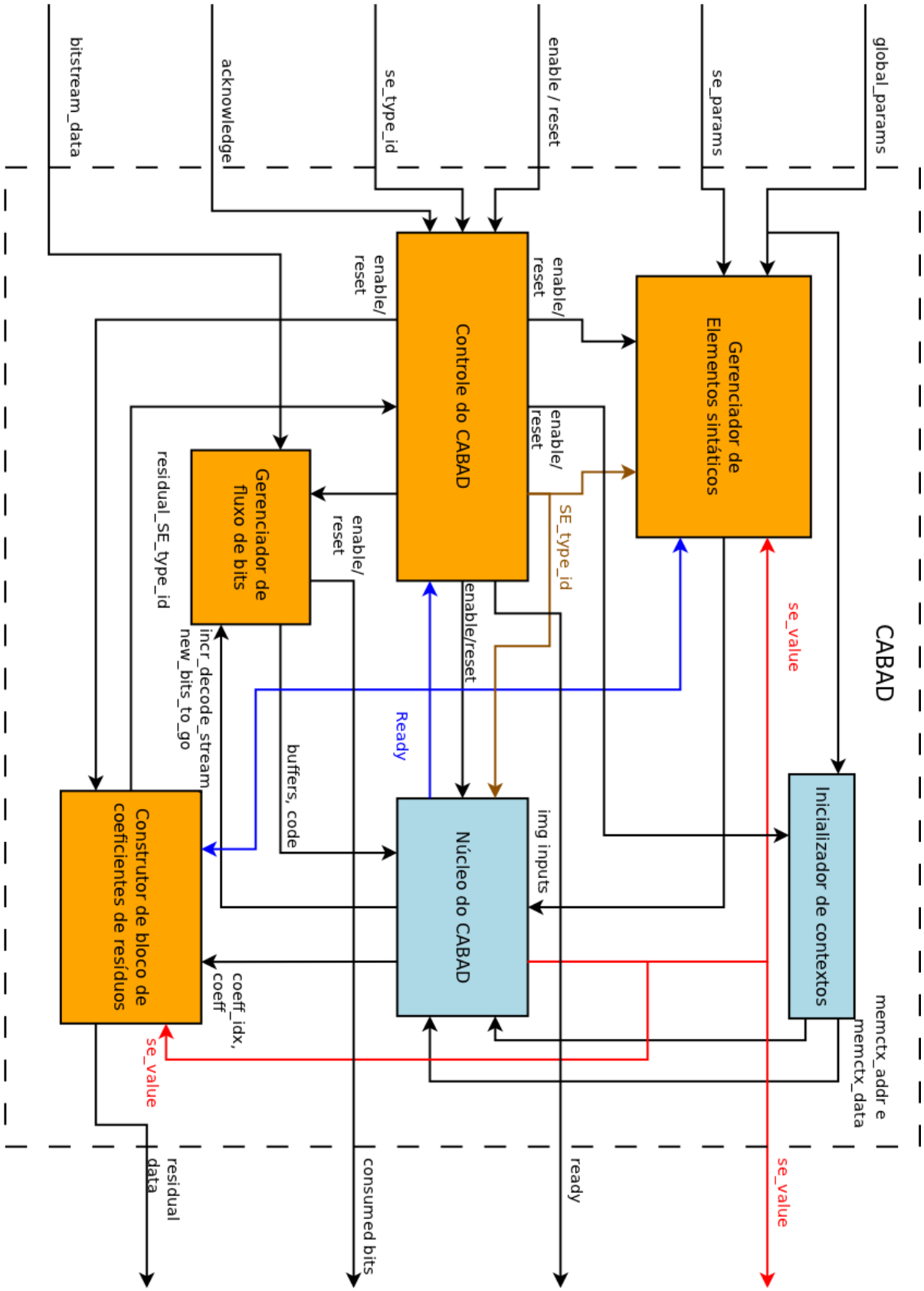

Figura 23: Blocos e interfaces da arquitetura do CABAD

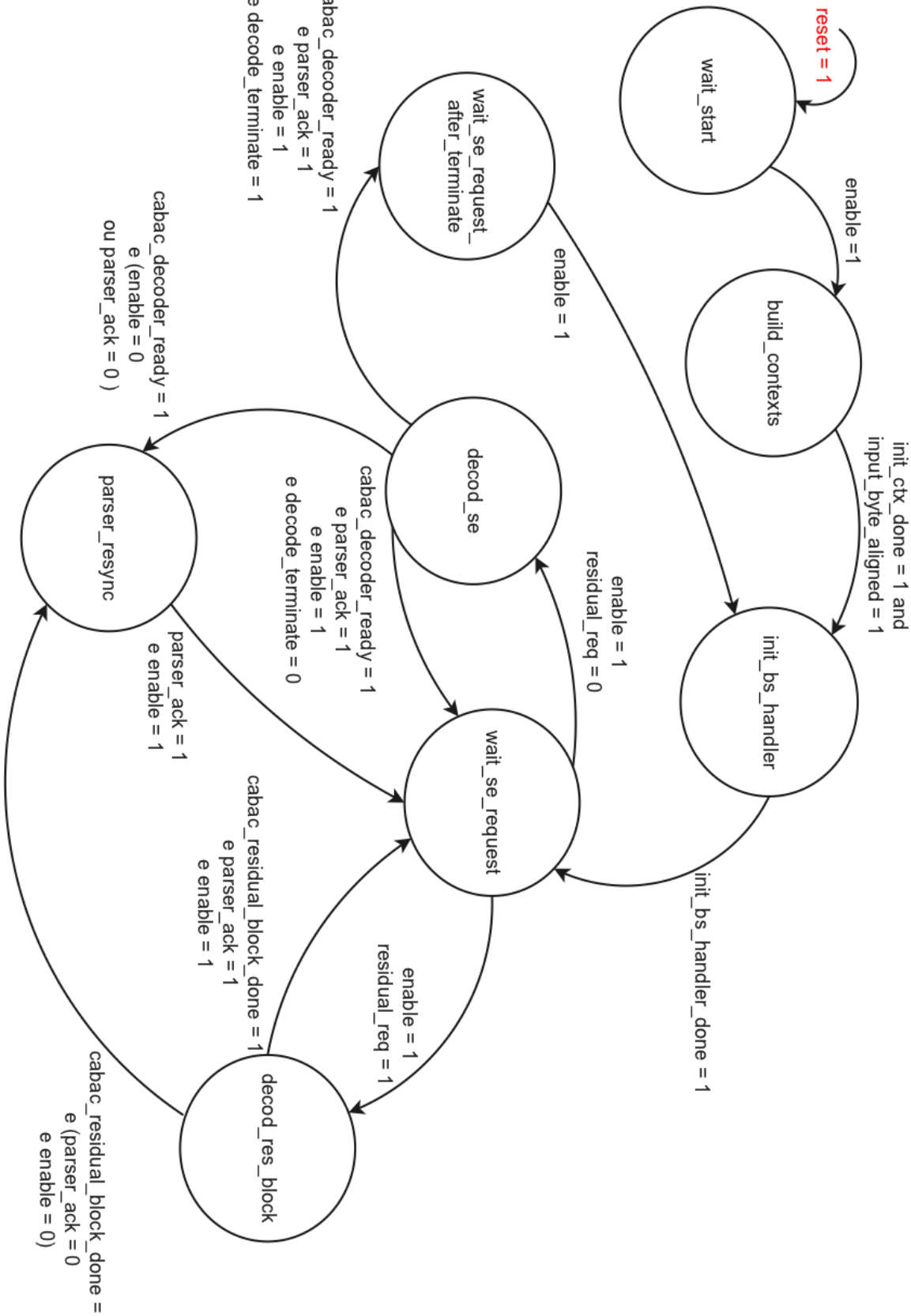

Figura 24: Diagrama de estados do módulo controle do CABAD

5.2.1.1 wait_start

No primeiro estado, que se entra com o *reset* do CABAD, o controle aguarda início da decodificação de dados do *slice* com a habilitação pelo *parser*. Os demais módulos do CABAD estão resetados. Quando o controle recebe o sinal de habilitação pelo *parser*, o ele passa para o estado *build_contexts*.

5.2.1.2 build_contexts

Neste estado, o módulo de inicialização das variáveis de contextos é mantido habilitado até que ele envie um sinal de pronto (*init_ctx_done* = 1) e o fluxo de *bits* esteja alinhado a *bytes*. Durante este estado todas as variáveis de contexto necessárias para a decodificação de elementos sintáticos são inicializadas, tendo como entrada os valores *slice_QP*, *slice_type* e *cabac_init_idc*, obtidos no cabeçalho do *slice*.

5.2.1.3 init_bs_handler

No estado de inicialização do gerenciador de fluxo de *bits*, são lidos 9 *bits* do código no fluxo de *bits*. Este valor, *codIOffset*, representa a localização do código no intervalo de codificação aritmética. Ao valor que representa o tamanho do intervalo, *codIRange*, é atribuído o valor 510.

Além disso, neste estado é inicializado o ponteiro do bitstream para a posição do *bit* no *byte* atual.

5.2.1.4 wait_se_request

O estado *wait_se_request* é um estado de espera do controle, em que o CABAD já foi inicializado. Durante este estado, o núcleo do CABAD é mantido resetado, conforme exige o seu ciclo de operação, e as entradas correspondentes ao elemento sintático a ser decodificado são obtidas pelo módulo gerenciador de elementos sintáticos (ver seção 5.2.3).

Quando o *parser* enviar solicitação de decodificação de elemento sintático com o sinal de habilitação do CABAD, o controle entra num estado de decodificação, dependendo se há ou não solicitação por decodificação de resíduo (*resid_req*).

5.2.1.5 decod_se

No estado de decodificação de elemento sintático, o núcleo do CABAD é mantido habilitado até que ele conclua a sua tarefa com *cabac_decoder_ready* = 1. Neste momento, se o *parser* estiver pronto para recuperar o valor do elemento sintático decodificado, indicando *parser_ack* = 1, retorna-se para o estado *wait_se__request*. Caso contrário, o valor do elemento sintático é guardado num registrador e entra-se a seguir no estado *parser_resync*.

Num caso incomum, pode ser decodificada uma decisão de término de decodificação com o CABAD (*decode_terminate*), o que ocorre no final do *slice* ou quando um macrobloco do tipo *I_PCM* é decodificado. O controle entra no estado *wait_se_request_after_terminate*.

5.2.1.6 decod_res_block

Este é o estado de decodificação de resíduo. Neste estado o núcleo do CABAD passa a ser controlado pelo módulo construtor de blocos de coeficientes de resíduo, o *residual_block*, que também tem ligação direta com a FIFO de resíduos. O CABAD implementa um algoritmo para obtenção de blocos de resíduo próprio, que é abstraído pelo *parser*, portanto os elementos sintáticos passam a ser solicitados pelo *residual_block*.

5.2.1.7 parser_resync

Neste estado, o valor do último elemento sintático é mantido num registrador especial para ser disponibilizado ao *parser*. O núcleo do CABAD é desabilitado e mantido resetado como exige o seu ciclo de operação. O controle permanece neste estado até que o *parser* levante o sinal *parser_ack* para fazer transição ao estado *wait_se_request*.

5.2.1.8 wait_se_request_after_terminate

O controle mantém-se neste estado enquanto o emprego do CABAD é interrompido com o sinal *enable = 0* que deve ser estabelecido pelo *parser*. O núcleo do CABAD é mantido resetado. Se o *parser* habilitar o CABAD sem antes resetá-lo, ele é reinicializado sem realizar a reinicialização dos contextos, mudando para o estado *init_bs_handler*.

5.2.2 Gerenciador de fluxo de *bits*

O gerenciador de fluxo de *bits* é responsável por marcar os *bits* consumidos pelo CABAD e inicializar os códigos de intervalo [13]. Logo após a sua inicialização, ele lê

9 *bits* para o valor da variável *codIOffset* e atribui o valor 510 a *codIRange* conforme é determinado em norma. Após a inicialização, esses valores são recuperados do núcleo do CABAD, sendo necessário fornecer num ciclo de operação seguinte.

O gerenciador do fluxo de *bits* os fornece 3 primeiros *buffers* conforme exige a interface de Albuquerque [12]. Para atualizá-los, o núcleo do CABAD determina as saídas *new_incr_decode_stream_lenght* para informar um incremento no ponteiro do fluxo de *bits* à nível de *byte* e *new_bits_to_go* para informar em que *bit* encontra-se no o ponteiro no *byte* atual. O gerenciador então acompanha este sinais a atualiza os *buffers* para o CABAD em em tempo real. Sua parte operativa é mostrada na figura 25

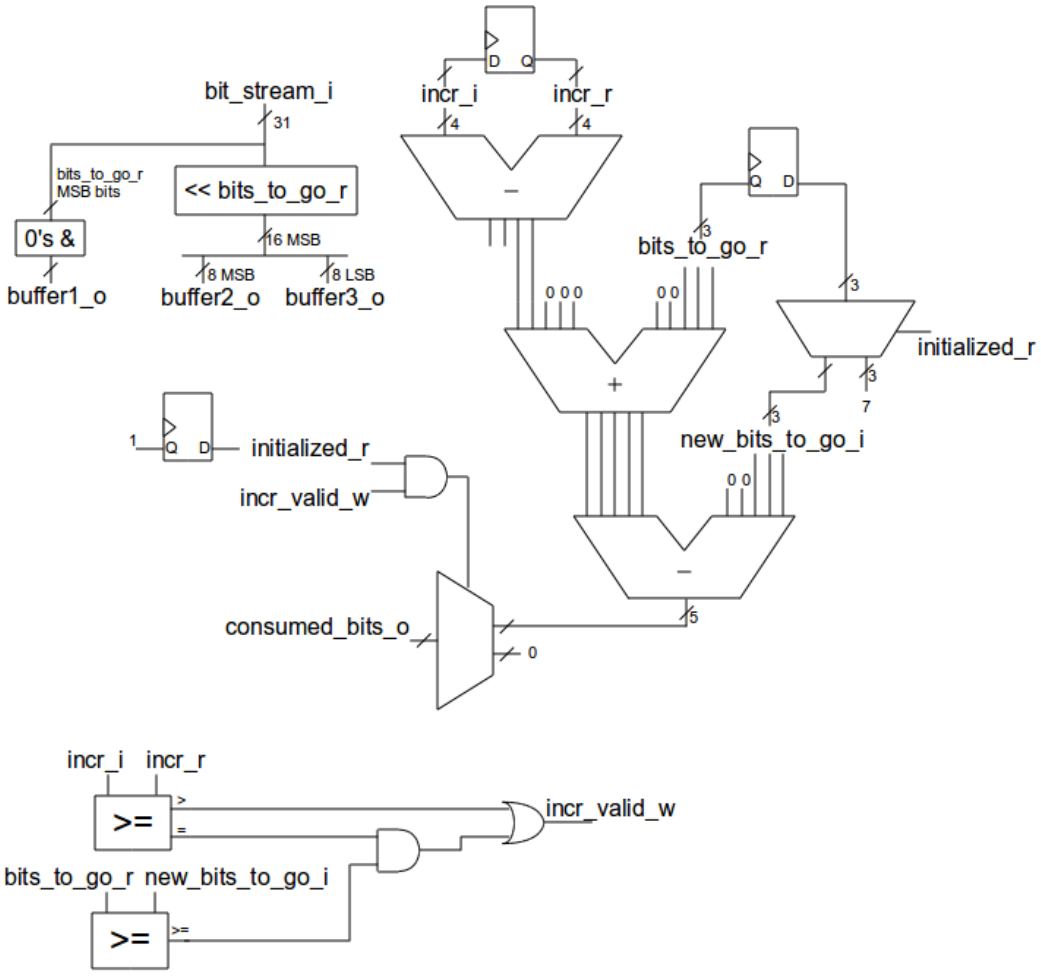

Figura 25: Parte operativa do gerenciador de fluxo de *bits*

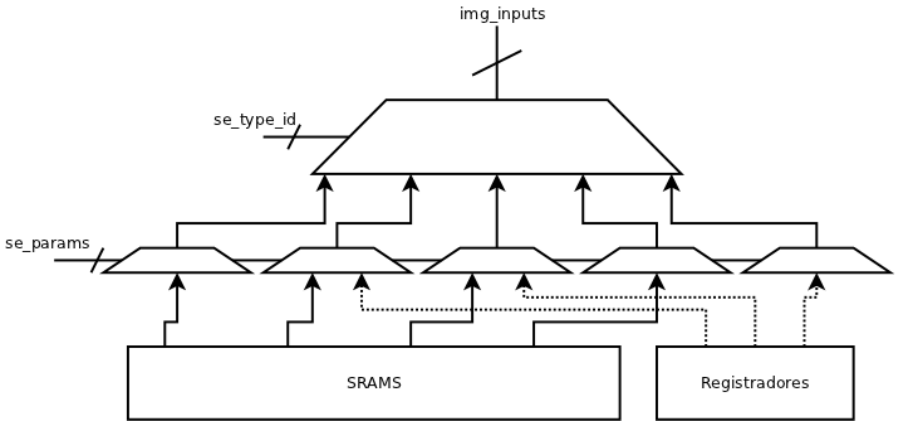

Figura 26: Visão geral da arquitetura do gerenciador de elementos sintáticos

5.2.3 Gerenciador de elementos sintáticos

O gerenciador de elementos sintáticos é o módulo responsável por fornecer as entradas necessárias para seleção do contexto pelo CABAD. Para a maioria dos tipos de elementos sintáticos a seleção do contexto toma como base os valores decodificados na vizinhança acima e à esquerda (ver seção3.3). Portanto, valores de elementos sintáticos devem ser armazenados.

A determinação da forma de armazenamento de valores de elementos sintáticos deve levar em consideração as suas possibilidades de serem referenciados como vizinhos. Eles podem ser armazenados em registradores ou memórias SRAM. DRAM não é apropriado pois como é mostrado adiante a quantidade de dados não é tamanha que inviabilize o uso dos outros dois tipos de memória por ocupação excessiva de área. Os registradores são usados para armazenar dados que devem ser lidos rapidamente num curto período de tempo após o seu armazenamento. As SRAM são usadas para armazenar mais informações, que são lidas um período de tempo mais longo após seu armazenamento.

Na figura 26 é mostrado uma visão geral da arquitetura do módulo gerenciador de elementos sintáticos. Parâmetros do elemento sintático passados pelo *parser* (*se_params*) são entradas para selecionar para cada tipo de elemento sintático um valor em memória SRAM ou registrador que corresponda á vizinhança. O tipo de elemento sintático determina quais valores serão colocados na saída *img_inputs* para o núcleo do CABAD selecionar o contexto.

A quantidade de valores de elementos sintáticos que devem ser armazenada depende da largura do quadro, pois para um macrobloco pode-se referenciar o valor de elemento sintático de um macrobloco acima e a ordem de decodificação é da esquerda para direita

para então começar uma linha nova de macroblocos ou pares de macroblocos. Quando há emprego de MBAFF (ver seção 2.3.1.3) o macrobloco vizinho deve ser selecionado entre um dos dois que compõem o par de macroblocos acima. A figura 27 mostra um exemplo em que o vizinho acima é o do topo do par de macroblocos.

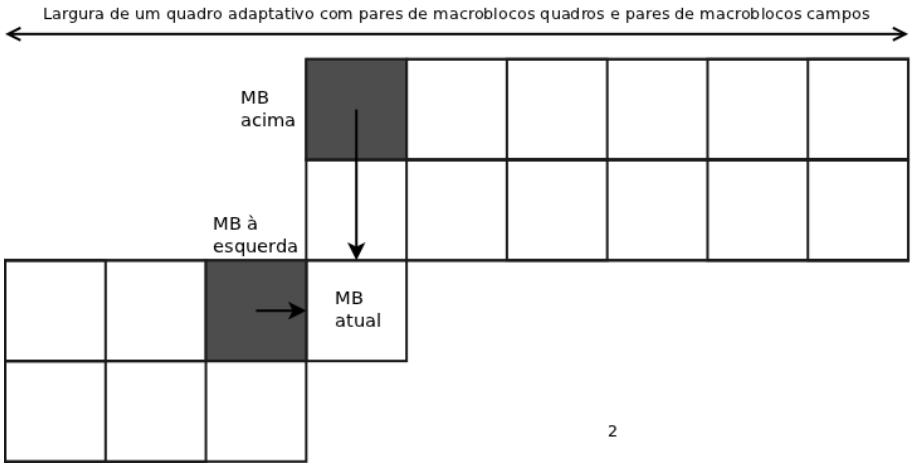

Figura 27: Exemplo de vizinhança de macroblocos com uso de MBAFF

5.2.3.1 Armazenamento de valores de elementos sintáticos em SRAM

Como o acesso individual a um macrobloco do par é independente do outro do mesmo par, foi arbitrado que uma palavra da memória comporia as informações apenas de um macrobloco. Assim, sua largura é definida como duas vezes a largura da imagem em número de macroblocos. Considerando um vídeo de máxima resolução (1920 x 1080) pixels, a largura da imagem é de 120 macroblocos, portanto a SRAM deve comportar 240 endereços. Logo a largura do endereço é de 8 *bits*.

Como mostrado na figura 27, elementos sintáticos podem ocorrer mais de uma vez por macrobloco quando tem-se sub-particionamento em submacroblocos (8x8), blocos (4x4) ou partições de tamanho variável. No caso destas, são feitas replicações para o menor tamanho de partição possível para o dado tipo de elemento sintático. Quando necessário, índices de referência (*ref_idx*) são replicados para submacroblocos e vetores diferenciais (*mvd*) de movimento para blocos.

Os elementos sintáticos de ocorrência única em um macrobloco foram agrupados, compondo uma palavra de 20 *bits* como mostrado na tabela 5. Para esta memória os elementos sintáticos decodificados são usados como referência de vizinhos acima da mesma forma de como vizinhos à esquerda, como pode ser visto na figura 28. Por esse motivo

ela foi feita de porta dupla para acessar a ambas vizinhanças ao mesmo tempo.

Tabela 5: Palavra de memória para elementos sintáticos de ocorrência única no macrobloco - *mem_mb_info*

Tipo de elemento sintático	Número de *bits*
mb_skip_flag	1
mb_type	6
chroma_intra_prediction_mode	3
coded_block_pattern	6
transform_size_8x8_flag	1
coded_block_flag Luminância DC	1
coded_block_flag Crominância DC Cb	1
coded_block_flag Crominância DC Cr	1
Total	20

O elemento sintático *coded_block_flag* ocorre mais de uma vez em cada macrobloco para blocos de coeficientes AC e portanto foi agrupado conforme está mostrado na tabela 6. É necessário armazenar o valor do elemento sintático para os 4 blocos de coeficientes de luminância AC, os 2 blocos de coeficientes de crominância AC Cb e os 2 blocos de coeficientes de crominância Cr da parte inferior do macrobloco. A memória para guardar valores deste tipo de elemento sintático possui uma portanto uma largura de palavra de 8 *bits*.

Tabela 6: Palavra de memória para *coded_block_flag* AC- *mem_cbf_ac_info*

Tipo	Quantidade por macrobloco	Número de *bits*
Luminância AC	4	4
Crominância AC Cb	2	2
Crominância AC Cr	2	2
Total	6	6

O elemento sintático *ref_idx_lx* tem ocorrência máxima de um por submacrobloco (8x8 amostras) por lista (*lx* = *L0* ou *L1*). Como a decodificação destes elementos sintáticos referentes a cada lista ocorre separadamente e dependendo do tipo de *slice* cada

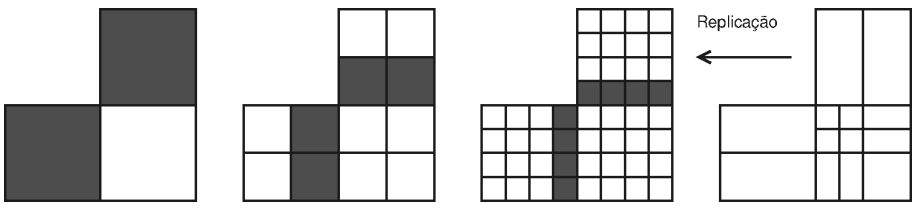

Figura 28: Elementos sintáticos num macrobloco e replicação para simplificação de vizinhanças

memória corresponde a uma lista. Em *slices* I as duas memórias ficam desativadas, em *slices* P somente a memória referente a lista 1 fica habilitada e em *slices B* as duas ficam habilitadas. Como mostrado na tabela 7 as memórias tem um total de 10 *bits* para a palavra, pois guarda-se dois valores por macrobloco de 5 *bits*.

Tabela 7: Palavra de memória para *ref_idx_lx*, *lx = L0* ou *L1 - mem_ref_idx_lx*

Elemento sintático	Quantidade por macrobloco	Número de *bits*
ref_idx_lx	2	10

O elemento sintático *mvd_lx* tem ocorrência máxima de um por bloco (4x4 amostras) por lista (*lx = L0* ou *L1*) por componente (horizontal ou vertical). Assim como o elemento sintático *ref_idx_lx*, houve separação em duas memórias para armazená-los, um para a lista *L0* e outro para a lista *L1* habilitados conforme o tipo de *slice*. A palavra de memória, como exibido na tabela 8, tem 64 *bits*.

Tabela 8: Palavra de memória para *mvd_lx*, *lx = L0* ou *L1 - mem_mvd_lx*

Tipo	Quantidade por macrobloco	Número de *bits*
mvd_lx_x	4	32
mvd_lx_y	4	32
Total	8	64

Resumidamente as informações das memórias são dadas na tabela 9, onde são informados os tipos, a quantidade, o tamanho da palavra e se é de porta dupla de leitura. O máximo que pode ser armazenado é 42240 *kbits* (240 endereços de 176 *bits*), o que é adequado para SRAM.

Tabela 9: Memórias SRAM usadas no gerenciador de fluxo de bits

Memória	Quantidade	Tamanho da palavra	Porta dupla
mem_mb_info	1	20	Sim
mem_cbf_ac_info	1	8	Não
mem_mvd_lx	2	64	Não
mem_ref_idx_lx	2	10	Não
Total	6	176	-

Na figura 29 é mostrado os sinais de interface para uma memória SRAM de porta única. A memória de porta dupla tem os mesmos sinais para uma segunda porta.

Durante a decodificação de um macrobloco, as memórias são mantidas com escrita desabilitada, com o endereço de leitura para fornecer valores para cálculo do contexto. Os valores elementos sintáticos que podem depois ser referenciados como vizinhança por

macroblocos abaixo são armazenados em registradores ligados diretamente às portas de dados de entrada das memórias. No final da decodificação do macrobloco, o endereço é mudado e a escrita é habilitada nas memórias. O endereço de leitura e escrita é comum para todas as memórias. Somente a memória *mem_mb_info* tem um endereço de leitura diferente para a segunda porta, que serve para ler valores de elementos sintáticos da esquerda.

5.2.3.2 Armazenamento de valores de elementos sintáticos em registradores

Os elementos sintáticos que devem ser referenciados no macrobloco que está sendo atualmente decodificado são armazenados em registradores. Nesta categoria enquadram-se os que ocorrem à nível de partição, subpartição e bloco 4x4: *ref_idx_lx*, *mvd_lx* e *coded_block_flag* AC.

Para o elemento sintático *ref_idx_lx* há 6 registradores para guardar valores decodificados que podem ser referenciado dentro do mesmo macrobloco, sendo 3 para a *ref_idx_L0* e 3 para *ref_idx_L1*, pois pode-se ter até 4 partições 8x8 para as quais se decodifica este elemento sintático. A última, do canto inferior à direita não é referenciada no mesmo macrobloco.

Para o elemento sintático *mvd_lx* há somente dois registradores para armazenar valores que serão referenciados dentro de um submacrobloco, conforme está mostrado na figura 30. Na primeira linha da figura é mostra um submacrobloco particionado em 4 blocos 4x4 no decorrer da decodificação dos elementos sintáticos de *mvd_lx*. O bloco atual é marcado com um 'X', os vizinhos referenciados são destacados em verde e o o bloco que será futuramente vizinho é destacado em azul. Nas duas linhas abaixo na figura é mostrado que a lógica também se aplica para subpartições de tamanho 8x4 ou 4x8.

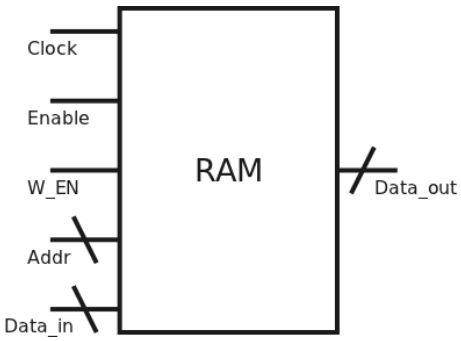

Figura 29: Sinais de interface da memória SRAM (uma porta)

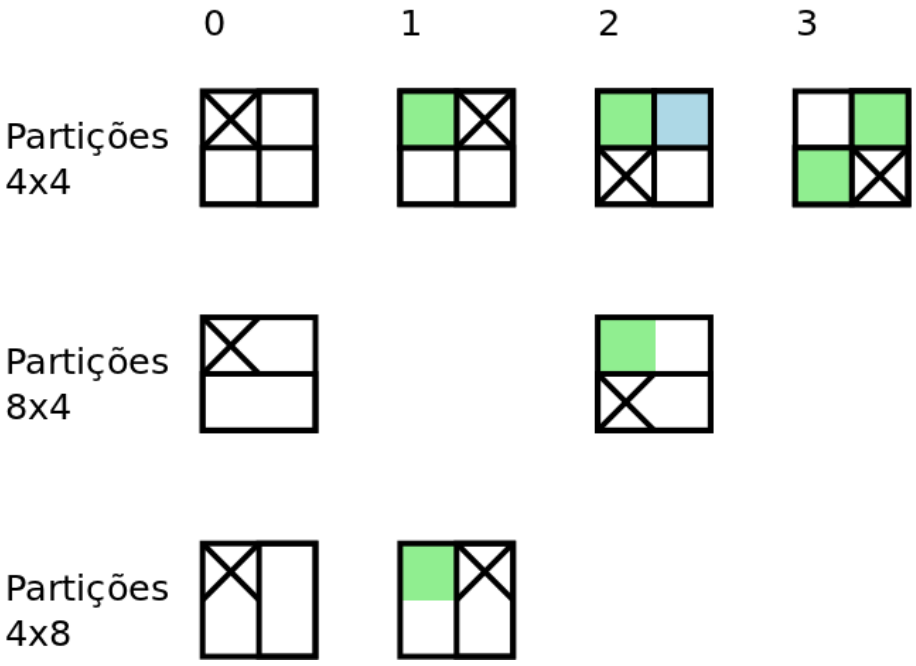

Figura 30: Armazenamento de *mvd_lx* para subpartições de submacroblocos (8x8)

Entre submacroblocos vizinhos são usados 14 registradores (7 para *mvd_L0* e 7 para *mvd_L1*) conforme é apresentado na figura 31. Os blocos 4x4 destacados são os que podem referenciados de um submacrobloco a outro. Quando um submacrobloco não é particionado em blocos 4x4, faz-se replicação dos valores decodificados.

O elemento sintático *coded_block_flag* AC ocorre para cada bloco 4x4 do macrobloco na ordem em que é apresentada na figura 32, onde o bloco atual é marcado com um 'X'. Destacado em verde é um bloco referenciado como vizinho, e em azul um bloco que pode ser referenciado como vizinho depois. Portanto somente 4 registradores de 1 *bit* são necessários para armazenar os valores vizinhos dentro do mesmo macrobloco.

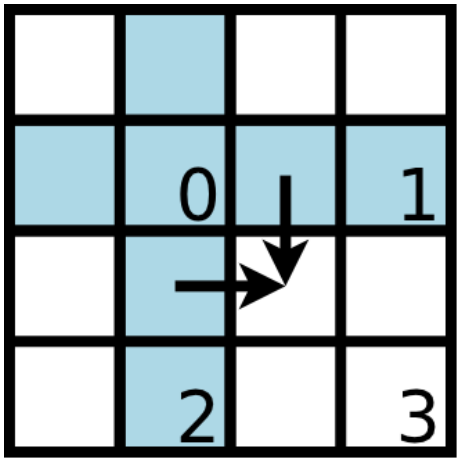

Figura 31: Armazenamento de *mvd_lx* para vizinhanças entre submacroblocos (8x8)

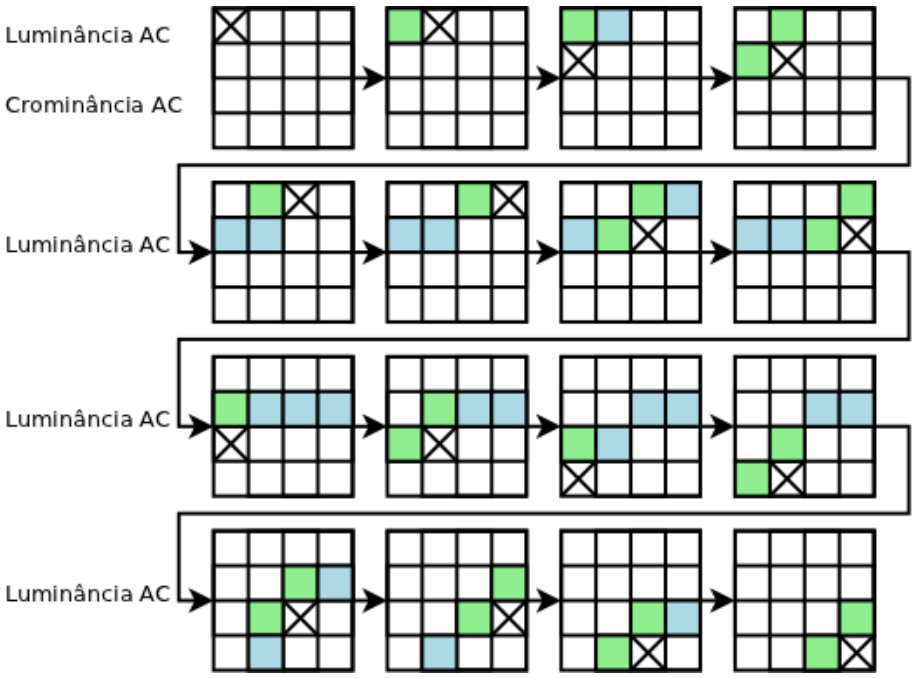

Figura 32: Armazenamento de *coded_block_flag* AC

5.2.4 Construtor de bloco de coeficientes de resíduo

Para o CABAD, é especificado na norma ITU um algoritmo que deve ser implementado para construção dos blocos de coeficientes de resíduo. Simplificadamente, este algoritmo determina:

1. Decodificação do elemento sintático *coded_block_flag* para verificar se há coeficientes no bloco.

2. Se há coeficientes, é feita decodificação de um mapa de significâncias (0 ou 1) para o bloco de coeficientes.

3. Decodificação dos níveis dos coeficientes para as posições do mapa de significâncias com valor '1', começando da última até a primeira.

O algoritmo foi implementado sob o controle da máquina de estados apresentada na figura 33. Os estados e as operações que ocorrem neles são apresentadas a seguir.

5.2.4.1 wait_start

Este é o estado inicial e final, em que o controle aguarda um sinal de habilitação. Dependendo do tamanho de bloco, informado pela entrada *blk_size* o próximo estado é *coded_block_flag* ou *map_sig*. Quando o tamanho de um bloco de coeficientes num

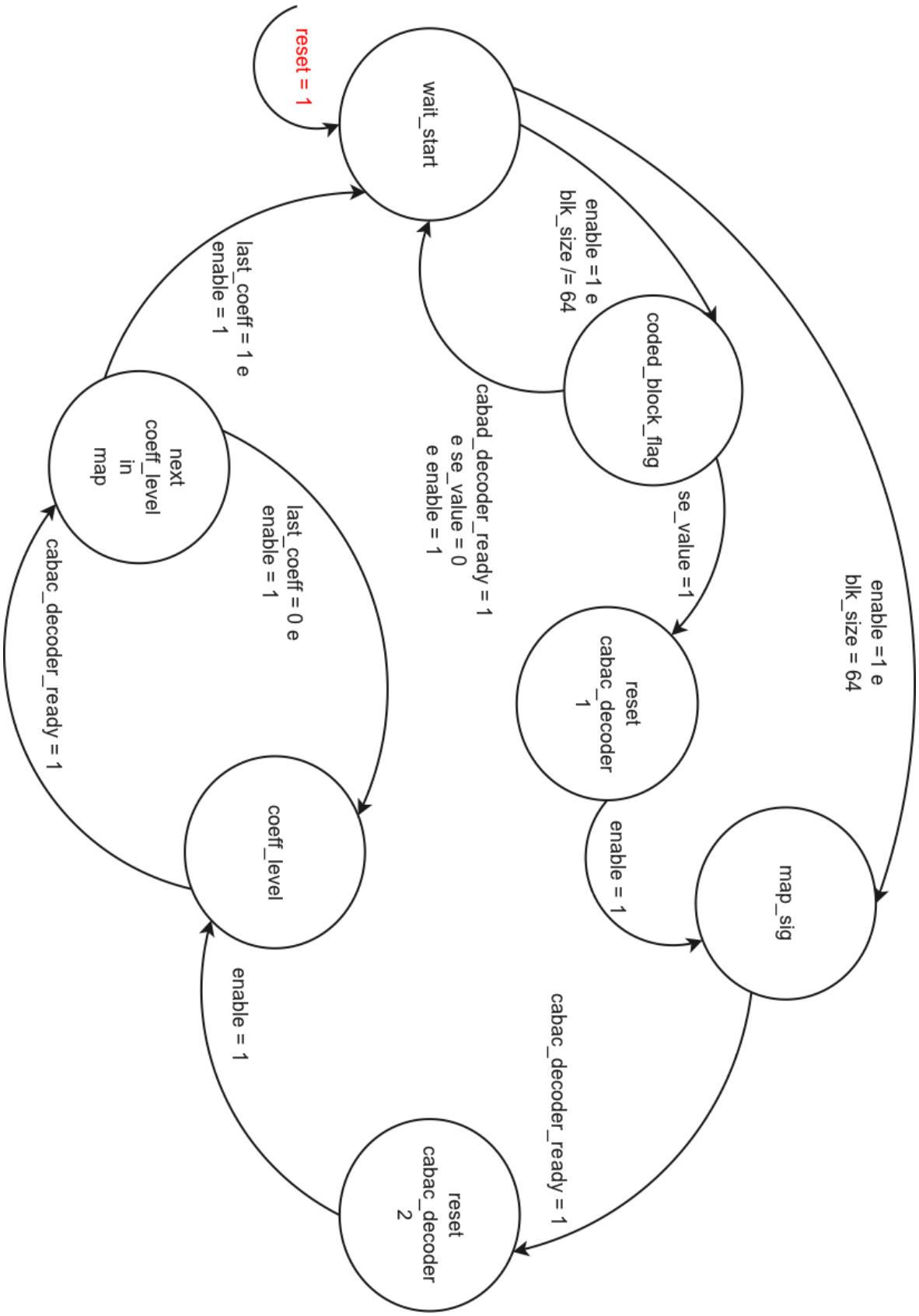

Figura 33: Diagrama de estados do módulo construtor de bloco de coeficientes

bloco é de 64, assume-se que o elemento sintático *coded_block_flag* é igual a 1 e não há necessidade de decodificá-lo.

Neste estado, o bloco de coeficientes de saída é zerado.

5.2.4.2 coded_block_flag

Neste estado o módulo solicita a decodificação do elemento sintático *coded_block_flag* ao núcleo do CABAD. Aguarda-se um sinal de 'pronto'. Se o valor do elemento sintático decodificado for 0, o módulo envia um sinal de 'pronto', e o bloco de coeficientes de saída é nulo. Caso contrário (se for decodificado 1), passa-se para o estado *reset_cabac_decoder_1*.

5.2.4.3 reset_cabac_decoder_1

Este estado somente é necessário porque o ciclo de operação do núcleo do CABAD exige um sinal de *reset* antes da decodificação de todo elemento sintático. Se o módulo estiver habilitado, muda-se para o estado *map_sig*.

5.2.4.4 map_sig

Neste estado o núcleo do CABAD é mantido habilitado até decodificar todo o mapa de significâncias. Enquanto isso, monitora-se as suas saídas *coeff_index*, *coeff1*, *coeff2* e *cabac_decoder_ready*. O número de coeficientes é contado, e uma tabela que usa este número como índice é preenchida com o valor de *coeff_index* ou *coeff_index* +1. Esta tabela informa, portanto, a posição do próximo coeficiente no mapa de significâncias.

Quando o núcleo do CABAD envia o sinal de 'pronto', se tem construído o mapa de significância completo e uma tabela de próximas posições. Muda-se para o estado *reset_cabac_decoder_2*.

5.2.4.5 reset_cabac_decoder_2

Similar a *reset_cabac_decoder_1*. Quando habilitado muda-se para o estado *coeff_level*.

5.2.4.6 coeff_level

Neste estado o núcleo do CABAD é habilitado para que seja decodificado um nível de coeficiente. Este nível é guardado na saída informada pela tabela. O próximo estado é

next_coeff_level_in_map.

5.2.4.7 next_coeff_level_in_map

Neste estado o número de coeficientes é decrementado a não ser que já seja nulo, para atualizar o valor da posição do próximo coeficiente na tabela. O núcleo do CABAD é mantido resetado. Quando não há outro nível de coeficiente para decodificar e o módulo está habilitado, é emitido um pulso de 'pronto' e o próximo estado é *wait_start*. Caso contrário, entra-se no estado *coeff_level* para decodificar o próximo nível de coeficiente.

5.3 Simulação e verificações funcionais

A simulação foi feita a partir de um *testbench* que abre fluxo de *bits* de vídeo codificado com o *software* de referência JM [8]. O vídeo usado é progressivo e contém *slices* do tipo P, B e I. A sequência foi codificada a partir do vídeo bruto *akiyo* [14], com parâmetro de quantização fixo em 28.

A verificação do correto funcionamento do CABAD foi feita comparando-se os valores de elementos sintáticos decodificados com os esperados que pode se extrair com o JM ou PRH.264. Também foi feita a comparação dos valores de intervalo de decodificação aritmética binária *codIRange* e *codIOffset*, que aparecem nas simulações, como *range* e *value*.

Na figura 34 é mostrado um trecho de simulação selecionado em que sendo feito decodificação de elementos sintáticos de modo de predição intra 4x4. Por exigência de projeto do *parser* [10], cada estado tem quatro tempos, denotado por *state_timer*. No tempo de 0 para 1, o valor do elemento sintático é decodificado. Se o núcleo do CABAD não estiver pronto no tempo 0, o contador é congelado.

Na figura 35 pode ser verificado que a decodificação do elemento sintático começa no estado anterior, com *state_timer* = 2, onde está marcado. Ela termina com *state_timer* = 1 no estado que corresponde ao elemento sintático no *parser*. Desta forma não o ciclo de operação do núcleo do CABAD é executado de forma enxuta e não há introdução de muito atraso no *parser*.

Em simulação também foi verificado o correto funcionamento do CABAD quando o *parser* deve ser desabilitado por faltar dados na FIFO de entrada por exemplo. Como pode ser visto na figura 36, em que é feita decodificação do mapa de significâncias para

um bloco de coeficientes de resíduo, num dado momento o decodificador *slice data* é desabilitado. Porém o núcleo do CABAD não pode ser interrompido durante a decodificação do elemento sintático. Então ela continua, e o gerenciador de fluxo de *bits* acumula o número de *bits* que deve ser consumido quando decodificador *slice data* for habilitado novamente e fornece apenas dados de *buffers*. Na marca da figura, foram acumulados 10 *bits*.

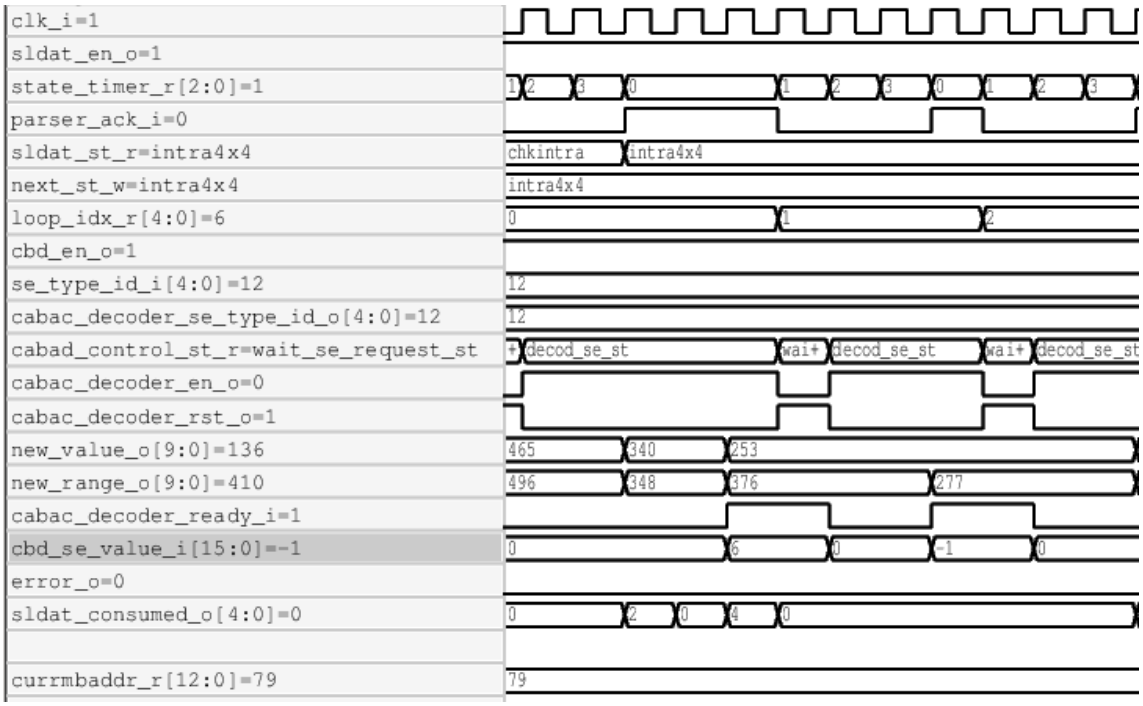

Figura 34: Simulação com testbench

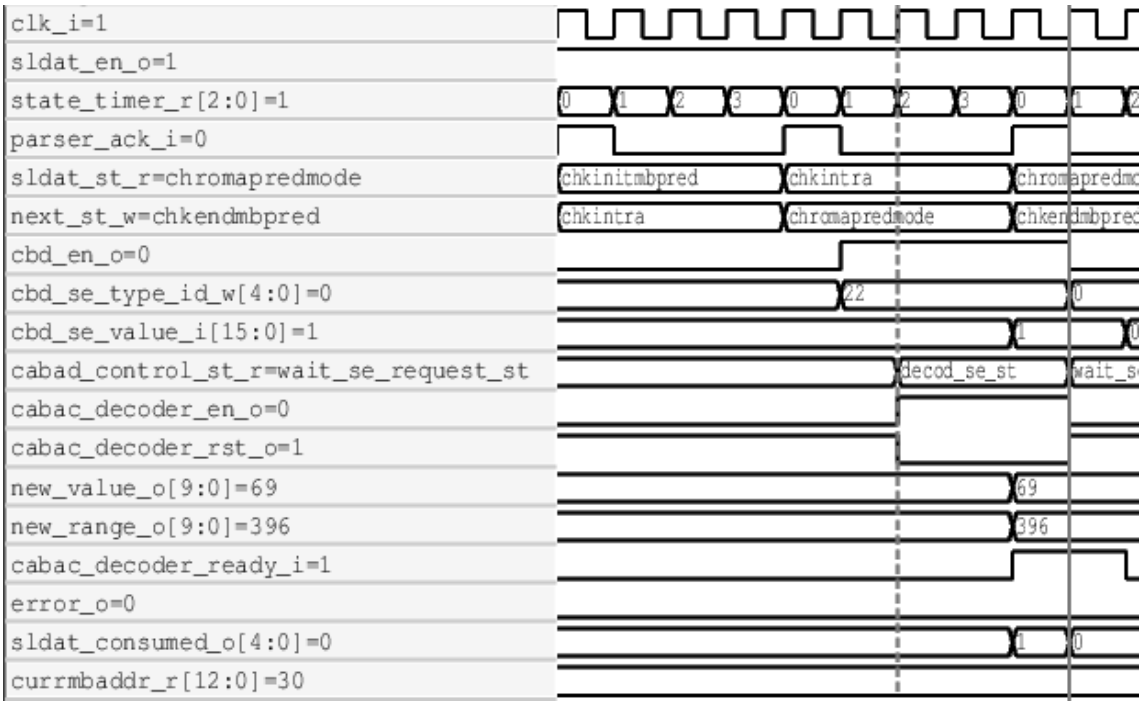

Figura 35: Verificação do sincronismo entre parser e CABAD

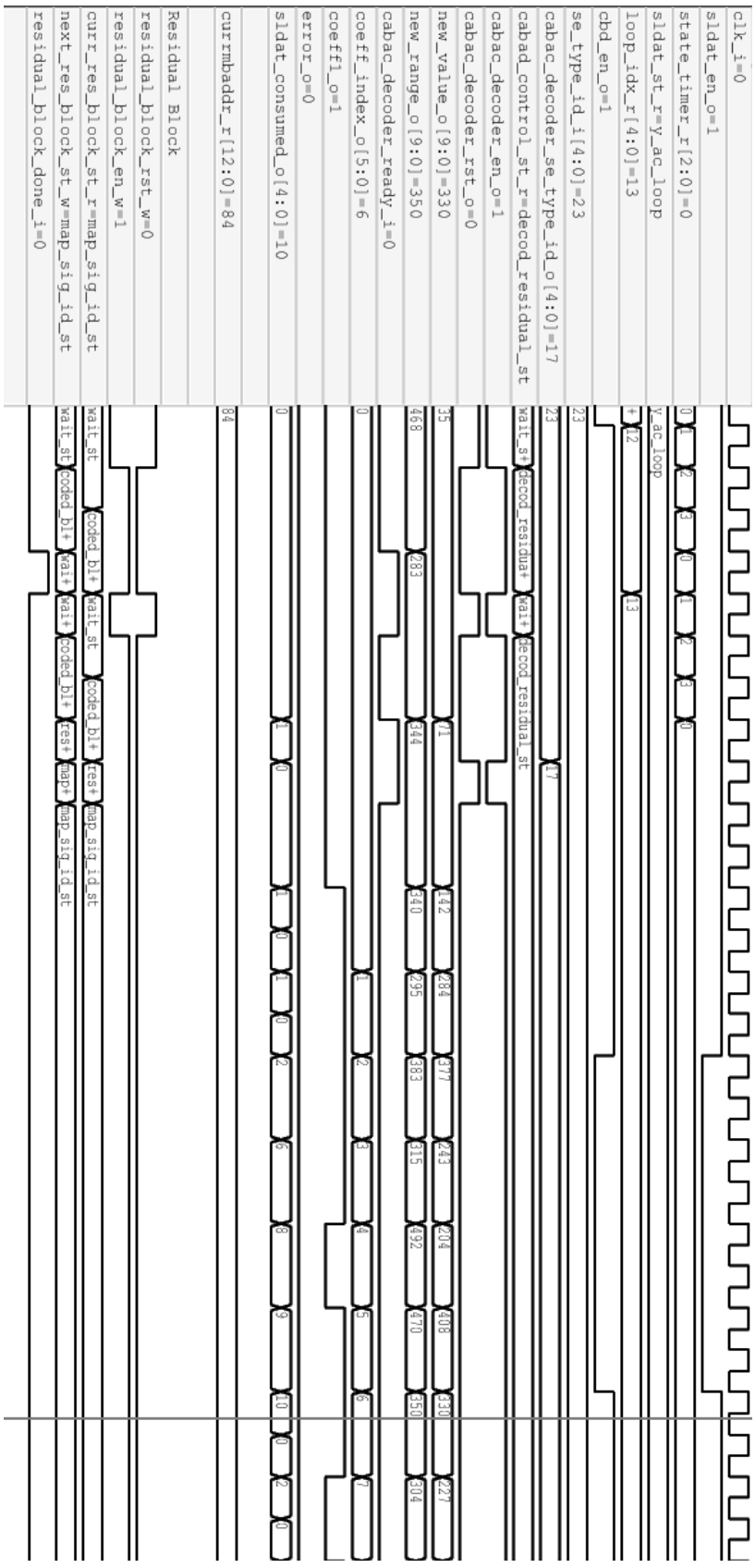

Figura 36: Verificação do sincronismo quando o barrel shifter é desabilitado

6 RESULTADOS ALCANÇADOS

Neste capítulo são reunidos os resultados atingidos pelo projeto. Na primeira seção são apresentados dados de síntese em FPGA. Na seção seguinte, tendo obtido a máxima frequência de operação da arquitetura do *parser* na síntese, é feita uma avaliação de desempenho.

6.1 Síntese em FPGA

A síntese em FPGA foi realizada para o *parser* após a integração do CABAD e separadamente para cada um dos principais módulos desenvolvidos. Também foi feita síntese do núcleo do CABAD, onde foi verificado que é o módulo limitador da frequência de operação. O dispositivo alvo de síntese foi a Xilinx Virtex-5 XC5VLX110T, disponível no LaPSI. Neste FPGA cada *slice* contém 4 *LUTs* e 4 *Flip-flops* [4]. As *LUTs* são de 6 entradas. Nesta placa está previsto o protótipo decodificador de vídeo H.264/AVC inteiro. O *software* utilizado para síntese foi o Xilinx ISE 10.1.

Os resultados de síntese para o *parser* após a integração do CABAD são apresentados na tabela 10. Nota-se que houve uma alta redução na frequência máxima de operação quando comparada ao estado antes da integração de 127 MHz [10]. O gargalo é introduzido pelo núcleo do CABAD que apresenta a menor frequência máxima, como pode ser observado na tabela 11 que apresenta seus resultados de síntese.

Os resultados de síntese para o módulo de controle são apresentados na tabela 12. Este módulo, por não apresentar elevada complexidade atinge uma alta frequência de operação e ocupa pouca área.

Na tabela 13 são apresentados os resultados de síntese do gerenciador de elementos sintáticos. Este constituiu o mais expressivo módulo desenvolvido para integração, ocu-

pando um número de *slices* dentro do FPGA próximo ao número de *slices* ocupado pelo núcleo.

A tabela 14 mostra os resultados de síntese do gerenciador de fluxo de *bits*, que apresentou baixa ocupação de área do FPGA.

Finalmente, os resultados de síntese para o construtor de bloco são mostrados na tabela 15. O alto consumo de área do FPGA deste módulo é devido à possibilidade de blocos de 64 coeficientes, cada um com 13 *bits*. Como não se conhecia no momento de projeto a interface exigida pelo módulo de quantização e transformada inversa que ainda não trata blocos deste tamanho, foi considerado o pior caso de que os coeficientes pudessem ser acessados diretamente de registradores, com uma entrada de seleção. Portanto, foram sintetizados muitos *Flip-Flops* para armazenar os coeficientes e *LUTs* para selecioná-los através de multiplexação.

Tabela 10: Resultados de síntese do *parser* com a integração do CABAD

Elemento	**Usado**	**Disponível**	**Utilização**
Slices	3.992	17.280	23%
Flip-flops	3.776	69.120	5%
LUTs	11.074	69.120	16%
Block RAMs	216 Kb	5.328 Kb	4%
Frequência	56,484 MHz		

Tabela 11: Resultados de síntese do núcleo do CABAD

Elemento	**Usado**	**Disponível**	**Utilização**
Slices	1.199	17.280	6%
Flip-flops	441	69.120	<1%
LUTs	3.014	69.120	4%
Block RAMs	90 Kb	5.328 Kb	1%
Frequência	56,294 MHz		

Tabela 12: Resultados de síntese do módulo de controle do CABAD

Elemento	**Usado**	**Disponível**	**Utilização**
Slices	52	17.280	<1%
Flip-flops	25	69.120	<1%
LUTs	110	69.120 Kb	<1%
Frequência	255,493 MHz		

Tabela 13: Resultados de síntese do gerenciador de elementos sintáticos

Elemento	**Usado**	**Disponível**	**Utilização**
Slices	1.061	17.280	6%
Flip-flops	1240	69.120	1%
LUTs	2.515	69.120	3%
Block RAMs	36 Kb	5.328 Kb	<1%
Frequência	188,679 MHz		

Tabela 14: Resultados de síntese do gerenciador de fluxo de *bits*

Elemento	**Usado**	**Disponível**	**Utilização**
Slices	114	17.280	<1%
Flip-flops	38	69.120	<1%
LUTs	227	69.120	<1%
Frequência	313,480 MHz		

Tabela 15: Resultados de síntese do construtor de blocos de resíduo

Elemento	**Usado**	**Disponível**	**Utilização**
Slices	747	17.280	4%
Flip-Flops	1233	69.120	1%
LUTs	2138	69.120	3%
Frequência	187,223 MHz		

6.2 Verificação de desempenho

Uma verificação de desempenho foi feita com a sequência usada para simulação e verificação funcional da arquitetura. As medições realizadas são resumidas na tabela 16, que apresenta em suas colunas:

1. O tipo de quadro e sua colocação ordem de decodificação
2. O número de *bits* codificados contidos no quadro
3. O número de ciclos consumidos em simulação para decodificar o quadro
4. A taxa de processamento de macroblocos por segundo para o quadro
5. A taxa de processamento de dados codificados em Kb/s do *parser*
6. A taxa de processamento *necessária* de dados codificados em Kb/s fazendo-se uma extrapolação para HD 1080p a 30,1 quadros por segundo.

Na tabela 17 são apresentadas:

1. A taxa de quadros por segundo obtida na resolução QCIF (176×144)
2. A taxa de quadros por segundo que pode ser decodificada pelo *parser* extrapolando para resolução HD 1080p

Tabela 16: Análise de desempenho para uma sequência de 3 quadros QCIF (akiyo)

Quadro	**Número de *bits***	**Ciclos**	**Taxa MB/s**	**Taxa Kb/s**	**Taxa Kb/s em HD 1080p**
I (0)	18378	57.500	86.087	*15.980,869*	45.595
P (1)	145	3.000	1.650.000	2.416,666	360
B (2)	11	1.584	3.125.000	347,222	27

Para obter os dados da tabela 16 assume-se que o *parser* opera na taxa de 50 MHz. O tempo de inicialização de contextos não é contado nos ciclos porque ele pode ser menosprezado quando se compara ao tempo de decodificação de todos os macroblocos de um *slice* de sequência HD 1080p.

Os dados da segunda e terceira colunas da tabela 16 foram medidos diretamente. Sendo F_{op} a frequência de operação de 50 MHz e N_{ciclos} o número de ciclos para decodificar o quadro, os dados da quarta coluna são obtidos a partir de:

$$\frac{F_{op} \times 99}{N_{ciclos}}$$

Sendo N_{bits} o número de *bits* contido no quadro, os dados da quinta coluna da tabela 16 são obtidos a partir de:

$$N_{bits} \times \frac{F_{op}}{N_{ciclos}}$$

A extrapolação de Kb/s da penúltima coluna da tabela 16 é feita multiplicando-se a proporção do número de macroblocos de um vídeo HD 1080P (8160 macroblocos) em relação a uma sequência QCIF (99 macroblocos), sem considerar a taxa de processamento do *parser* obtida em simulação. É a taxa necessária para imagens com a mesma média de *bits* por macrobloco do quadro. Os dados são obtidos a partir de:

$$N_{bits} \times \frac{8160 \times 30.1}{99 N_{ciclos}}$$

Na tabela 17, os dados são obtidos dividindo-se o a frequência pelo número de ciclos necessários para decodificar o quadro. Para a resolução HD 1080p é aplicada a proporção de 99 para 8160 macroblocos.

Tabela 17: Taxa de quadros por segundo para a sequência (akiyo)

Quadro	**Quadros por segundo QCIF**	**Quadros por segundo HD 1080p**
I (0)	869,56	10,55
P (1)	16.666,66	202,16
B (2)	31.565,65	382,96

Tomando como referência o nível necessário para o projeto, os dados obtidos para o segundo e terceiro quadro satisfazem a necessidade de processamento de 245.760 MB/s com muita folga, porém devem ser desconsiderados porque a codificação tem uma taxa muito baixa de Kb/s.

O primeiro quadro, do tipo I, por sua vez apresenta taxa mais condizentes. O que é explicado na seção 4.1.3, comprova-se uma vez que se extrapola a taxa de Kb/s para o caso de uma sequência HD 1080p, que ultrapassa o limite do nível de 20 Mbit/s. A taxa de macroblocos por segundo deve ser descartada. A taxa mais significante é portanto a de Kb/s para o primeiro quadro, que se aproximou dos 20 Mb/s do nível 4 de codificação. Ela é porém maior que 14 Mb/s necessários para vídeo HD 720p no perfil 3.1 como especificado pela norma ITU.

7 CONCLUSÕES

Para a frequência máxima de operação foi constatado que a arquitetura não alcançou desempenho suficiente para decodificar vídeos *Full-HD*, quando sintetizada para o FPGA disponível para uso. Porém a meta final do projeto é o produto em ASIC, que devido às suas características atinge desempenho mais elevado para um mesmo *netlist* e tecnologia de fabricação [15], portanto cumprindo os requisitos de desempenho.

Este trabalho representou um importante avanço no desenvolvimento do decodificador de vídeo H.264/AVC para o Sistema Brasileiro de Televisão Digital, uma vez que foram previstas no projeto as necessidades do perfis mais avançados do padrão. Além do CABAD, no projeto foi incluído suporte a entrelaçamento.

Ao possibilitar que o *parser* extraia parâmetros com o CABAD e uma vez que as opções mais avançadas de codificação normalmente são usadas em conjunto com a CABAC, a integração também contribui para testar outros módulos de forma mais completo como por exemplo a compensação de movimento com duas imagens.

A abordagem adotada para o projeto foi pragmática, no sentido que houve aproveitamento de módulos já desenvolvidos e foi buscado atingir suas metas o quanto antes, tendo em vista os prazos para encerramento do projeto Rede H.264.

Como trabalho futuro, é preciso realizar uma validação rigorosa com sequências maiores, através da decodificação de fluxo de *bits* das emissoras que já trabalham dentro das normas do SBTVD. É necessário também que o protótipo do decodificador seja simulado por completo e verificado operando na placa e gerando saída de vídeo.

REFERÊNCIAS

[1] ITU-T. *ITU-T Recommendation H.264 (03/05): Advanced video coding for generic audiovisual services*. [S.l.]: International Telecommunication Union, 2005. http://www.itu.int/rec/T-REC-H.264-200503-S.

[2] RICHARDSON, I. E. *H.264 and MPEG-4 Video Compression: Video Coding for the Next-generation Multimedia*. [S.l.]: John Wiley and Sons, 2003.

[3] MARPE, D.; SCHWARZ, H.; WIEGAND, T. Context-based adaptive binary arithmetic coding in the h.264/avc video compression standard. *Circuits and Systems for Video Technology, IEEE Transactions on*, v. 13, n. 7, p. 620 – 636, julho 2003. ISSN 1051-8215.

[4] XILINX. *Virtex-5 Family Overview*. February 2009. Http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf.

[5] SBTVD, R. H. *Wiki Rede H.264 SBTVD*. Julho 2010. http://www.lapsi.eletro.ufrgs.br/h264/wiki.

[6] ABNT. *NBR15602 Televisão Digital Terrestre - Codificação de vídeo, áudio e multiplexação*. [S.l.]: ABNT, 2007.

[7] WIKIPEDIA. *Exponential-Golomb Coding*. Maio 2011. http://en.wikipedia.org/wiki/Exponential-Golomb_coding.

[8] Coordination, H. S. *JM Software*. Março 2011. http://iphome.hhi.de/suehring/tml/.

[9] SCHMIDT, A. A. de A. et al. Development of a software model for an H.264/AVC progressive main profile hardware video decoder. In: *10th Students Forum on Microelectronics*. [S.l.: s.n.], 2010.

[10] LORENCETTI, M. A. Parser em vhdl para decodificador de vídeo h.264 para sbtvd. 2010. 52 páginas. Projeto de Diplomação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2010. 2010.

[11] DEPRA, D. A. *Algoritmos e Desenvolvimento de Arquitetura para a Codificação Binária Adaptativa ao Contexto para o Decodificador H.264/AVC*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul, UFRGS, 2009.

[12] CARVALHO, J. P. A. de. *Arquitetura Dedicada para Decodificação CABAC H.264/AVC em Sistema em Silício*. Dissertação (Mestrado) — Universidade de Brasília, UnB, 2009.

[13] SCHMIDT, A. A. de A.; SUSIN, A. A. Cabac integration into an intra-only h.264/avc hardware video decoder. In: *26º Simpósio Sul de Microeletrônica*. [S.l.: s.n.], 2011.

[14] AKIYO YUV video sequence. março 2011. Http://trace.eas.asu.edu/yuv/.

[15] KUON, I.; ROSE, J. *Measuring the Gap between FPGAs and ASICs*. [S.l.], 2006.