UNIVERSIDADE FEDERAL DO PAMPA

CAMILA DE MATOS ALONSO

DESENVOLVIMENTO DE UMA ARQUITETURA EM HARDWARE DO BLOCO DE BINARIZAÇÃO DO CABAC BASEADO NO PADRÃO HEVC

Bagé
2016

CAMILA DE MATOS ALONSO

DESENVOLVIMENTO DE UMA ARQUITETURA EM HARDWARE DO BLOCO DE BINARIZAÇÃO DO CABAC BASEADO NO PADRÃO HEVC

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Computação.

Orientador: Prof. MSc. Fábio Luís Livi Ramos

Bagé
2016

Ficha catalográfica elaborada automaticamente com os dados fornecidos
pelo(a) autor(a) através do Módulo de Biblioteca do
Sistema GURI (Gestão Unificada de Recursos Institucionais).

A454d Alonso, Camila de Matos
Desenvolvimento de uma Arquitetura em Hardware do Bloco de Binarização do CABAC Baseado no Padrão HEVC / Camila de Matos Alonso.
79 p.

Trabalho de Conclusão de Curso(Graduação) -- Universidade Federal do Pampa, ENGENHARIA DE COMPUTAÇÃO, 2016.
"Orientação: Fábio Luís Livi Ramos".

1. Codificador de Entropia. 2. CABAC. 3. Bloco de Binarização. I. Título.

CAMILA DE MATOS ALONSO

DESENVOLVIMENTO DE UMA ARQUITETURA EM HARDWARE DO BLOCO DE BINARIZAÇÃO DO CABAC BASEADO NO PADRÃO HEVC

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Computação.

Trabalho de Conclusão de Curso defendido e aprovado em: 07 de dezembro de 2016.

Banca examinadora:

_____

Prof. MSc. Fábio Luís Livi Ramos
Orientador
UNIPAMPA

_____

Prof. MSc. Julio Saraçol Domingues Júnior
UNIPAMPA

_____

Prof. Dr. Bruno Silveira Neves
UNIPAMPA

Dedico este trabalho ao meu avô Antônio que mesmo não estando mais presente sempre será lembrado com muito carinho.

AGRADECIMENTOS

Primeiramente gostaria de agradecer aos meus pais e a minha irmã que sempre me apoiaram e me incentivaram em todas as minhas escolhas. A minha mãe, por todo o cuidado dedicado a mim durante este período, sempre se preocupando com o meu bem estar. Ao meu pai por todo o suporte e confiança sempre acreditando que eu era capaz. Gostaria de agradecer também ao meu namorado por estar comigo durante todo este tempo, sempre me apoiando e me dando força para seguir em frente.

A minha família por sempre estar presente. Aos meus avós que sempre estiveram do meu lado. As minhas tias queridas, Rosani, Regina, Roseli e Rosaura, por toda preocupação e carinho que sempre tiveram comigo, todos os telefonemas após alguma prova ou apresentação difícil. Aos meus primos que considero como irmãos e sei que torcem por mim da mesma forma que torço por eles.

Gostaria de agradecer também ao meu orientador, por todos os ensinamentos durante o desenvolvimento deste trabalho, por toda a preocupação em me ajudar nos momentos em que estive com dúvida. Com certeza este trabalho só foi possível, pois tive o senhor como orientador. Muito obrigado.

Aos demais professores do curso, também agradeço, por todos os ensinamentos e conselhos. Com certeza sou muito melhor hoje do que quando entrei no curso, graças a vocês. Em especial ao professor Érico, por ter me dado a oportunidade de fazer parte de um projeto de pesquisa, me ensinando a escrever o meu primeiro artigo e sempre me orientando e incentivando a escrever trabalhos e a participar de eventos.

Por fim, gostaria de agradecer as minhas colegas e novas amigas, que fiz durante o curso, por estarem comigo durante este caminho. A Leticia e a Fernanda, obrigada por terem entrado na minha vida e me ajudado em momentos que precisei. A Luana que esteve comigo antes mesmo do curso começar e compartilhou comigo momentos de nervosismo durante a primeira apresentação e momentos de alegria quando algo importante dava certo. A Liliane que é uma pessoa incrível, que me ajudou muito em vários momentos, se tornado uma grande amiga.

RESUMO

O codificador de entropia é responsável pela representação simbólica de dados de forma a representá-los com um número menor de bits. No padrão HEVC existe apenas um tipo de codificador de entropia, o CABAC (*Context Adaptive Binary Arithmetic Coding* – Codificação Aritmética Binária Adaptativa ao Contexto), que é similar ao utilizado no padrão anterior, o H.264/AVC, porém foi modificado para facilitar o processamento paralelo. Este trabalho apresenta uma arquitetura em hardware para o bloco de binarização do CABAC, que é o primeiro bloco executado no processo do codificador de entropia. Este bloco tem como objetivo reduzir o tamanho do alfabeto de símbolos, simplificando assim os custos da modelagem de contexto e facilitando a tarefa da codificação aritmética. Como resultado deste trabalho, buscou-se uma arquitetura eficiente em termos de desempenho e com redução no consumo de potência. Para realizar a síntese da arquitetura, foi utilizada a ferramenta RTL Compiler da Cadence e a biblioteca de células de 65 nm da ST. Os resultados da análise do consumo de potência em cima do *gate-level netlist* mostram que a arquitetura final proposta teve redução de consumo de potência de até 41% em relação a arquitetura inicial.

Palavras-Chave: Codificador de entropia, CABAC, bloco de binarização.

ABSTRACT

The entropy encoder is responsible for the symbolic data representation in order to represent it with a smaller number of bits. In HEVC standard, there is only one type of entropy coder, the CABAC (Context Adaptive Binary Arithmetic Coding), which is similar to that used in the previous standard, H.264/AVC, but it has been modified to facilitate the parallel processing. This work presents a hardware architecture for the binarization block of CABAC, which is the first block in the entropy encoding process. This block aims to reduce the alphabet symbols size, thus simplifying the costs of context modeling and facilitating the task of the arithmetic coding. As a result of this work, an architecture with efficient performance and reduced power consumption was sought. The synthesis of architecture was performed by using, the RTL Compiler from Cadence tool and 65 nm ST gates library. The results show that synthesis of the final proposed architecture has reached a reduction in power consumption up to 41% compared to the initial architecture.

Keywords: entropy coder, CABAC, binarization block.

LISTA DE FIGURAS

Figura 1 - Codificação de vídeo de uma chamada de vídeo. 18
Figura 2 - Amostras temporais e Amostras espaciais. 19
Figura 3 - Espaço de cor RGB para YCbCr. 20
Figura 4 - Subamostragem de Cores. 21
Figura 5 -Compressão de dados com perdas de informação. 22
Figura 6 - Redundância Entrópica. 23
Figura 7 - Redundância Espacial e Temporal. 24
Figura 8 - Diagrama de blocos do codificador do HEVC. 26
Figura 9 - Partições de movimento (a) simétricas e (b) assimétricas. 27
Figura 10 - Modo de intra-predição. 28
Figura 11 - Possíveis tipos de partição na intra-predição. 29
Figura 12 - Bloco 4x4 (a) Original (b) Após passar pelo processo de transformada. 29
Figura 13- Bloco 4x4 (a) Após o processo de transformada (b) Após o processo de quantização. 30
Figura 14 - Bloco de binarização do CABAC. 32
Figura 15 - Etapas adotadas para o desenvolvimento da pesquisa. 43
Figura 16 - Arquitetura do bloco de binarização. 45
Figura 17 - Diagrama do bloco *analyzer*. 46
Figura 18 - Diagrama dos blocos de binarização. 49
Figura 19 – Saídas do Método Truncado Rice. 50
Figura 20 - Cálculos do Método EGk para os dois primeiros intervalos. 51
Figura 21 - Cálculos realizados para o Custom 4 do intervalo de 6 a 7. 56
Figura 22 - Arquitetura em Paralelo. 58
Figura 23 - Técnica de redução de consumo dinâmico de potência. 59
Figura 24 - Arquitetura final de um núcleo de binarização. 60
Figura 25 - Valor absoluto e média de chamadas para PeopleOnStreet. 61
Figura 26 - Valor absoluto e média de chamadas para BasketballDrive. 62
Figura 27 - Simulação da arquitetura. 63
Figura 28 - Ambiente de validação para o Bloco de Binarização. 64
Figura 29 - Comparação de dois arquivos .txt. 65

LISTA DE TABELAS

Tabela 1- Exemplos de binarização unário, truncado unário e tamanho fixo. 35
Tabela 2 - Exemplo de binarização truncado rice. 36
Tabela 3 - Exemplo de binarização Exp-Golomb. 37
Tabela 4 - Exemplo de binarização do elemento sintático *cu_qp_delta_abs*. 38
Tabela 5 - Processo de binarização do *intra_chroma_pred_mode*. 38
Tabela 6 - Comparação dos Trabalhos Correlatos.....41
Tabela 7 - Tipos de binarização e sua representação em binário. 47
Tabela 8 - cMax e sua representação em binário. 48
Tabela 9 - Cálculo do cMax. 49
Tabela 10 - Cálculo realizado para o EGk. 52
Tabela 11 - Processo de binarização para o *part_mode*. 53
Tabela 12 - Processo de binarização para o inter_pred_idc. 54
Tabela 13 - Processo de binarização dos elementos sintáticos de 0 a 5. 55
Tabela 14 - Processo de binarização do Custom 4. 56
Tabela 15 - Casos possíveis a partir da variável cRiceParam. 57
Tabela 16 - Resultados da síntese ASIC para frequência e área.....66
Tabela 17 - Resultados da Síntese para a Arquitetura Inicial.....67
Tabela 18 - Resultados da Síntese para a Arquitetura com *Operand Isolation*.....67
Tabela 19 - Resultados da Síntese para a Arquitetura Inicial em Paralelo. 68
Tabela 20 - Resultados da Síntese para a Arquitetura em Paralelo com *Operand Isolation*. 69
Tabela 21 - Comparação entre os resultados dos Trabalhos Correlatos. 70

LISTA DE ABREVIATURAS E SIGLAS

ACM - *Association for Computing Machinery*

ASIC - *Application-Specific Integrated Circuit*

AVC - *Advanced Video Coding*

BAE - *Binary Arithmetic Encoder*

BCM - *Binarization and Context Modeling*

BPBS - *Bypass bin splitting*

BPCC - *Bins per clock cycle*

BS - *Boundary Strength*

CABAC - *Context Adaptive Binary Arithmetic Coding*

CTU - *Coding-Tree Unit*

CU - *Coding Units*

DBF - *Deblocking Filter*

DCT - *Discrete Cosine Transform*

DST - *Discrete Sine Transform*

EGk - *kth-order Exp-Golomb*

FL - *Fixed Length*

FPGA - *Field Programmable Gate Array*

HEVC - *High Efficiency Video Coding*

HPC - *Hvbrid Path Coverage*

IEEE - *Institute of Electrical and Electronics Engineers*

ISO/IEC - *International Organization for Standardization – International Electrotechnial Commission*

ITU-T - *International Telecommunication Union – Telecommunication sector*

JCT-VC - *Joint Collaborative Team on Video Coding*

LCU - *Largest Coding Unit*

MCP - *Motion Compensation Prediction*

ME - *Motion Estimation*

MPEG - *Moving Pictures Experts Group*

PMA - *Asymmetric movement partitions*

PMS - *Symmetrical movement of partitions*

PU - *Prediction Units*

QP - *Quantization Parameter*

RGB - *Red, Green, Blue*

SAO - *Sample Adaptative Offset*

SDT - *State Dual Transistion*

SE - *Sintax Elements*

TR - *Truncated Rice*

TU - *Truncated Unary*

U - *Unary*

VCEG - *Video Coding Experts Group*

VHDL - *VHSIC Hardware Description Language*

VHSIC - *Very High Speed Integrated Circuits*

YCbCr - *Luminance*, *Chrominance Blue*, *Chrominance Red*

SUMÁRIO

1 INTRODUÇÃO 15
1.1 Objetivos 17
1.2 Organização do Texto 17
2 CONCEITOS GERAIS E REVISÃO DE LITERATURA 18
2.1 Conceitos de Vídeo Digital e Compressão de Dados 18
2.1.1 Vídeo Digital 19
2.1.2 Compressão de Vídeo Digital 21
2.1.3 Redundância de Dados 22
2.2 O Padrão HEVC 24
2.2.1 Blocos do Padrão HEVC 25
2.2.1.1 Inter-predição 26
2.2.1.2 Intra-predição 28
2.2.1.3 Transformadas e Quantização 29
2.2.1.4 Filtros 30
2.2.1.5 Codificador de Entropia 31
3 BLOCO DE BINARIZAÇÃO 34
3.1 Binarização para códigos Unário, Truncado Unário e Tamanho Fixo 34
3.2 Binarização para Truncado Rice 35
3.3 Binarização para Exp-Golomb 36
3.4 Outros métodos de Binarização 37
3.5 Trabalhos Correlatos 39
3.5.1 Trabalho de Martins 39
3.5.2 Trabalho de Zhou *et al* 39
3.5.3 Trabalho de Peng *et al* 40
3.5.4 Trabalho de Vizzotto *et al* 40
3.5.5 Conclusão dos Trabalhos Correlatos 41
4 METODOLOGIA 43
5 ARQUITETURA PROPOSTA PARA O BLOCO DE BINARIZAÇÃO DO CABAC 45
5.1 Bloco *Analyzer* 46
5.2 Blocos de Binarização 48
5.2.1 Blocos de binarização FL, TR e EGk 49
5.2.2 Blocos de binarização Custom 1, Custom 2 e Custom 3 53

5.2.3 Blocos de binarização Custom 4 e Custom 5 54
5.3 Arquitetura em Paralelo 57
5.4 Arquitetura voltada para Redução de Consumo de Potência 59
6 RESULTADOS E DISCUSSÕES 63
6.1 Simulação e Validação da Arquitetura 63
6.2 Síntese da Arquitetura 65
6.3 Comparação com os Trabalhos Correlatos 69
7 CONCLUSÃO 72
REFERÊNCIAS 74
ANEXO A – ELEMENTOS SINTÁTICOS E SEUS VALORES DE CMAX E FORMATO 78

1 INTRODUÇÃO

Atualmente, existe no mundo uma crescente demanda de aplicações de vídeo digital, seja em dispositivos móveis seja em dispositivos fixos, tais como *smartphones*, câmeras digitais, computadores pessoais e portáteis, televisores de alta definição, entre outros (DINIZ, 2009). O crescimento da demanda já ultrapassou 50% nos últimos tempos, e tende a aumentar ainda mais nos próximos anos[1]. Além disso, a evolução das tecnologias de aquisição, compressão e transmissão de dados tem facilitado a maneira como vídeos digitais são criados, armazenados e distribuídos.

Um vídeo digital é composto, basicamente, por uma série de imagens estáticas apresentadas em sequência a uma determinada taxa de amostragem. Segundo Corrêa (2010), quanto maior a taxa de amostragem, melhor a qualidade do vídeo e maior a quantidade de dados utilizados para representá-lo. Grande parte dos dados que compõem o vídeo digital é redundante e o seu volume pode ser diminuído através de técnicas de compressão. Sendo assim, o alto volume de dados presentes nos vídeos digitais tem impulsionado a pesquisa sobre técnicas de compressão a fim de facilitar a transmissão de tais vídeos pelos dispositivos existentes no mercado.

A codificação para compressão de dados de imagem e vídeo é indispensável, não só para aplicações relacionadas à televisão digital, ou seja, a transmissão e reprodução em tempo real de vídeos de alta definição, mas também às mais diversas aplicações multimídia. Sendo assim, a codificação de vídeo é o processo de representação digital de um vídeo, que tem como principal objetivo a compressão dos dados para otimizar seu armazenamento e transmissão (ARAUJO, 2010).

Alguns padrões de codificação de vídeo foram criados com o intuito de fornecer uma maneira única de tratamento dos dados e permitir a integração entre diferentes implementações. Estes padrões, normalmente, especificam como deve ser implementado o decodificador ou como deve ser organizada o *bitstream* (saída do codificador), deixando em aberto algumas questões que permitem flexibilidade na implementação, principalmente no codificador (DARONCO, 2009).

---

[1] http://socialtimes.com/cisco-predicts-that-90-of-all-internet-traffic-will-be-video-in-the-nextthreeyears_b82819

De acordo com Vianna, *et al.* (2012), a eficiência de um padrão de codificação de vídeo é, geralmente, medida pela sua capacidade de compressão para uma dada qualidade de imagem. Uma codificação mais eficiente permite a transmissão ou armazenamento de mais conteúdo com a mesma qualidade, objetivando melhorar a resolução ou a definição de um vídeo utilizando a mesma taxa de bits. Por esta razão, há um esforço constante para desenvolver padrões de codificação de vídeo novos e mais eficientes. Duas organizações internacionais são tidas como referência pelos seus padrões desenvolvidos: a ITU-T (*International Telecommunication Union – Telecommunication sector*) e a ISO/IEC (*International Organization for Standardization – International Electrotechnial Commission*). Cada uma possui um grupo destinado, especificamente, à codificação de vídeo. Eles são o VCEG (*Video Coding Experts Group*) da ITU-T e o MPEG (*Moving Pictures Experts Group*) da ISO/IEC (DARONCO, 2009).

O padrão de codificação de vídeo mais recente é o HEVC (*High Efficiency Video Coding* – Codificação de Vídeo de Alta Eficiência) publicado pela ITU-T e ISO/IEC. Segundo, Sullivan, *et al.* (2012), a principal característica do padrão HEVC é a sua melhoria significativa no desempenho de compressão quando comparado com o padrão anterior (H.264/AVC), com redução de pelo menos 50% na taxa de bits, porém, mantendo a mesma qualidade visual. Este padrão alcança duas grandes realizações adicionais, as quais são: lidar com maior resolução de vídeo através da introdução de tamanho de blocos de codificação maiores e capitalizar sobre arquiteturas digitais de processamento paralelo no projeto de codificador de vídeo para diminuir o tempo de codificação (TEW, *et al.* 2014).

O CABAC é um método de codificação de entropia introduzido pela primeira vez no H.264/AVC e agora utilizado no padrão mais recente, HEVC (SZE, *et al.* 2014). A codificação de entropia é um esquema de compressão sem perdas que usa as propriedades estatísticas para comprimir dados, tal que o número de bits utilizados para representar os dados é proporcional à probabilidade dos mesmos aparecerem mais ou menos vezes. Assim, os elementos sintáticos de vídeo que tendem a aparecer menos vezes, tendem a ter códigos maiores, enquanto que os que aparecem mais vezes tendem a ter códigos menores.

1.1 Objetivos

Este trabalho tem como objetivo geral desenvolver e validar uma arquitetura em hardware do bloco de binarização do CABAC, baseado no padrão de codificação de vídeo HEVC, procurando-se obter redução no consumo da arquitetura final ao se utilizar técnicas específicas para baixo consumo de potência. Para este fim, têm-se os seguintes objetivos específicos:

- Realizar um estudo sobre codificação de vídeo e seu padrão mais recente: o HEVC;
- Estudar da literatura sobre o CABAC e soluções do estado da arte para o bloco de binarização;
- Desenvolver da descrição da arquitetura em VHDL;
- Utilizar o software de referência do padrão HEVC como modelo para simular e validar a arquitetura;
- Apresentar uma análise sobre o seu desempenho e consumo de potência após sintetizar a arquitetura para ASIC (*Application Specific Integrated Circuit*).

1.2 Organização do Texto

O trabalho está organizado da seguinte forma: No capítulo dois serão apresentados os conceitos gerais e revisão da literatura. O capítulo três apresenta o bloco de binarização, onde serão descritos o seu funcionamento e os tipos de binarização, além dos trabalhos correlatos. A metodologia adotada neste trabalho será apresentada no capítulo 4. No capítulo 5, será apresentada a arquitetura proposta para o bloco de binarização do CABAC, onde será exibido o funcionamento dos blocos desenvolvidos. No capítulo 6, serão apresentados os resultados e discussões. Por fim, no capítulo 7, será apresentada a conclusão.

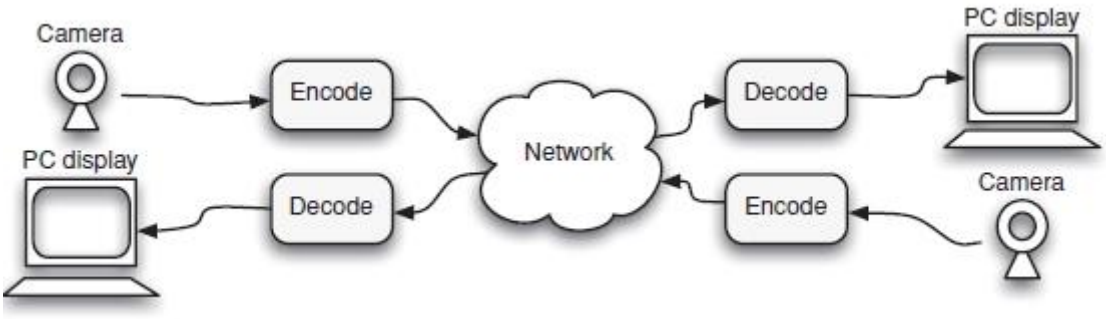
2 CONCEITOS GERAIS E REVISÃO DE LITERATURA

Este capítulo apresenta a revisão bibliográfica construída para o desenvolvimento do presente trabalho. Primeiro, serão exibidos os conceitos gerais de vídeo digital e compressão de dados. Em seguida, será feita uma descrição do padrão que é utilizado como base neste trabalho, o HEVC, onde serão apresentados aspectos básicos da sua estrutura. Por fim, será descrito como funciona o CABAC, apresentando as suas principais características e os blocos que o compõem.

2.1 Conceitos de Vídeo Digital e Compressão de Dados

A compressão de vídeos digitais é um dos assuntos mais discutidos atualmente, tanto pela academia quanto pela indústria, e possui um grande número de aplicações. Com ela, a quantidade de bits utilizada para representar as sequências de vídeo, pode ser reduzida drasticamente, facilitando o armazenamento ou a transmissão das mesmas (PORTO, 2008). Um exemplo de compressão de vídeo digital é apresentado na Figura 1. Para realizar uma chamada de vídeo, cada participante contém um codificador e um decodificador. O vídeo de uma câmera é codificado e transmitido através de uma rede, quando chegar ao seu destino, será decodificado e exibido. Isto ocorre em duas direções ao mesmo tempo (RICHARDSON, 2010).

Figura 1 - Codificação de vídeo de uma chamada de vídeo.



Fonte: Richardson, 2010.

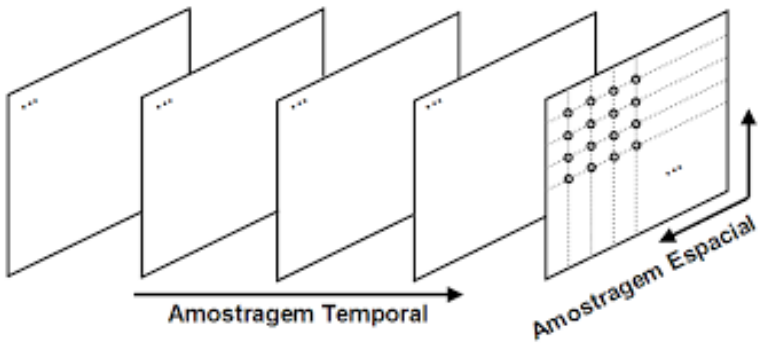
Assim, para compreender melhor os conceitos de codificação de vídeo, esta seção foi organizada da seguinte maneira. Primeiro, será explicado o que é um vídeo digital, o que é um *pixel* e o espaço de cores. Na seção seguinte, os conceitos

de compressão de vídeo digital são apresentados, onde serão exibidas as duas técnicas de compressão de dados, em geral, sem perdas e com perdas. Por fim, será apresentado os tipos de redundância de dados, que são: espacial, temporal e entrópica (ou estatística).

2.1.1 Vídeo Digital

Um vídeo digital é composto por uma série de imagens exibidas em sequência sob uma determinada taxa temporal. Uma imagem digital é formada por vários pontos discretos, chamados de *pixel* (*Picture Element* – Elemento da imagem), agrupados em uma matriz bidimensional. Cada imagem que compõe o vídeo é chamada de quadro (*frame*). A frequência em que os quadros são reproduzidos tem que ser suficiente para fornecer a percepção de movimento ao espectador (MARTINS, 2011). Em um sinal de vídeo digital, esses quadros passam também por uma amostragem espacial, na qual cada *pixel* representa uma determinada intensidade de brilho ou de cor (MANOEL, 2007). Um exemplo de amostragem temporal e espacial é ilustrado na Figura 2.

Figura 2 - Amostras temporais e Amostras espaciais.



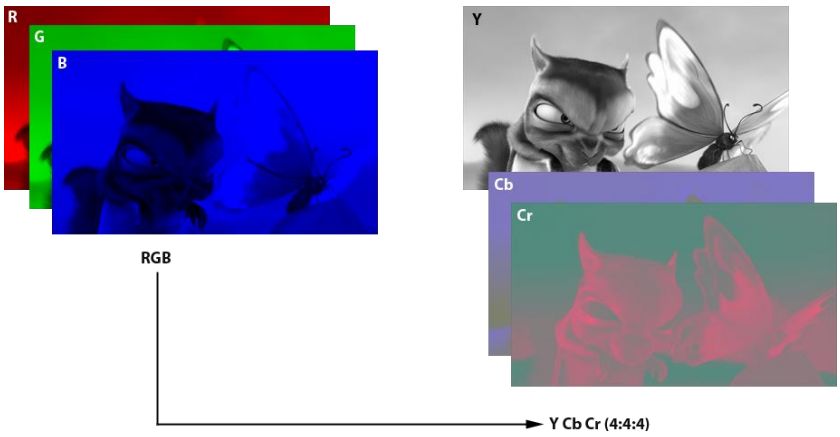
Fonte: Hung, 2007.

É possível representar cada *pixel* de uma imagem de vídeo por três componentes ou amostras de cores, a partir da fisiologia do sistema visual humano e da teoria das cores. Segundo Junior (2011), um espaço de cor é uma especificação de um sistema de coordenadas tridimensionais e um subespaço dentro desse sistema, no qual cada cor é representada por um único ponto. O espaço de cores mais comum e conhecido para representar imagens digitais é o RGB (*Red*, *Green*,

Blue). O espaço de cor RGB é um cubo, no qual a diagonal principal representa valores de cinza do preto ao branco e qualquer ponto (cor) é representado pela soma ponderada de vermelho (R), verde (G) e azul (B).

O espaço de cores para compressão de vídeo mais usado, por outro lado, é o YCbCr (luminância, crominância azul e crominância vermelha, do inglês *Luminance*, *Chrominance Blue*, *Chrominance Red*). De acordo com Diniz (2009), este espaço de cores é usado para compressão porque, a informação de cor (crominância) está separada da informação de brilho (luminância), ao contrário do RGB. Podendo ser aplicadas ferramentas de compressão distintas para luminância e crominância neste caso. Na Figura 3, temos um exemplo de passagem do espaço de cor RGB para o YCbCr.

Figura 3 - Espaço de cor RGB para YCbCr.

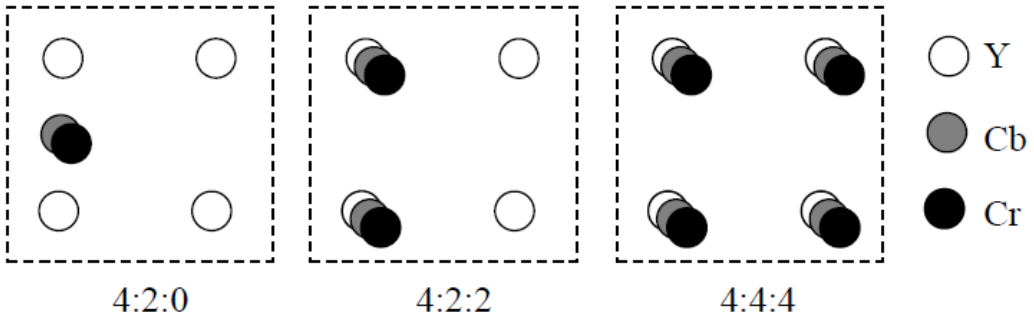


Fonte: Spears & Munsil – Hand Forged Video.

Essa característica do espaço de cores YCbCr pode ser utilizada para reduzir a quantidade de informação necessária a ser processada através de uma operação conhecida como subamostragem de cores. Segundo Martins (2011), a subamostragem de cores é obtida através da redução da resolução espacial da informação de crominância no espaço YCbCr. Existem várias formas de relacionar os componentes de crominância com o componente de luminância para realizar a subamostragem. Os formatos mais comuns são o 4:4:4, o 4:2:2 e o 4:2:0. No formato 4:4:4, para cada quatro amostras de luminância (Y), existem quatro amostras de crominância azul (Cb) e quatro amostras de crominância vermelha (Cr) (isto é, a subamostragem não foi aplicada). No formato 4:2:2, para cada quatro

amostras de (Y) na direção horizontal, existem apenas duas amostras de Cb e duas amostras de Cr. Neste caso, as amostras de crominância possuem a mesma resolução vertical das amostras de luminância, mas possuem metade da resolução horizontal. No formato 4:2:0, para cada quatro amostras de Y, existe apenas uma amostra de Cb e uma amostra de Cr. Neste caso, as amostras de crominância possuem metade da resolução horizontal e metade da resolução vertical do que as amostras de luminância (ROSA, 2010). A Figura 4 apresenta a localização geográfica da informação de crominância em um bloco de 2x2 *pixels* da imagem, que formam o padrão 4:2:0, 4:2:2 e 4:4:4. Para imagens de qualquer tamanho, este padrão de 2x2 *pixels* se repete.

Figura 4 - Subamostragem de Cores.



Fonte: Rosa, 2010.

2.1.2 Compressão de Vídeo Digital

A compressão de vídeo é essencial para o sucesso das aplicações que manipulam vídeos digitais, pois um vídeo não comprimido utiliza uma quantidade de bits muito elevada (AGOSTINI, 2007). Isso implica em custos maiores em termos de armazenamento e transmissão destas informações. Segundo Richardson (2003), a compressão de vídeo tem duas vantagens importantes. Em primeiro lugar, torna-se possível a utilização de vídeo digital em ambientes de transmissão e de armazenamento que não irá dar suporte a vídeos não comprimidos. Por exemplo, taxas de transferência de Internet atuais são insuficientes para lidar com vídeo não comprimido em tempo real (mesmo em baixas taxas de *frames* e/ou tamanho do *frame* pequeno). Em segundo lugar, a compressão de vídeo permite a utilização mais eficiente de transmissão e armazenamento.

Existem dois tipos de técnicas de compressão de dados em geral: sem perdas de informação (*lossless compression*) ou com perdas de informação (*lossy compression*). Técnicas de compressão sem perdas de informação codificam os dados de forma que possam ser reconstruídos de maneira idêntica aos dados originais. Este tipo de compressão é utilizado para codificação de arquivos onde não é admitida a perda de informação (DINIZ, 2009).

Em técnicas de compressão com perdas de informação, os dados descomprimidos não são idênticos aos dados originais. Segundo Sanches (2001), esta técnica é usada na compressão de imagens e vídeos que procura tirar vantagem das limitações da visão humana, uma vez que a perda de algumas informações pode não ser percebida decorrente do fato da visão humana ser mais perceptível a luminosidade do que a variação de cores.

Figura 5 -Compressão de dados com perdas de informação.

Fonte: Oliveira, 2009.

A Figura 5 apresenta um exemplo de compressão de dados com perdas de informação, onde é possível observar que houve perda de resolução da primeira imagem para a segunda. Segundo Porto (2008), a compressão de vídeo está baseada em três tipos de redundância: entrópica, espacial e temporal. Na próxima seção será detalhado cada um desses tipos.

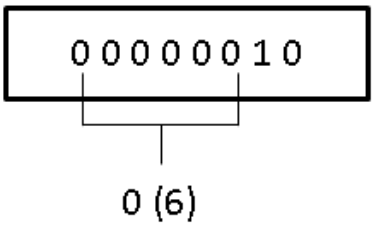
2.1.3 Redundância de Dados

A principal atividade dos algoritmos de compressão de dados de qualquer natureza é tentar identificar redundâncias que se manifestam através de correlações, repetições e distribuições estatísticas não uniformes existentes em um conjunto de

dados e encontrar um novo conjunto (o menor possível) que represente o conjunto original através de regras definidas. Segundo Rosa (2010), a compressão de vídeo é uma aplicação específica da compressão de dados onde uma série de características do conjunto de dados é conhecida, facilitando o trabalho dos algoritmos de compressão. Existem três tipos de redundância de dados que podem ser exploradas na compressão de vídeo: redundância entrópica, redundância espacial e redundância temporal.

- **Redundância Entrópica:** Está relacionada com a forma de representação computacional dos símbolos codificados e não se relaciona diretamente ao conteúdo da imagem. Segundo Martins (2011), a redundância entrópica existe quando as frequências de ocorrência dos símbolos são diferentes entre si. Não há redundância entrópica a ser explorada, quando for assumido que a distribuição de ocorrência de símbolos é uniforme, ou seja, uma média. Por outro lado, se a distribuição dos dados é uma curva não uniforme como, por exemplo, uma curva normal ou exponencial, a redundância entrópica existe e quanto menor a variância dos dados nas curvas não uniformes, mais redundância entrópica existe entre os dados e mais esses dados poderão ser comprimidos. Um exemplo disso pode ser visto na Figura 6. Onde a sequência de zeros pode ser representada por um bit 0 e um byte na sequência indica o número de zeros.

Figura 6 - Redundância Entrópica.

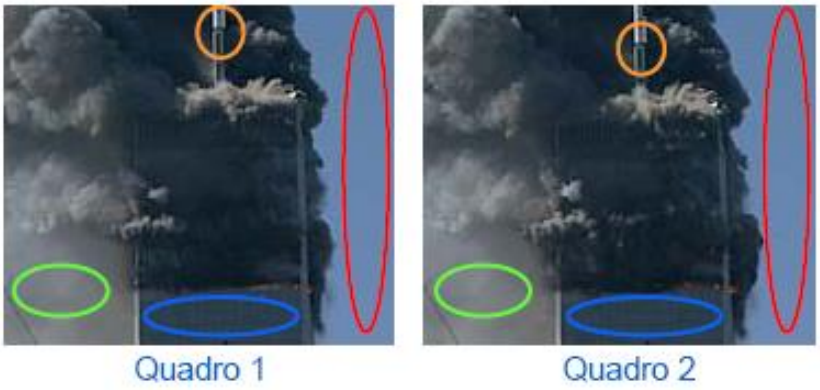


Fonte: Próprio autor, 2016.

- **Redundância Espacial:** Este tipo de redundância é também denominada de redundância intra-quadro e é a correlação entre os *pixels* vizinhos de um mesmo quadro. Ao longo de uma linha ou coluna os *pixels* vizinhos variam gradativamente de intensidade, possuindo valores semelhantes entre si, com exceção das regiões de borda. Desta maneira, os compressores exploram esta

redundância através de funções de auto correlação (THIELE, 2012). Um exemplo de redundância espacial pode ser visto na parte circulada em vermelho do quadro 1 da Figura 7. Onde os valores dos *pixels* no espaço são muito parecidos ou iguais.

Figura 7 - Redundância Espacial e Temporal.



Fonte: Dorothée, 2013.

- **Redundância Temporal:** Este tipo de redundância também é chamado de redundância inter-quadros e é causada pela correlação existente entre quadros temporalmente próximos em um vídeo. Muitos blocos de *pixel* simplesmente não mudam de valor de um quadro para outro em um vídeo (como é apresentado na Figura 7, onde as partes circuladas permaneceram iguais do quadro 1 para o quadro 2). Outros *pixels* apresentam uma pequena variação de valores causada, por exemplo, por uma variação de iluminação. Por fim, também é possível que o bloco de *pixels* simplesmente tenha se deslocado de um quadro para o outro. De acordo com Agostini (2007), todos os padrões atuais de codificação de vídeo visam aproveitar a redundância temporal durante a codificação. A exploração eficiente da redundância temporal conduz a elevadas taxas de compressão, o que é fundamental para o sucesso dos codificadores.

2.2 O Padrão HEVC

O padrão H.264/AVC foi lançado em 2003 e atingiu seu objetivo, que era dobrar as taxas de compressão quando comparado com os padrões anteriores, tais como o MPEG-2 (VIANNA, *et al.* 2012). No entanto, a procura de resoluções mais

elevadas e melhor qualidade de imagem em vídeos digitais continuam a crescer, enquanto que as redes de comunicação de dados, especialmente por meio de tecnologia sem fio, continuam a limitar o fator para a transmissão de vídeos em HD com alta qualidade.

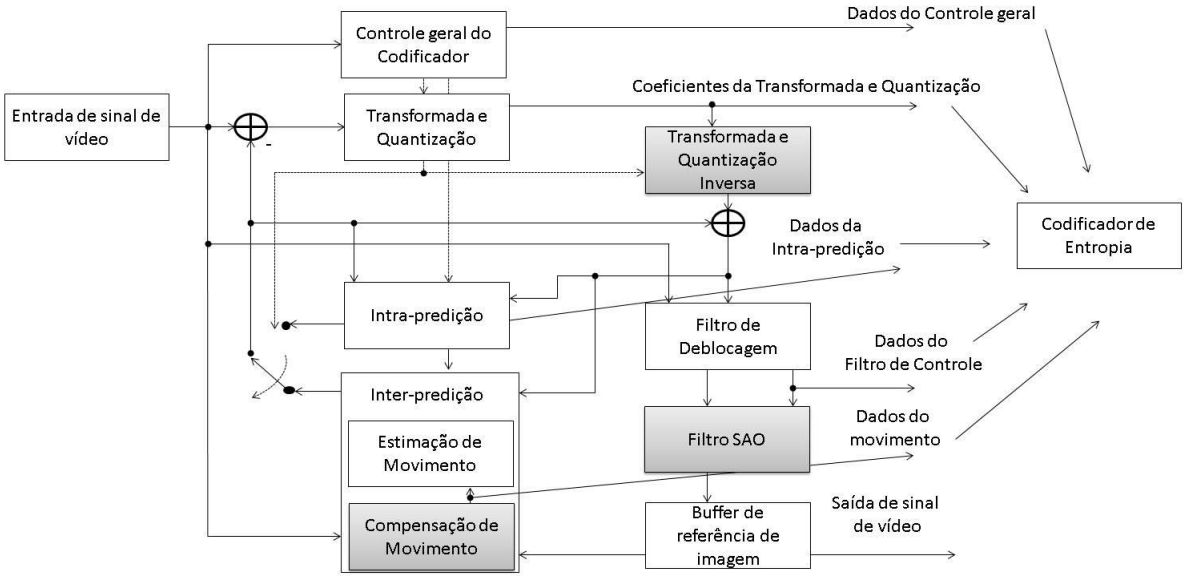
Com isso em mente, os especialistas da ITU-T VCEG (*Video Coding Experts Group*) e ISO/IEC MPEG (*Moving Picture Experts Group*) uniram esforços para desenvolver um novo padrão para a próxima geração de codificação de vídeo, formando o JCT-VC (*Joint Collaborative Team on Video Coding*). Segundo Oliveira e Alencar (2014), a primeira edição do padrão HEVC foi publicada em janeiro de 2013, em duas normas com mesmo conteúdo divulgadas pelos órgãos envolvidos.

O principal objetivo do padrão HEVC é reduzir em cerca de 50% a taxa de bits necessária quando comparado com o H.264/AVC, porém, mantendo a mesma qualidade de imagem. De acordo com Sze e Budagavi (2012), para melhorar a eficiência de codificação, o HEVC usa blocos de transformadas maiores e filtros adicionais, dentre outros melhoramentos.

2.2.1 Blocos do Padrão HEVC

Esta seção irá apresentar os principais blocos do codificador do HEVC, mostrando as suas funcionalidades. A codificação híbrida baseada em blocos usada no padrão HEVC é a mesma que foi utilizada nos codificadores de vídeo anteriores, como o (H.264/AVC). Segundo Moreira (2014), para codificar o conteúdo, os quadros de vídeo são divididos em blocos e esses blocos são codificados individualmente. Para isso, técnicas de predição são aplicadas, baseadas em blocos vizinhos da mesma imagem (predição intra-blocos ou intra-predição) ou de imagens previamente codificadas (por estimativa de compensação de movimento, isto é, inter-predição). A diferença entre o resultado previsto e os dados do vídeo original é posteriormente codificada por uma aplicação dos blocos de Transformada e Quantização. Desta forma, um bloco pode ser representado por apenas alguns coeficientes diferentes de zero. A quantização de coeficientes de transformação, vetores de movimento, direções de predição, modos de blocos e outros tipos de informação (isto é, elementos sintáticos do codificador) são então codificados via o codificador de entropia sem perdas na codificação.

Figura 8 - Diagrama de blocos do codificador do HEVC.



Fonte: Próprio autor, 2016.

Os blocos do codificador são brevemente detalhados nas próximas seções, de acordo com o fluxo de dados do HEVC, que é apresentado na Figura 8. Como o foco deste trabalho trata do estudo do bloco de binarização da codificação de entropia, este será descrito detalhadamente no capítulo 3.

2.2.1.1 Inter-predição

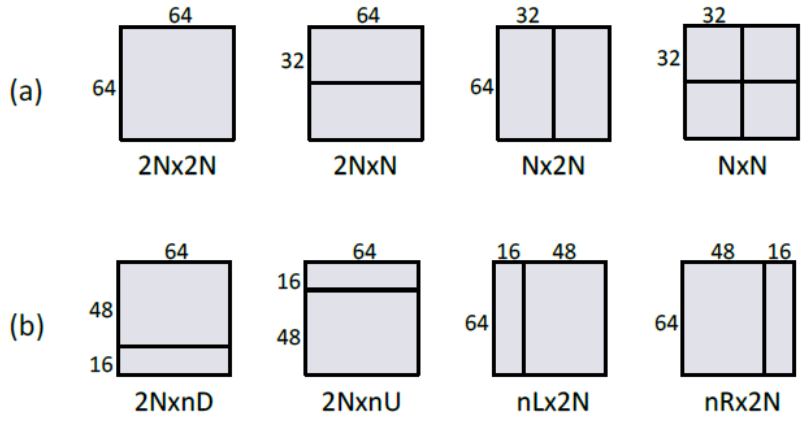
A Inter-predição faz uso da correlação temporal entre as imagens a fim de obter uma previsão de compensação de movimento (MCP – *Motion Compensation Prediction*) para blocos de amostra de imagens. Segundo Moreira (2014), para os padrões anteriores, a maior entidade que poderia ser codificada era um macrobloco (16 x 16 *pixels*). Já para o HEVC, foram introduzidas estruturas de blocos maiores, com mecanismos flexíveis de particionamento. Dessa forma o bloco básico do HEVC é chamado de LCU (*Largest Coding Unit*) “grande unidade de codificação” ou CTU (*Coding-Tree Unit*), que possui um tamanho máximo de 64 x 64 *pixels*.

Cada CTU é a raiz de uma “*quadtree*” (uma espécie de super macrobloco) e pode ser, de forma recursiva, dividida em outras pequenas unidades de codificação chamadas de CU (*Coding Units*), que por sua vez, ainda podem ser divididas em pequenas Unidades de Predição, as “PUs” (*Prediction Units*). O HEVC define que a

inter-predição deve ser realizada para cada partição de PU quando quadros do tipo preditivo (P) ou bi-preditivo (B) são codificados (MOREIRA, 2014).

Inicialmente, quatro tipos de partições de PU foram definidas. Esse conjunto é composto por partições de movimento simétricas (PMS) quadradas e retangulares: 2Nx2N, 2NxN, Nx2N e NxN, onde 2N é equivalente à dimensão da CU. De acordo com Grellert (2012), quatro partições de movimento assimétricas (PMA) adicionais foram propostas, podendo aumentar a eficiência da codificação, pois padrões irregulares de imagem podem ocorrer e as PMS não representariam uma solução adequada.

Figura 9 - Partições de movimento (a) simétricas e (b) assimétricas.



Fonte: Grellert *et al.*, 2012.

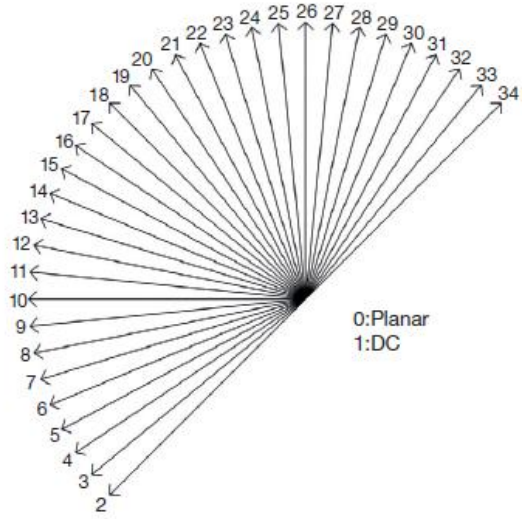
A Figura 9 ilustra as partições de movimento simétricas (a) e assimétricas (b) possíveis para uma CU de tamanho de 64x64. De acordo com Grellert (2012), quando a predição inter-quadros é processada, a estimação de movimento (ME – *Motion Estimation*) é chamada para cada partição de PU possível. Essa ferramenta consiste em tentar descobrir para onde um objeto se deslocou entre um quadro e outro no vídeo.

De modo geral, o quadro sendo processado em blocos é dividido, buscando-se encontrar a região mais semelhante a ele dentro de um quadro previamente codificado (chamado quadro de referência). Quando o melhor casamento é encontrado, ao invés de reproduzir todas as amostras do bloco, um vetor de movimento que aponta para o melhor casamento no quadro de referência é gerado.

2.2.1.2 Intra-predição

A intra-predição é uma técnica que permite dentro de um mesmo *frame* prever o bloco a partir de um bloco adjacente. Os *pixels* adjacentes ao bloco que está sendo decodificado, nesta técnica, são utilizados como referência para prever os *pixels* deste bloco (TEIXEIRA, 2014). Desta forma, as imagens onde o conteúdo é relativamente contínuo e repetitivo possuem taxas de compressão ainda maiores.

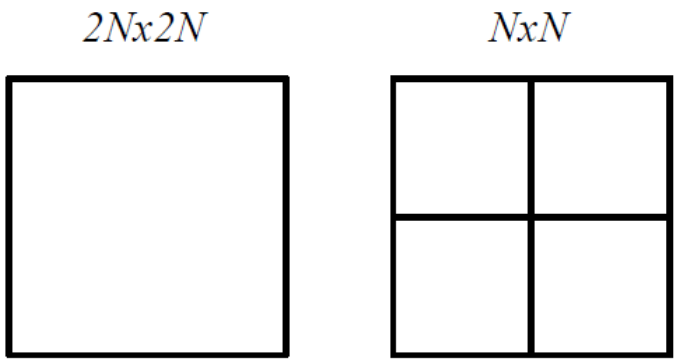
Figura 10 - Modo de intra-predição.



Fonte: Lin e Lai, 2016.

A especificação HEVC contém mais modos de intra-predição do que o H.264/AVC. Na Figura 10 é possível ver estes modos, onde 0 é o modo planar, 1 é o modo DC e 2 a 34 são modos de predição angulares com diferentes direções. Cada vetor mostrado na Figura 10, representa a direção dentro de um mesmo quadro a partir de um *pixel* central, de onde se vai derivar os valores ponderados para cada um daqueles *pixels* no vetor a partir dos demais. De acordo com Palomino (2013), na intra-predição, cada CU pode ser dividido em PUs, onde os passos de predição são realmente aplicados. Dois tamanhos de partição são suportados na intra-predição no HEVC, 2Nx2N e NxN, onde N é ½ do tamanho do CU (exemplo: para uma CU 64x64, N é 32). A Figura 11 mostra os dois tipos de partição.

Figura 11 - Possíveis tipos de partição na intra-predição.

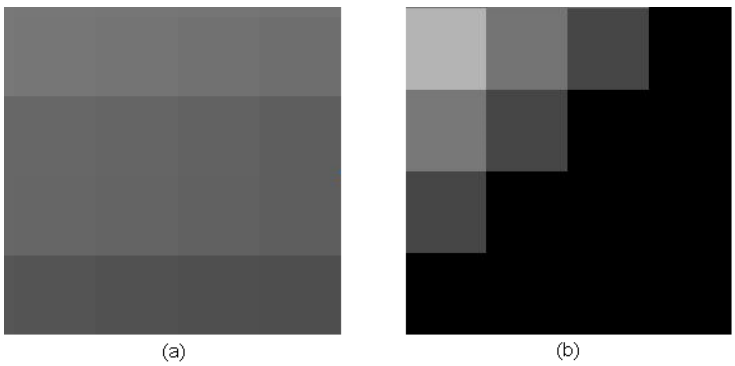


Fonte: Palomino, 2013.

2.2.1.3 Transformadas e Quantização

Os *pixels* que estão sendo codificados na intra-predição ou inter-predição, podem não ser idênticos aos *pixels* do quadro de referência. Essa diferença gera os resíduos, e esses então irão passar pelo processo de transformada e quantização. As transformadas são responsáveis por traduzir valores de resíduos para o domínio de frequência. Este processo também compacta a energia no lado superior esquerdo do bloco, o que é útil para a quantização e a entropia que o seguem. Como na predição, as transformadas no HEVC também têm modos diferentes (DA SILVA, 2014). A Figura 12 mostra um exemplo de um bloco 4x4 após o processo de transformadas. Segundo Ramos (2010), é possível observar que, após passar pelo processo, os dados de mais alta frequência são zerados, e isto faz com que os valores diferentes de zero se concentrem no canto superior esquerdo do bloco.

Figura 12 - Bloco 4x4 (a) Original (b) Após passar pelo processo de transformada.



Fonte: Ramos, 2010.

As unidades de transformação (TUs) são a unidade básica para os processos de transformação e quantização, e podem assumir tamanhos de amostras de 4x4 até 32x32. Existem várias funções de transformada que são utilizadas para a compressão de imagens. Uma transformada comum que pode ser utilizada é a transformada de cossenos discreta, chamada também de DCT (*Discrete Cosine Transform*). Esta transformada é baseada na transformada de Fourier e é muito utilizada na sua forma original ou em variação baseadas que levam em conta a redução da complexidade da implementação e uma melhor compressão. No caso específico de uma transformada 4x4 existe também uma transformada discreta dos senos (DST – *Discrete Sine Transform*) para um modo específico da intra-predição.

Segundo Teixeira (2014), um passo importante para aumentar a compressão do bloco é a quantização. É a partir dela que temos uma redução significativa do número de bits codificados. A quantização é análoga a uma divisão inteira dos resíduos pós-transformada. Por outro lado, a quantização causa uma perda na qualidade visual que pode ser significativa, pois insere perdas no processo. A Figura 13 apresenta um exemplo de um bloco 4x4 após o processo de quantização.

Figura 13- Bloco 4x4 (a) Após o processo de transformada (b) Após o processo de quantização.

Fonte: Ramos, 2010.

2.2.1.4 Filtros

O HEVC especifica dois filtros, um filtro de deblocagem (DBF – *Deblocking Filter*) e um filtro de amostra de deslocamento adaptativo (SAO – *Sample Adaptative Offset*). Os filtros são aplicados nos laços de codificação e decodificação, após a quantização inversa e antes de salvar a imagem no buffer de imagem decodificada. O filtro de deblocagem é aplicado em primeiro lugar. Ele diminui descontinuidades

na predição e transforma limites de bloco. O segundo filtro, SAO, é aplicado na saída do filtro de deblocagem e melhora ainda mais a qualidade da imagem decodificada por meio da atenuação de artefatos e mudanças na intensidade da amostra de algumas áreas de uma imagem (SZE, *et al.* 2014).

No padrão HEVC, o filtro de deblocagem é aplicado apenas aos limites da PU e TU, que contam com uma grade de amostras tanto para luminância quanto para crominância. De acordo com Souza, *et al*. (2015), para cada limite, uma filtragem *Boundary Strength* (BS) é avaliada, de acordo com várias condições de blocos vizinhos. O valor resultante BS varia entre 0 e 2, em que 0 significa que nenhum filtro de deblocagem deverá ser aplicado. Quando um dos blocos vizinhos é de intra-predição, o valor BS é ajustado para 2. Além disso, apenas quando o valor BS é dois, as amostras de crominância são filtradas. Para amostras de luminância, condições adicionais são verificadas para determinar se o DBF deve ser aplicado.

As amostras reconstruídas são processadas pelo filtro SAO logo depois de ser filtrado pelo DBF. As amostras são posteriormente modificadas pela adição de um valor de deslocamento cuja magnitude depende de um conjunto de parâmetros do SAO, nomeados de tipo, quatro valores de deslocamento, e posição da banda/classe de borda. Estes parâmetros SAO são codificados no fluxo de bits para cada CTU e podem possuir valores diferentes para os componentes de luminância e crominância de cada CTU (SOUZA, *et al*. 2015).

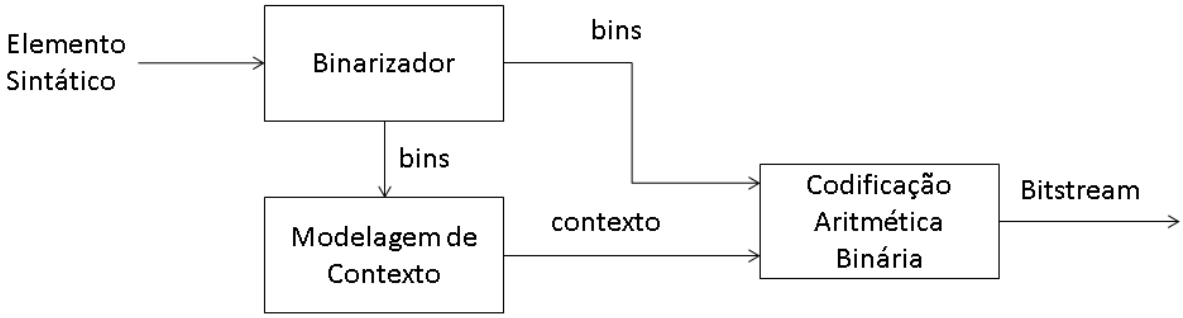
A vantagem mais importante dos filtros é melhorar a qualidade das imagens reconstruídas. Segundo Sze, *et al*. (2014), utilizando os filtros no circuito de decodificação também aumenta a qualidade das imagens de referência e, portanto, também a eficiência de compressão.

2.2.1.5 Codificador de Entropia

Após as transformadas e a quantização, além de todos os demais processos de codificação, a codificação de entropia é aplicada para codificar todos os elementos e coeficientes que foram quantificados. Estes elementos são chamados de elementos sintáticos (SE – *Sintax Elements*). Segundo Moreira (2014), no HEVC há um único codificador de entropia, ele é denominado de CABAC e é similar ao utilizado no H.264/AVC, mas com algumas modificações devido a atualizações no novo padrão. O CABAC é uma técnica de compressão sem perdas, e é superior a

outros algoritmos de codificação de entropia utilizados nos padrões anteriores. O CABAC envolve três etapas principais que são apresentadas na Figura 14.

Figura 14 - Bloco de binarização do CABAC.



Fonte: Próprio autor, 2016.

1) **Binarizador**: Na etapa da binarização, uma cadeia binária é gerada para cada elemento sintático não binário. Cada bit gerado por este processo é chamado de "*bin*". Vários formatos básicos de binarização são usados no HEVC, como unário (U), truncado unário (TU), truncado rice (TR), kth-order Exp-Golomb (EGk) e tamanho fixo (FL). A maioria dos elementos sintáticos usam os tipos básicos de binarização, ou alguma combinação dos mesmos. Alguns outros usam formatos de binarização personalizado. Comparando com o H.264/AVC, os formatos básicos de binarização são quase iguais, mas os processos de combinação e binarização personalizado são diferentes (ZHOU *et al*., 2013).
2) **Modelagem de Contexto**: A modelagem de contexto fornece uma estimativa da probabilidade exata necessária para se obter uma elevada eficiência de codificação. Assim, é altamente adaptável e diferentes modelos de contexto podem ser utilizados para *bins* diferentes e a probabilidade do modelo de contexto é atualizada com base nos valores dos *bins* previamente codificados. *Bins* com distribuições semelhantes, muitas vezes compartilham o mesmo modelo de contexto. De acordo com Sze, *et al*. (2012), o HEVC usa o mesmo método de atualização de probabilidade do H.264/AVC. No entanto, a lógica de seleção de contexto foi modificada para melhorar o rendimento.
3) **Codificação Aritmética Binária**: Os *bins* são comprimidos em bits utilizando codificação aritmética (isto é, vários *bins* podem ser representados por um único bit). Isto permite que os elementos sintáticos sejam representados por um número

fracionário de bits, o que melhora a eficiência de codificação. Segundo Sze e Budagavi (2013), a codificação aritmética envolve subintervalo de divisão recursiva, onde um intervalo é dividido em dois subintervalos com base na probabilidade do símbolo que está sendo comprimido. Os bits codificados representam um deslocamento que, quando convertido para uma fração de binário, seleciona um dos dois subintervalos, que indica o valor do *bin* decodificado. De acordo com Martins (2011), existem dois modos de codificação aritmética, o regular e o bypass. O modo regular de codificação aritmética utiliza a modelagem de contexto para definir os limites dos valores de intervalo e deslocamento para o cálculo aritmético. O bypass é um modo especial para *bins* com distribuição probabilística considerada pelo CABAC como uniforme. Assim, não precisam de modelagem de contexto, pois são considerados equiprováveis.

Os blocos de binarização e modelagem de contexto foram modificados no HEVC, enquanto a codificação aritmética binária permaneceu a mesma do H.264/AVC. Entre as técnicas que foram utilizadas para melhorar a taxa de transferência do CABAC no HEVC estão: redução do número total de *bins*, redução do número de *bins* que precisam de modelagem de contexto e agrupamento de *bins* bypass (SZE e BUDAGAVI, 2013).

3 BLOCO DE BINARIZAÇÃO

O processo de binarização consiste no mapeamento de valores inteiros em uma sequência de bits que representam o valor original. Segundo Martins (2011), esse mapeamento é feito para reduzir o tamanho do alfabeto de símbolos, simplificando assim os custos da modelagem de contexto e facilitando a tarefa da codificação aritmética binária. O tamanho do "*binstring*" gerado para cada valor de entrada é variável e depende do tipo de elemento sintático que está sendo processado e do contexto atual. O mapeamento binário com tamanhos variáveis de *bins* é utilizado com o objetivo de gerar a menor representação possível para os valores de elementos sintáticos que ocorrem com mais frequência e ainda ter um modelo de probabilidade consistente para cada *bin*.

Existe um grande conjunto de métodos usados para converter os SEs em *binstring*. O CABAC define um conjunto de dez métodos de binarização, alguns deles compostos pela combinação de dois métodos através de um esquema de prefixo-sufixo. Os formatos básicos são unário (U), truncado unário (TU), truncado rice (TR), tamanho fixo (FL) e Exp-Golomb (EGk) (VIZZOTTO *et al*., 2015). Além disso, alguns SEs são codificados com métodos derivados destes ou formatos de binarização personalizados.

3.1 Binarização para códigos Unário, Truncado Unário e Tamanho Fixo

O método Unário converte um número inteiro sem sinal numa sequência de '1's acrescido de um '0' ao final, com tantos '1's quanto for o valor do número. O método Truncado Unário funciona da mesma maneira que o método Unário, mas limitado por uma variável chamada de cMax que determina o tamanho máximo do *binstring*. Quando o valor unário extrapola o valor de cMax a sequência binária acaba sendo representada somente por '1's (THIELE, 2012). O método Tamanho Fixo não recodifica o valor do SE, apenas limita o número de bits da cadeia através da variável cMax (MARTINS, 2011). Para calcular o tamanho máximo do *binstring* nesse método, a variável cMax é utilizada como apresenta a equação (1):

$$fixedLenght = (\mathrm{Log2}(\mathrm{cMax} + 1)) \quad (1)$$

A Tabela 1 apresenta exemplos de binarização utilizando os métodos descritos anteriormente. Onde, o ‘N’ representa o valor do elemento sintático, nesta tabela e, nas próximas.

Tabela 1- Exemplos de binarização unário, truncado unário e tamanho fixo.

N	Unário (U)	Truncado Unário (TU) cMax = 7	Tamanho Fixo (FL) cMax = 7
0	0	0	000
1	10	10	001
2	110	110	010
3	1110	1110	011
4	11110	11110	100
5	111110	111110	101
6	1111110	1111110	110
7	11111110	1111111	111

Fonte: Sze *et al*. 2014.

3.2 Binarização para Truncado Rice

O Truncado Rice é um código Rice parametrizado composto por um prefixo e um sufixo. O prefixo é uma *string,* truncado unário, que é definida na equação (2), onde “>>” representa um deslocamento à direita e, ‘N’ representa o elemento sintático,

$$N \gg \mathrm{cRiceParam} \tag{2}$$

onde o maior valor possível é o cMax. O sufixo é uma representação binária de tamanho fixo dos *bins* menos significativos de N. O parâmetro cRiceParam indica o número de *bins* menos significativos (SZE *et al*., 2014). Para cRiceParam = 0, o truncado rice é igual a binarização truncado unária. A Tabela 2 contém dois exemplos de binarização deste método. No primeiro exemplo, o cMax = 7 e o cRiceParam = 0. Neste caso, o truncado rice é igual ao truncado unário. É possível comprovar isto olhando para o exemplo de truncado unário efetuado na Tabela 1,

onde o processo de binarização foi realizado da mesma forma, uma vez que ambos possuem o mesmo valor de cMax.

Tabela 2 - Exemplo de binarização truncado rice.

N	cMax = 7, cRiceParam = 0		cMax = 7, cRiceParam = 1	
	Prefixo	Sufixo	Prefixo	Sufixo
0	0	-	0	0
1	10	-	0	1
2	110	-	10	0
3	1110	-	10	1
4	11110	-	110	0
5	111110	-	110	1
6	1111110	-	111	0
7	1111111	-	111	1

Fonte: Próprio autor, 2016.

No segundo exemplo, o cMax = 7 e o cRiceParam = 1. Neste caso o sufixo está presente e o processo de binarização resultará da concatenação do prefixo com o sufixo. Ou seja, quando o elemento sintático for igual a 7, o elemento sintático binarizado será igual a “1111”.

3.3 Binarização para Exp-Golomb

O método de binarização Exp-Golomb é dado pela concatenação de um prefixo e um sufixo. A variável ‘k’ determina o tamanho inicial de bits do sufixo. Quando o ‘k’ for igual a zero, o sufixo vai iniciar sem nenhum valor e, por consequência, o prefixo também, uma vez que o prefixo é uma sequência de ‘0’s que depende do sufixo. O prefixo vai começar com ‘0’ quando houver um sufixo e vai aumentar quando o sufixo também aumentar o tamanho em bits de sua representação (THIELE, 2012). Ou seja, quando o sufixo tiver 1 bit, o prefixo vai conter um ‘0’, já quando o sufixo tiver 2 bits o prefixo vai ter dois ‘0’s e assim por diante. Além disso, entre o prefixo e sufixo deve ser adicionado o valor 1. Isto é feito somente quando o ‘k’ for igual a zero. A Tabela 3 apresenta dois exemplos de binarização deste método.

Tabela 3 - Exemplo de binarização Exp-Golomb.

N	k = 0			k = 1		
	Prefixo	**/**	**Sufixo**	**Prefixo**	**/**	**Sufixo**
0	-	1	-	-	0	0
1	0	1	0	-	0	1
2	0	1	1	1	0	00
3	00	1	00	1	0	01
4	00	1	01	1	0	10
5	00	1	10	1	0	11
6	00	1	11	11	0	000
7	000	1	000	11	0	001

Fonte: Próprio autor, 2016.

Quando ‘k’ for igual ou maior a um, o prefixo vai ser composto por uma sequência de ‘1’s e, entre o prefixo e o sufixo deverá ser adicionado o valor ‘0’. Através dos exemplos apresentados na Tabela 3, nota-se que o ‘k’ além de determinar o tamanho inicial de bits do sufixo, determina quando o prefixo vai iniciar a sua sequência de ‘0’s. Além disso, é possível perceber que o sufixo é igual à binarização do tamanho fixo, porém não possui valor de cMax.

3.4 Outros métodos de Binarização

A maioria dos elementos sintáticos usam os processos de binarização citados acima. Porém, cinco elementos sintáticos utilizam métodos chamados de Custom, que são tipos de binarização personalizados desenvolvidos exclusivamente para estes elementos sintáticos. Alguns destes custom processam apenas os valores específicos dos elementos sintáticos e, outros são dados pela concatenação dos métodos básicos. Por exemplo, o elemento sintático *cu_qp_delta_abs* usa o TR como prefixo, com cMax = 5 e cRiceParam = 0, mais o EGk como sufixo, com k = 0 (ITU-T, 2015). O valor do prefixo, prefixVal, é derivado a partir da equação (3):

$$\text{prefixVal} = \text{Min}\,(cu\_qp\_delta\_abs, 5) \qquad (3)$$

Quando o prefixVal for maior do que quatro, a cadeia de bin do sufixo vai estar presente. O valor do sufixo, suffixVal, é definido na equação (4):

$$\text{suffixVal} = cu\_qp\_delta\_abs \ - 5 \tag{4}$$

A Tabela 4 apresenta um exemplo deste tipo de binarização.

Tabela 4 - Exemplo de binarização do elemento sintático *cu_qp_delta_abs*.

N	**Prefixo** **cMax = 5 e cRiceParam = 0**	**Sufixo** **k = 0**
0	0	-
1	10	-
2	110	-
3	1110	-
4	11110	-
5	11111	1
6	11111	010
7	11111	011

Fonte: Próprio autor, 2016.

Segundo Sze, *et al*. (2014), alguns elementos sintáticos utilizam processos de binarização personalizados. A Tabela 5 apresenta o processo de binarização do elemento sintático *intra_chroma_pred_mode*.

Tabela 5 - Processo de binarização do *intra_chroma_pred_mode*.

Valor de ***intra_chroma_pred_mode***	***Binstring***
4	0
0	100
1	101
2	110
3	111

Fonte: ITU-T, 2015.

3.5 Trabalhos Correlatos

Para o desenvolvimento desta pesquisa, foram estudados alguns trabalhos correlatos, a fim de auxiliar na construção do conhecimento sobre o tema. A seguir são descritas quatro propostas relevantes relacionadas ao presente trabalho.

3.5.1 Trabalho de Martins

Martins (2011), busca alcançar maior eficiência em área. Neste trabalho, é feita a implementação dos blocos de binarização e modelagem de contexto (BCM – *Binarization and Context Modeling*) do CABAC para o padrão H.264/AVC. Com foco na redução dos recursos de *hardware,* uma arquitetura multiciclo é proposta, pois permite a reutilização de alguns operadores. A arquitetura multiciclo de binarizador suporta todos os métodos de binarização.

Para comparar quais as melhorias atingidas com a arquitetura multiciclo, foi desenvolvida uma arquitetura ciclo único sem a otimização da binarização. Além disso, foi desenvolvida uma terceira arquitetura chamada *context-aware,* que tem como objetivo facilitar a modelagem de contexto. As arquiteturas foram sintetizadas para FPGA. A arquitetura multiciclo apresentou uma redução de área de 58,3% em relação a arquitetura ciclo único. Assim, a arquitetura multiciclo foi a que obteve maior redução de área em relação as três arquiteturas implementadas. Já para frequência, a arquitetura ciclo único atingiu 142,4 MHz, enquanto que a arquitetura multiciclo atingiu 247,5 MHz e a arquitetura *contexto-aware* atingiu 250,9 MHz.

3.5.2 Trabalho de Zhou *et al*

Zhou, *et al*. (2015), propõem otimizações como, redução do caminho crítico da codificação aritmética binária (BAE – *Binary Arithmetic Encoder*), aumento de desempenho do número de *bins* entregue por ciclo de clock (BPCC – *Bins per clock cycle*), e soluções para os problemas da modelagem de contexto. Para isto, é realizada uma análise do gargalo do CABAC e das dependências de dados. Uma pré renormalização (PN) de atualização de valores da codificação aritmética e técnicas de cobertura de caminho crítico (HPC – *Hvbrid Path Coverage*) são propostas para reduzir o caminho crítico do BAE.

Para aumentar o desempenho do BPCC, propõem uma técnica de divisão de *bin* bypass (BPBS – *Bypass bin splitting*), que permite o paralelismo total entre *bins* regulares e *bins* bypass na fase crítica de pipeline do BAE. Com a melhora do BAE, a modelagem de contexto pode se tornar o novo gargalo. Para este problema, é proposta uma solução de estado dual-transição (SDT – *State Dual Transistion*).

Como o bloco de binarização não é o gargalo do CABAC, neste trabalho eles desenvolvem a arquitetura de binarização superestimada, de modo que a sua capacidade de processamento média seja maior do que a do BAE quase todo o tempo. Para isto, eles realizam a binarização de vários SEs em paralelo.

Como resultado, o codificador do CABAC oferece uma média de 4,37 *bins* por ciclo de clock. A frequência máxima do clock atinge 420 MHz, quando sintetizado em tecnologia de 90 nm. A taxa de transferência total correspondente é 1836 Mbins/s que é 62,5% maior do que o estado da arte da arquitetura.

3.5.3 Trabalho de Peng *et al*

Peng *et al.* (2013), propõem um projeto de hardware para o CABAC onde, métodos de codificação que visem aumentar o paralelismo e reduzir o custo de hardware do CABAC são discutidos. Neste trabalho, o BAE é compatível com o padrão H.264/AVC enquanto que os blocos de binarização e modelagem de contexto são redesenhados para o padrão HEVC.

Para o bloco de binarização, foi proposto melhorias para o método Custom 5 (este é um dos métodos utilizados na codificação dos resíduos de transformada, que tende a gerar uma grande parte dos elementos sintáticos da codificação (STANKOWSKI, 2014)). Onde, ao invés de simplesmente mapear a entrada do SE para uma sequência de *bin*, com uma função de mapeamento fixo para o tipo de SE, o processo de binarização é parametrizado por cRiceParam. Assim, a taxa de transferência do pior caso melhora e, os recursos de hardware reservados para o pior caso podem ser salvos. Este trabalho, foi implementado com CMOS de 0.13 um, podendo operar a 357 MHz.

3.5.4 Trabalho de Vizzotto *et al*

Vizzotto, *et al*. (2015), propõem otimizações na renormalização explorando o paralelismo e, melhoras na codificação aritmética binária (BAE), reduzindo o atraso

do caminho crítico e aumentando a taxa de transferência. Para isto, uma arquitetura de binarização simplificada com oito núcleos é apresentada para aumentar o rendimento *binstring* para alcançar o conteúdo UHD e reduzir a área total do CABAC. Na solução proposta, um núcleo é responsável para cada TR e FL enquanto que, dois núcleos são responsáveis pelos Customs, TU e Exp-Golomb. Assim, cada ciclo de clock pode manipular seis entradas de SE, no máximo.

Além disso, eles utilizam circuitos em paralelo no processo de renormalização para melhorar o rendimento. Esta técnica aumenta os *bins* por ciclo de clock para uma média de 2,37. Para realizar a simulação, foram utilizadas sequências de vídeo FullHD (1920x1080 pixels) e WQXGA (2560x1600 pixels). As sequências de vídeo utilizadas na simulação foram: *BasketballDrive, PeopleOnStreet, BQTerrace* e *Kimono*. Os resultados de síntese apresentam uma frequência de 380MHz com 31.180 *gates* no processo 0,13µm CMOS.

3.5.5 Conclusão dos Trabalhos Correlatos

Os trabalhos apresentados na seção 3.5, foram utilizados como referência para o desenvolvimento deste trabalho. Embora nenhum deles seja voltado exclusivamente para a binarização, todos apresentam uma descrição de como foi implementado este bloco. A Tabela 6 apresenta uma comparação dos trabalhos correlatos apresentados nesta seção.

Tabela 6 - Comparação dos Trabalhos Correlatos.

Trabalhos	Padrão	Bloco de Binarização
Martins	H.264/AVC	Propõe uma arquitetura multiciclo buscando alcançar maior eficiência em área.
Zhou e*t al.*	HEVC	Superestima o bloco de binarização desenvolvendo uma arquitetura em paralelo.
Peng *et al.*	HEVC	Propõe melhorias para o método Custom 5, melhorando a taxa de transferência do pior caso.
Vizzotto *et al.*	HEVC	Propõe uma arquitetura de binarização simplificada com oito núcleos para aumentar o rendimento de *binstring* e redução de área.

Fonte: Próprio autor, 2016.

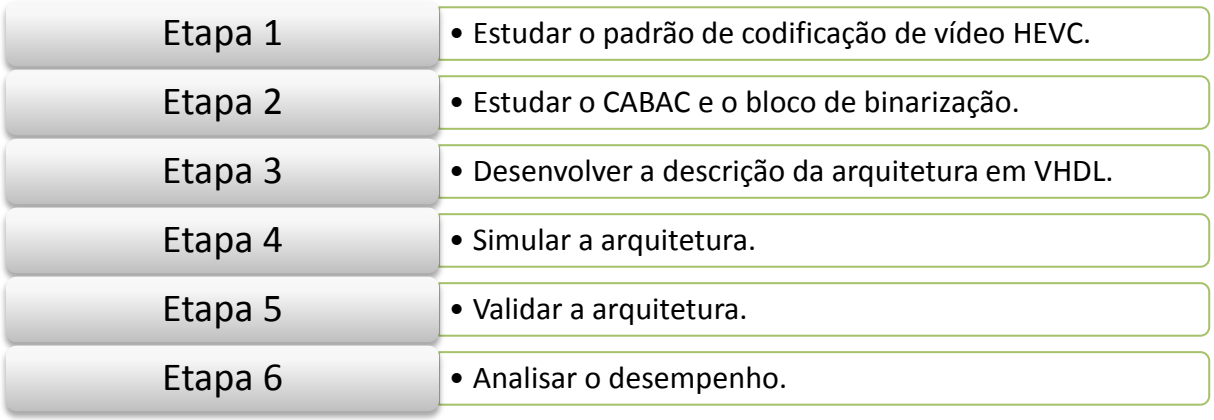
Dos quatro trabalhos correlatos apresentados, três foram desenvolvidos para o padrão atual, o HEVC. O trabalho do Martins (2011), apesar de ser do padrão H.264/AVC, possui uma descrição bem detalhada do funcionamento dos blocos de binarização e da forma como ele foi desenvolvido, sendo muito importante, principalmente para o entendimento dos métodos de binarização básicos. O trabalho do Zhou *et al.* (2015), apresenta uma visão geral da arquitetura do bloco de binarização, sendo utilizado como base para o desenvolvimento das arquiteturas deste trabalho. Por fim, os trabalhos de Peng *et al.* (2013) e Vizzotto *et al.* (2015), apresentam descrições das arquiteturas implementadas, além de explicações sobre o bloco de binarização, que também auxiliaram para o desenvolvimento deste trabalho.

4 METODOLOGIA

Para realizar este trabalho foi definido um conjunto de etapas que serviu de base para o desenvolvimento da pesquisa. As etapas foram divididas em seis partes, seguindo a ordem em que as mesmas deverão ser executadas. A Figura 15 apresenta estas etapas.

A primeira etapa consistiu em um estudo sobre o padrão HEVC, que é o padrão de codificação de vídeo do estado da arte, onde foram buscados artigos, teses e trabalhos que abordem sobre este assunto. Para realizar esta busca foi realizada uma pesquisa bibliográfica, em bibliotecas digitais como, IEEE, ACM, entre outras. Em seguida, foi estudado o bloco de binarização do CABAC, que é o bloco alvo deste trabalho. Desta forma, este estudo serviu de base para a implementação do bloco de binarização.

Figura 15 - Etapas adotadas para o desenvolvimento da pesquisa.



Fonte: Próprio autor, 2015.

O próximo passo foi desenvolver a descrição da arquitetura em VHDL do bloco de binarização do CABAC. Para realizar está etapa foi utilizada a ferramenta ModelSim, que é uma ferramenta da *Mentor Graphics* voltado para simulação de hardware digital, em especial de sistemas voltados para dispositivos FPGA e também projeto para ASIC. Logo após, foi desenvolvida a descrição de um *testbench*, para o bloco de binarização, que é um arquivo de descrição VHDL voltado para a emulação do comportamento real a qual a arquitetura estará submetida na prática. Com a descrição do *testbench* foi possível realizar a simulação da arquitetura.

A etapa 5 foi responsável pela validação da arquitetura. No processo de validação, o software de referência do padrão HEVC foi utilizado como modelo, onde os dados de entrada do processo de binarização no software foram usados também como entradas da arquitetura digital, utilizando-se sequências reais de vídeos para a geração dos estímulos. Assim, uma comparação foi realizada entre os dados de saída obtidos no software de referência com os dados de saída obtidos no *testbench*. Desta forma, foi realizada uma pesquisa quantitativa, onde traduzimos em números os resultados encontrados, sendo possível validar a nível funcional a arquitetura.

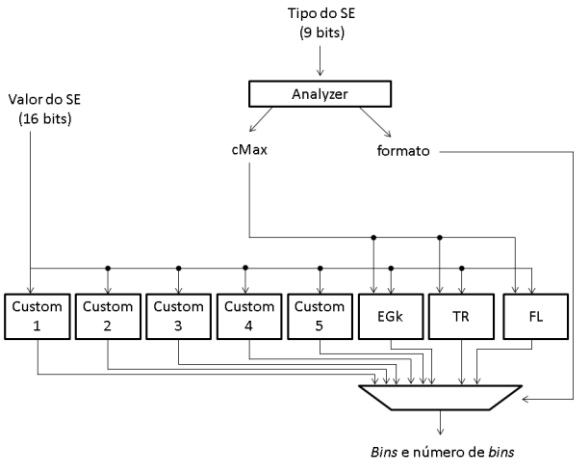
Sintetizar a arquitetura para ASIC utilizando ferramental apropriado foi a próxima etapa. Os dados da síntese foram utilizados para o cálculo do desempenho, comparando as arquiteturas implementadas neste trabalho com as demais encontradas na literatura. Algumas variáveis utilizadas para a análise comparativa foram: vazão da arquitetura, consumo de potência e área do projeto.

5 ARQUITETURA PROPOSTA PARA O BLOCO DE BINARIZAÇÃO DO CABAC

A arquitetura proposta em hardware para implementar o bloco de binarização do CABAC é apresentada neste capítulo. Em linhas gerais, o funcionamento do bloco de binarização foi apresentado ao longo do capítulo 3, no qual os métodos e a forma como eles funcionam foram discutidos. A arquitetura foi dividida em nove blocos: um bloco *analyzer* e outros oito blocos para os métodos de binarização. Desta forma, a seção 5.1 trata do bloco *analyzer*, a seção 5.2 apresenta os oito blocos dos métodos de binarização, a seção 5.3 apresenta a arquitetura em paralelo e, na seção 5.4 será visto a arquitetura voltada para redução de consumo de potência.

A função geral da arquitetura proposta para o bloco de binarização consiste em binarizar vários elementos sintáticos (isto é, passar os valores para representações usando apenas ‘0’s ou ‘1’s). O bloco *analyzer* gera os primeiros parâmetros relacionados com a forma como o elemento sintático deve ser binarizado, incluindo o cMax. Depois, oito blocos são utilizados para os processos de binarização individuais. O método unário é usado durante o processo do método Exp-Golomb, assim como, o método truncado unário é usado durante o processo do método truncado rice e, portanto, não existe necessidade de criar um bloco separado para estes métodos. A Figura 16 apresenta o diagrama de blocos da arquitetura que foi desenvolvida neste trabalho para um núcleo de binarização, que equivale a um FSE (*Full syntax engine* – Mecanismo sintático completo).

Figura 16 - Arquitetura do bloco de binarização.



Fonte: Próprio autor, 2016.

5.1 Bloco *Analyzer*

O bloco *analyzer* desenvolvido apresenta duas entradas. A entrada *se_type* de nove bits, que determina o tipo do elemento sintático, e a entrada *se_value* que é o valor do elemento sintático, sendo esta uma entrada de 16 bits. O bloco apresenta uma saída de quatro bits do formato do elemento sintático e uma saída de quatro bits do valor de cMax. Além disso, contém mais quatro saídas de 16 bits que são valores de elementos sintáticos que precisam ser usados no processo de binarização e, portanto, são registrados dentro desse bloco. A Figura 17 apresenta um diagrama do bloco *analyzer*.

Figura 17 - Diagrama do bloco *analyzer*.

Fonte: Próprio autor, 2016.

Para determinar qual o tipo de elemento sintático, foi criado um código de nove bits onde cada valor representa um elemento sintático. São usados nove bits devido a uma possível extensão para os elementos sintáticos do padrão H.264 e extensões do HEVC. Por exemplo, o elemento sintático *end_of_slice_segment_flag* é representado pelo número “000000000”. No total existem 56 elementos sintáticos para o HEVC, onde cada um é representado por um valor em binário. As informações dos elementos sintáticos, como o valor do cMax, o seu formato e a quantidade de SEs, foram retiradas da norma ITU-T, 2015. O anexo A apresenta todos os elementos sintáticos com a sua representação em nove bits, o valor de cMax e o formato.

Cada tipo de binarização foi representado por um número em binário de quatro bits. A Tabela 7 apresenta todos os tipos de binarização com a sua

representação em binário gerada pelo bloco *analyser* (isto é, a saída formato). Analisando o anexo A, é possível perceber que os tipos de binarização mais utilizados são o FL e o TR, em termos teóricos. O método TR se comporta como um TU quando a variável cRiceParam é igual a zero, sendo assim, embora o método TU não apareça como sendo utilizado em algum elemento sintático, ele é bastante utilizado no método TR.

Tabela 7 - Tipos de binarização e sua representação em binário.

Formato	Representação em Binário (4 bits)
Tamanho Fixo (FL)	0001
Exp-Golomb (EGk)	0011
Truncado Rice (TR)	0100
Custom 1	0101
Custom 2	0110
Custom 3	0111
Custom 4	1001
Custom 5	1010

Fonte: Próprio autor, 2016.

O cMax é representado por um número em binário de quatro bits, onde o valor pode não representar o valor final de cMax propriamente dito. Isto acontece porque nem todos os cMax possuem valor fixo. Alguns deles dependem de outros elementos sintáticos ou de variáveis externas. Para os cMax que não possuem valor fixo, pré cálculos foram realizados para poder determinar o valor de cMax no pior caso e assim realizar o processo de binarização em cima desses valores. Estes cálculos foram realizados no bloco de binarização correspondente a cada cMax. As saídas *se_value_a* e *se_value_b* registram dois elementos sintáticos que são usados no cálculo do cMax (como visto na Figura 17). Analisando o anexo A, é possível perceber que o cMax mais comum é o “0001” e que alguns cMax são utilizados apenas por um elemento sintático. A Tabela 8 apresenta todos os cMax com a sua representação em binário, onde “<<” e “>>” representam deslocamentos à esquerda e à direita, respectivamente.

Tabela 8 - cMax e sua representação em binário.

cMax	Representação em Binário (4 bits)
1	0001
2	0010
(1 << (Min (bitDepth,10)-5)) -1	0011
31	0100
3	0101
MaxNumMergeCand -1	0110
num_ref_idx_10_active_minus1	0111
num_ref_idx_11_active_minus1	1000
chroma_qp_offset_list_len_minus1	1001
4	1010
(log2TrafoSize << 1) -1	1011
(1 << ((last_sig_coeff_x_prefix >>1)-1)-1)	1100
(1 << ((last_sig_coeff_y_prefix >>1)-1)-1)	1101

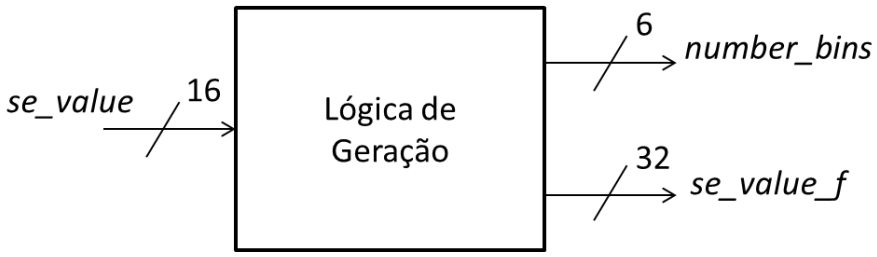
Fonte: Próprio autor, 2016.

As saídas *se_value_c* e *se_value_d* registram dois elementos sintáticos que são utilizados para o método de binarização custom 5 (Figura 17). Sendo assim, o bloco *analyzer* gera todos os parâmetros dos elementos sintáticos para ser possível realizar a binarização nos próximos blocos.

5.2 Blocos de Binarização

No total foram desenvolvidos oito blocos de binarização, onde cada bloco é responsável por um método. Todos os blocos possuem pelo menos uma entrada e duas saídas. A entrada *se_value* de 16 bits, que é o valor do elemento sintático; uma saída chamada *se_value_f* de 32 bits, que contém o elemento sintático binarizado; e uma saída *number_bins,* que determina a quantidade de *bins* que deve ser considerado. A Figura 18 apresenta o diagrama geral dos blocos de binarização.

Figura 18 - Diagrama dos blocos de binarização.



Fonte: Próprio autor, 2016.

Para melhor entendimento de como funciona cada bloco, os tipos de binarização foram agrupados da forma que se segue: a seção 5.2.1 apresenta os blocos FL, TR e EGk, a seção 5.2.2 apresentará os blocos custom 1, custom 2 e custom 3 e, a seção 5.2.3 explica os blocos custom 4 e custom 5.

5.2.1 Blocos de binarização FL, TR e EGk

Os formatos FL, TR e EGk são considerados métodos básicos de binarização. Para os métodos FL e TR foram adicionadas outras entradas (em relação as que foram apresentadas na Figura 18), para o cálculo do cMax.

O método FL define que o tamanho máximo do *binstring* é calculado a partir da equação (1). A Tabela 9 apresenta todos os valores de cMax possível e, a partir destes valores, foi realizado o cálculo para determinar o tamanho do *binstring*.

Tabela 9 - Cálculo do cMax.

cMax	**Tamanho máximo de *Binstring***
1	1
3	2
31	5
(1<<((*last_sig_coeff_x_prefix* >>1)-1)-1)	Depende do elemento sintático
(1<<((*last_sig_coeff_y_prefix* >>1)-1)-1)	Depende do elemento sintático

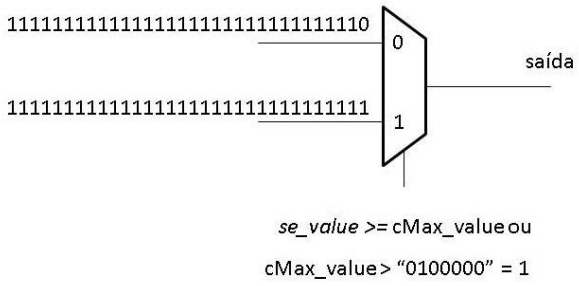
Fonte: Próprio autor, 2016.

Para o cMax que depende de outros elementos sintáticos, foi considerado o pior caso (o maior valor que o SE poderia ter) e o cálculo foi realizado para todos os valores. Como o método FL não recodifica o valor do SE, o *se_value_f* será, neste

caso, igual ao *se_value* e o *number_bins* vai ser igual ao cálculo para determinar o tamanho máximo de *binstring*.

O método TR foi desenvolvido de forma semelhante ao FL. Sabendo o código do cMax (código de quatro bits apresentado na Tabela 8), foi criado um sinal cMax_value que continha o valor de cada cMax. Para os cMax 2 e 4, a variável cMax_value recebe o próprio valor do cMax. Para os demais cMax que dependiam de variáveis externas, foi realizado um pré cálculo considerando o pior caso (maior valor que estas variáveis poderiam ter). No fim, foi realizada a binarização para todos os casos. Segundo a norma, o valor de cRiceParam é igual a zero para todos elementos sintáticos não-customizados. Sendo assim, o *se_value_f* poderá ter duas saídas que são apresentadas na Figura 19.

Figura 19 – Saídas do Método Truncado Rice.

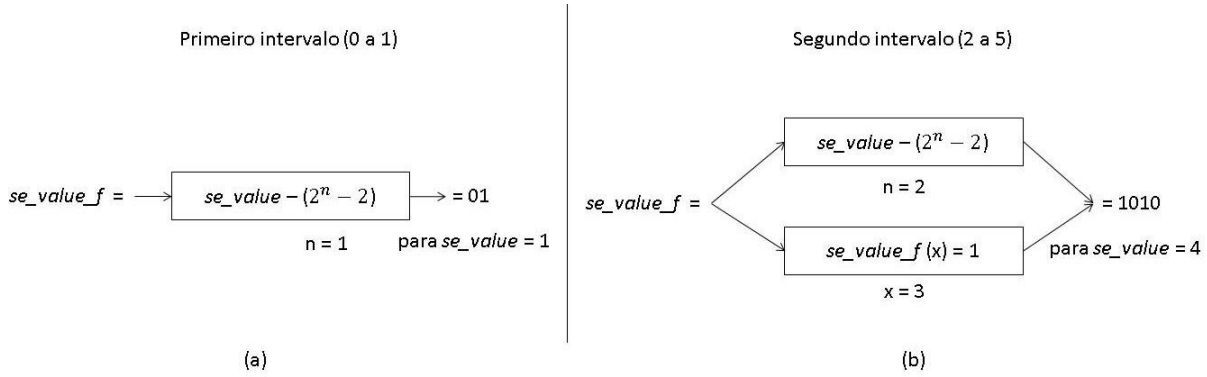


Fonte: Próprio autor, 2016.

Se o *se_value* for maior ou igual ao cMax_value ou se o cMax_value for maior que 32, a saída vai conter apenas ‘1’s, caso contrário a saída vai ser uma sequência de ‘1’s com um zero no final. O que determina o número de *bins* a ser considerado nessas saídas é a variável *number_bins* que vai de zero até o valor do cMax_value.

O método EGk não depende da variável cMax, ao invés disso, a única variável que deve ser considerada é o ‘k’, que determina o tamanho inicial de bits do sufixo. Tirando os métodos customs, existe apenas um elemento sintático que utiliza o método EGk. Este elemento sintático já determina que ‘k’=1 e, portanto, o processo de binarização foi feito considerando que ‘k’=1 e o limite máximo dele vai ser de 32 bits, que é o tamanho máximo da saída. A Tabela 3 apresentou como este método funciona quando ‘k’=1. Para a implementação deste método, foram realizados pré cálculos considerando intervalos que se comportassem da mesma forma. A Figura 20 apresenta estes cálculos para os dois primeiros intervalos.

Figura 20 - Cálculos do Método EGk para os dois primeiros intervalos.



Fonte: Próprio autor, 2016.

Os intervalos foram definidos pelo tamanho de bits do prefixo. Para saber qual valor inicial do intervalo, deve-se utilizar a equação (5), onde a variável ‘n’ começa em 1, e vai até o valor limite de entrada do SE (considerando 16 bits):

$$\text{início do intervalo} = (2^n - 2) \quad (5)$$

O primeiro intervalo, que vai de 0 a 1, não possui bits no prefixo e, portanto, a saída vai ser igual a entrada, pois o cálculo entre parênteses neste primeiro intervalo é igual a zero e, para este primeiro intervalo, nenhum cálculo a mais precisa ser efetuado. No entanto, do segundo intervalo em diante, o prefixo vai estar presente e, assim, outros cálculos devem ser considerados. O que vai mudar deste intervalo para o primeiro, é que a variável ‘n’ vai ser incrementada (onde, ‘n’ é a mesma variável citada em (5), sendo ela interna ao método, que define em qual intervalo de valores o atual elemento sintático está). Além disso, o que muda do primeiro intervalo para o segundo, é que o índice de valor 3 da saída vai receber o valor 1 para quaisquer valores de elementos sintáticos que estejam contidos dentro desse intervalo (isto é, elementos sintáticos com valores de 2 a 5). Desta forma, a posição de índice 3 da saída vai receber o valor do prefixo para este intervalo. No exemplo da Figura 20(b), o elemento sintático tem valor 4 (ou seja, está contido no intervalo de 2 a 5), a saída vai receber o elemento sintático menos dois (que é o resultado encontrado ao resolver o cálculo dentro dos parênteses na Figura 20(b)). Além disso, vai ser adicionado na posição de índice 3 da saída o valor 1 e, assim, obtemos a saída 1010 (isto é, o valor 4 do elemento menos 2^2 - 2 iria gerar, em

binário, o valor 0010 e, ao adicionar o valor ‘1’ no vetor de saída com índice 3, obtemos 1010).

Para os próximos intervalos, o processo de binarização vai ser o mesmo do segundo, mudando apenas os valores do ‘n’ e do ‘x’. O ‘n’ vai ser incrementado mais uma vez para cada intervalo, conforme já mencionado. Enquanto que o ‘x’ para o próximo intervalo vai começar uma posição acima do que no intervalo anterior, e vai ser adicionada uma posição a mais. Ou seja, no intervalo de 2 a 5, temos x = 3; para o intervalo de 6 a 13, o x vai começar uma posição acima (x = 4) e, além disso, o próximo índice no vetor de saída (x = 5) também terá o valor ‘1’. Os próximos intervalos funcionam da mesma forma, começando uma posição acima do que no intervalo anterior e adicionando uma posição a mais. O *number_bins* possui valor fixo para cada intervalo, uma vez que os elementos sintáticos de um mesmo intervalo possuem tamanho igual. Assim, para calcular o *number_bins* de cada intervalo, é utilizada a equação (6):

$$number\_bins = \mathrm{n} * 2 \tag{6}$$

A Tabela 10 apresenta os cálculos do método EGk para os primeiros intervalos.

Tabela 10 - Cálculo realizado para o EGk.

Intervalo	**n**	***se_value_f***	***se_value_f* (x)**	***number_bins***
0 -1	1	*se_value*	-	000010
2 - 5	2	*se_value* - 2	*se_value_f*(3) = ‘1’	000100
6 - 13	3	*se_value* - 6	*se_value_f*(4) = ‘1’ *se_value_f*(5) = ‘1’	000110
14 - 29	4	*se_value* - 14	*se_value_f*(5) = ‘1’ *se_value_f*(6) = ‘1’ *se_value_f*(7) = ‘1’	001000
...		...	...	...

Fonte: Próprio autor, 2016.

5.2.2 Blocos de binarização Custom 1, Custom 2 e Custom 3

Os customs 1, 2 e 3 são os tipos de binarização considerados personalizados. Eles foram desenvolvidos para um elemento sintático específico e, portanto, o processo de binarização é exclusivo para os valores destes elementos sintáticos. O custom 2 foi desenvolvido para o elemento sintático *intra_chroma_pred_mode* e a forma de binarização deste elemento sintático já foi apresentada na Tabela 5.

O custom 1 foi desenvolvido para o elemento sintático *part_mode*. Ele possui quatro entradas adicionais além da entrada vista na Figura 18, que são: *CuPredMode*, *log2CbSize*, *MinCbLog2SizeY* e *amp_enabled_flag*. Estas variáveis externas auxiliam no processo de binarização do método custom 1. A Tabela 11 apresenta o processo de binarização para o elemento sintático *part_mode* para os modos intra e inter.

Tabela 11 - Processo de binarização para o *part_mode*.

CuPredMode [xCb][yCb]	**part_mode**	***Binstring***			
		log2CbSize > MinCbLog2SizeY		**log2CbSize == MinCbLog2SizeY**	
		AMP disabled	**AMP enabled**	**log2CbSize == 3**	**log2CbSize > 3**
Modo intra	0	-	-	1	1
	1	-	-	0	0
Modo inter	0	1	1	1	1
	1	01	011	01	01
	2	00	001	00	001
	3	-	-	-	000
	4	-	0100	-	-
	5	-	0101	-	-
	6	-	0000	-	-
	7	-	0001	-	-

Fonte: ITU-T,2015.

O processo de binarização personalizado determina a saída de todos os valores possíveis de SE e, para qualquer valor maior do que os previstos na Tabela

11 (no caso 7), a saída será igual a zero, pois não vai existir processo de binarização para estes valores (segundo a norma, não são previstos valores maiores que esse, na prática).

O custom 3 é responsável pelo processo de binarização do elemento sintático *inter_pred_idc*. Além da entrada apresentada na Figura 18 este elemento sintático possui mais três entradas, que são: *partitionsize, size* e *height*. Estas variáveis auxiliam no processo de binarização. Para implementar este método foi utilizado como base o software de referência (HM, 2016). A Tabela 12 apresenta o processo de binarização deste elemento sintático. Para valores maiores do que o da Tabela 12 (no caso 3), a saída será igual a zero.

Tabela 12 - Processo de binarização para o inter_pred_idc.

Valor do *inter_pred_idc*	***Binstring***	
	(*partitionsize* == *size*) ou (*height* != 8)	**(*partitionsize* != *size*) ou (*height* == 8)**
0	00	0
1	01	1
2	1	-

Fonte: HM, 2016.

5.2.3 Blocos de binarização Custom 4 e Custom 5

Os customs 4 e 5 são métodos de binarização compostos pela combinação de dois métodos através de um esquema de prefixo-sufixo. O custom 4 é usado para binarizar o elemento sintático *cu_qp_delta_abs*. Ele é formado pela concatenação do método TR para o prefixo e o método EGk para o sufixo. Os parâmetros de entrada deste método de binarização já foram definidos onde para o TR o cMax = 5 e o cRiceParam = 0; e para o EGk o ‘k’ = 0. Além disso, para a cadeia de *bin* sufixo estar presente, o valor do prefixVal tem que ser maior do que quatro. O prefixVal é definido a partir da equação (3). O processo de binarização deste método foi apresentado na Tabela 4.

Para implementar este bloco, foi analisado os parâmetros de entrada do método. Quando o elemento sintático tem valor 0 até 4, o sufixo não está presente e, portanto, até o valor igual a 4 do elemento sintático, será realizado apenas o

processo de binarização do TR. Como o TR tem cRiceParam = 0, ele se comporta como um TU, (conforme já mencionado anteriormente). Quando o prefixVal for maior que quatro, o sufixo estará presente. Como o tamanho máximo do prefixo é 5, quando o sufixo fizer parte do processo de binarização, o prefixo passará a ter o mesmo valor, que é: “11111”.

O processo de binarização do método EGk quando ‘k’ = 0 pôde ser observado na Tabela 3. Para a implementação deste método para o sufixo, foram realizados pré cálculos considerando intervalos que se comportassem da mesma forma. Os intervalos foram definidos pelo tamanho de bits do prefixo do método EGk. Para saber qual o valor inicial do intervalo, deve-se usar a equação (7):

$$\text{início do intervalo} = (4 + 2^n) \tag{7}$$

Quando o elemento sintático for igual a 5, o sufixo vai estar presente, mas o prefixo e o sufixo do EGk não vai existir, tendo apenas o ‘1’ que separa o prefixo do sufixo. Assim, este valor não vai estar em um intervalo, ao invés disso, o *se_value_f* vai receber a concatenação do prefixo do custom 4 que é “11111” com o sufixo que é “1”, ficando igual a “111111”. O processo de binarização dos elementos sintáticos de 0 a 5 são apresentados na Tabela 13.

Tabela 13 - Processo de binarização dos elementos sintáticos de 0 a 5.

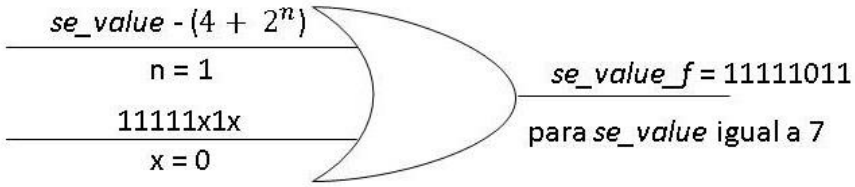
se_value	**Prefixo**	**Sufixo**	***se_value_f***
0	0	-	0
1	10	-	10
2	110	-	110
3	1110	-	1110
4	11110	-	11110
5	11111	1	111111

Fonte: Próprio autor, 2016.

A partir do elemento sintático de valor 6, os intervalos foram definidos e, algumas considerações têm que ser levadas em conta para os cálculos de binarização pois, agora o prefixo e o sufixo do método EGk vão estar presentes.

Para essa explicação, a Figura 21 apresenta os cálculos para o intervalo de 6 a 7. Onde, ‘x’ representa o valor do prefixo e do sufixo do método EGk.

Figura 21 - Cálculos realizados para o Custom 4 do intervalo de 6 a 7.



Fonte: Próprio autor, 2016.

Para o primeiro intervalo, que vai de 6 a 7, a saída vai receber o elemento sintático menos 6 (que é o valor encontrado ao realizar o cálculo que está entre parênteses na Figura 21), sendo feito um OR *bit-wise* com o valor “11111010” (isto é, a OR é feita considerando o valor em binário “11111x1x”, onde os ‘x’s vão conter valor ‘0’, para o intervalo de 6 a 7). Por exemplo, se o elemento sintático for igual a 7, a saída vai receber “11111011”, conforme consta na Figura 21. Para cada novo intervalo, os cálculos vão ser os mesmos apresentados na Figura 21. Porém, o ‘n’ vai ser incrementado a cada intervalo e, o ‘x’ vai passar a ter mais um bit de ‘0’ a cada intervalo, ou seja, as posições em ‘x’ vão ter ‘0’s de acordo com cada intervalo calculado. A Tabela 14 apresenta os primeiros intervalos e seus respectivos cálculos.

Tabela 14 - Processo de binarização do Custom 4.

Intervalo	**n**	***se_value* – $(4 + 2^n)$**	**x**	**11111x1x**
6 - 7	1	*se_value* - (6)	0	11111010
8 - 11	2	*se_value* - (8)	00	1111100100
12 - 19	3	*se_value* - (12)	000	111110001000
20 - 35	4	*se_value* - (20)	0000	11111000010000
...	...	...	...	...

Fonte: Próprio autor, 2016.

O custom 5 é o método de binarização do elemento sintático *coeff_abs_level_remaining*. Ele é formado pela concatenação do método TR para o

prefixo e o método EGk para o sufixo. O que difere este método do custom 4 é que os parâmetros de entrada não são definidos (isto é, eles variam ao longo do processamento). De acordo com a norma, a variável cRiceParam do método TR pode variar de 0 a 4. E, a partir desta variável será possível determinar os valores do 'k'. Para determinar o 'k', é utilizada a equação (8):

$$\mathrm{k} = \mathrm{cRiceParam} + 1 \tag{8}$$

O sufixo só vai estar presente neste caso, quando o prefixo possuir tamanho igual a quatro e todos os seus bits tiverem valor igual a 1. Esta condição irá acontecer apenas quando o cRiceParam for igual a 0 e, portanto, o método EGk só vai estar presente neste caso. Para implementar este método, foi considerado o valor do cRiceParam e, a partir dele o processo de binarização foi feito considerando todos os casos possíveis. A Tabela 15 apresenta os casos possíveis.

Tabela 15 - Casos possíveis a partir da variável cRiceParam.

cRiceParam	k	Métodos presentes
0	1	TR + EGk
1	2	TR
2	3	TR
3	4	TR
4	5	TR

Fonte: Próprio autor, 2016.

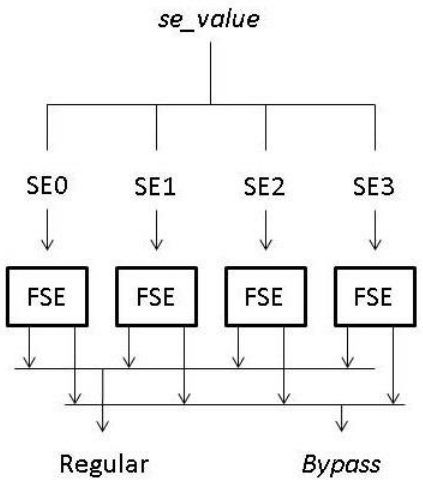
Assim sendo, a mesma lógica apresentada para os demais tipos de binarização, foi replicada para ser utilizada para esse elemento sintático. Onde, para o cRiceParam = 0, foi efetuada a concatenação dos métodos TR e EGk e, para os demais valores de cRiceParam, a binarização foi feita com o método TR.

5.3 Arquitetura em Paralelo

Em busca de melhores resultados em relação a desempenho, a arquitetura do bloco de binarização vai processar vários elementos sintáticos em paralelo. Para isto, foi utilizado como base o trabalho do Zhou, *et al.* (2015). Neste trabalho,

existiam quatro núcleos, duas FSEs (*Full syntax engine* – Mecanismo sintático completo) que contém todos os tipos de binarização e, duas SSEs (*Simplified syntax engine* – Mecanismo sintático simplificado) que contém apenas alguns tipos, que são: os custom, o método tamanho fixo e o método truncado unário. Isto foi feito devido à probabilidade de uso dos tipos de binarização. Além disso, o SSE contém mais uma restrição. Ele só processa elementos sintáticos que não ultrapassem os dois bits. Assim, no máximo quatro elementos sintáticos podem ser inseridos por ciclo de clock. Os dois primeiros elementos sintáticos serão sempre distribuídos aos FSEs e, os SSEs podem processar até mais dois elementos sintáticos se os seus valores e formatos atenderem as restrições do SSE.

Figura 22 - Arquitetura em Paralelo.



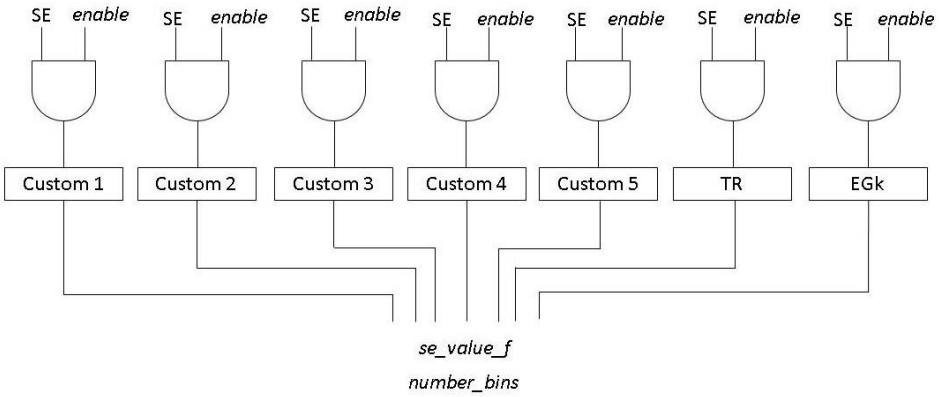
Fonte: Próprio autor, 2016.

A Figura 22 apresenta a arquitetura em paralelo desenvolvida para este trabalho. Como foi feito no trabalho do Zhou, *et al.* (2015), a arquitetura desenvolvida em paralelo possui quatro núcleos. Porém, os quatro núcleos são completos e não possuem nenhuma restrição. Esta decisão foi tomada, pois se verificou que a ausência dos métodos de binarização na SSE pouco impactavam em termos de consumo de potência, mas poderiam ter um impacto maior considerando o desempenho final de *throughput*. Assim, quaisquer quatro elementos sintáticos podem ser inseridos por ciclo de clock na arquitetura em paralelo do binarizador.

5.4 Arquitetura voltada para Redução de Consumo de Potência

Após realizar a implementação da arquitetura, a proposta era conseguir reduzir o consumo de potência. Além disso, era esperado manter o desempenho com o mínimo possível de degradação em relação à arquitetura original. Na arquitetura apresentada na Figura 16, para cada processo de binarização o elemento sintático era binarizado para todos os métodos (conforme a proposta de Zhou *et al.* (2015)). Sendo que apenas um destes métodos de binarização correspondia ao elemento sintático que de fato deveria ser binarizado naquele instante (ou seja, todos os blocos de binarização dentro dos quatro núcleos estariam ativos a todo instante, mesmo que o tipo de binarização a ser usado fosse apenas um dentro de cada um dos núcleos). Para melhorar a arquitetura implementada, o formato do elemento sintático passou a ser informado antes do processo de binarização, desligando as entradas dos blocos que não serão utilizados (e assim evitando chaveamento nos blocos), ativando apenas um dos blocos de binarização em cada núcleo. Esse processo é feito através de inserções de ANDs nas entradas de dados do bloco de binarização junto do sinal de *enable* gerado pelo *Analyser*, essa técnica conhecida como *Operand Isolation*, é utilizada em lógicas puramente combinacionais para redução de consumo dinâmico de potência (CORREALE, 1995). A Figura 23 apresenta como funciona a técnica *Operand Isolation*.

Figura 23 - Técnica de redução de consumo dinâmico de potência.

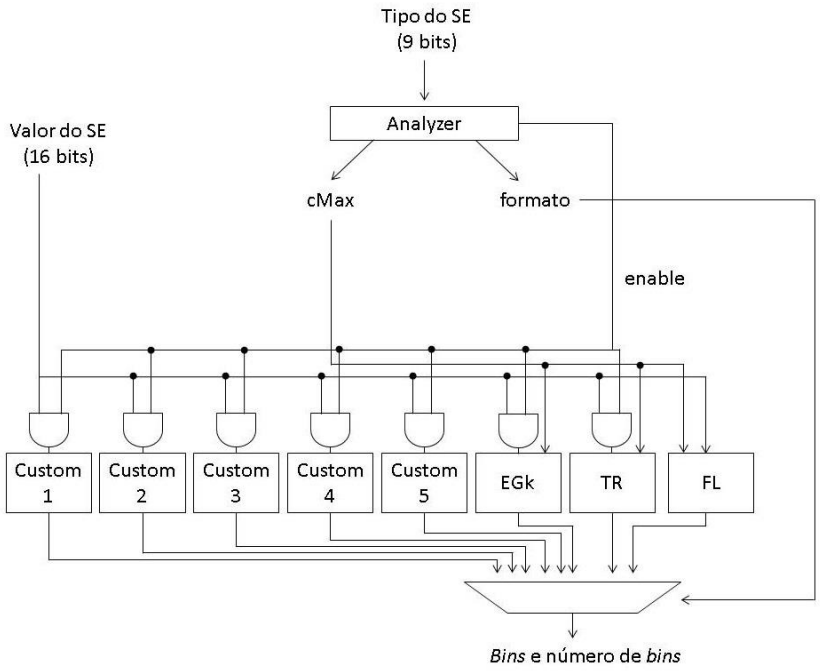


Fonte: Próprio autor, 2016.

O sinal de *enable* foi inserido para todos os métodos de binarização, exceto para o método Tamanho Fixo. Isto porque a lógica de desligamento não seria eficiente para esse bloco, dada a simplicidade do método de binarização citado (isto

é, só iria gerar mais consumo, ao invés de reduzir). A Figura 24 apresenta a arquitetura final deste trabalho para um dos núcleos completos de binarização.

Figura 24 - Arquitetura final de um núcleo de binarização.

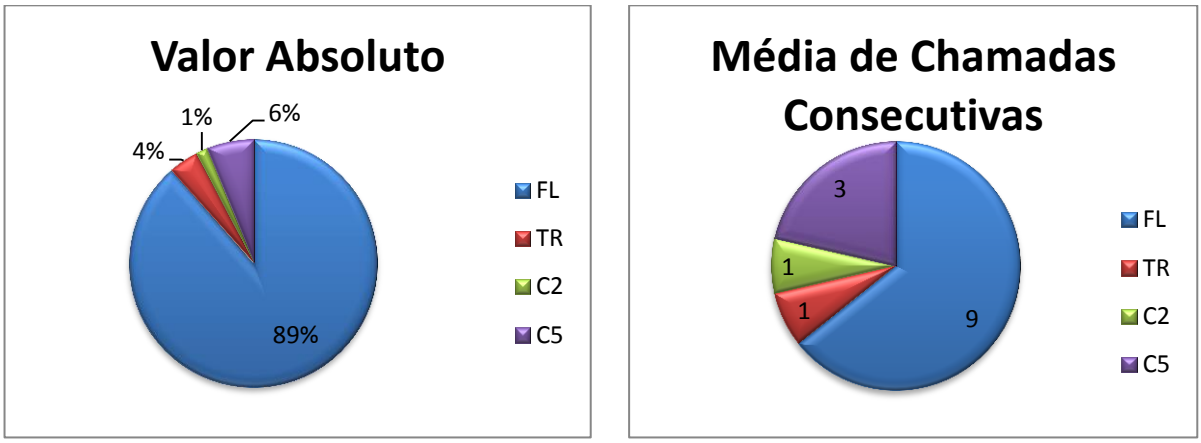


Fonte: Próprio autor, 2016.

Para justificar o uso da técnica *Operand Isolation,* foram rodadas sequências de teste de vídeo para PeopleOnStreet com resolução WQXGA (2560x1600), e a BasketballDrive, com resolução Full HD (1920x1080). Os testes foram gerados para duas configurações diferentes, o *Low Delay* e o *Random Access* e, para dois parâmetros de quantização (*Quantization Parameter* - QP) com valores de 22 e 37. As sequências teste de vídeo e as configurações escolhidas, são recomendadas pelo padrão HEVC. Existe uma série de sequências de diferentes resoluções, recomendas pelo padrão, bem como 4 valores de QPs. Sendo escolhido duas sequências entre as disponíveis e os dois valores extremos do QP, o 22 e 37.

A partir dos testes realizados, foram criados gráficos contendo a quantidade de vezes e o valor médio de vezes seguidas que um determinado tipo de binarização é chamado. A Figura 25 apresenta estes gráficos para a sequência de teste de vídeo PeopleOnStreet.

Figura 25 - Valor absoluto e média de chamadas para PeopleOnStreet.

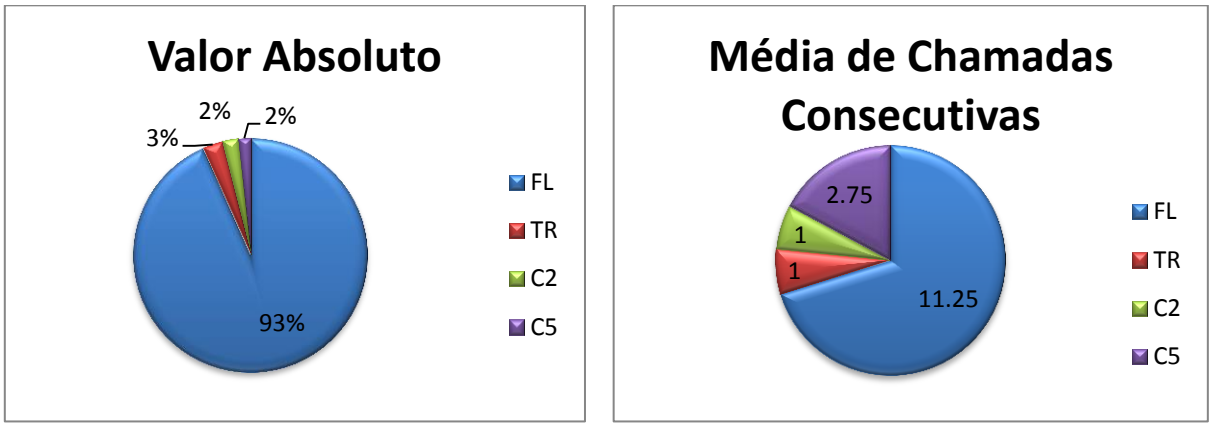


Fonte: Próprio autor, 2016.

Os gráficos da Figura 25 foram gerados a partir da sequência teste de vídeo PeopleOnStreet executados a partir do software de referência do padrão. Os testes foram gerados para duas configurações e para dois QPs diferentes (totalizando quatro sequências). Os resultados apresentados no gráfico são uma média de todas as análises. É possível perceber que apenas quatro tipos de binarização foram chamados, considerando que as sequências não foram rodadas de forma completa, devido à dificuldade de se gerar dados desse tipo para uma sequência inteira de vídeo (isto é, escassez de recursos de processamento para a análise). Apesar disso, é possível perceber que o método Tamanho Fixo é o mais comum deles, sendo o método mais chamado entre os apresentados no gráfico e, o que possui maior média de vezes seguida chamada. Também é possível perceber, que embora o método Custom 5 seja utilizado para um elemento sintático específico, ele é chamado muitas vezes e possui uma média de chamadas de até 3 vezes seguidas (o que era esperado, devido ao método ser utilizado por resíduos de transformada, conforme já foi mencionado em Stankowski (2014)).

Os mesmos gráficos foram gerados para a sequência teste de vídeo BasketballDrive. Para esta sequência de vídeo, foram utilizados os mesmos perfis e valor de QP da primeira sequência de vídeo, fazendo uma média de todas as análises. A Figura 26 apresenta estes gráficos para o valor absoluto e a média de chamadas.

Figura 26 - Valor absoluto e média de chamadas para BasketballDrive.



Fonte: Próprio autor, 2016.

Ao compararmos os gráficos da Figura 26 com os da Figura 25, é possível perceber que os valores ficaram bem próximos. A partir dos gráficos apresentados, é analisado que alguns métodos de binarização são chamados várias vezes seguidas, o que justifica o uso da técnica de *Operand Isolation* utilizando ANDs de forma eficiente para a redução de consumo, pois assim teremos as entradas travadas em zero por vários ciclos seguidos nos tipos de binarização que não estão sendo utilizados durante uma rajada de chamadas de um dos tipos mais comuns, evitando consumo de potência desnecessários nesses instantes.

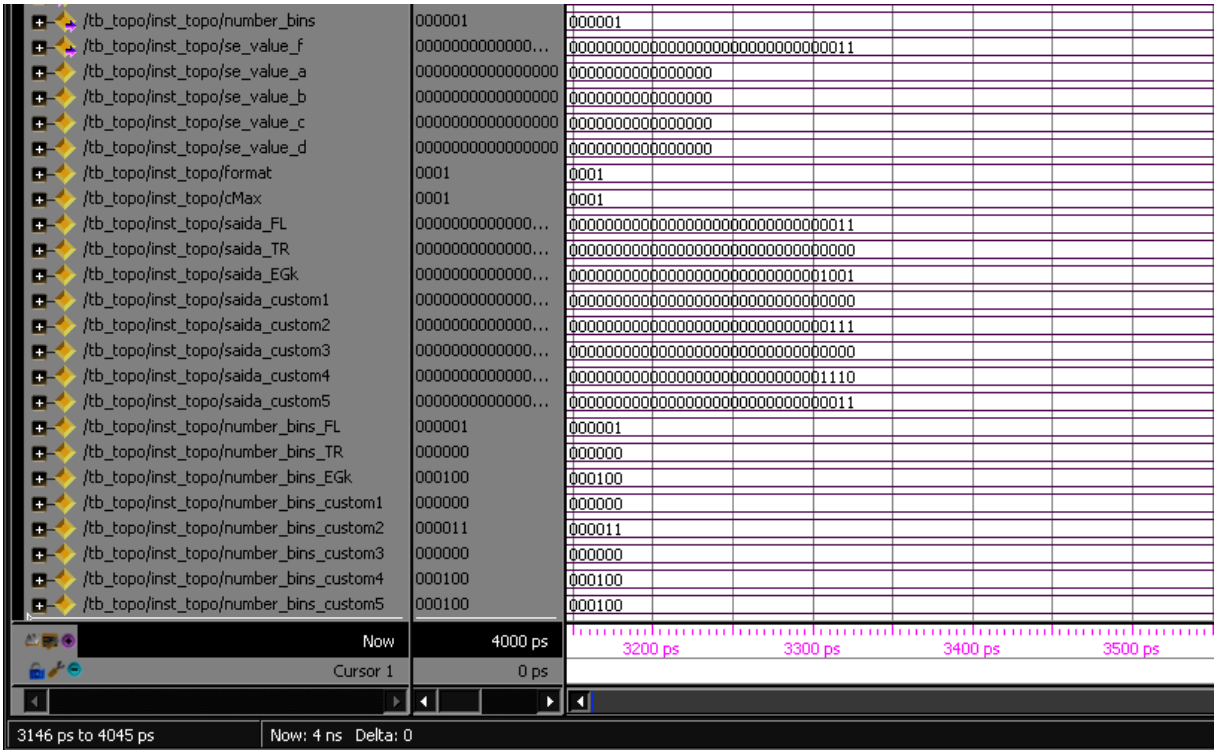
6 RESULTADOS E DISCUSSÕES

Este capítulo aborda os resultados da simulação, validação e síntese da arquitetura. Além disso, é feita a análise de desempenho e de consumo de potência em relação a arquitetura desenvolvida com e sem as técnicas de baixo consumo. Para fazer a validação da arquitetura em VHDL, foi utilizada a ferramenta ModelSim da Mentor Graphics. Já para a síntese, foi utilizada a ferramenta RTL Compiler da Cadence para síntese lógica ASIC, utilizando a biblioteca de células de 65nm da ST. A seção 6.1 apresenta a simulação e validação da arquitetura e, a seção 6.2 apresenta os resultados de síntese da arquitetura.

6.1 Simulação e Validação da Arquitetura

Para realizar a simulação da arquitetura implementada foi feita a descrição de um *testbench* do bloco de binarização, que é um arquivo de descrição VHDL voltado para a emulação do comportamento real ao qual a arquitetura estará submetida na prática. A Figura 27 apresenta uma das simulações da arquitetura.

Figura 27 - Simulação da arquitetura.

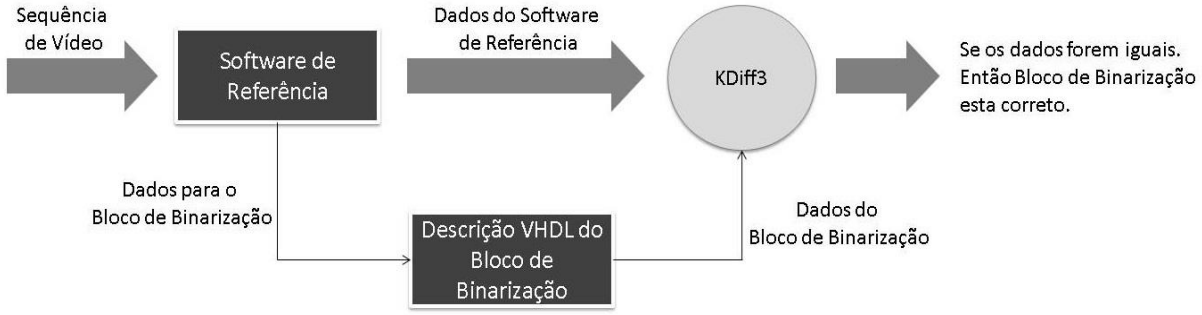


Fonte: Próprio autor, 2016.

A simulação ocorreu de duas formas diferentes. Em um primeiro momento foi feita a simulação de cada um dos tipos de binarização separado, podendo assim, testar cada método de forma individual. Logo após, foi feita a descrição do *testbench* contendo todos os métodos de binarização. A Figura 27 apresenta a simulação realizada a partir do *testbench*. O valor do formato para este exemplo é “0001” e, portanto, o *se_value_f* vai receber a saída do método tamanho fixo. A saída deste método é representada pela variável saída_FL e, o número de *bins* é representado pela variável number_bins_FL.

Depois de realizar a simulação da arquitetura, a próxima atividade consistiu em realizar a validação da arquitetura. Para isso, foi utilizado o software de referência do padrão HEVC como modelo, onde as entradas do software foram usadas como entradas da arquitetura digital, utilizando sequências de teste de vídeo reais já mencionadas anteriormente. A Figura 28 apresenta o ambiente de validação para o bloco de binarização.

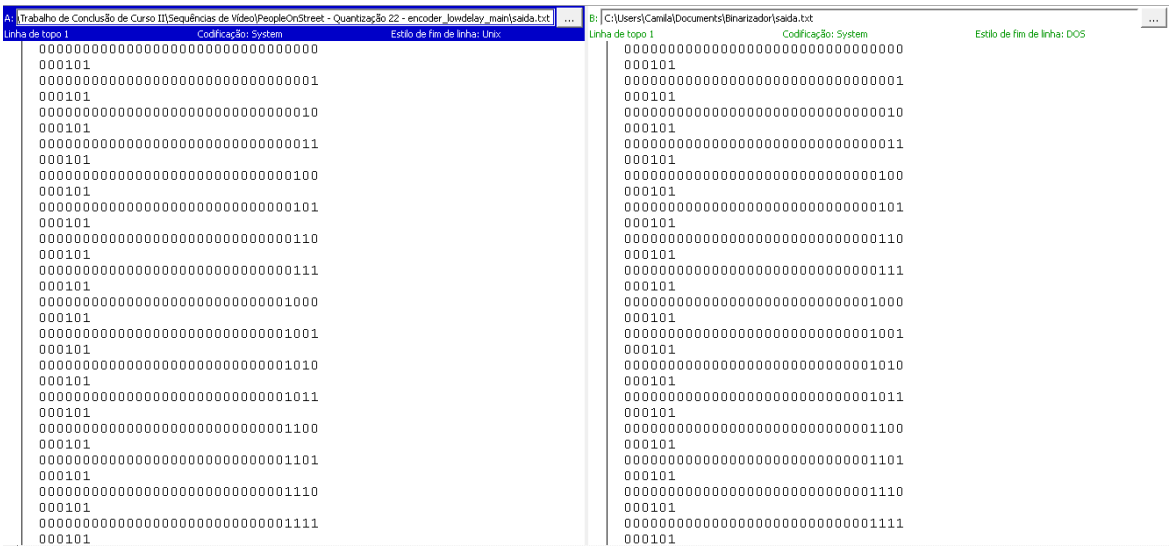
Figura 28 - Ambiente de validação para o Bloco de Binarização.



Fonte: Próprio autor, 2016.

Tendo os resultados obtidos no software de referência e os resultados obtidos no *testbench,* foi realizada uma comparação entre eles. Para realizar esta comparação, foi utilizado a ferramenta KDiff3 que compara o arquivo obtido a partir do software de referência com o arquivo gerado na simulação e, indica se existe diferença entre as linhas dos arquivos selecionados. A Figura 29 apresenta a comparação de dois arquivos .txt.

Figura 29 - Comparação de dois arquivos .txt.



Fonte: Próprio autor, 2016.

No lado esquerdo da Figura 29, está o arquivo que contém as saídas obtidas no software de referência e, no lado direito da figura, está o arquivo que contém as saídas obtidas da arquitetura implementada neste trabalho. As saídas foram organizadas para serem impressas da mesma forma nos dois arquivos. Primeiro sempre será impresso o valor do elemento sintático binarizado com tamanho igual a 32 bits. Depois, será gerado o número de *bins* com tamanho igual a 6 bits. O programa irá comparar cada linha dos dois arquivos buscando por diferenças entre eles. Com isso, se garante que a funcionalidade da arquitetura está correta, frente o software de referência do padrão.

6.2 Síntese da Arquitetura

A síntese da arquitetura foi feita com a ferramenta RTL Compiler da Cadence. Esta ferramenta tem como entrada o código RTL e como saída um *Netlist*, que descreve a conectividade entre os componentes do sistema. Além disso, segundo Marques (2015), esta ferramenta realiza a otimização simultânea de temporização e área, visando reduzir o consumo de energia e aumentar sua qualidade e eficiência. No processo da síntese lógica, o RTL Compiler utiliza uma biblioteca de células de uma determinada tecnologia, afim de mapear as funcionalidades do código RTL. Para realizar esta síntese, foi utilizada a biblioteca de célula de 65nm da ST. Os resultados da síntese ASIC para frequência e área, são apresentados na Tabela 16.

Tabela 16 - Resultados da síntese ASIC para frequência e área.

Arquitetura	Frequência	Área
Arquitetura Inicial	850 MHz	3.08 kgates
Arquitetura com *Operand Isolation*	858 MHz	3.18 kgates
Arquitetura em Paralelo	826 MHz	11.67 kgates
Arquitetura em Paralelo com *Operand Isolation*	834 MHz	11.85 kgates

Fonte: Próprio autor, 2016.

A frequência obtida na arquitetura com *Operand Isolation* foi de 858 MHz, ocorrendo uma melhora quando comparado com a frequência obtida na arquitetura inicial que foi de 850 MHz. Esta melhora pode ter ocorrido devido a biblioteca de células junto com a ferramenta de síntese conseguirem ser mais eficientes ao escolher uma célula que faria toda a função anterior junto do *Operand Isolation,* enquanto que na arquitetura inicial, provavelmente uma ou mais células menos eficientes foram utilizadas. A arquitetura em paralelo com *Operand Isolation* também apresentou melhorias em relação a arquitetura inicial em paralelo.

Em relação aos resultados da área, as duas arquiteturas com *Operand Isolation,* apresentaram um aumento de área em relação às arquiteturas iniciais. Este resultado é esperando, uma vez que, para aplicar a técnica de *Operand Isolation,* foi feita a inserção de uma série de lógicas AND.

Para realizar a simulação da *Netlist*, foram utilizadas duas sequências de vídeo: a PeopleOnStreet, com resolução WQXGA (2560x1600); e a BasketballDrive com resolução Full HD (1920x1080). Foram considerados dois QPs com valores de 22 e 37. Além disso, foram consideradas as configurações *Low Delay* e *Random Access*. A frequência escolhida para rodar os testes, foi de 500 MHz. Isto porque a arquitetura do Zhou *et al.* (2015) alcança 420 MHz e, assim, foi feita a análise considerando um valor extremamente pessimista, supondo o pior caso possível em relação ao consumo, pois na realidade para o CABAC como um todo, o gargalo não estará na binarização (isto é, usando uma frequência próxima a que o bloco crítico alcança, que é o BAE). Os resultados da análise do consumo de potência para a arquitetura inicial são apresentados na Tabela 17.

Tabela 17 - Resultados da Síntese para a Arquitetura Inicial.

Arquitetura	Vídeos	Perfil	QP	Consumo (mW) @500MHZ
Arquitetura Inicial	BasketballDrive	*Low Delay*	22	0.596
			37	0.640
		Random Access	22	0.518
			37	0.639
	PeopleOnStreet	*Low Delay*	22	0.524
			37	0.610
		Random Access	22	0.522
			37	0.606

Fonte: Próprio autor, 2016.

Analisando os resultados obtidos para a arquitetura inicial, é possível perceber que, para as duas sequências de vídeo, o melhor resultado encontrado foi com o perfil *Random Access* e o parâmetro de quantização 22. Além disso, o pior resultado encontrado para as duas sequências de vídeo foi com o perfil *Low Delay* e parâmetro de quantização 37. Após realizar a síntese da arquitetura inicial, foi feita a síntese da arquitetura com *Operand Isolation*. Os resultados podem ser vistos na Tabela 18.

Tabela 18 - Resultados da Síntese para a Arquitetura com *Operand Isolation*.

Arquitetura	Vídeos	Perfil	QP	Consumo (mW) @500MHz	Relação de Otimização
Arquitetura com *Operand Isolation*	BasketballDrive	*Low Delay*	22	0.486	17%
			37	0.454	41%
		Random Access	22	0.450	15%
			37	0.452	41%
	PeopleOnStreet	*Low Delay*	22	0.469	12%
			37	0.476	28%
		Random Access	22	0.468	12%
			37	0.475	28%

Fonte: Próprio autor, 2016.

Com a aplicação da técnica de baixo consumo, explicada na seção 5.4, foi possível obter uma melhora de até 41% em relação a arquitetura inicial. Ao analisar os resultados encontrados, percebemos que o QP com maior consumo de potência na arquitetura inicial, que era 37, foi o que obteve melhora de 41%. Este resultado era esperado, com base nas análises estatísticas realizadas no capítulo anterior, havendo uma redução na prática no consumo de potência.

Depois de realizar a síntese para a arquitetura inicial e arquitetura com *Operand Isolation,* foi realizada a síntese das arquiteturas em paralelo. A Tabela 19 apresenta os resultados da síntese para a arquitetura inicial em paralelo.

Tabela 19 - Resultados da Síntese para a Arquitetura Inicial em Paralelo.

Arquitetura	**Vídeos**	**Perfil**	**QP**	**Consumo (mW) @500MHz**
Arquitetura Inicial em Paralelo	BasketballDrive	*Low Delay*	22	2.46
			37	2.38
		Random Access	22	2.34
			37	2.37
	PeopleOnStreet	*Low Delay*	22	2.38
			37	2.51
		Random Access	22	2.35
			37	2.50

Fonte: Próprio autor, 2016.

Ao analisar os resultados da síntese da arquitetura inicial em paralelo, é possível perceber que o perfil *Low Delay* foi o que obteve maior consumo de potência para as duas sequências de vídeo. Além disso, ao compararmos estes resultados com o da Tabela 16, é possível perceber que houve um aumento de consumo de quase quatro vezes. Este resultado era esperado, pois na arquitetura em paralelo, temos quatro núcleos de binarização, que correspondem a quatro vezes a arquitetura inicial, sempre com todas as entradas sendo alimentadas pelos elementos sintáticos que estão sendo processados, da forma como expresso no trabalho de Zhou *et al.,* (2015) (ou seja, todos os blocos de binarização dentro dos quatro núcleos estão sempre ativos).

Logo após realizar a síntese para a arquitetura inicial em paralelo, foi realizada a síntese da arquitetura em paralelo com *Operand Isolation*. A Tabela 20 apresenta os resultados para esta arquitetura em paralelo.

Tabela 20 - Resultados da Síntese para a Arquitetura em Paralelo com *Operand Isolation*.

Arquitetura	**Vídeos**	**Perfil**	**QP**	**Consumo (mW) @500MHz**	**Relação de Otimização**
Arquitetura em Paralelo com *Operand Isolation*	BasketballDrive	*Low Delay*	22	1.97	25%
			37	1.74	37%
		Random Access	22	1.90	23%
			37	1.73	37%
	PeopleOnStreet	*Low Delay*	22	1.96	21%
			37	1.86	35%
		Random Access	22	1.94	21%
			37	1.88	33%

Fonte: Próprio autor, 2016.

A arquitetura em paralelo com *Operand Isolation*, apresentou uma melhora de até 37% em relação a arquitetura inicial em paralelo. Com um consumo de potência no melhor caso de 1.74 mW, para o perfil *Low Delay* e QP igual a 37. Esta redução no consumo de potência era esperada, devido novamente a análise teórica realizada no capítulo anterior. É alcançada também uma redução considerável no consumo de potência na arquitetura em paralelo, lembrando que agora temos quatro núcleos, que anteriormente estariam com todos os blocos internos ligados ao mesmo tempo.

6.3 Comparação com os Trabalhos Correlatos

Os trabalhos apresentados na seção 3.5, foram utilizados como referência para o desenvolvimento deste trabalho, conforme mencionado. A Tabela 21 apresenta uma comparação entre os trabalhos pesquisados na literatura, considerando apenas aqueles que contêm resultados para o bloco de binarização, e a arquitetura em paralelo com *Operand Isolation*.

Tabela 21 - Comparação entre os resultados dos Trabalhos Correlatos.

Autor	Tecnologia	Padrão	Frequência	Área	*Throughput*
Martins	FPGA	H.264/AVC	350 MHz	-	-
Liu *et al.*	ASIC 90 nm	H.264/AVC	250 MHz	19.3 kgates	1 a 2 SE/ciclo
Zhou *et al.*	ASIC 90 nm	HEVC	420 MHz*	-	4 SE/ciclo
Este trabalho	**ASIC 65 nm**	**HEVC**	**834 MHZ**	**11.85 kgates**	**4 SE/ciclo**

* Frequência de todo o CABAC.

Fonte: Próprio autor, 2016.

Uma comparação dos resultados da síntese realizada com os trabalhos correlatos não é necessariamente simples, uma vez que quase todos os trabalhos não apresentam resultados somente para o bloco de binarização.

O trabalho de Martins (2011) apresenta os resultados de síntese, para o padrão H.264/AVC, de cada um dos blocos do CABAC. A síntese foi realizada em FPGA e não foram apresentados resultados de consumo de potência. A frequência apresentada foi de 350 MHz. O trabalho do Liu *et al.* (2011), apresenta resultados de área, frequência e *throughput.* A frequência apresentada foi de 250 MHz, sendo pior em relação ao trabalho do Martins, que foi desenvolvido para o mesmo padrão. A área foi de 19.3 kgates, tendo uma área maior em comparação com a encontrada neste trabalho. O *throughput* é de 1 a 2 SE/ciclo, enquanto que neste trabalho a taxa de processamento foi de 4 SE/ciclo. O trabalho do Zhou *et al.* (2015), não apresenta resultados de consumo de potência, e apresenta resultados de frequência considerando o CABAC como um todo. O *throughput* é de 4 SE/ciclo, porém, apresenta restrições em relação ao tamanho de bits e o formato de binarização para dois deles. A arquitetura em paralelo com *Operand Isolation* apresenta *throughput* de 4 SE/ciclo, sem restrições, alcançando uma frequência de 834 MHz.

Em relação ao consumo de potência, não foi possível comparar os resultados obtidos, pois, nenhum trabalho encontrado na literatura apresenta dados de consumo (sendo o diferencial deste trabalho, onde foi utilizada sequências reais de vídeo para gerar os dados encontrados). Mesmo assim, ao comparar os resultados de consumo entre as arquiteturas desenvolvidas neste trabalho, é notável o ganho

que se obteve com a técnica, podendo ser uma excelente alternativa para projetos de codificadores de vídeo de baixo consumo.

7 CONCLUSÃO

Este trabalho apresenta uma breve descrição sobre codificação de vídeo, vídeos digitais e suas redundâncias. Além disso, foi apresentado o diagrama da codificação de vídeo explicando em linhas gerais o funcionamento dos principais blocos. Posteriormente, foi explicado o bloco de binarização, que é o bloco que foi o foco deste trabalho, onde foi descrito porque ele é realizado, o seu funcionamento e os tipos de binarização.

Com base nos estudos realizados sobre o padrão HEVC e CABAC, foi proposta uma arquitetura para o bloco de binarização. Esta arquitetura visa redução do consumo de potência, mantendo alto desempenho. Para isto, foi utilizada a técnica *Operand Isolation* que é utilizada em lógicas puramente combinacionais para redução de consumo dinâmico de potência. Esta técnica foi utilizada, pois ao se realizar um estudo estatístico, foi determinado que alguns métodos de binarização são chamados várias vezes seguidas, o que implica que nem todos os blocos de binarização precisam estar ativos ao mesmo tempo, já que não são todos necessários simultaneamente.

Para utilizar esta técnica, foi realizada a inserção de ANDs utilizando uma lógica própria de habilitação dos blocos. Assim, o formato do elemento sintático passou a ser informado antes do processo de binarização, desligando os blocos que não serão utilizados, gerando apenas um elemento sintático binarizado. Foi desenvolvido também, uma arquitetura com quatro núcleos em paralelo. Assim, quaisquer quatro elementos sintáticos podem ser inseridos por ciclo de clock, que é uma alternativa para impedir que a binarização possa ser o gargalo ao ser integrada a um codificador de entropia completo.

A arquitetura descrita em VHDL foi sintetizada com a ferramenta RTL Compiler da Cadence. A síntese foi realizada para ASIC e a biblioteca de células utilizada foi a 65nm da ST. Para analisar os resultados encontrados, foi feita uma comparação entre as quatro arquiteturas desenvolvidas neste trabalho.

Ao analisar os resultados de consumo de potência, verificou-se que a arquitetura com a técnica de *Operand Isolation,* obteve uma redução de consumo de potência de até 41%, com frequência de 858 MHz. Enquanto que, a arquitetura em paralelo com a técnica de *Operand Isolation,* obteve uma redução de consumo de potência de até 37%, sendo apta a processar quatro elementos sintáticos quaisquer

por ciclo de operação e com 834 MHz de frequência alcançada, atingido os objetivos traçados para este trabalho.

Além disso, realizou-se uma comparação entre os resultados alcançados neste trabalho com os trabalhos correlatos. O trabalho desenvolvido apresentou melhor resultado de frequência e área, sendo o único trabalho que apresenta resultados de consumo de potência, baseados em sequência reais de vídeo. Desta forma, este trabalho se apresenta como alternativa de alto potencial para ser utilizado em um CABAC e um codificador completo de vídeo, visando aplicações de baixo consumo e alto desempenho.

REFERÊNCIAS

AGOSTINI, Luciano Volcan. **Desenvolvimento de Arquiteturas de Alto Desempenho Dedicadas à Compressão de Vídeo Segundo o Padrão H. 264/AVC**. Tese de Doutorado - Instituto de Informática. Universidade Federal do Rio Grande do Sul, 2007.

ARAUJO, André. **Uma Proposta de Estimação de Movimento para o Codificador de Vídeo Dirac**. Tese de Doutorado . Universidade Estadual de Campinas, 2010.

CHEN, Yu-Hsin, *et al*. SZE, Vivienne**.** A deeply pipelined CABAC decoder for HEVC supporting level 6.2 high-tier applications. **IEEE Transactions on Circuits and Systems for Video Technology**. p. 856-868, 2015.

CORRÊA, Guilherme Ribeiro. **Estudo e Desenvolvimento de Heurísticas e Arquiteturas de Hardware para Decisão Rápida do Modo de Codificação de Bloco para o Padrão H. 264/AVC**. Dissertação de Mestrado - Instituto de Informática. Universidade Federal do Rio Grande do Sul, 2010.

CORREALE JR, Anthony. Overview of the power minimization techniques employed in the IBM PowerPC 4xx embedded controllers. In: International Symposium on Low Power Design. **Anais...** ACM. p. 75-80, 1995.

DA SILVA, Mateus Grellert. **Computational Effort Analysis and Control in High Efficiency Video Coding**. Dissertação de Mestrado. Universidade Federal do Rio Grande do Sul, 2014.

DARONCO, Leonardo Crauss. **Avaliação subjetiva de qualidade aplicada à codificação de vídeo escalável**. Dissertação de Mestrado. Universidade Federal do Rio Grande do Sul. 2009.

DE ALMEIDA JUNIOR, Jurandy Gomes. **Recuperação de vídeos comprimidos por conteúdo**. Tese de Doutorado. Universidade Estadual de Campinas (UNICAMP). Instituto de Computação, 2011.

DE OLIVEIRA, Jean Felipe Fonseca; DE ALENCAR, Marcelo Sampaio. Padrão HEVC–Novas Tecnologias para Aplicações de Elevadas Taxas de Compressão de Vídeo. **Revista De Tecnologia Da Informação e Comunicação,** Vol. 4, No. 2, Outubro, 2014.

DINIZ, Claudio Machado. **Arquitetura de hardware dedicada para a predição intra-quadro em codificadores do padrão H. 264/AVC de compressão de vídeo**. Dissertação de Mestrado - Instituto de Informática. Universidade Federal do Rio Grande do Sul, 2009.

GRELLERT, Mateus, *et al*. MSBSD: um Algoritmo para Decisão Rápida na Inter-Predição em Codificadores HEVC. **Revista de Iniciação Científica**, v. 12, n. 3, 2012.

HEVC Test Model, HM 16. Disponível em: <https://hevc.hhi.fraunhofer.de/svn/svnHEVCSoftware/tags/HM-16.6>.

ITU-T. **High Efficiency Video Coding (HEVC) Version 3**, 2015.

LIU, Yizhong; SONG, Tian; SHIMAMOTO, Takashi. High performance binarizer for H. 264/AVC CABAC. In: Electric Information and Control Engineering (ICEICE), 2011 International Conference on. **Anais...** IEEE, 2011. p. 2237-2240, 2011.

MANOEL, Edson Tadeu Monteiro. **Codificação de vídeo H. 264: estudo de codificação mista de macroblocos**. Dissertação de Mestrado - Centro Tecnológico. Universidade Federal de Santa Catarina, 2007.

MARQUES, Bruno Bezerra. **Concepção e simulação em alto nível de sistemas eletrônicos para aplicações de RSSF, com síntese direta em circuitos integrados.** Monografia. Universidade de Brasília, 2015.

MARTINS, André Luis Del Mestre. **Projeto da arquitetura de hardware para binarização e modelagem de contextos para o CABAC do padrão de compressão de vídeo H. 264/AVC**. Dissertação de Mestrado - Instituto de Informática. Universidade Federal do Rio Grande do Sul, 2011.

MOREIRA, Tom Jones. Conceitos e Fundamentos do HEVC/H. 265. **Revista da SET: Sociedade Brasileira de Engenharia de Televisão**, 2014.

PALOMINO, Daniel Munari. **Algorithm and Hardware Based Architectural Design Targeting the Intra-Frame Prediction of the HEVC Video Coding Standard**. Tese de Doutorado - Instituto de Informática. Universidade Federal do Rio Grande do Sul, 2013.

PENG, Bin et al. A hardware CABAC encoder for HEVC. In: IEEE International Symposium on Circuits and Systems (ISCAS). **Anais...** IEEE, 2013. p. 1372-1375, 2013.

PORTO, Roger Endrigo Carvalho. **Desenvolvimento arquitetural para estimação de movimento de blocos de tamanhos variáveis segundo padrão H. 264/AVC de compressão de vídeo digital**. Dissertação de Mestrado - Instituto de Informática. Universidade Federal do Rio Grande do Sul, 2008.

RICHARDSON, Iain E. **H. 264 and MPEG-4 video compression: video coding for next-generation multimedia**. John Wiley & Sons, 2003.

RICHARDSON, Iain E. **The H. 264 advanced video compression standard**. John Wiley & Sons, 2010.

SANCHES, Ionildo José. **Compressão sem Perdas de Projeções de Tomografia Computadorizada usando a Transformada Wavelet**. Dissertação de Mestrado - Setor de Ciências Exatas. Universidade Federal do Paraná, 2001.

STANKOWSKI, Jakub et al. Bitrate distribution of syntax elements in the HEVC encoded video. In: Signals and Electronic Systems (ICSES), 2014 International Conference on. **Anais...** IEEE, 2014. p. 1-4, 2014.

SULLIVAN, Gary J., *et. al.* Overview of the High Efficiency Video Coding (HEVC) Standard*.* **IEEE Transactions on Circuits and Systems for Video Technology***,* 2012*.* P. 1649-1668, 2012.

SZE, Vivienne; BUDAGAVI, Madhukar. A comparison of CABAC throughput for HEVC/H. 265 VS. AVC/H. 264. In: IEEE Workshop on Signal Processing Systems **Anais...** 2013.

SZE, Vivienne; BUDAGAVI, Madhukar; SULLIVAN, Gary J. **High Efficiency Video Coding (HEVC)**. Springer, 2014.

SZE, Vivienne; BUDAGAVI, Madhukar. High throughput CABAC entropy coding in HEVC. **IEEE Transactions on Circuits and Systems for Video Technology**, 2012. p. 1778-1791, 2012.

SZE, Vivienne and BUDAGAVI, Madhukar. Parallelization of CABAC transform coefficient coding for HEVC. In: IEEE Picture Coding Symposium (PCS). **Anais...** IEEE, 2012.

TEIXEIRA, Gabriel Diego. **Desenvolvimento de uma arquitetura de hardware de um estimador de vetores de movimento de precisão sub-pixel seguindo o padrão HEVC.** Trabalho de Conclusão de Graduação – Instituto de Informática. Universidade Federal do Rio Grande do Sul, 2014.

TEW, Yiqi and KOKSHEIK Wong. Information hiding in HEVC standard using adaptive coding block size decision. In: IEEE International Conference on Image Processing (ICIP). **Anais...** IEEE, 2014.

THIELE, Cristiano. **Desenvolvimento da arquitetura dos codificadores de entropia adaptativos CAVLC e CABAC do padrão H. 264/AVC.** Dissertação de Mestrado - Instituto de Informática. Universidade Federal do Rio Grande do Sul, 2012.

VIANNA, Henrique, *et al*. Very high throughput FPGA design for vertical rotational transform of HEVC emergent video coding standard. In: VIII Southern Conference on, Programmable Logic (SPL). **Anais...** IEEE, 2012. p. 1-5, 2012.

VIZZOTTO, Bruno; MAZUI, Volnei; BAMPI, Sergio. Area efficient and high throughput CABAC encoder architecture for HEVC. In: 2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS). **Anais...** IEEE, 2015. p. 572-575, 2015.

ZHOU, Dajiang *et al*. Ultra-high-throughput VLSI architecture of H. 265/HEVC CABAC encoder for UHDTV applications. **IEEE Transactions on Circuits and Systems for Video Technology.** v. 25, n. 3, p. 497-507, 2015.

ZHOU, Jinjia *et al*. A high-performance CABAC encoder architecture for HEVC and H. 264/AVC. In: 20th IEEE International Conference on - Image Processing (ICIP), 2013. **Anais...** IEEE, 2013. p. 1568-1572, 2013.

ANEXO A – ELEMENTOS SINTÁTICOS E SEUS VALORES DE CMAX E FORMATO

Elemento Sintático	Representação em Binário (9 bits)	cMax	Formato
end_of_slice_segment_flag	000000000	0001	0001
end_of_subset_one_bit	000000001	0001	0001
sao_merge_left_flag	000000010	0001	0001
sao_merge_up_flag	000000011	0001	0001
sao_type_idx_luma	000000100	0010	0101
sao_type_idx_chroma	000000101	0010	0101
sao_offset_abs	000000110	0011	0101
sao_offset_sign	000000111	0001	0001
sao_band_position	000001000	0100	0001
sao_eo_class_luma	000001001	0101	0001
sao_eo_class_chroma	000001010	0101	0001
split_cu_flag	000001011	0001	0001
cu_transquant_bypass_flag	000001100	0001	0001
cu_skip_flag	000001101	0001	0001
pred_mode_flag	000001110	0001	0001
part_mode	000001111	-	0101
pcm_flag	000010000	0001	0001
prev_intra_luma_pred_flag	000010001	0001	0001
mpm_idx	000010010	0010	0100
rem_intra_luma_pred_mode	000010011	0100	0001
intra_chroma_pred_mode	000010100	-	0110
rqt_root_cbf	000010101	0001	0001
merge_flag	000010110	0001	0001
merge_idx	000010111	0110	0100
inter_pred_idc[x0][y0]	000011000	-	0111
ref_idx_10	000011001	0111	0100
mvp_10_flag	000011010	0001	0001

ref_idx_11	000011011	0111	0100
mvp_11_flag	000011100	0001	0001
split_transform_flag	000011101	0001	0001
cbf_luma	000011110	0001	0001
cbf_cb	000011111	0001	0001
cbf_cr	000100000	0001	0001
abs_mvd_greater0_flag	000100001	0001	0001
abs_mvd_greater1_flag	000100010	0001	0001
abs_mvd_minus2	000100011	-	0011
mvd_sign_flag	000100100	0001	0001
cu_qp_delta_abs	000100101	-	1001
cu_qp_delta_sign_flag	000100110	0001	0001
cu_chroma_qp_offset_flag	000100111	0001	0001
cu_chroma_qp_offset_idx	000101000	1001	0100
log2_res_scale_abs_plus1	000101001	1010	0100
res_scale_sign_flag	000101010	0001	0001
transform_skip_flag	000101011	0001	0001
explicit_rdpcm_flag	000101100	0001	0001
explicit_rdpcm_dir_flag	000101101	0001	0001
last_sig_coeff_x_prefix	000101110	1011	0100
last_sig_coeff_y_prefix	000101111	1011	0100
last_sig_coeff_x_suffix	000110000	1100	0001
last_sig_coeff_y_suffix	000110001	1101	0001
coded_sub_block_flag	000110010	0001	0001
sig_coeff_flag	000110011	0001	0001
coeff_abs_level_greater1_flag	000110100	0001	0001
coeff_abs_level_greater2_flag	000110101	0001	0001
coeff_abs_level_remaining	000110110	-	1010
coeff_sign_flag	000110111	0001	0001

Fonte: Próprio autor, 2016.