UNIVERSIDADE FEDERAL DO PAMPA

HEITOR HUARACHI

DESENVOLVIMENTO DE UMA ARQUITETURA EM HARDWARE DO BLOCO DE BINARIZAÇÃO E DO ELEMENTO SINTÁTICO RESIDUAL ESPECIAL BASEADO NO PADRÃO VVC

Bagé
2025

HEITOR HUARACHI

DESENVOLVIMENTO DE UMA ARQUITETURA EM HARDWARE DO BLOCO DE BINARIZAÇÃO E DO ELEMENTO SINTÁTICO RESIDUAL ESPECIAL BASEADO NO PADRÃO VVC

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Engenharia de Computação como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Fábio Luís Livi Ramos

Bagé
2025

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).

H874d HUARACHI, HEITOR

DESENVOLVIMENTO DE UMA ARQUITETURA EM HARDWARE DO BLOCO DE BINARIZAÇÃO E DO ELEMENTO SINTÁTICO RESIDUAL ESPECIAL BASEADO NO PADRÃO VVC / HEITOR HUARACHI.

79 f.: il.

Orientador: Fábio Luís Livi Ramos

Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Pampa, Engenharia de Computação, 2025.

1. Binarização. 2. Elementos Sintáticos Residuais. 3. VVC. I. Título.

HEITOR MAURO CHAVEZ HUARACHI

DESENVOLVIMENTO DE UMA ARQUITETURA EM HARDWARE DO BLOCO DE BINARIZAÇÃO E DO DECABSLEVEL BASEADO NO PADRÃO VVC

Trabalho de Conclusão de Curso apresentado ao curso de Engenharia de Computação como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Computação.

Dissertação defendida e aprovada em: 10 de dezembro de 2025.

Banca examinadora:

_____

Prof. Dr. Fábio Luís Livi Ramos

Orientador

(UNIPAMPA)

_____

Prof. Dr. Bruno Silveira Neves

(UNIPAMPA)

_____

Prof. Dr. Carlos Michel Betemps

(UNIPAMPA)

Assinado eletronicamente por **FABIO LUIS LIVI RAMOS**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 18/12/2025, às 18:59, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **CARLOS MICHEL BETEMPS**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 18/12/2025, às 20:43, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **BRUNO SILVEIRA NEVES**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 19/12/2025, às 16:21, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Dedico este trabalho a minha mãe Silvia, e meu pai Mauro, por serem as pessoas mais corajosas que conheço, por sempre estarem me incentivando ao estudo, e me dando a oportunidade de concluir minha graduação em uma universidade publica.

AGRADECIMENTOS

Agradeço profundamente aos meus pais, Silvia e Mauro, por sempre acreditarem no meu potencial, incentivarem meus estudos e estarem ao meu lado em todos os momentos. Sem o apoio constante de vocês, esta etapa não teria sido possível.

À minha namorada, Julia, que esteve comigo nos períodos mais desafiadores da faculdade, oferecendo apoio, compreensão e motivação quando eu mais precisei. Sua presença foi essencial para que eu chegasse até aqui, e sou imensamente grato por tudo que fez por mim durante esta jornada.

Aos meus amigos Cardoso, Dias, Fredes, Guilherme, Gustavo, Renan, Renato, Tomás e Vinícius, que estiveram ao meu lado durante toda essa caminhada, compartilhando dificuldades, aprendizados e conquistas. Cada um contribuiu para tornar essa trajetória mais leve e especial.

Agradeço também ao lendário "graxo", um lanche excepcional que conheci em Bagé e que, sem dúvida, permanecerá para sempre na memória quando eu recordar da graduação e da cidade.

Ao meu gato Banguela, que se tornou um grande apoio emocional neste último ano, trazendo conforto e companhia nos momentos de maior pressão e cansaço.

Por fim, expresso meu sincero agradecimento ao meu orientador, professor Fábio, cuja dedicação e incentivo à pesquisa me acompanharam por muitos anos. Seu comprometimento com o trabalho, sua orientação atenta e sua confiança foram fundamentais para a conclusão deste estudo. Hoje, além de orientador, considero-o um grande amigo, e sou profundamente grato por todo o aprendizado e apoio.

“If I have seen farther than others, it is because I stood on the shoulders of giants.”
— Sir Isaac Newton

RESUMO

Nos últimos anos, a demanda por vídeo aumentou significativamente devido ao uso intensivo de plataformas de *streaming* e ao crescimento do trabalho remoto. Para atender a esse cenário, torna-se necessário empregar soluções de codificação mais eficientes. O *Versatile Video Coding* (VVC) é um padrão moderno projetado para alcançar altas taxas de compressão mantendo a qualidade do vídeo, porém essa eficiência implica maior complexidade computacional. Para reduzir essa complexidade, o uso de arquiteturas dedicadas em hardware é uma abordagem amplamente adotada, especialmente nas etapas críticas do processo de codificação. Entre essas etapas, destacam-se a geração dos elementos sintáticos residuais e o processo de binarização, responsáveis por grande parte dos dados manipulados pelo módulo *CABAC*.

Este trabalho apresenta o desenvolvimento completo de uma arquitetura dedicada que integra o bloco *DecAbsLevel*, responsável pela geração de uma parte dos elementos sintáticos residuais, e todos os métodos de binarização utilizados no processo de codificação. A validação funcional foi realizada por meio de simulações, utilizando valores extraídos diretamente do codificador de referência do VVC. Além disso, foi conduzida uma síntese para ASIC, permitindo analisar área, potência e frequência de operação da arquitetura. Por fim, os resultados obtidos foram comparados com trabalhos correlatos presentes na literatura, possibilitando avaliar o desempenho da proposta em relação a outras implementações existentes.

Palavras-chave: Binarização. Elementos Sintáticos Residuais. VVC.

ABSTRACT

In recent years, the demand for video has increased significantly due to the intensive use of streaming platforms and the growth of remote work. To address this scenario, it becomes necessary to employ more efficient coding solutions. Versatile Video Coding (VVC) is a modern standard designed to achieve high compression rates while maintaining video quality; however, this efficiency results in increased computational complexity. To mitigate this complexity, the use of dedicated hardware architectures is a widely adopted approach, especially in the critical stages of the encoding process. Among these stages, the generation of residual syntax elements and the binarization process stand out, as they are responsible for a significant portion of the data handled by the CABAC module.

This work presents the complete development of a dedicated architecture that integrates the *DecAbsLevel* block—responsible for generating part of the residual syntax elements—and all binarization methods used in the coding process. Functional validation was carried out through simulations using values extracted directly from the VVC reference encoder. In addition, an ASIC synthesis was performed, enabling the analysis of area, power consumption, and operating frequency of the architecture. Finally, the obtained results were compared with related works from the literature, allowing the evaluation of the proposed architecture in relation to existing implementations.

Keywords: Binarization. Residual Syntactic Elements. VVC.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AOMedia	Alliance for Open Media
ASIC	Application Specific Integrated Circuit
AV1	AOMedia Video 1
AVC	Advanced Video Coding
CABAC	Context-based Adaptive Binary Arithmetic Coding
Cb	Coding Block (largura e altura do bloco luma)
CTU	Coding Tree Unit
CU	Unidade de Codificação (Coding Unit)
DC	Direct Current prediction
DCT	Transformada Discreta do Cosseno (Discrete Cosine Transform)
EGk	k-th order Exponential-Golomb
FL	Comprimento Fixo (Fixed-Length)
fps	Frames Per Second
GHz	Gigahertz
HEVC	High Efficiency Video Coding
ISO/IEC	International Organization for Standardization / International Electrotechnical Commission
ITU-T	International Telecommunication Union - Telecommunication Standardization Sector
JVET	Joint Video Experts Team
Kgates	Kilo Gates
LD	Low Delay
MCP	Motion Compensated Prediction
ME	Motion Estimation
MHz	Megahertz

mW	Milliwatt
MPEG	Moving Picture Experts Group
MUX	Multiplexador (Multiplexer)
MV	Motion Vector
ns	Nanosecond
PDK	Process Design Kit
QP	Quantization Parameter
RGB	Red, Green, Blue
SE	Syntax Element
SEs	Syntax Elements
TB	Truncado Binário
TR	Truncado Rice
TU	Truncado Unário
U	Unário
UHD	Ultra High Definition
UNIPAMPA	Universidade Federal do Pampa
VHDL	VHSIC Hardware Description Language
VVC	Versatile Video Coding
YCbCr	Luma, Chroma Blue, Chroma Red
°C	Graus Celsius

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, a demanda por serviços de *streaming* vem aumentando significativamente, impulsionada pelo consumo de vídeos em celulares, computadores, *smart TVs* e outros dispositivos. Estudos recentes indicam que o *streaming* é o tipo de conteúdo dominante na internet, independentemente da plataforma utilizada, representando entre 40% e 50% do tráfego global em 2024. Entre os principais responsáveis por esse volume estão plataformas como Netflix, Amazon e YouTube, que somam aproximadamente 30% do tráfego mundial [Sandvine 2024].

As resoluções de vídeo também evoluíram de forma acentuada, chegando atualmente aos formatos 4K e 8K. Para fins de exemplificação, um vídeo em resolução 1080p (Full HD) pode aumentar seu tamanho em cerca de 10,6 GB por minuto. Considerando conteúdos como filmes, transmissões ao vivo e séries, que geralmente possuem duração média de uma hora, o volume total pode atingir aproximadamente 636 GB por vídeo. Tamanhos dessa magnitude seriam inviáveis para as condições de banda disponíveis atualmente. Como forma de contornar essa limitação, empresas e pesquisadores em todo o mundo têm buscado tecnologias mais eficientes de compressão de vídeo [Zhang et al. 2013].

A compressão de vídeo consiste em reduzir o tamanho de arquivos por meio de técnicas que exploram redundâncias temporais e espaciais presentes nos dados. Essa redução é obtida por métodos padronizados de codificação, implementados via software ou hardware, permitindo diminuir significativamente o tamanho do vídeo sem comprometer seu conteúdo essencial.

Existem diversos padrões de codificação de vídeo, como AV1, HEVC e VVC, cada um projetado para necessidades específicas do mercado. Um exemplo é o codec (codificação e decodificação) AV1, desenvolvido por grandes empresas como Amazon, Google e Netflix, que formaram a *Alliance for Open Media* (AOMedia) em 2015 com o objetivo de criar um codec livre de *royalties*, voltado especialmente para transmissões pela internet [Norkin et al. 2022].

O estado da arte em codificação de vídeo é representado pelo *Versatile Video Coding* (VVC) [ITU-T e ISO/IEC 2020], lançado em 2020. Esse padrão oferece compressão significativamente superior, reduzindo pela metade a taxa de bits necessária em comparação com codecs anteriores, sem comprometer a qualidade visual. Apesar de sua alta eficiência, o VVC apresenta maior tempo de codificação devido à complexidade

de seus algoritmos [Uhrina et al. 2024]. Ainda assim, por ter sido projetado para lidar com resoluções elevadas, como 4K e 8K, sua adoção é especialmente vantajosa em aplicações que exigem alta qualidade visual e compressão eficiente.

O processo de codificação de vídeo é composto por várias etapas. Inicialmente, cada quadro é segmentado em blocos e submetido a técnicas de predição, seja intraquadro, baseada no próprio quadro, ou interquadro, utilizando quadros adjacentes. Em seguida, aplica-se uma transformada para representar os dados no domínio da frequência, gerando coeficientes que são posteriormente quantizados. Ao longo desse fluxo, diversos elementos sintáticos são produzidos para estruturar o conjunto de informações que será transmitido ou armazenado. Entre esses elementos, destacam-se os Elementos Sintáticos Residuais (ESRs), originados após a quantização. Por fim, a codificação de entropia reduz a redundância final do fluxo binário.

O tratamento final desses elementos sintáticos ocorre no módulo de entropia, que realiza a binarização e posterior codificação aritmética. Nesse contexto, a importância dos ESRs é relevante: estudos realizados no padrão HEVC indicam que eles podem representar até 94% do fluxo codificado [Ramos et al. 2020].

O *Context-based Adaptive Binary Arithmetic Coding* (CABAC) é a técnica utilizada no VVC para a codificação de entropia. O método explora propriedades estatísticas dos dados, atribuindo códigos menores aos símbolos mais frequentes e códigos maiores aos menos frequentes. O CABAC é fundamental para a compressão de elementos sintáticos como resíduos quantizados e vetores de movimento, garantindo alta eficiência principalmente em vídeos de alta resolução e altas taxas de quadros [Marpe, Schwarz. H e Wiegand 2003].

Embora o VVC apresente ganhos expressivos em eficiência de compressão, esses avanços implicam um aumento significativo da complexidade computacional do processo de codificação. A implementação do VVC em software torna-se, portanto, onerosa em termos de desempenho e consumo energético, especialmente em aplicações que exigem processamento em tempo real. Como alternativa, a utilização de hardware dedicado permite tratar etapas específicas do codec de forma especializada, reduzindo a complexidade computacional e viabilizando o uso do VVC em cenários com restrições de desempenho e eficiência.

1.1 Objetivos

Este trabalho tem como objetivo desenvolver duas arquiteturas distintas: uma dedicada ao processamento de elementos sintáticos residuais, especificamente à geração do *DecAbsLevel* - um tipo especial de ESR, e outra destinada ao bloco de binarização do CABAC, ambas conforme o padrão VVC.

As duas arquiteturas serão posteriormente integradas em uma única solução, compondo um sistema unificado.

1.1.1 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Compreender o funcionamento do padrão de compressão de vídeo VVC, com foco nos elementos sintáticos residuais;
- Estudar o codificador CABAC e suas formas de binarização no padrão VVC;
- Desenvolver uma arquitetura digital em VHDL dedicada à geração de um elemento sintático residual especial;
- Implementar uma arquitetura para o módulo de binarização CABAC em VHDL;
- Integrar as arquiteturas desenvolvidas e validar seu funcionamento em conjunto;
- Realizar simulações e síntese em tecnologia ASIC (*Application Specific Integrated Circuit*), avaliando os resultados obtidos em termos de área, potência e desempenho, e assim comparar com trabalhos correlatos.

1.2 Organização do Texto

Este trabalho está estruturado em seis capítulos, além desta introdução. O Capítulo dois apresenta os conceitos gerais relacionados à compressão de vídeo e ao padrão VVC, bem como a revisão bibliográfica. O Capítulo três aborda inicialmente a estrutura do elemento sintático residual e, em seguida, apresenta os principais tipos de binarização utilizados no padrão, além dos trabalhos correlatos. O Capítulo quatro descreve a

metodologia adotada para o desenvolvimento deste projeto. O Capítulo cinco detalha o desenvolvimento da arquitetura voltada à geração do *DecAbsLevel* e à construção das binarizações. O Capítulo seis apresenta os resultados obtidos pela arquitetura e suas comparações com os trabalhos correlatos. Por fim, o Capítulo sete traz as considerações finais.

2 CONCEITOS GERAIS DE VÍDEO E REVISÃO BIBLIOGRÁFICA

Nesta seção serão apresentados alguns entendimentos prévios necessários para as próximas seções, incluindo conceitos sobre codificação de vídeo, aspectos funcionais e decodificação. Também serão abordados conteúdos específicos do padrão utilizado neste trabalho, o VVC, com ênfase em processos como binarização e elementos sintáticos.

2.1 Conceitos de Vídeo Digital e Compressão de Video

Para fornecer o devido entendimento sobre o conteúdo deste trabalho e seu embasamento na área de codificação de vídeo, é necessário, primeiramente, compreender como um vídeo é formado. Um vídeo é composto por uma sequência de imagens estáticas individuais, tecnicamente chamadas de quadros (*frames*). A percepção de movimento é gerada quando esses quadros são exibidos sucessivamente por um determinado tempo, e em alta velocidade [Gonzalez e Woods 2000]. A fluidez dessa sensação de movimento é quantificada pela taxa de quadros, ou fps *(frames per second)*, que indica o número de imagens exibidas por segundo. Padrões comuns, como 30 a 60 fps, são amplamente adotados para assegurar que a transição entre os quadros seja rápida o suficiente para criar uma representação de movimento contínuo e natural.

Dentro dessas imagens que são usadas para formar os *frames*, temos os *pixels* que pelo autor Ramachandran (2018), são diversos pontos, sendo a menor parte de uma imagem digital. Ele armazena dados sobre a cor e a intensidade luminosa que formam a imagem. Quando muitos *pixels* são agrupados, eles criam a imagem completa.

2.1.1 Espaço de cores YCbCr

Existem diversas formas para representar cores, no contexto de imagem digital, se utiliza muito o RGB *(red,green,blue)*, onde as cores são denominados nessa intensidade de cores vermelha, verde e azul.

Na área de codificação de vídeo, há uma preferência pelo modelo YCbCr, que divide as informações da imagem em três componentes: (Y) luminância e (Cb e Cr) crominância. A luminância (Y) representa a intensidade de luz, enquanto a crominância (Cb e Cr) está relacionada às informações de cor. Esse tipo de separação ocorre por

conta da limitação humana, onde o ser humano é mais sensível a variação de luz, do que a variações de cor [Gonzalez e Woods 2000]. A Figura 1 ilustra a divisão dos três componentes do espaço de cor YCbCr. Esse modelo de cor é amplamente adotado devido a propriedades que contribuem para a diminuição da quantidade de bits requeridos na compressão.

Figura 1 – Exemplo de YCbCr

Fonte: Microsoft (2021).

Essas separações entre crominância e luminância, como mostrado na Figura 1, tornam-se mais evidentes nas subamostragens, nas quais os elementos responsáveis pela cor podem ser representados com menor precisão do que os elementos de luz (luminância). Os principais tipos de subamostragem são:

- Formato 4:4:4

  Cada pixel possui informações completas de luminância e crominância, ou seja, os três componentes(Y,Cb,Cr), são mostrados com a mesma resolução, tanto na horizontal quanto na vertical, garantindo extrema qualidade e fidelidade de cor, como mostrado na Figura 2.

- Formato 4:2:2 Equilíbrio entre qualidade e economia de dados. Luminância em todos os pixels, já os componentes que são utilizados de cor são amostrados com metade da resolução na direção horizontal.

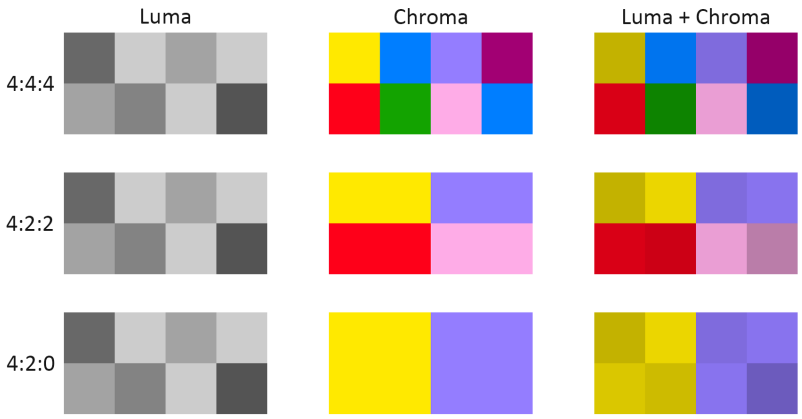
- Formato 4:2:0

  Os componentes de crominância são sub-amostrados na horizontal e na vertical, onde apenas uma amostragem de Cb (vermelha) e uma amostragem de Cr(azul) para cada bloco de quatro pixels (dois por dois), ou seja três amostras das cores

[0]Fonte: Microsoft Learn. Disponível em: <https://learn.microsoft.com/en-us/windows/win32/wic/jpeg-ycbcr-support>

azul e vermelha são descartadas, gerando uma grande redução no volume de dados para esse tipo subamostragem [Bruno 2003].

Figura 2 – Formatos de subamostragem de crominância.



Fonte: Babcock (2019)

2.2 Codificação de video

A codificação de vídeo tem como objetivo principal reduzir o tamanho do vídeo para armazenamento ou transmissão. Em uma transmissão, quando um lado está atuando como transmissor e enviando o vídeo, é necessário codificar o vídeo para que ele possa ser enviado de forma eficiente ao receptor. O receptor, por sua vez, precisará descompactar (ou decodificar) o vídeo recebido, como ilustrado na Figura 3.

Vale ressaltar que tanto o transmissor quanto o receptor possuem tanto o codificador (codec) quanto o decodificador (decoder), permitindo que ambos se comuniquem entre si. Assim, os papéis de transmissor e receptor podem ser alternados, funcionando de maneira semelhante a uma conexão duplex, como mostrado na Figura 3 [Richardson 2010].

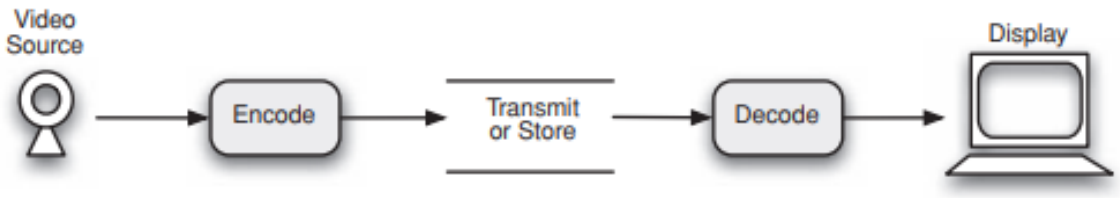


Figura 3 – Richardson (2010)

A necessidade da codificação de vídeo torna-se evidente ao se analisar a quantidade de dados envolvidos em um vídeo sem compressão. Por exemplo, um vídeo em resolução 1080p Full HD contém uma grande quantidade de dados, o que gera desafios para seu armazenamento e transmissão. Considerando que cada frame possui 1920x1080 pixels e que cada pixel é representado por 24 bits (ou 3 bytes), cada frame ocupa aproximadamente 6 megabytes de dados. Com uma taxa de 30 frames por segundo, o fluxo de dados chega a cerca de 180 megabytes por segundo. Para enviar um minuto desse vídeo sem compressão, seria necessário transmitir cerca de 10,8 gigabytes de informação. Essa enorme quantidade de dados exige uma largura de banda muito alta e resultaria em longos tempos de transmissão em redes convencionais, tornando inviável o envio direto sem alguma forma de compressão ou otimização Zhang et al. (2013). Existe dois tipos de técnicas de compressão no geral, segundo Mentzer, Gool e Tschannen (2020), e podem ser classificadas como: compressão com perdas *(lossy)* e compressão sem perdas *(lossless)*.

- Compressão com perdas *(lossy)*

  Ocorre quando na compressão se remove partes dos dados considerados imperceptíveis ao olho humano como comentado na seção anterior, não sendo idênticos aos dados originais. Um exemplo deste tipo de perda é alteração na cor ou detalhes em regiões com movimento, sendo extremamente eficiente em compressão de video. [Mentzer, Gool e Tschannen 2020].

  A Figura 4 é um exemplo de compressão com perdas *(lossy)*, onde se consegue observar as perdas na compressão na segunda imagem.

- Compressão sem perdas *(lossless)*

  Ocorre quando nenhuma informação da imagem original é descartada, isto é, todo os pixels da imagem podem ser recuperados, exatamente como antes da codificação, sem qualquer alteração [Diniz 2009].

Figura 4 – Compressão de dados com perdas de informação.

Fonte: Oliveira, 2009.

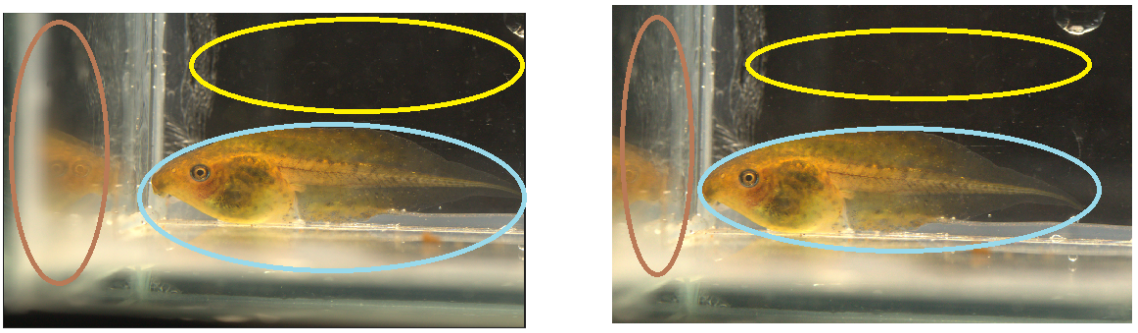
2.3 Redundâncias de dados

A compressão de vídeo tem como objetivo reduzir a quantidade de dados necessários para representar uma sequência de quadros, explorando diferentes formas de redundância visual presentes nas imagens ao longo do tempo [Santos 2020]. As formas como esses recursos são utilizados, segundo Gonzalez e Woods (2000), são a redundância temporal, a espacial e a entrópica.

- Redundância Temporal

  Se refere a informações de quadros sucessivos em uma sequência de vídeo/frames. Geralmente alguns pontos dentro do frame estão estáticos, ou seja, não ocorre muita mudança (água, nuvem, local escuro), tendo somente mudanças de deslocamento ou iluminação. Segundo Soares (2022), esta redundância é onde, atualmente, se ganha altas taxas de compressão, devido às diferentes situações presentes no vídeo, como é apresentado na Figura 5, onde as partes circuladas permaneceram praticamente inalteradas entre os quadros analisados.

Figura 5 – Redundância Temporal e Espacial.



Fonte: Castro, 2024.

- Redundância Espacial

  A redundância espacial está associada à similaridade existente entre pixels vizinhos (intra-quadro). Imagens geralmente possuem áreas com pixels semelhantes ou com variações graduais, tanto de cor como de intensidade luminosa. Os métodos de compressão intra-frame exploram essa característica ao prever o conteúdo de blocos com base nas informações dos blocos vizinhos já codificados. Um exemplo de redundância espacial pode ser visto na parte circulada em amarelo no frame 1 da Figura 5, onde os valores dos pixels são muito semelhantes no espaço.

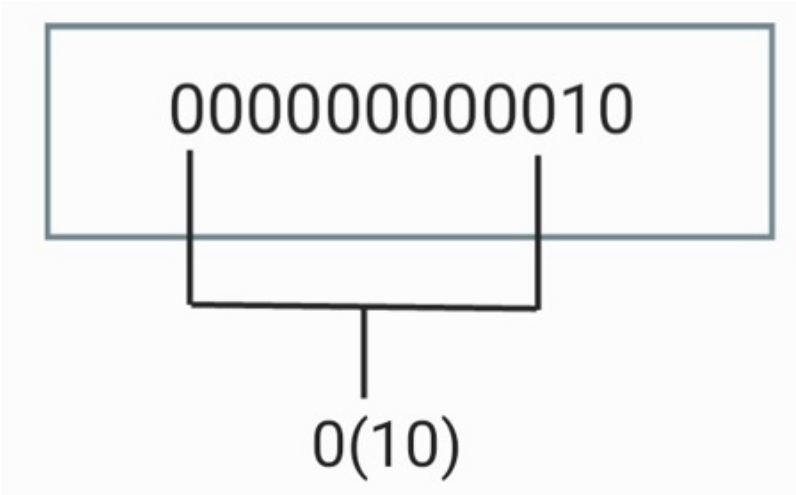
- Redundância Entropica.

  Esta relacionada com distribuição dos símbolos presentes no conjunto de fluxo de dados. Dentro de uma codificação, símbolos de mesma frequência tendem a ter representações com o mesmo número de bits, na prática, determinados valores são mais comuns de aparecer que outros. A compressão entrópica se aproveita dessa prática, para atribuir valores menores, para isso, os codificadores procuram símbolos mais frequentes para serem representados através de um único simbolo codificado.

  A Figura 6 ilustra justamente esse conceito: mesmo sendo um bloco de 12 bits, a repetição de padrões como uma sequência inicial de bits iguais, nesse caso o '0',pode ser codificada de forma mais eficiente. Em vez de transmitir todos os bits individualmente, técnicas de compressão entrópica permitem representar o padrão

de repetição com menos bits, possibilitando uma representação compacta [Agostini 2007].

Figura 6 – Redundância Entrópica.



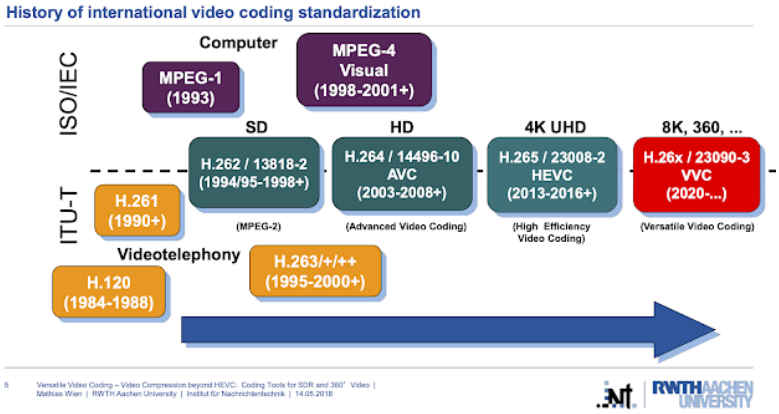
Fonte: Autor (2025).

2.4 Padrão VVC

2.4.1 Historia dos codificadores

A Figura 7 apresenta a evolução cronológica dos principais padrões de codificação de video, com destaque para o padrão mais recente de codificação de video que é o VVC (*Versatile Video Coding*/ H.266) [ITU-T e ISO/IEC 2020]. Esse novo padrão representa um avanço significativo na busca por maior eficiência de compressão e qualidade visual, especialmente diante do crescimento exponencial de conteúdos em alta resolução, como vídeos em 4K, 8K.

Figura 7 – Histórico da padronização internacional da codificação de vídeo.



Fonte: Mathias Wien(2018).

Assim como os padrões anteriores, o VVC foi desenvolvido em um esforço conjunto entre a ITU-T (International Telecommunication Union) e a ISO/IEC (MPEG – Moving Picture Expert Group), por meio do grupo JVET (Joint Video Experts Team). Esse grupo já havia sido responsável pelo desenvolvimento do H.264/AVC e do H.265/HEVC, e agora apresenta o VVC como sucessor natural na linha evolutiva dos codecs da série H.26x.

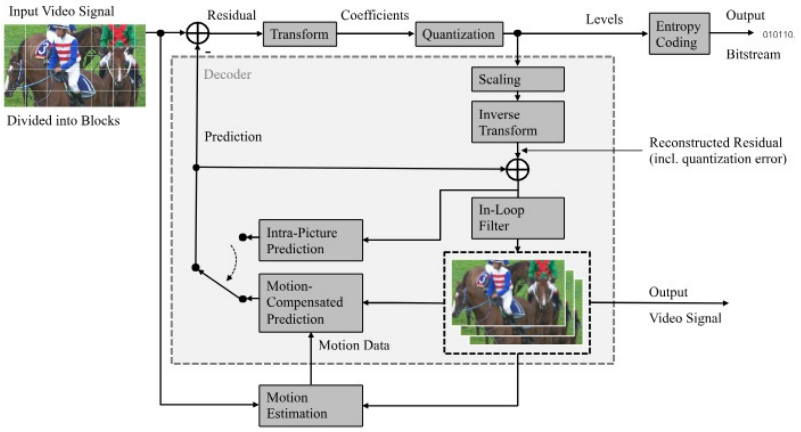
O padrão Versatile Video Coding (VVC) foi finalizado em 2020 com o objetivo de oferecer até 50% de redução de taxa de bits em relação ao HEVC, mantendo a mesma qualidade perceptiva de vídeo. Para isso, o VVC introduziu uma série de inovações, como unidades de codificação mais flexíveis, melhores algoritmos de predição intra e interquadros, técnicas mais refinadas de compensação de movimento e uma estrutura de codificação adaptável a diferentes resoluções e dispositivos [Bross et al. 2021]

2.5 Blocos do padrão VVC

Essa seção apresentará os principais blocos do VVC, identificando suas funcionalidades. No VVC cada quadro do video a ser codificado, primeiramente é dividido em blocos de tamanhos hierárquicos que são chamados de *Coding Tree Units* (CTU), contendo um numero de pixels que varia de 32x32 a 128x128. Um maior número de CTUs permite reduzir a necessidade de subdividir a imagem em regiões homogêneas [Fraga 2023]. Após o CTU, o bloco é particionado em Coding Units (CUs), que representam a unidade básica de decisão no processo de codificação, incluindo a

escolha do tipo de predição (intra ou inter). Adicionalmente, são aplicadas operações de transformada e quantização. A Figura 8 apresenta o fluxo completo do processo, desde a imagem de entrada até a geração do *bitstream*, que corresponde à sequência de bits resultante da codificação.

Figura 8 – Diagrama de blocos de um codificador de vídeo genérico.



Fonte: (BROSS et al., 2021).

Na Figura 8, nota-se a existência da quantização inversa e da aplicação de filtros. Esse fluxo inverso é necessário porque, durante a decodificação, os quadros reconstruídos mesmo com perdas decorrentes da quantização precisam ser utilizados como referência. Como o decoder só possui acesso aos quadros com perdas, a quantização inversa permite reconstruir a imagem ou vídeo de forma aproximada, possibilitando a continuidade do processo de decodificação.

2.5.1 Inter-Predição

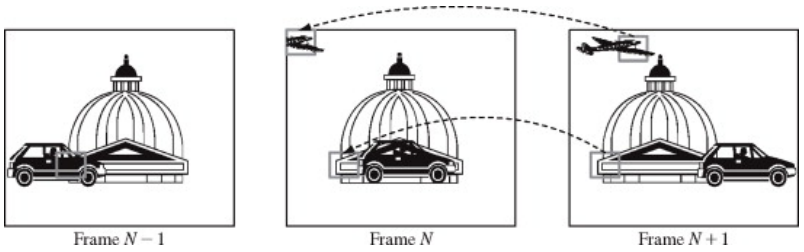
A predição inter-quadro tem como objetivo reduzir a quantidade de dados necessários para representar quadros consecutivos. Em vez de codificar cada quadro individualmente, essa técnica utiliza a semelhança entre quadros vizinhos para prever o conteúdo do quadro atual com base em quadros anteriores ou subsequentes. Dessa forma, é possível codificar apenas as diferenças entre o quadro atual e a predição, ao invés de codificar o quadro completo.

Esse processo envolve três etapas: *Motion Estimation (ME), Motion Vector (MV) e Motion Compensated Prediction (MCP)*. Na *Motion Estimation*, o quadro atual é dividido em blocos de pixels, e cada bloco é comparado com blocos do quadro de referência para

encontrar o melhor deslocamento entre eles. Isso permite calcular como cada bloco de pixels se move de um quadro para o outro, ou seja, como ele se desloca em relação ao quadro anterior ou subsequente. O vetor de movimento (MV) é o resultado desse cálculo e descreve a direção e a distância desse deslocamento, sendo uma das etapas com mais custo computacional [Ramachandran 2018].

Com os vetores de movimento definidos, o MCP é utilizado para prever a posição dos blocos no quadro atual. A ideia é deslocar os blocos do quadro de referência com base nos vetores de movimento, criando uma previsão de como o quadro atual deveria se parecer. A diferença entre a previsão e o quadro real é chamada de resíduos, e é essa diferença que será codificada. Assim, ao invés de codificar o quadro inteiro, o sistema codifica apenas o erro de predição. A Figura 9 é um exemplo utilizando MCP.

Figura 9 – Exemplo do MCP.



Fonte: Aramvith, 2009

2.5.2 Intra-predição

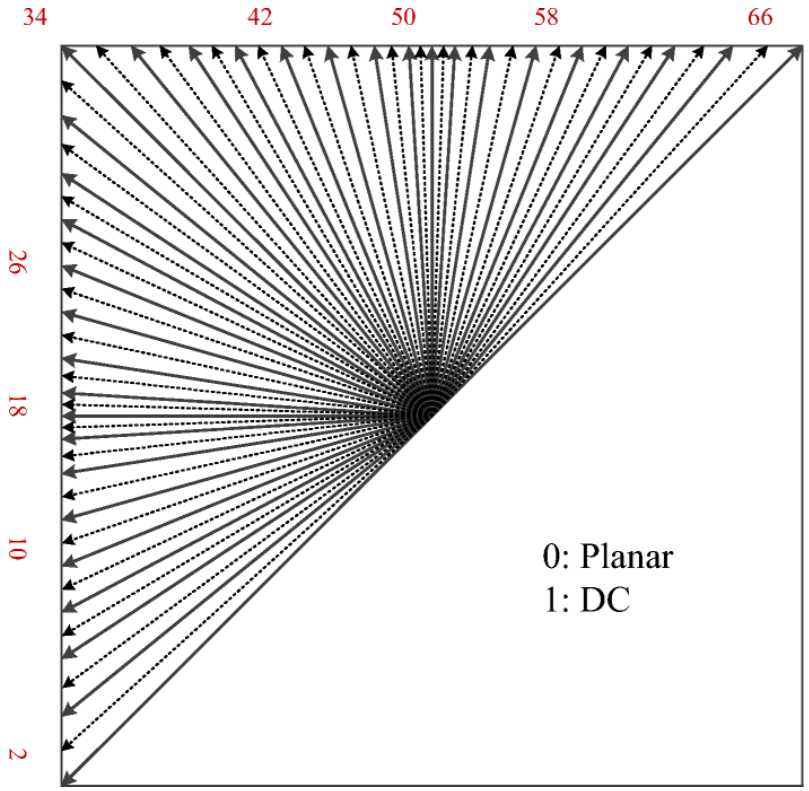
A predição intra-quadro no VVC explora a forte relação entre pixels próximos dentro de uma mesma imagem, permitindo antecipar muitos valores sem a necessidade de codificá-los explicitamente. Esse processo utiliza principalmente os pixels localizados acima e à esquerda de cada bloco como referência para prever o conteúdo do bloco atual, o que ajuda a reduzir a redundância espacial comum nas imagens [Gonçalves 2021]. Após a criação dessa previsão, ela é subtraída do quadro original, gerando um quadro residual. Esse quadro residual contém as diferenças entre a previsão e o quadro original, sendo então enviado para as etapas seguintes do processo de codificação.

Um dos grandes diferenciais do VVC está na sua capacidade de ajustar o tamanho dos blocos utilizados nesse processo, variando de blocos grandes, como 64x64, até blocos pequenos de 4x4 pixels. Essa flexibilidade permite que o codificador adapte a predição conforme a complexidade de cada região da imagem, seja para áreas com muitos detalhes

ou zonas mais uniformes [Duarte 2021].

O VVC oferece uma ampla variedade de modos de predição. Entre os mais simples estão os chamados modos não angulares, como o DC, que utiliza a média dos pixels de referência, e o Planar, que realiza uma interpolação suave entre eles. Já os modos angulares extrapolam os valores de referência em diversas direções, sendo especialmente eficazes para prever bordas e padrões de textura. Enquanto seu antecessor, o HEVC, contava com 33 modos angulares, o VVC expande para 65 modos angulares, totalizando 67 opções com os dois modos não angulares, o que representa um avanço significativo, conforme ilustrado na Figura 10.

Figura 10 – Modos e Ângulos da predição intra angular.



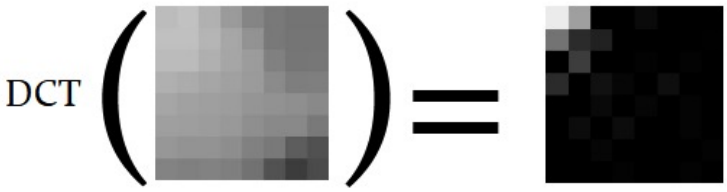
Fonte: (ITU-T; ISO/IEC, 2020).

2.5.3 Transformadas e Quantização

Os pixels que estão sendo codificados, sejam por intra-predição ou inter-predição, raramente são idênticos aos valores dos blocos de referência. A diferença resultante, chamada de resíduo, precisa ser processada por transformadas e quantização antes da codificação entropia.

A transformada é uma etapa essencial na compressão de vídeo, na qual os dados espaciais da imagem (como os pixels) são convertidos para o domínio de frequência. Uma das transformadas mais comuns utilizadas neste processo é a Transformada Discreta de Cosseno (DCT). A DCT converte blocos de pixels (na maior parte do tempo 8x8 ou 4x4) em coeficientes de frequência, representando a variação de intensidade de luz na imagem [Schwarz et al. 2021]. A principal vantagem da DCT é que ela concentra a maior parte da energia da imagem nos coeficientes de baixa frequência, localizados no canto superior esquerdo da matriz de coeficientes [Prangnell 2018]. Já os coeficientes de alta frequência, são geralmente muito pequenos ou zero após a transformada, e ficam localizados no canto inferior direito da matriz de coeficientes, a Figura 11 apresenta um exemplo de transformada por DCT.

Figura 11 – Transformada DCT.



Fonte: Rosa, 2010

Durante o processo de compressão, os coeficientes de alta frequência podem ser zerados ou simplificados, aproveitando o fato de que esses coeficientes não têm um grande impacto visual na imagem ou vídeo. Essa propriedade da DCT permite que as partes mais significativas da imagem sejam concentradas em uma pequena quantidade de coeficientes de baixa frequência, facilitando a compressão.

A quantização, que ocorre após a transformada, visa reduzir a precisão dos coeficientes gerados. Durante esse processo, os coeficientes são divididos por um valor predeterminado e, em seguida, arredondados para um número limitado de valores representáveis. Quanto maior o Nível de Quantização, maiores serão as perdas, já que os coeficientes têm maior chance de se tornarem zero. Isso acontece porque a quantização reduz a quantidade de detalhes representados, fazendo com que muitos coeficientes se aproximem de zero. Por outro lado, essa redução na precisão dos dados também resulta em maior compressão, uma vez que os coeficientes zero são mais facilmente compactados, diminuindo o tamanho do arquivo. A quantização, portanto, equilibra qualidade e compressão, e o grau de quantização é fundamental para ajustar

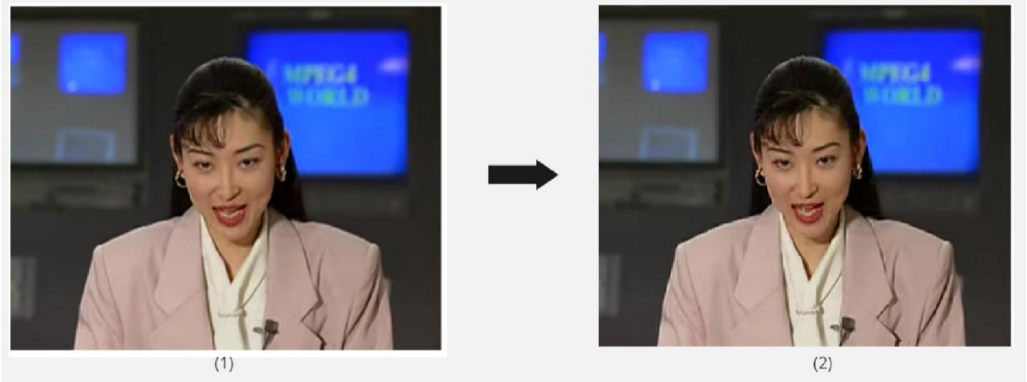
esse equilíbrio.

2.5.4 Filtros

A quantização durante a compressão de vídeo introduz perdas nos dados, o que pode resultar em bordas excessivamente quadradas ou distorções visuais nas imagens reconstruídas. Para reduzir essas perdas e melhorar a qualidade visual, utiliza-se a filtragem em loop, um processo aplicado durante a reconstrução dos quadros de referência. Essa técnica melhora o desempenho da inter-predição, suavizando as áreas afetadas pela quantização.

Um filtro comum utilizado nesse processo é o Filtro de deblocagem (Deblocking Filter), que tem como objetivo suavizar as bordas nos quadros, eliminando as descontinuidades visuais [Lee e Jeong 2020]. A filtragem em loop ocorre durante o fluxo de codificação, permitindo que os quadros reconstruídos sejam usados como referência para previsões futuras, o que resulta em uma compressão mais eficiente e uma melhor qualidade visual. A Figura 12 mostra um exemplo utilizando filtro de deblocagem.

Figura 12 – Aplicação de filtro de deblocagem. (1) Sem filtro. (2) Com filtro.



Fonte: Souza, 2023

2.5.5 Codificador de entropia

Assim como no HEVC, o VVC emprega o CABAC (*Context-Adaptive Binary Arithmetic Coding*) na etapa de codificação de entropia, com alguns aprimoramentos em seu comportamento por conta do novo padrão. Utilizado na última etapa do processo de compressão, mais precisamente depois da transformada e quantização, a saída desse processo são os elementos sintáticos residuais (SE-Sintax Elements) [Bross et al. 2021].

A funcionalidade do CABAC envolve três fases principais que garantem sua elevada eficiência, conforme apresentadas a seguir.

- Binarizador

  Etapa na qual os valores dos elementos sintáticos são convertidos em cadeias de bits, chamadas bins, essa conversão não é arbitraria: diferentes esquemas de binarização são usados conforme o tipo e a distribuição do valor a ser codificado, alguns tipo são *Unary, Truncated Unary, Exponential-Golomb*. Sendo comparado com o HEVC, são idênticos o formato do VVC com o HEVC [Bross et al. 2021].

- Modelagem de contexto

  A modelagem de contexto no VVC tem como objetivo estimar com maior precisão a probabilidade de ocorrência de cada bin, o que contribui diretamente para uma compressão mais eficiente. Esse processo é altamente dinâmico, permitindo o uso de múltiplos modelos de contexto adaptados aos diferentes tipos de dados sintáticos presentes no padrão. A atualização das probabilidades dos modelos de contexto é feita com base no comportamento dos bins previamente codificados, ajustando-se constantemente ao conteúdo da sequência de vídeo. No VVC, a variedade de elementos sintáticos e a complexidade das estruturas exigem um conjunto mais amplo e especializado de contextos, o que permite uma codificação mais precisa [Bross et al. 2021].

- Codificação Aritmética Binária

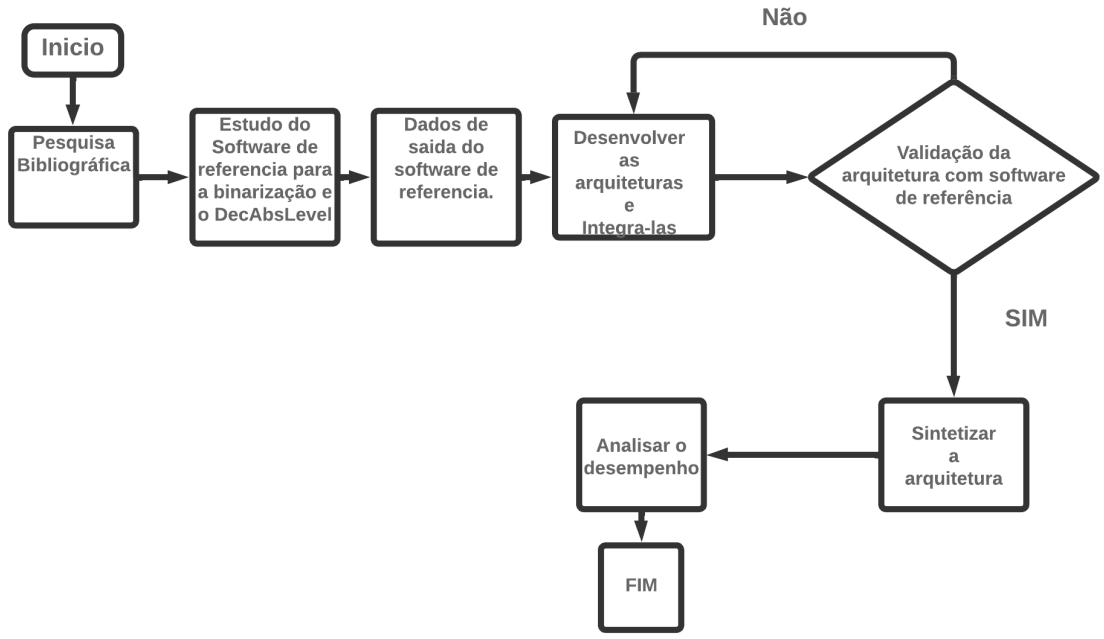
  Os bins são comprimidos em bits utilizando codificação aritmética, o que possibilita representar uma sequência de bins por uma quantidade reduzida de bits. Esse processo permite que os elementos sintáticos sejam armazenados de forma compacta, otimizando a ocupação de memória e a taxa de transmissão. A codificação aritmética opera por meio da segmentação progressiva de um intervalo contínuo entre 0 e 1. A cada bin processado, esse intervalo é particionado em duas regiões, cujas proporções são determinadas pelas probabilidades associadas fornecidas pelo modelo de contexto. A seleção do subintervalo correspondente ao valor do bin atual define a nova faixa de codificação, que será usada na próxima iteração. Ao final do processo, a posição acumulada dentro dos intervalos gera um número fracionário que representa toda a sequência de bins [Rosa 2010].

No VVC, os blocos de binarização e de modelagem de contexto foram aprimorados em relação ao HEVC, e a codificação aritmética binária passou a empregar multiplicação explícita. Além disso, foram aplicadas técnicas para reorganizar os bins e otimizar o uso do modo bypass, no qual os símbolos são codificados sem adaptação de contexto, utilizando probabilidades fixas, o que reduz a complexidade computacional em elementos sintáticos menos dependentes do contexto. A binarização e os elementos sintáticos residuais são tratados com mais detalhes no Capitulo 4.

3 MATERIAL E MÉTODOS

A metodologia adotada para o desenvolvimento deste trabalho foi estruturada em etapas bem definidas, com o objetivo de garantir uma compreensão sólida do padrão Versatile Video Coding (VVC) e sua implementação eficiente em hardware. A Figura 13 mostra as etapas em que foi estruturado o andamento do trabalho.

Figura 13 – Fluxograma das etapas da metodologia.



Fonte: (Autor,2025).

Inicialmente, foi realizado um estudo aprofundado sobre codificação de vídeo, com ênfase no funcionamento e nos princípios que regem o VVC. Essa etapa envolveu uma revisão bibliográfica abrangente, abordando conceitos fundamentais de compressão de dados, estrutura de quadros, predição intraquatros e interquadros, transformações, quantização e codificação por entropia. Além disso, foi conduzida uma análise detalhada da arquitetura e operação do VVC, com atenção especial à etapa de binarização, à estrutura do fluxo de bits e aos elementos sintáticos residuais com destaque para o DecAbsLevel, que representa coeficientes absolutos não nulos usados na reconstrução do sinal de vídeo. Essa compreensão aprofundada será essencial para a posterior modelagem e implementação em hardware, garantindo fidelidade ao padrão e eficiência na execução.

Na sequência como mostrado na Figura 13, foram realizados testes práticos

utilizando o software de referência do VVC [Joint Video Experts Team (JVET) 2020], com o intuito de compreender o funcionamento interno do processo de binarização. Essa análise foi apoiada tanto pela observação direta do comportamento do software quanto pelo estudo do documento de especificação técnica do padrão VVC, que detalha as regras e estruturas envolvidas no codec. O objetivo era entender como as variáveis relacionadas à binarização se comportam durante a codificação de diferentes blocos de dados e de que forma os elementos sintáticos residuais, como o DecAbsLevel, são processados na codificação. Foram investigadas as regras de binarização aplicadas em distintos contextos da compressão de vídeo, buscando identificar padrões consistentes que possam ser replicados de maneira eficiente em uma futura implementação em hardware.

Com base nas informações obtidas, foi projetada uma arquitetura de hardware dedicada à binarização no VVC. Esta arquitetura foi descrita em VHDL, com o objetivo de replicar o comportamento do software de referência do VVC, mas visando otimizar o tempo de execução. Como o hardware é mais rápido que o software, a implementação em circuitos integrados (ASIC) permitira uma execução mais eficiente, reduzindo a latência e acelerando o processamento em tempo real. A funcionalidade da arquitetura foi validada por meio de simulações, garantindo que os bits gerados pelo módulo de hardware correspondam exatamente aos resultados do software VVC. Além disso, foram exploradas possibilidades de otimização para aumentar a eficiência do sistema, garantindo um desempenho superior em relação às soluções baseadas em software.

Posteriormente, foi realizada a modelagem em hardware dos elementos sintáticos residuais, mais especificamente no DecAbsLevel. A implementação dessa modelagem foi feita de maneira semelhante à da binarização, utilizando a linguagem VHDL, que é amplamente empregada para descrever circuitos digitais e sistemas de hardware. Para garantir a precisão e a conformidade com o comportamento esperado, a arquitetura foi validada por meio de testes e simulações rigorosas, utilizando testbenches nos quais são aplicadas como entradas as informações obtidas a partir do software de referência. As saídas geradas pela arquitetura são então comparadas com os resultados desse software, permitindo a verificação funcional e a validação do correto funcionamento do sistema.

Após a validação, foi realizada a síntese das arquiteturas de binarização e dos elementos sintáticos residuais, visando sua implementação em circuitos integrados ASIC (Application-Specific Integrated Circuit).

A etapa final do trabalho consistiu na análise de desempenho das arquiteturas sintetizadas, utilizando o Cadence Genus. Nessa fase, foram avaliados parâmetros

cruciais como latência, consumo de energia e área, a fim de garantir que a implementação seja não apenas eficiente, mas também viável para aplicações práticas em tempo real. A adoção de hardware dedicado para a arquitetura proposta justifica-se principalmente por dois fatores, a necessidade de desempenho em tempo real, e a otimização do consumo energético. O processamento em tempo real é essencial em aplicações de codificação de vídeo, uma vez que sistemas que manipulam fluxos de vídeo continuamente requerem baixa latência e alta eficiência. A implementação em hardware proporciona aceleração significativa, garantindo que essas exigências de processamento sejam atendidas de forma consistente.

Além disso, o consumo energético é um fator crítico em dispositivos embarcados, como smartphones e outros equipamentos móveis, onde a eficiência energética é determinante. A utilização de hardware dedicado permite reduzir substancialmente o gasto de energia, contribuindo para a extensão da autonomia da bateria desses dispositivos.

Dessa forma, a implementação em hardware, quando adequadamente projetada para o algoritmo em questão, viabiliza a aceleração do processo de codificação em tempo real e pode apresentar ganhos de eficiência energética em relação a abordagens puramente em software. A avaliação desses parâmetros permitiu identificar possíveis pontos de otimização, possibilitando que trabalhos e artigos futuros utilizem esta arquitetura como referência.

4 BINARIZAÇÃO E ELEMENTOS SINTÁTICOS RESIDUAIS

Neste capitulo é apresentado o que são resíduos, elemento sintático, elemento sintático residual mais precisamente sobre o DecAbsLevel e conceitos de tipos de binarização.

4.1 Resíduos

Como mencionado brevemente no capítulo 2, os resíduos são gerados durante o processo de compressão de vídeo e representam as diferenças entre o quadro original e o quadro predito. Esses resíduos são o que sobra após a predição de uma parte da imagem, e são os dados que ainda precisam ser codificados para reconstruir o vídeo de forma eficiente. Ao processar esses resíduos, é possível obter uma compressão mais eficaz sem perder a qualidade perceptível da imagem.

Os resíduos são gerados durante a etapa de predição, que estima o quadro atual com base em quadros anteriores ou subsequentes. A diferença entre o quadro real e o quadro predito forma os resíduos. Esses resíduos são, então, submetidos a um processo de transformação, como a transformada discreta de cosseno (DCT), que converte os resíduos para o domínio da frequência. Após a transformação, ocorre a quantização, após a quantização os resíduos são denominados por outro nome, chamados de coeficientes.

4.2 Elementos sintáticos

Um elemento sintático (SE) é utilizado para representar de maneira compacta as características importantes de um quadro ou de um bloco de dados. Em vez de armazenar ou transmitir todos os dados brutos de um quadro de vídeo, como os valores dos pixels, os elementos sintáticos descrevem informações essenciais sobre os dados, como o movimento de blocos entre quadros ou os parâmetros usados na quantização, que ajudam a prever ou reconstruir a imagem.

Os elementos sintáticos são gerados nas etapas de predição, transformação e quantização, que ocorrem antes da codificação de entropia. Na predição, os elementos sintáticos podem incluir informações sobre o movimento entre quadros (como o vetor de movimento). Na transformação, eles podem envolver os coeficientes transformados.

E na quantização, os SEs descrevem os parâmetros usados para reduzir a precisão dos coeficientes. Após essas etapas, os elementos sintáticos são usados na codificação de entropia, que transforma a informação em uma sequência de bits compacta. Esses elementos são geralmente representados de forma simples, como flags binários, ou como valores compactos que descrevem parâmetros, como a quantização aplicada aos coeficientes de frequência.

Vale ressaltar como mencionado no Capitulo 2, após a transformada, os elementos sintáticos já estão organizados de forma a otimizar a codificação. Os coeficientes de maior valor ficam localizados no canto superior esquerdo do bloco, enquanto os coeficientes próximos de zero, ficam no canto inferior direito. Isso facilita a leitura e a codificação, já que a maior parte da informação útil está concentrada nos coeficientes de alta frequência no canto superior esquerdo. A leitura do bloco é feita da esquerda para a direita e de baixo para cima, como descrito na Figura 14a.

4.3 Elementos Sintáticos Residuais

Antes de abordar o conceito de elemento sintático residual (ESR) que será tratado neste trabalho, é importante entender a codificação aritmética no padrão VVC. A codificação aritmética é realizada através de dois modos distintos: modo regular e modo bypass. No modo regular, os limites dos intervalos são ajustados com base nas probabilidades adaptativas estimadas para cada bin, utilizando contextos previamente definidos. Já no modo bypass, a codificação é feita assumindo uma distribuição uniforme de probabilidade, sem o uso de modelagem de contexto, sendo aplicável a sinais binários com comportamento mais aleatório ou baixa correlação com os dados codificados anteriormente. Ambos os modos operam de maneira integrada no CABAC, sendo escolhidos conforme o tipo de elemento sintático e sua relevância para a eficiência da compressão [Martins 2011].

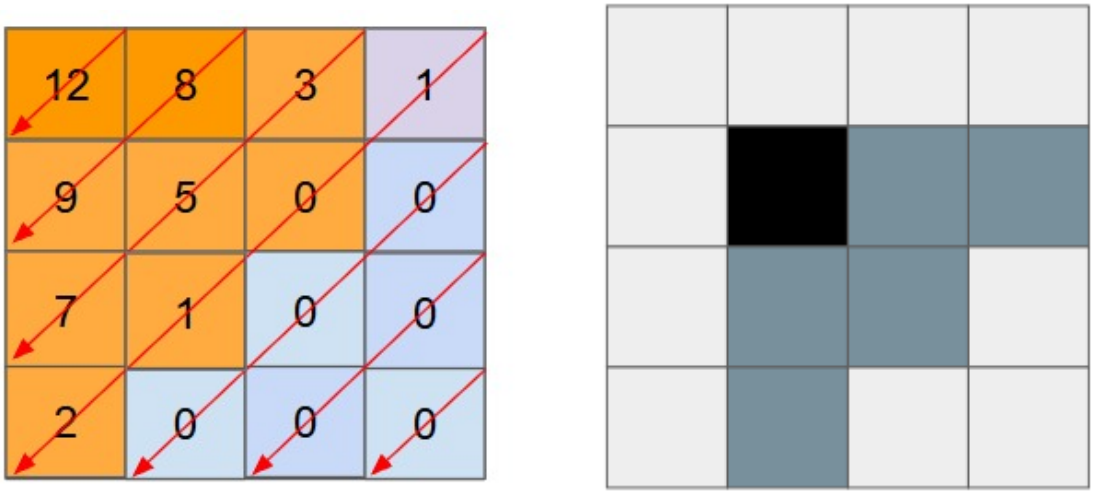
Uma etapa importante, como mencionado na sub-seção anterior, ocorre após os processos de predição, transformação e quantização: a geração dos elementos sintáticos (ESs). Esses elementos representam as decisões e dados necessários para que o decodificador consiga reconstruir o vídeo corretamente. Entre eles, os elementos sintáticos residuais (ESRs) desempenham um papel fundamental na eficiência da compressão [Cardoso et al. 2024].

Os Elementos Sintáticos Residuais (ESRs) podem ser classificados em dois tipos:

baseados em transformada (Tb) e sem transformada (Ts). Ambos os tipos seguem a granularidade dos coeficientes da Unidade de Transformação (UT), sendo geralmente processados em blocos de 4x4.

No caso dos ESRs sem transformada (Ts), ocorre o processo chamado Transform Skip, onde a etapa de transformação é intencionalmente ignorada. Esse método é aplicado em situações específicas, como blocos com baixa variação espacial ou características que não se beneficiariam da transformação. Nesses casos, os coeficientes residuais são mantidos no domínio espacial, e a codificação desses ESRs segue regras mais simples, porém com menor eficiência em termos de compressão.

No padrão VVC, existe um limite de 28 *bins* Regulares permitidos por transformada em uma UT [Schwarz et al. 2021]. Esse limite tem como objetivo equilibrar a eficiência da codificação dos *bins* Regulares e melhorar o desempenho dos *bins* Bypass. Quando esse número é excedido, nenhum outro ESR é gerado para os coeficientes restantes . Nesse ponto, um novo ESR, chamado **DecAbsLevel** (ou **dec**), entra em ação [Cardoso et al. 2024].



(a) Template e ordem de leitura utilizados na geração do Dec ESR.

(b) Padrão de varredura diagonal dos coeficientes.

Figura 14 – Template para o cálculo de S_t.

Fonte: Cardoso (2024).

A leitura dos coeficientes segue um padrão de varredura diagonal dentro da Unidade de Transformada. O processo inicia no canto inferior esquerdo do bloco e avança em direção ao canto superior direito. Cada diagonal é percorrida integralmente antes da transição para a diagonal seguinte, sempre conforme o sentido indicado pelas setas

da Figura 14a. Com os coeficientes já organizados conforme esse padrão de leitura, o **dec** utiliza os vizinhos imediatos seguindo o *template* da Figura 14b. Nesse template, o quadrado preto representa a posição do coeficiente atual, enquanto os quadrados azuis indicam as posições cujos valores absolutos devem ser somados. A soma desses coeficientes gera $\mathbf{S}_t$, utilizada para derivar a variável **m** (conforme a Equação 1), que por sua vez define $\mathbf{pos}_0$ (2) e, finalmente, o valor do **dec** (3). Além disso, a variável $\mathbf{S}_k$ representa o valor de quantização associado ao processo.

$$m = \begin{cases} 0, & S_T < 7 \\ 1, & 7 \leq S_T < 14 \\ 2, & 14 \leq S_T < 28 \\ 3, & S_T \geq 28 \end{cases} \tag{1}$$

$$pos_0 = 2^m . \begin{cases} 1, & S_k < 2 \\ 2, & S_k \geq 2 \end{cases} \tag{2}$$

$$dec = \begin{cases} pos_0, & |q| = 0 \\ |q| - 1, & 0 < |q| \leq pos_0 \\ |q|, & |q| > pos_0 \end{cases} \tag{3}$$

Neste trabalho, é abordado exclusivamente o caso dos elementos sintáticos residuais com aplicação de transformada (Tb), com ênfase no funcionamento do DecAbsLevel. Esse mecanismo é empregado na representação de coeficientes cujo valor absoluto excede os limites definidos para codificação por *bins* regulares, permitindo a continuidade do processo de codificação em conformidade com as restrições do padrão.

4.4 Binarização

A binarização é o procedimento no qual valores inteiros, resultantes de unidades sintáticas, são representados como sequências de bits denominadas binstrings. Essas representações binárias são utilizadas pelo codificador aritmético binário adaptativo baseado em contexto (CABAC), que realiza a compressão com base em modelos estatísticos condicionados ao conteúdo local. O tamanho da sequência binária resultante varia de acordo com o valor codificado e com a estrutura sintática do dado.

Para realizar essa conversão, são adotados diferentes esquemas de binarização, selecionados conforme o comportamento estatístico dos dados. Os métodos utilizados incluem: *Unary* (U), *Truncated Unary* (TU), *Fixed-Length* (FL), *Truncated Rice* (TR), *Truncated Binary* (TB) e *k-th order Exp-Golomb* (EGk). Além desses, também são empregados formatos combinados, em que diferentes métodos são aplicados separadamente ao prefixo e ao sufixo do valor binarizado, e formatos customizados, definidos por tabelas específicas para certos elementos sintáticos. A seleção do método adequado depende do tipo de dado e do seu papel na codificação.

4.5 Binarização Unary

O método de binarização unário representa valores inteiros não negativos por meio de uma sequência composta por bits de valor igual a um, finalizada por um único bit de valor zero que atua como delimitador. A quantidade de bits de valor um é proporcional ao valor numérico a ser codificado. Trata-se de um esquema de codificação de comprimento variável, no qual valores menores geram sequências mais curtas, favorecendo a compactação quando há predominância desses valores na distribuição estatística dos dados.

4.6 Truncated Unary

O método de binarização unário truncado é uma modificação do código unário tradicional, utilizado para representar valores inteiros dentro de um intervalo finito. Essa limitação é controlada por um parâmetro denominado cMax, que define o valor máximo a ser representado. A codificação consiste em uma sequência de bits de valor um, cujo comprimento é proporcional ao valor do dado, seguida ou não de um bit de terminação. Para valores estritamente menores que cMax, a sequência é encerrada com um bit de valor zero, indicando o fim da codificação. Quando o valor atinge exatamente cMax, o bit de terminação é omitido, de modo a evitar codificações mais longas do que o necessário.

4.7 Fixed-Length

O método de binarização por comprimento fixo, conhecido como *Fixed-Length*, é utilizado para representar valores inteiros dentro de um intervalo previamente definido, no qual cada valor é codificado com a mesma quantidade de bits. O número de bits necessários para representar cada valor é determinado por uma expressão matemática que depende do valor máximo a ser codificado, denotado por c_{Max}. Como todas as representações possuem o mesmo comprimento, esse método proporciona simplicidade na decodificação, acesso direto e previsibilidade no tamanho da codificação.

A quantidade de bits necessários para codificar os valores com esse método é calculada pela equação 4:

$$\text{FL} = \lceil \log_2(c_{\text{Max}} + 1) \rceil \tag{4}$$

onde FL representa o número de bits por valor e c_{Max} é o valor máximo possível dentro do intervalo considerado.

4.8 Exemplo Unario, Truncado Unario e Fixed Length

A Tabela 1 apresenta exemplos de binarização utilizando os métodos descritos anteriormente. Nessa tabela, assim como nas seguintes, o termo *N* representa o valor do elemento sintático utilizado nos exemplos.

Tabela 1 – Exemplos de binarização unário, truncado unário e tamanho fixo

N	Unário (U)	Truncado Unário (TU) ($c_{\text{Max}} = 7$)	Fixed-Length (FL) ($c_{\text{Max}} = 7$, 3 bits)
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: (Autor, 2025).

4.9 Binarização Truncado Rice

A binarização Truncated Rice representa um valor inteiro por meio da concatenação de um prefixo e um sufixo, definidos a partir de dois parâmetros: `cRiceParam` e `cMax`. O parâmetro `cRiceParam` determina a quantidade de bits menos significativos que serão isolados do valor original e codificados como sufixo. O parâmetro `cMax` define o valor máximo que pode ser representado, servindo como limite superior para a codificação.

O valor inteiro a ser codificado é representado por N. Esse valor é dividido em duas partes. A primeira parte corresponde ao quociente da divisão inteira de N por $2^{\texttt{cRiceParam}}$, sendo esse resultado codificado por meio do método unário truncado, formando o prefixo. A segunda parte corresponde aos `cRiceParam` bits menos significativos de N, os quais são codificados em formato binário de comprimento fixo, constituindo o sufixo. A junção do prefixo com o sufixo resulta na cadeia binária final [Alonso 2016].

$$\text{Prefixo: } \left\lfloor \frac{N}{2^{\texttt{cRiceParam}}} \right\rfloor \qquad \text{Sufixo: } N \bmod 2^{\texttt{cRiceParam}} \tag{5}$$

Quando `cRiceParam` é igual a zero, não há bits alocados para o sufixo, e o valor

é representado apenas pela codificação unária truncada. Nesse caso, o método Truncated Rice se torna equivalente ao método unário truncado, desde que o valor esteja contido no intervalo definido por `cMax`. A Tabela 2 nos dá um exemplo utilizando o Truncado Rice.

Tabela 2 – Exemplo de binarização Truncado Rice com $c_{\text{Max}} = 7$ e $cRiceParam = 1$.

N	Prefixo	Sufixo
0	0	0
1	0	1
2	10	0
3	10	1
4	110	0
5	110	1
6	1110	0
7	1110	1

Fonte: (Autor, 2025).

4.10 Binarização para Exp-Golomb

O método de binarização Exp-Golomb é estruturado a partir da concatenação de um prefixo e um sufixo, sendo controlado por um parâmetro k. Esse parâmetro determina o tamanho inicial do sufixo binário utilizado na codificação. Quando $k = 0$, o sufixo pode ser inexistente ou conter poucos bits, o que implica também em um prefixo mais curto. À medida que o valor a ser codificado aumenta, o sufixo requer mais bits, e o prefixo é estendido por uma sequência de bits '0', encerrada por um único bit '1', que funciona como marcador.

O sufixo representa os bits menos significativos de um valor ajustado, sendo seu comprimento diretamente relacionado à quantidade de zeros presentes no prefixo. Assim, valores menores resultam em representações binárias mais curtas, enquanto valores maiores geram cadeias mais longas.

Tabela 3 – Binarização Exp–Golomb para $k = 0$

N	Prefixo	/	Sufixo
0	–	1	–
1	0	1	0
2	0	1	1
3	00	1	00
4	00	1	01
5	00	1	10
6	00	1	11
7	000	1	000

Fonte: (Autor, 2025).

4.11 Trabalhos Correlatos

Para fundamentar teoricamente esta pesquisa, foram analisados estudos anteriores que abordam a compressão de vídeo, com ênfase nos elementos sintáticos residuais, esses elementos desempenham um papel central na eficiência do sistema de compressão, sendo diretamente afetados pelas estratégias de binarização aplicadas.

Também foram considerados trabalhos que exploram técnicas de binarização e codificação aritmética, em especial aquelas baseadas no codificador aritmético binário adaptativo baseado em contexto (CABAC). A análise desses estudos permitiu compreender a evolução dos mecanismos de compressão em padrões modernos e identificar abordagens relevantes que influenciaram o desenvolvimento deste trabalho.

A seguir, são apresentadas algumas propostas que se relacionam diretamente com os tópicos abordados nesta pesquisa e que serviram de referência para a análise realizada.

4.11.1 Trabalho Alonso

O trabalho de Alonso (2016) descreve a criação de uma arquitetura de hardware voltada para a etapa de binarização no processo de codificação do HEVC. Esse trabalho propõe o desenvolvimento de um módulo dedicado ao CABAC, com foco em otimizar

o consumo energético e reduzir a área da implementação em VHDL. A pesquisa foi centrada em como aprimorar essas métricas de desempenho sem comprometer a qualidade da codificação de vídeo.

Durante a análise dos resultados de consumo de energia, foi identificado que a técnica Operand Isolation conseguiu uma redução de até 41% no consumo de energia, enquanto operava a uma frequência de 858 MHz, em comparação a primeira arquitetura construída. Em outra configuração, utilizando a mesma técnica, mas em uma arquitetura paralela, a redução de consumo de energia foi de 37%, com a capacidade de processar até quatro elementos sintáticos simultaneamente, operando a 834 MHz. Esses resultados indicam uma excelente performance no que diz respeito à eficiência energética, alcançando os objetivos de desempenho do estudo.

Embora o foco principal do estudo tenha sido o padrão HEVC, os resultados e abordagens podem ser aplicados ao desenvolvimento de arquiteturas equivalentes no padrão VVC. Apesar das diferenças entre os dois padrões, os mecanismos de binarização compartilham princípios fundamentais, o que torna esse trabalho uma boa referência para a adaptação das técnicas de binarização em hardware no contexto do VVC, pois são as mesmas binarizações (*Unary* (U), *Truncated Unary* (TU), *Fixed-Length* (FL), *Truncated Rice* (TR) e *k-th order Exp-Golomb* (EGk). O trabalho também destaca-se pela contribuição prática na redução de consumo de potência e pela proposta de soluções viáveis para sistemas de codificação de vídeo com alto desempenho e baixo consumo energético, como é o caso de codificadores completos de vídeo que utilizam o CABAC.

4.11.2 Trabalho do Cardoso

O artigo de Cardoso et al. (2024) apresenta uma análise estatística detalhada da codificação residual e entropia no padrão VVC, com foco em orientar o desenvolvimento de arquiteturas de hardware mais eficientes para codificadores de vídeo. O estudo identifica os elementos sintáticos residuais (RSEs) como componentes críticos do fluxo de codificação, sendo responsáveis, em média, por cerca de 60% dos elementos sintáticos totais podendo ultrapassar 80% dependendo da sequência e do parâmetro de quantização (QP).

Com base nesses dados, os autores propõem uma arquitetura de hardware parcial dedicada à geração dos RSEs transformados, implementada em Verilog e sintetizada em tecnologia TSMC 40nm, sendo a primeira do tipo direcionada ao padrão VVC. A

arquitetura considera o orçamento de regular bins exigido pela especificação e adapta dinamicamente a geração dos ESRs, suspendendo parte deles quando o limite de bins é atingido.

Este trabalho é especialmente relevante, pois trata diretamente da estrutura e geração dos elementos sintáticos residuais, incluindo os mecanismos que levam à substituição dos bins regulares pelo elemento DecAbsLevel, justamente o foco desta pesquisa. O DecAbsLevel é ativado quando o limite de 28 bins regulares se esgota, conforme descrito no artigo, e passa a ser codificado como *bypass bin*, comportamento que será aprofundado e analisado no presente trabalho. Dessa forma, este artigo fornece subsídios técnicos e conceituais fundamentais para a compreensão do funcionamento, importância e implementação eficiente dos elementos sintáticos tratados neste trabalho.

4.11.3 Trabalho de Nagaraju

O artigo de Nagaraju, Gupta e Bhadauria (2022) apresenta uma arquitetura de hardware com alta taxa de transferência e eficiência em área para a etapa de binarização no padrão VVC. A proposta permite o processamento paralelo de múltiplos elementos sintáticos (SEs) por ciclo de clock, utilizando técnicas de otimização, como buffers e compartilhamento de recursos. A arquitetura foi projetada para ser escalável e adaptativa, e foi implementada tanto em FPGA quanto em ASIC, demonstrando um throughput de 3,14 Gbin/s a uma frequência de 282 MHz. A eficiência da área foi significativamente melhorada em comparação com arquiteturas de ponta, o que é um avanço importante para o processamento de vídeos em alta definição.

Este trabalho é relevante para o desenvolvimento deste TCC, pois aborda diretamente a binarização dos elementos sintáticos residuais, com destaque para o *coeff-abs-level-remaining*, um elemento associado ao DecAbsLevel no VVC. A proposta de codificação aritmética também foi otimizada utilizando uma abordagem de binarização adaptativa, que ajusta o processo de codificação com base em parâmetros como o cRiceParam.

A leitura e análise desse artigo oferecem uma compreensão detalhada dos métodos de binarização aplicados no VVC e da arquitetura proposta para a codificação eficiente. Essas informações fornecem uma base técnica sólida para o desenvolvimento do trabalho deste TCC, especialmente no que diz respeito à estrutura e codificação dos elementos sintáticos residuais no VVC, e na implementação de soluções de alto desempenho e

eficiência em área.

4.11.4 Trabalho de Tran

O artigo de Tran et al. (2020), foca no desenvolvimento de uma arquitetura de binarização eficiente para o processo de codificação de vídeo, especialmente na otimização do módulo de binarização do codificador HEVC CABAC. O objetivo principal é reduzir o consumo de energia e o custo de área, mantendo um alto desempenho em termos de throughput. A binarização dos elementos sintáticos residuais (SEs) é uma parte essencial da codificação de vídeo, principalmente para vídeos em ultra-alta definição (UHD).

Os resultados apresentados demonstram que a arquitetura proposta consegue processar 3,5 SEs por ciclo de clock a uma frequência de 500 MHz, com consumo de energia de 0,239 mW. O custo total de área é de 9,45 Kgates, sendo 6,41 Kgates destinados exclusivamente ao núcleo da binarização. Além disso, a eficiência energética é destacada, com uma taxa de 8288 Mbins/mW, o que supera outras abordagens existentes.

Embora o trabalho se concentre no padrão HEVC, muitos dos conceitos e técnicas apresentados podem ser adaptados para o VVC, uma vez que ambos os padrões compartilham semelhanças em suas arquiteturas de codificação e no processo de binarização de elementos sintáticos residuais. A pesquisa oferece uma base sólida para direcionar o desenvolvimento de uma arquitetura de binarização eficiente, que pode ser aplicada no contexto do VVC, além de se poder comparar área e eficiência energética.

4.11.5 Trabalho Gomes *et al.*

O trabalho de Gomes et al. (2023) apresenta a primeira arquitetura dedicada à geração dos elementos sintáticos residuais do codec AV1, denominada *Residual Syntax Elements Generator* (RSEG). A partir de uma análise estatística do comportamento dos coeficientes residuais e da distribuição dos tamanhos de transformadas, os autores demonstram que essa categoria de símbolos representa a maior parte do fluxo processado pelo módulo de entropia, tornando sua otimização essencial para reduzir custos computacionais e energéticos.

Com base nesses resultados, o RSEG é projetado como uma arquitetura pipeline

que opera a 1,1 GHz em tecnologia CMOS de 40 nm, com aproximadamente 6,9 k células e consumo de cerca de 10,92 mW. Os autores enfatizam que o módulo foi planejado para sustentar taxas elevadas de processamento sem comprometer área ou potência, podendo alimentar arquiteturas modernas de codificação aritmética do AV1.

A relevância desse trabalho para a presente pesquisa reside no fato de ambos explorarem a implementação eficiente de elementos sintáticos residuais em hardware, ainda que em estágios distintos do fluxo de codificação. Enquanto o RSEG gera diretamente os elementos sintáticos residuais do AV1 antes da etapa de entropia, o presente trabalho foca no elemento *decAbsLevel* do VVC, que é primeiramente reconstruído pelo bloco *Dec* e, somente então, encaminhado para os métodos de binarização do CABAC. Dessa forma, a comparação evidencia abordagens diferentes para tratar a mesma classe de informações.

4.11.6 Trabalho de Liu

O artigo de Liu et al. (2024) apresentam uma arquitetura de hardware para estimativa de taxa no padrão VVC, abordando especialmente a codificação dos coeficientes residuais. Devido ao maior número de elementos sintáticos, modelos probabilísticos mais complexos e atualizações de contexto mais frequentes, o processo de *rate estimation* no VVC apresenta fortes dependências seriais que dificultam o paralelismo em hardware.

Para enfrentar essas limitações, os autores propõem três otimizações: (i) *Localized State Update (LSU)*, que reduz as atualizações de contexto aos coeficientes de baixa frequência; (ii) *MCCB Dependency Removal (MCCBDR)*, que permite processar diferentes grupos de sintaxes em paralelo; e (iii) *Rate Estimation Table Compression (RETC)*, que diminui o tamanho da tabela de custo sem comprometer a precisão.

Com essas técnicas, a arquitetura alcança 500 MHz e codificando $7680\times4320@30$ fps, com perda de apenas 0,29% de BD-rate. Como o estudo trata diretamente de sintaxes residuais incluindo elementos como *decabslevel* oferecendo uma referência relevante para comparação com este trabalho, que aprofunda especificamente a etapa de binarização desse elemento sintático no contexto do VVC.

4.11.7 Síntese dos Trabalhos Correlatos

As propostas analisadas incluem implementações em HEVC e VVC, com foco em arquiteturas de hardware para a etapa de binarização e na análise dos elementos sintáticos residuais. A Tabela 4 apresenta uma síntese comparativa dos principais trabalhos utilizados como referência:

Tabela 4 – Comparação dos Trabalhos Correlatos

Trabalhos	**Padrão**	**Bloco de Binarização / Elementos Sintáticos**
Alonso	HEVC	Desenvolve uma arquitetura de hardware para a binarização no CABAC, reduzindo o consumo de energia em 41% a 858 MHz e 37% a 834 MHz com processamento paralelo de quatro elementos sintáticos.
Cardoso et al.	VVC	Realiza análise estatística dos elementos sintáticos residuais e propõe uma arquitetura parcial para geração de RSEs, demonstrando que esses elementos podem representar até 80% dos SEs totais.
Nagaraju et al.	VVC	Propõe uma arquitetura de alta taxa de transferência para binarização, incluindo o tratamento do *coeff_abs_level_remaining*, elemento diretamente associado ao *DecAbsLevel*.
Tran et al.	HEVC	Desenvolve uma arquitetura de binarização eficiente, capaz de processar 3,5 SEs por ciclo a 500 MHz, com consumo de 0,239 mW e área de 9,45 Kgates, destacando alta eficiência energética.
Gomes et al.	AV1	Apresenta a arquitetura RSEG para geração dos elementos sintáticos residuais do AV1, operando a 1,1 GHz em CMOS 40 nm com cerca de 6,9k células e 10,92 mW, destacando otimização para alto desempenho.
Liu et al.	VVC	Propõe uma arquitetura de estimativa de taxa para sintaxes residuais no VVC, incluindo elementos como *decAbsLevel*, alcançando 500 MHz e suporte a 8K@30fps com apenas 0,29% de perda de *BD-rate.*

5 DESENVOLVIMENTO DAS ARQUITETURAS PARA BINARIZAÇÃO E RSE ESPECIAL DO VVC

Neste capítulo é apresentada a construção da arquitetura proposta para este trabalho. O objetivo é implementar o bloco de binarização do VVC e o cálculo do elemento sintático *decAbsLevel*. Toda a fundamentação teórica e os cálculos referentes às binarizações e ao *decAbsLevel* foram discutidos no Capítulo 3, servindo de base para o desenvolvimento em hardware. O bloco do *decAbsLevel* foi implementado como uma única arquitetura, responsável por gerar o valor utilizado diretamente pelos módulos de binarização para formar o binário final e seu respectivo número de *bins*. No total, a arquitetura completa possui nove módulos: um bloco para o *dec*, sete blocos de binarização e um módulo *Analyser*, responsável por identificar o elemento sintático e escolher qual método de binarização deve ser utilizado.

O bloco de binarização desenvolvido neste trabalho opera transformando diversos elementos sintáticos em sequências de bits binários (0 ou 1), informando também o tamanho total dessa representação. Cada elemento sintático possui um método específico de binarização definido pelo padrão VVC, e cabe ao módulo *Analyser* identificar qual técnica deve ser aplicada em cada caso. Esse módulo interpreta o tipo do elemento sintático, determina o formato correspondente apresentado no Capítulo 3 e encaminha aos blocos internos todos os parâmetros necessários. Após essa etapa de análise, a arquitetura utiliza sete blocos dedicados às binarizações individuais.

Alguns métodos simples não exigiram implementação independente, pois são empregados apenas como partes internas de outras binarizações mais complexas — como ocorre com a binarização unária dentro do *Exp-Golomb* ou com o unário truncado utilizado no *Truncated Rice*. Da mesma forma, certos elementos sintáticos específicos não demandam a criação de um bloco próprio, uma vez que podem ser completamente tratados pelas binarizações já existentes. Esses casos foram incorporados às binarizações *Custom* ou absorvidos diretamente pelos módulos mais gerais, especialmente os blocos de *Truncated Binary* e *Truncated Rice*, que possuem flexibilidade estrutural para representar adequadamente tais elementos conforme as regras da norma.

A Figura 15 apresenta um *datapath* simplificado que ilustra o comportamento geral da arquitetura proposta. Nela é possível observar o fluxo entre a entrada do elemento sintático, o processamento realizado pelo *Analyser*, o roteamento para o bloco de binarização selecionado e, finalmente, a geração do código binarizado e da quantidade

final de *bins*. Vale destacar que, antes de chegar às binarizações, o elemento sintático passa por um multiplexador responsável por definir sua origem. Esse *MUX* permite escolher se o valor utilizado na binarização será aquele calculado pelo bloco *decAbsLevel* ou se será fornecido diretamente por uma entrada externa da arquitetura.

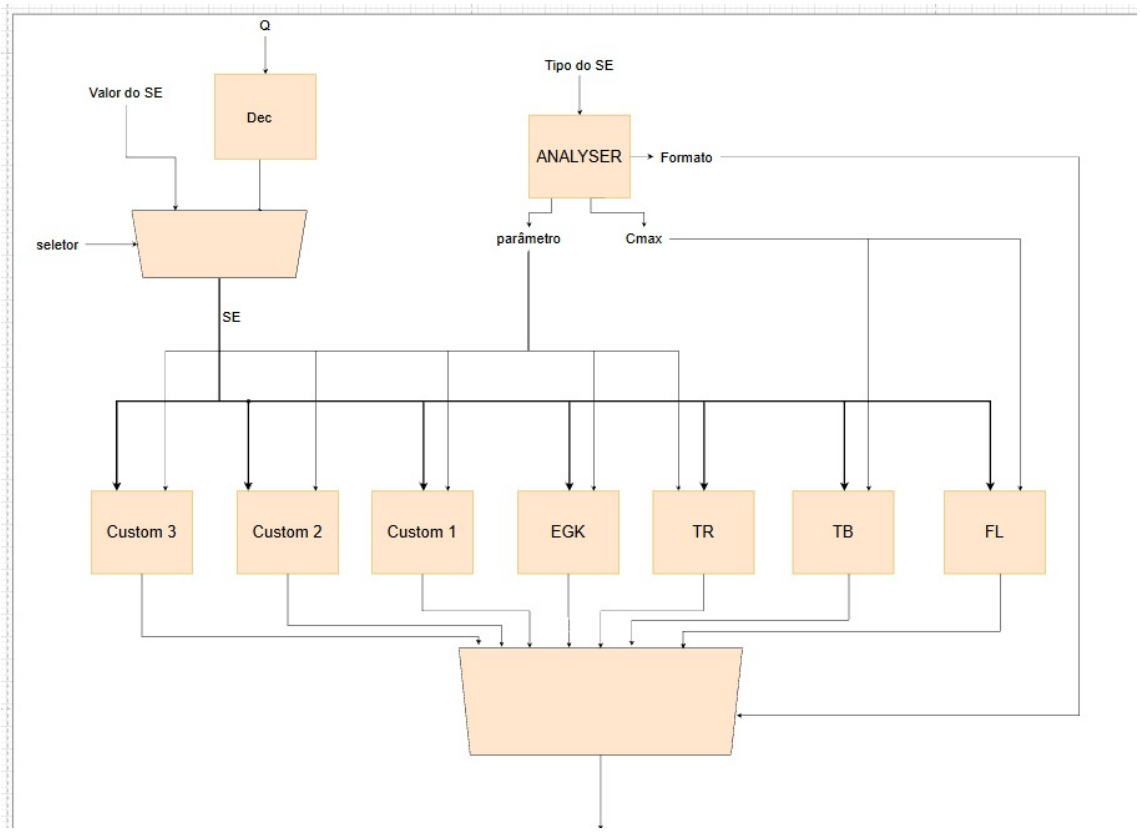


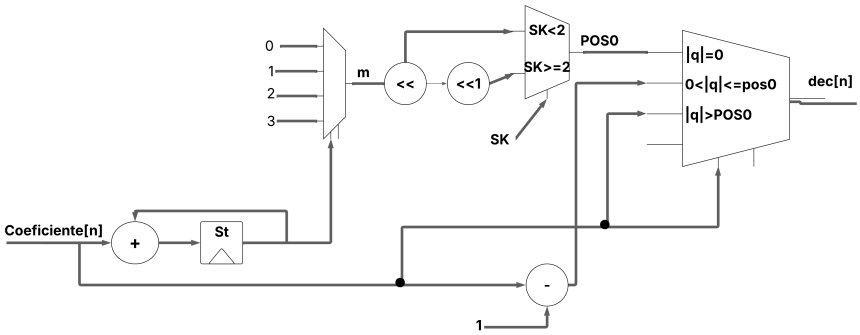
Figura 15 – Bloco topo.

Fonte: Autor (2025).

5.1 Arquitetura do Decabslevel

O bloco básico do *Dec-Tb* está ilustrado na Fig. 16. Sua ativação ocorre quando o *limite* de *bins regulares* é excedido, conforme discutido no Capítulo 4. A partir desse ponto, o processo de codificação passa a utilizar o mecanismo alternativo baseado no *DecAbsLevel*, codificado por meio de *bypass bins*.

Figura 16 – Bloco Básico do Dec-Tb



Fonte: Autor (2025).

A arquitetura responsável pela geração do **dec**, conforme definido pelas Equações 1, 2 e 3, inicia com um registrador dedicado ao armazenamento dos coeficientes necessários para o cálculo da variável S_t. Esse valor corresponde à soma descrita anteriormente e serve como entrada para o primeiro bloco lógico da arquitetura. Com o valor de S_t disponível, a Equação 1 é avaliada pelo primeiro multiplexador lógico, cuja função é determinar o valor de m de acordo com os intervalos estabelecidos na própria equação. Em seguida, a Equação 2 é aplicada em um segundo multiplexador lógico, responsável por gerar o valor de pos_0 a partir da variável de quantização S_k. Uma vez obtidos m e pos_0, o coeficiente corrente é processado utilizando seu valor absoluto. O último multiplexador lógico implementa diretamente a Equação 3, selecionando a saída final do **dec** conforme as três condições definidas para o valor absoluto do coeficiente $|q|$.

A lógica completa apresentada na Fig. 16 é replicada diversas vezes dentro da Unidade de Transformação (UT). O número de instâncias necessárias depende da quantidade de coeficientes que podem receber um valor **dec**. Neste projeto, foram implementadas dezesseis unidades independentes, cada uma responsável pelo processamento de um sub-bloco 4×4.

5.2 Binarização do CABAC

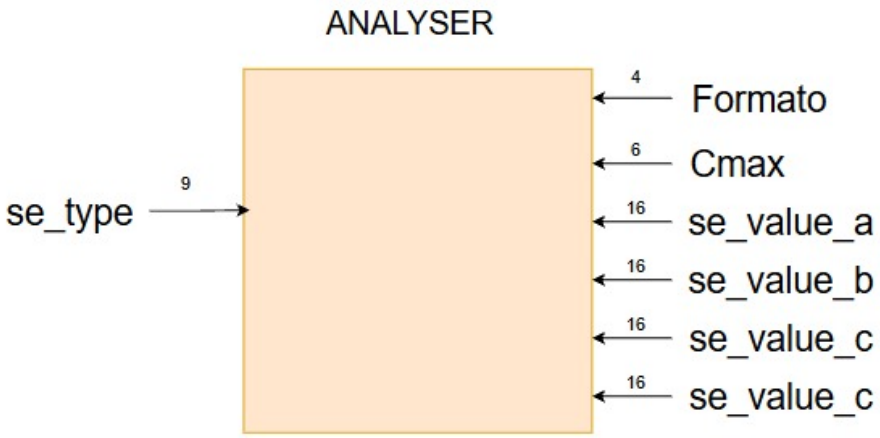
5.2.1 Bloco Analyser

O módulo Analyser tem a função de identificar qual binarização deve ser aplicada a cada elemento sintático, atuando como o bloco de controle que direciona o fluxo interno da arquitetura. Sua entrada principal é o sinal se_type, composto por 9 bits, responsável

por indicar o tipo do elemento sintático. A partir desse identificador, o Analyser consulta o mapeamento estabelecido, e determina qual método de binarização deve ser utilizado. O bloco produz um conjunto de seis saídas. A primeira delas é cMax, um sinal de 6 bits que representa o valor máximo permitido para o símbolo em binarizações que exigem essa saída, como é o caso do Truncated Binary. Em seguida, o Analyser gera o sinal formato, com 4 bits, que indica diretamente qual das binarizações implementadas será utilizada para processar o elemento sintático. Além disso, o módulo fornece quatro parâmetros auxiliares, cada um com 16 bits, que são repassados aos blocos de binarização para configurar corretamente cálculos internos, tais como larguras, limites, valores auxiliares ou parâmetros específicos, conforme cada método requer.

A Figura 17 apresenta o diagrama do módulo Analyser, destacando suas entradas, suas seis saídas e a lógica de seleção utilizada para determinar, de forma determinística, a binarização adequada e os parâmetros correspondentes para cada elemento sintático.

Figura 17 – Bloco Básico do analyser



Fonte: Autor (2025).

A entrada se_type, composta por 9 bits, é responsável por identificar qual elemento sintático está sendo processado. No total, o padrão VVC define 110 elementos sintáticos, e cada valor possível de se_type corresponde diretamente a um desses elementos. O Apêndice A apresenta a lista completa desses elementos, incluindo o formato de binarização associado e o respectivo valor de cMax utilizado quando aplicável.

A seleção do método de binarização é realizada por meio da saída formato, que possui 4 bits e representa o tipo de binarização escolhido para aquele elemento sintático. Cada combinação binária de 4 bits corresponde a um método específico entre os implementados. A Tabela 5 apresenta todos os tipos de binarização utilizados nesta

arquitetura, juntamente com sua codificação binária no sinal formato. Vale destacar que, conforme definido pela norma, os métodos Fixed-Length (FL) e Truncated Rice (TR) são os mais recorrentes ao longo da codificação dos elementos sintáticos.

Tabela 5 – Formatos de binarização utilizados na arquitetura.

Código (4 bits)	Binarização	Descrição
0001	FL	Fixed-Length
0010	TB	Truncated Binary
0011	TR	Truncated Rice
0100	EGk	Exponential-Golomb (k)
0101	Custom1	Binarização customizada 1
0110	Custom2	Binarização customizada 2
0111	Custom3	Binarização customizada 3

O sinal cMax, representado por 6 bits, é utilizado para indicar o valor máximo permitido para o símbolo em determinadas binarizações. Esse valor é encaminhado diretamente para as arquiteturas *Fixed-Length* (FL) e *Truncated Binary* (TB), nas quais o cMax é utilizado de forma fixa e não necessita de cálculos adicionais para sua determinação. Nesses casos, o Analyser simplesmente fornece o valor já definido pela norma.

Entretanto, há métodos de binarização que não utilizam um cMax fixo, exigindo cálculos intermediários para determinar seu valor final. Um exemplo é o Truncated Rice (TR), no qual o valor efetivo de cMax depende de operações realizadas a partir do símbolo de entrada e de parâmetros auxiliares. Para esses casos, o Analyser não transmite diretamente o cMax, mas sim um conjunto de quatro sinais auxiliares (se_value_a, se_value_b, se_value_c e se_value_d), cada um com 16 bits. Esses sinais fornecem às arquiteturas internas todas as informações necessárias para que o cálculo do cMax seja realizado. Para elementos sintáticos cujo cMax não é constante, adota-se sempre o pior caso de saída. Essa estratégia permite lidar de forma padronizada com elementos sintáticos que apresentam dependências aritméticas, sobretudo nas binarizações mais complexas.

A partir da análise do Apêndice A, observa-se que o valor de cMax mais recorrente entre os elementos sintáticos é “0001”, enquanto alguns valores aparecem apenas uma vez, geralmente associados a flags específicas da codificação. A Tabela 8 apresenta todos os valores possíveis de cMax com sua representação em binário. Nessa tabela,

os símbolos “«” e “»” indicam operações de deslocamento à esquerda e à direita, respectivamente, quando aplicáveis ao cálculo interno desses valores.

5.2.2 Binarizações

Foram desenvolvidos sete blocos de binarização, onde cada bloco é responsável por um método de binarização. Cada bloco recebe como entrada o sinal `se_value`, que contém o valor do elemento sintático a ser binarizado. Independentemente do método utilizado, o resultado do processo é sempre representado por duas saídas principais: `se_value_f`, um vetor de 32 bits que armazena a sequência de bins gerada, e `number_bins`, um vetor de 8 bits que informa a quantidade total de bins produzidos para aquele símbolo. A Figura 18, o padrão de entrada e saídas, utilizado nas binarizações.

Figura 18 – Bloco padrão das binarizações

Fonte: Autor (2025).

As binarizações estão agrupadas da seguinte forma. A Seção 5.2.2.1 apresenta os blocos FL, TB, TR e EGK, já a Seção 5.2.2 apresentará os blocos custom 1 e custom 2.

5.2.2.1 Blocos de binarização FL,TB,TR e EGK

Os métodos de binarização FL, TB e TR dependem do valor cMax, que define o valor máximo possível do símbolo a ser codificado e, portanto, determina o número de bins gerados. Esse parâmetro varia conforme o elemento sintático processado, se for os métodos FL ou TB, é mandado valores fixos de CMAX como mostra o Apêndice A.

O método FL (*Fixed-Length*) define o tamanho máximo do *binstring* a partir da Equação 4. Vale ressaltar que o valor de cMax para o FL é sempre fixo. Portanto, a saída cMax do bloco Analyser é repassada diretamente ao bloco FL, que utiliza esse parâmetro para determinar o comprimento da representação binária do símbolo. Para os métodos em que cMax depende de um cálculo dinâmico, adota-se o pior caso desses valores, obtido a partir das saídas `se_value_a` e `se_value_b` geradas pelo Analyser. Nesses cenários, a saída cMax do Analyser não é utilizada, uma vez que ela se aplica apenas aos casos em que o valor é fixo. Assim, os sinais `se_value_a` e `se_value_b` são diretamente encaminhados aos blocos de binarização correspondentes, permitindo que cada método obtenha internamente o valor de cMax necessário ao seu processamento.

O processo de binarização *Truncated Binary* (TB), é aplicado a elementos sintáticos utilizando como entrada o valor `se_value` e o valor máximo permitido `cMax`. Esse método emprega o processo *Fixed-Length* (FL) de forma condicionada, utilizando dois comprimentos possíveis de binarização, definidos a partir de `cMax`.

Os valores auxiliares utilizados para determinar o tamanho do binário gerado são calculados conforme as equações 6, 7, 8:

$$n = cMax + 1 \tag{6}$$

$$k = \lfloor \log_2(n) \rfloor \tag{7}$$

$$u = 2^{(k+1)} - n \tag{8}$$

Com base nesses valores, o método TB define a binarização da seguinte forma:

- Se `se_value` $< u$, a codificação é obtida aplicando o método FL com k bits, e limite $(2^k) - 1$.
- Se `se_value` $\geq u$, aplica-se o método FL com $k+1$ bits, codificando o valor `se_value` $+u$, com limite $(2^{(k+1)}) - 1$.

Ao final do processo, o bloco de binarização TB produz as saídas `se_value_f` (32 bits), contendo a cadeia de bins resultante, e `number_bins` (8 bits), indicando a quantidade total de bins gerados.

A binarização *Truncated Rice* (TR) depende de três entradas: o valor sintático a ser binarizado (*se_value*), o parâmetro Rice (*goRicePar*), que determina o tamanho do

sufixo, e o valor de truncamento ($cutoff$), responsável por delimitar a região truncada. Esses parâmetros não são fixos no padrão VVC: diferentes elementos sintáticos utilizam combinações distintas de *goRicePar* e $cutoff$. Contudo, segundo a norma, a maior parte dos elementos sintáticos que utilizam TR adota $goRicePar = 0$, valor aplicado a todos os elementos não-customizados. Nesses casos, a TR reduz-se ao comportamento de um código unário truncado, pois não há sufixo a ser sinalizado. Por outro lado, alguns elementos sintáticos customizados utilizam valores de $goRicePar > 0$ e, em certos casos, valores de $cutoff$ mais elevados, exigindo a aplicação completa do processo TR. Para que a arquitetura de hardware seja capaz de atender simultaneamente a todos esses cenários, é necessário dimensionar o bloco TR considerando o pior caso entre esses parâmetros. Dessa forma, a lógica implementada torna-se capaz de reproduzir todas as combinações previstas pela norma, garantindo que a saída codificada seja correta independentemente do elemento sintático processado.

A escolha do formato binário empregado é determinada pela comparação entre o valor de entrada e um limite denominado *threshold*, calculado como

$$threshold = cutoff \cdot 2^{goRicePar}. \tag{9}$$

Com base nesse limite, a binarização TR apresenta dois comportamentos distintos:

1. **Região truncada**: ocorre quando conforme a Equação 10

$$se\_value < threshold. \tag{10}$$

Nessa condição, é gerado um prefixo construído por uma sequência de bits `1` seguida por um único bit `0`, compondo uma binarização unária. O número de bits `1` corresponde ao quociente de *se_value* dividido por $2^{goRicePar}$. Em seguida, são acrescentados *goRicePar* bits de sufixo, que representam os bits menos significativos de *se_value*. Assim, a *binstring* resultante é dada pela Equação 11.

$$TR(se\_value) = \underbrace{\texttt{11...110}}_{\text{prefixo unário}} \parallel FL(suffixVal), \tag{11}$$

em que o valor do sufixo é definido pela Equação 12.

$$suffixVal = se\_value - \left(\left\lfloor \frac{se\_value}{2^{goRicePar}} \right\rfloor \cdot 2^{goRicePar} \right). \tag{12}$$

2. **Região não truncada**: ocorre conforme a Equação 13.

$$se\_value \geq threshold. \tag{13}$$

Nessa condição, o código inicia com um *prefixo saturado*, formado por uma sequência de bits `1` cujo comprimento corresponde ao limite máximo permitido pela região truncada. Esse prefixo indica que o valor ultrapassou o intervalo em que o código unário é suficiente.

Após o prefixo saturado, a TR ainda precisa representar a parte excedente do valor de entrada, isto é, o quanto *se_value* ultrapassa o limite definido por $threshold$. Essa parcela excedente é convertida em uma pequena sequência de bits, cujo tamanho aumenta conforme o valor de entrada cresce. Esse processo ocorre inclusive quando $goRicePar = 0$, pois mesmo nesse caso a TR completa continua a codificar o excedente.

A arquitetura implementa o código Exponential-Golomb de ordem k (EGk), no qual o valor de entrada *se_value* é convertido em um *binstring* formado por um prefixo unário e um sufixo de k bits. O valor inicial de k é fornecido pelo bloco *Analyser*, sendo necessário suportar os casos $k = 0$ e $k = 5$.

O processo inicia verificando quantos múltiplos de 2^k podem ser removidos do valor de entrada. Enquanto, conforme a Equação 14.

$$se\_value \geq 2^k, \tag{14}$$

um bit "1" é adicionado ao prefixo e subtrai-se 2^k do valor. Quando a condição deixa de ser satisfeita, o prefixo é finalizado com um bit "0". Em seguida, o valor restante é representado utilizando exatamente k bits, formando o sufixo. A estrutura final do código consiste em um prefixo unário seguido dos k bits do sufixo.

De momento, o formato obtido não se assemelha diretamente ao apresentado na Tabela 3 apresentada no Capítulo 4, pois ali é descrito apenas o funcionamento conceitual do Exp-Golomb. Já o método EGk utilizado aqui segue a forma prática de binarização empregada na arquitetura, resultando em códigos equivalentes porém organizados de forma diferente, conforme mostra na Tabela 6.

Tabela 6 – Exemplos de binarização EGk.

se_value	*k*	Resultado (EGk)
0	0	0
1	0	10
2	0	110
3	0	1110
0	5	000000
1	5	000001
5	5	000101
16	5	010000
32	5	1000000
40	5	1001000
64	5	11000000

5.2.2.2 Blocos de binarização Custom 1 e Custom 2 e Custom 3

O elemento sintático *intra_chroma* utiliza uma binarização do tipo **Custom 1**. Nesse método, cada valor possível do elemento sintático é associado diretamente a uma sequência binária fixa, sem cálculos adicionais ou uso de parâmetros. Assim, o processo consiste apenas em mapear o valor de entrada para o seu respectivo *binstring*, conforme apresentado na Tabela 7.

Tabela 7 – Processo de binarização do *intra_chroma_pred_mode*.

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

Fonte: ITU-T, 2021.

O método de binarização aplicado ao elemento sintático *inter_pred_idc* é do tipo **Custom 2**. Nesse método, a seleção do *binstring* depende das dimensões atuais do bloco de codificação luma, utilizando os valores *cbWidth* e *cbHeight*, os quais são fornecidos como entradas da arquitetura. A binarização consiste em mapear diretamente o valor de entrada de *inter_pred_idc* para uma sequência binária, aplicando-se uma condição sobre a soma $(cbWidth + cbHeight)$. Quando $(cbWidth + cbHeight) > 12$, utiliza-se uma representação de dois bits. Para blocos em que $(cbWidth + cbHeight) = 12$, utiliza-se

uma representação de apenas um bit. A Tabela 8 apresenta os mapeamentos definidos para esse método de binarização Custom 2.

Tabela 8 – Binarização do elemento sintático *inter_pred_idc*.

Valor de *inter_pred_idc*	**($cbWidth + cbHeight$ > 12)**	**($cbWidth + cbHeight$ = 12)**
0	00	0
1	01	1
2	1	–

Fonte: ITU-T, 2020.

O método de binarização **Custom 3**, aplicado ao elemento sintático *cu_qp_delta_abs*, combina dois processos distintos e é estruturado pela concatenação de um **prefixo** e um **sufixo**. Para valores do elemento sintático menores ou iguais a cinco, a binarização é inteiramente realizada pelo método **TR**, conforme apresentado na Tabela 9. Para valores superiores a cinco, utiliza-se um formato híbrido: primeiro é emitido um prefixo fixo formado pelo padrão `11111`, e em seguida aplica-se o método **EG0 (Exp-Golomb com** $k = 0$**)** para gerar o **sufixo**, utilizando como entrada o termo $(se\_value - 5)$. Dessa forma, a binarização produzida pelo Custom 3 consiste em um prefixo truncado para valores pequenos e, para valores maiores, em um prefixo fixo seguido pelo sufixo obtido por meio de EG0 (conforme definido na Equação 15).

$$\text{bin}(se\_value) = \begin{cases} \text{UnaryTrunc}(se\_value), & se\_value \leq 5 \\ 11111 \,\|\, EG0(se\_value - 5), & se\_value > 5 \end{cases} \tag{15}$$

onde $EG0$ corresponde ao processo de binarização Exp-Golomb com $k = 0$.

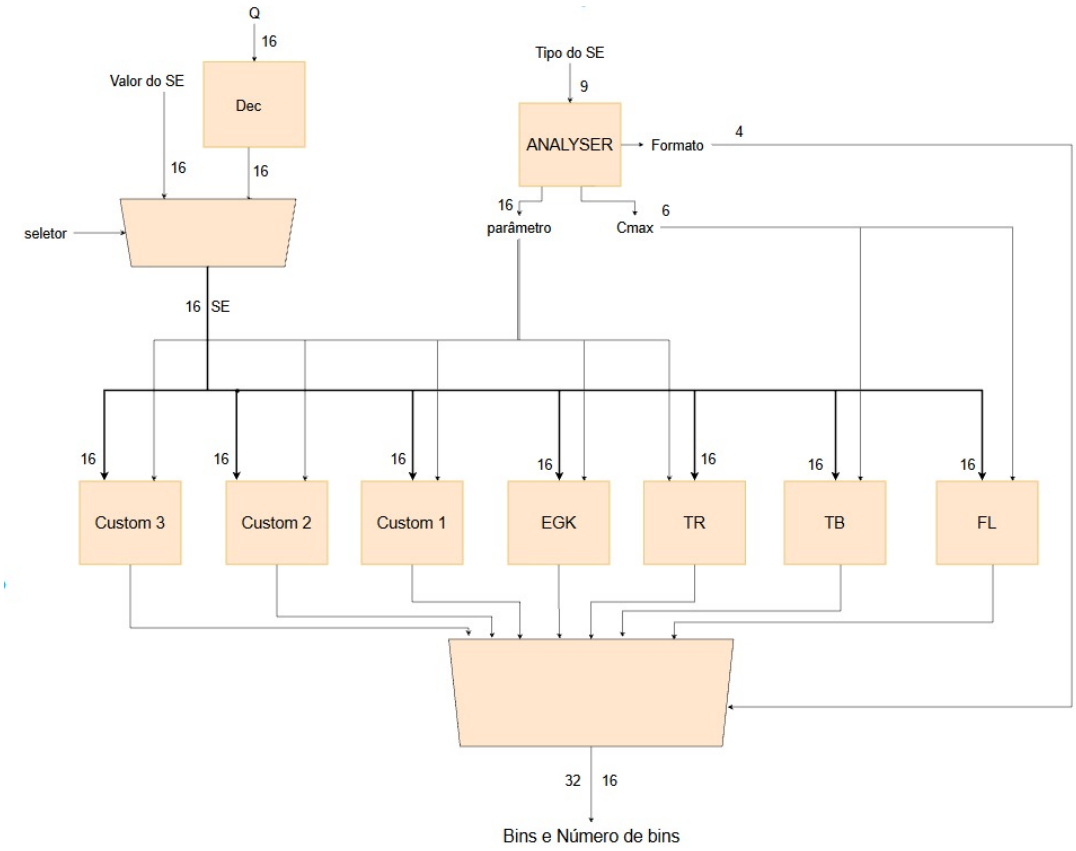
Tabela 9 – Processo de binarização do Custom 3 (*cu_qp_delta_abs*).

Intervalo	*x* **(EG0)**	**Binário Final**
0	–	0
1	–	10
2	–	110
3	–	1110
4	–	11110
5	–	11111
6–7	0	111111010
8–11	00	11111100100
12–19	000	1111110001000
20–35	0000	111111000010000

5.3 Arquitetura Completa e Tamanhos dos Barramentos

A Figura 19 apresenta uma variação da arquitetura completa do núcleo de binarização, utilizada exclusivamente para evidenciar o tamanho dos barramentos de entrada e saída. A estrutura funcional do núcleo já foi apresentada anteriormente; na Figura 19 o objetivo é apenas detalhar a largura de cada sinal que compõe o caminho de dados.

Figura 19 – Tamanhos dos barramentos das entradas e saídas do núcleo de binarização.



Fonte: (Autor, 2025).

Nesta representação, destaca-se que o valor sintático (*se_value*) possui largura de 16 bits, enquanto o tipo do elemento sintático (*se_type*) é composto por 9 bits. As saídas geradas pelos blocos de binarização também seguem a padronização definida anteriormente: o vetor codificado (*se_value_f*) é composto por 32 bits, e o número de *bins* produzidos é representado por um barramento de 8 bits. Esses tamanhos refletem diretamente o dimensionamento necessário para garantir compatibilidade entre o *Analyser*, os blocos de binarização e o multiplexador final.

Para validar o funcionamento da arquitetura e coletar dados reais de cada processo de binarização, foram geradas sequências de teste utilizando a sequência *PartyScene* (resolução 1080p) [F. Bossen, J. Boyce, K. Suehring, X. Li, V. Seregin 2020]. A codificação foi realizada empregando o perfil *Low Delay*. A quantização foi configurada com valores de QP variando de 10 até 22. A escolha de QPs mais baixos teve como objetivo estimular o acionamento de um conjunto mais amplo de funções internas do codificador de referência do VVC, permitindo observar o comportamento de múltiplos elementos sintáticos ativados em diferentes condições do algoritmo CABAC. Esses

valores mais reduzidos aumentam a quantidade de detalhes preservados na codificação e, consequentemente, a diversidade de caminhos percorridos pelo codificador, favorecendo a análise detalhada das binarizações produzidas.

Essa estratégia possibilitou capturar o funcionamento completo de todos os métodos de binarização implementados, além de permitir validar o fluxo de ativação entre o *Analyser* e cada bloco específico.

6 RESULTADOS E DISCUSSÕES

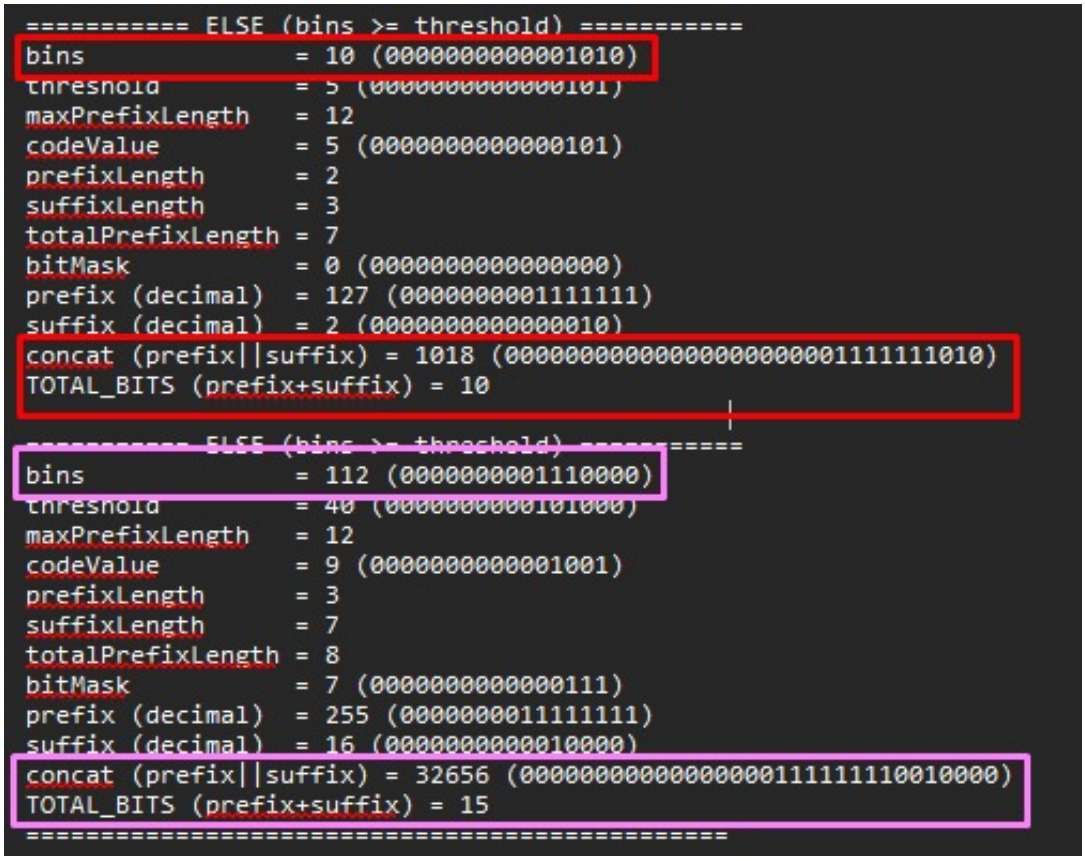
Este capítulo apresenta os resultados de simulação, validação e síntese da arquitetura desenvolvida. A validação funcional em VHDL foi realizada utilizando o *ModelSim* (Mentor Graphics) [Siemens EDA 2024], permitindo verificar o comportamento lógico dos blocos do DEC, binarização e *Analyser*. A etapa de síntese foi conduzida com o *Genus™ Synthesis Solution* [Cadence Design Systems, Inc. 2025], utilizado para estimar área e desempenho da arquitetura em um fluxo direcionado à síntese ASIC. As seções seguintes detalham os resultados obtidos em cada uma dessas etapas. A Seção 6.1 aborda os resultados de simulação e validação da arquitetura, enquanto a Seção 6.2 apresenta os resultados obtidos no processo de síntese.

6.1 Simulação e Validação da Arquitetura

A validação da arquitetura foi conduzida de forma progressiva, iniciando pela análise individual de cada bloco que compõe o sistema. Para isso, foram extraídos diretamente do código de referência do VVC os valores necessários para a verificação de cada método de binarização, o mesmo para validação do bloco *Dec*. A partir desses valores, tornou-se possível compreender detalhadamente o funcionamento interno de cada módulo e estabelecer um processo de validação fiel ao comportamento especificado no padrão.

Com base nesse entendimento, foram desenvolvidos testbenches específicos em VHDL para cada bloco, permitindo a aplicação controlada de entradas e a comparação direta das saídas geradas pela arquitetura com aquelas produzidas pelo software de referência. Esse procedimento assegurou que cada módulo funcionasse de maneira isolada e com total aderência à especificação. A Figura 20 apresenta um exemplo dos valores extraídos do VVC e utilizados como referência nos testes das arquiteturas.

Figura 20 – Resultados do Bloco TR no VVC



Fonte: (Autor, 2025).

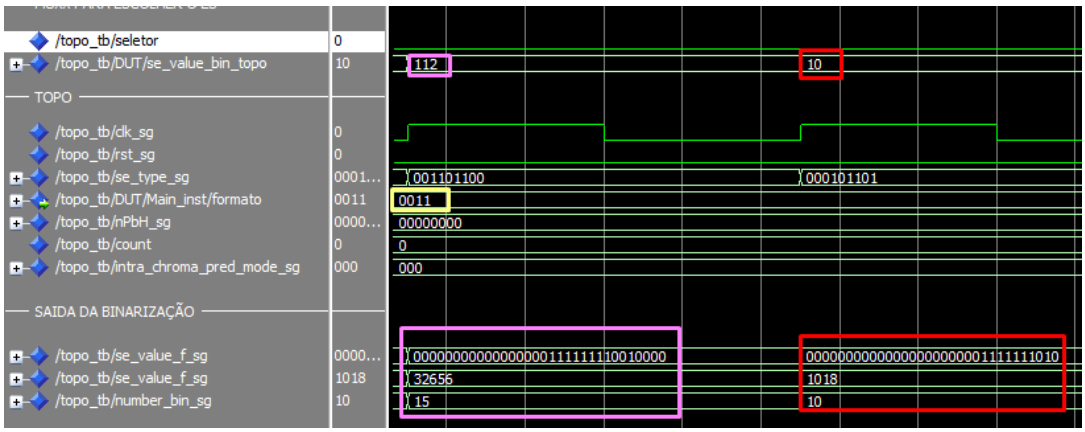
Após a validação unitária, os módulos foram integrados, formando a arquitetura completa. Para a avaliação desse cenário, foi desenvolvido um novo testbench dedicado à verificação do funcionamento conjunto dos blocos. Essa etapa foi essencial para identificar possíveis problemas de sincronização, atrasos indesejados ou incompatibilidades de comunicação entre módulos. Os resultados demonstraram que a integração não introduziu falhas e que o comportamento funcional global permaneceu consistente com o esperado. Parte dessa análise pode ser observada na Figura 21.

A Figura 21 apresenta uma simulação realizada no *ModelSim*. Nela, os sinais destacados em rosa correspondem aos valores de *se_value*, *se_value_f* e *number_bins*, lidos de cima para baixo, e estão referenciados pelos mesmos tons presentes na Figura 20. De maneira equivalente, os sinais destacados em vermelho correspondem aos valores mostrados em vermelho no conjunto de testes da Figura 20, permitindo a conferência direta entre entradas, saídas e o binário gerado pela arquitetura. Na Figura 21, há dois sinais associados a *se_value_f*, um exibido em formato binário, e outro em formato

decimal, facilitando a interpretação dos resultados.

Por fim, a região destacada em amarelo na Figura 21 indica o sinal *formato*. No exemplo apresentado, o valor binário ‘0011‘ identifica o método *TR* (*Truncated Rice*), confirmando que a arquitetura selecionou corretamente o formato de binarização para o elemento sintático em questão.

Figura 21 – Simulação e validação da arquitetura



Fonte: (Autor, 2025).

6.2 Síntese da Arquitetura

A síntese da arquitetura foi realizada utilizando a ferramenta *Cadence Genus*. O projeto foi implementado em tecnologia **45 nm**, operando com tensão de **1,08 V** e temperatura de **125 °C**. A partir dos resultados obtidos, verificou-se que a frequência máxima de operação alcançada pela arquitetura é de aproximadamente **398 MHz**, valor correspondente ao período crítico de **2,513 ns**, determinado pela cadeia combinacional responsável pela geração do vetor de saída *se_value_f*. No que se refere à ocupação de área, a síntese resultou em aproximadamente **18.170 gates**. O consumo energético estimado foi de aproximadamente **4,25 mW**, sendo a maior parcela proveniente da lógica combinacional, seguida pelos registradores e, em menor proporção, pela rede de distribuição de clock.

A Tabela 10 apresenta de forma consolidada todas as informações mencionadas nesta seção.

Tabela 10 – Resultados de síntese da arquitetura proposta.

Parâmetro	**Valor**
Tecnologia	45 nm (1.08 V, 125°C)
Frequência máxima	398 MHz
Período crítico	2.513 ns
Gate count	18,170 gates
Área total	18,641.770 unidades de área
Potência total	4.25 mW
Throughput	1-SE/cycle

6.3 Comparação com trabalhos correlatos

Os trabalhos apresentados na Seção 3.5, foram utilizados como referência para o desenvolvimento deste trabalho, conforme mencionado. A Tabela 11 apresenta uma comparação entre os trabalhos pesquisados na literatura, considerando apenas aqueles que contêm resultados para o bloco de binarização e elementos sintáticos residuais.

Tabela 11 – Resultados de síntese e comparação com trabalhos relacionados.

Autor / Trabalho	**Tecnologia**	**Padrão**	**Frequência**	**Área**	**Throughput**
Alonso	65 nm	HEVC	834 MHz	11.85 kgates	4 SE/ciclo
Tran	45 nm	HEVC	500 MHz	9.45 kgates	3.5 SE/ciclo
Gomes, J	40 nm	AV1	1.1 GHz	6.93 kgates	1 Coeff/ciclo
Nagaraju	90 nm	AVC/HEVC/VVC	282 MHz	7.09 kgates	5 SE/ciclo
LIU C HUANG	28 nm	VVC-based	500 MHz	52.03 kgates	12 Coeff/ciclo
Cardoso	40 nm	VVC	1 GHz	8.98 kgates	1 Coeff/ciclo
Este trabalho	45 nm	VVC	**398 MHz**	**18.17 kgates**	**1 SE/ciclo**

A partir dos resultados apresentados na Tabela 11, é possível comparar de forma direta as arquiteturas da literatura com a solução desenvolvida neste trabalho. Entre os projetos dedicados exclusivamente à etapa de binarização, os trabalhos de Alonso e Tran apresentam áreas de 11,85 kgates e 9,45 kgates, respectivamente, com throughputs de 4 SE/ciclo e 3,5 SE/ciclo. Esses resultados refletem o uso de paralelismo interno e o escopo restrito à binarização no contexto do HEVC. Em contraste, a arquitetura proposta processa 1 SE/ciclo de forma sequencial e possui 18,17 kgates, valor superior devido ao conjunto mais amplo de módulos implementados, que inclui tanto a etapa residual parcial

quanto todas as binarizações do VVC.

No processamento residual, os trabalhos de Cardoso e LIU constituem as referências mais adequadas, pois se tratam de elementos sintáticos residuais. A arquitetura do Cardoso apresenta área de 8,98 kgates e frequência de 1 GHz, operando de forma sequencial a 1 Coeff/ciclo. Já LIU emprega uma arquitetura paralela, alcançando 12 Coeff/ciclo ao custo de 52,03 kgates. O trabalho de Nagaraju, por sua vez, obtém 5 SE/ciclo com área de 7,09 kgates e 282 MHz, resultado de um escopo reduzido de símbolos e simplificações aplicadas ao fluxo residual. Em relação a esses trabalhos, a arquitetura desenvolvida neste projeto apresenta área e complexidade intermediárias, mantendo operação sequencial.

O trabalho de Gomes, focado no processamento residual do AV1, opera a 1,1 GHz com área de 6,93 kgates e throughput de 1 Coeff/ciclo. Esses valores resultam de um escopo específico e otimizações direcionadas ao conjunto reduzido de símbolos tratados pelo AV1, permitindo atingir alta frequência com baixo custo de área. Em comparação ao presente trabalho, o projeto de Gomes apresenta menor ocupação de hardware e frequência superior devido à menor complexidade funcional, enquanto a arquitetura desenvolvida aqui abrange um conjunto mais amplo de operações relacionadas ao fluxo residual e à binarização do VVC.

Diante das comparações realizadas, verifica-se que a arquitetura desenvolvida neste trabalho apresenta vantagens e limitações claras em relação às soluções da literatura. Em termos de área, permanece mais compacta que arquiteturas altamente paralelas, como a de LIU, e próxima de projetos sequenciais mais simples, como o Cardoso. A área superior observada decorre diretamente do fato de a arquitetura integrar, em um único bloco, a etapa residual parcial (DecAbsLevel) e todos os métodos de binarização, funcionalidade não implementada de forma conjunta nos demais trabalhos. No que se refere à frequência, obtém valores inferiores às arquiteturas mais otimizadas e específicas. Ainda assim, mesmo operando de forma sequencial, mantém 1 SE/ciclo de maneira estável, oferecendo um compromisso equilibrado entre área e complexidade.

7 CONSIDERAÇÕES FINAIS

A codificação de vídeo destaca-se atualmente como um dos campos de pesquisa mais relevantes dentro dos sistemas digitais. O crescimento contínuo do uso de plataformas de comunicação, entretenimento e transmissão impulsiona a demanda por codecs mais eficientes, tornando esse domínio estratégico tanto para a indústria quanto para a academia. Nesse cenário, compreender e aperfeiçoar os módulos internos de codecs modernos, como o VVC, é essencial para sustentar avanços em desempenho, eficiência e redução de complexidade.

Este trabalho apresentou uma análise detalhada do funcionamento do VVC, com foco na etapa de codificação por entropia realizada pelo módulo CABAC. A pesquisa concentrou-se em duas partes fundamentais desse processo. A primeira corresponde ao módulo *Dec*, responsável por executar parte da codificação aritmética e gerar o elemento sintático *decAbsLevel*, que representa o valor absoluto dos coeficientes residuais após a transformada. A segunda etapa refere-se à binarização, na qual os elementos sintáticos produzidos pelo *Dec* ou fornecidos externamente são convertidos em sequências de bins conforme os formatos definidos pelo padrão, conforme descrito no Apêndice A.

Com esse objetivo, foram desenvolvidas duas arquiteturas específicas. A primeira implementa o cálculo completo do *decAbsLevel*, reconstruindo o valor residual necessário para o fluxo de entropia. A segunda realiza a binarização de todos os cento e dez elementos sintáticos definidos pelo VVC, selecionando dinamicamente o formato correto para cada caso. Essas duas arquiteturas foram posteriormente integradas, formando um sistema único capaz de receber um elemento sintático, processá-lo no módulo *Dec* e encaminhá-lo ao módulo de binarização de forma contínua e sincronizada. Toda a modelagem foi descrita em VHDL e sintetizada utilizando a ferramenta Cadence Genus em um fluxo direcionado à implementação em ASIC.

Os resultados de síntese mostram que a arquitetura proposta apresenta desempenho estável e área compatível com o conjunto de funções implementadas. O projeto alcançou **18,170 gates** e **18,641.770 unidades de área**, valor naturalmente mais elevado devido à integração, em um único bloco, da etapa residual parcial (*decAbsLevel*) com todos os métodos de binarização do VVC. A frequência máxima obtida, **398 MHz** (período crítico de **2,513 ns**), assegurou operação sequencial em **1 SE/ciclo**, mantendo o fluxo interno correto em todas as simulações. Com potência de **4,25 mW**, a arquitetura demonstra viabilidade prática e um equilíbrio adequado entre área, funcionalidade e

simplicidade estrutural.

Conclui-se, portanto, que a arquitetura desenvolvida cumpre plenamente seu objetivo de integrar o processamento do *decAbsLevel* e a etapa de binarização em uma única solução de hardware. Os resultados confirmam que é possível obter estabilidade e eficiência mesmo reunindo essas funções em um único bloco, contribuindo para avanços nas implementações dedicadas ao VVC e reforçando a importância do estudo detalhado dos componentes internos do CABAC.

Tabela 12 – Exemplos de binarização TB com $cMax = 61$. Parâmetros: $n = 62$, $k = 5$, $u = 2$.

se_value	**Comparação com** $u = 2$	**Valor codificado**	**Bits usados**	**Binário FL**
1	$1 < 2$	1	5 bits	00001
2	$2 \geq 2$	$2 + 2 = 4$	6 bits	000100
3	$3 \geq 2$	$3 + 2 = 5$	6 bits	000101
4	$4 \geq 2$	$4 + 2 = 6$	6 bits	000110
16	$16 \geq 2$	$16 + 2 = 18$	6 bits	010010
32	$32 \geq 2$	$32 + 2 = 34$	6 bits	100010
64	$64 \geq 2$	$64 + 2 = 66$	6 bits*	1000010
128	$128 \geq 2$	$128 + 2 = 130$	6 bits*	10000010

REFERÊNCIAS

AGOSTINI, L. V. Desenvolvimento de arquiteturas de alto desempenho dedicadas à compressão de vídeo segundo o padrão h. 264/avc. 2007.

ALONSO, C. d. M. Desenvolvimento de uma arquitetura em hardware do bloco de binarização do cabac baseado no padrão hevc. Universidade Federal do Pampa, 2016.

BROSS, B. et al. Developments in international video coding standardization after avc, with an overview of versatile video coding (vvc). **Proceedings of the IEEE**, IEEE, v. 109, n. 9, p. 1463–1493, 2021.

BROSS, B. et al. Overview of the versatile video coding (vvc) standard and its applications. **IEEE Transactions on Circuits and Systems for Video Technology**, IEEE, v. 31, n. 10, p. 3736–3764, 2021.

BRUNO, G. G. E. Vebit: um algorítmo para codificação de vídeo com escalabilidade. 2003.

Cadence Design Systems, Inc. **Genus™ Synthesis Solution**. [S.l.], 2025. Ferramenta de síntese lógica para projetos digitais ASIC e FPGA. Disponível em: https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/synthesis/genus-synthesis-solution

CARDOSO, G. B. et al. Statistical analysis of vvc residual and entropy coding aiming efficient hardware design. In: **2024 IEEE 15th Latin America Symposium on Circuits and Systems (LASCAS)**. [S.l.: s.n.], 2024. p. 1–5.

DINIZ, C. M. Arquitetura de hardware dedicada para a predição intra-quadro em codificadores do padrão h. 264/avc de compressão de vídeo. 2009.

DUARTE, A. I. R. **Redução de complexidade do processo de decisão de modo da predição intra-quadro do codificador de vídeo VVC utilizando aprendizado de máquina**. Dissertação (Mestrado) — Universidade Federal de Pelotas, 2021.

F. Bossen, J. Boyce, K. Suehring, X. Li, V. Seregin . **VTM common test conditions and software reference configurations for SDR video**. [S.l.], 2020.

FRAGA, L. M. L. d. Acelerador de hardware com arquitetura de módulos especializados para predição intra-quadro do versatile video coding. 2023.

GOMES, J. S. et al. AV1 Residual Syntax Elements Assessment and Efficient VLSI Architecture. In: **2023 36th SBC/SBMicro/IEEE/ACM Symposium on Integrated Circuits and Systems Design (SBCCI)**. [S.l.: s.n.], 2023. p. 1–6.

GONÇALVES, P. H. R. **Um esquema rápido baseado em aprendizado de máquina para a predição interquadros do codificador de vídeo VVC**. Dissertação (Mestrado) — Universidade Federal de Pelotas, 2021.

GONZALEZ, R. C.; WOODS, R. E. **Processamento de imagens digitais**. [S.l.]: Editora Blucher, 2000.

ITU-T; ISO/IEC. **Infrastructure of audiovisual services – Coding of moving video (H.266)**. [S.l.], 2020.

Joint Video Experts Team (JVET). **VVC Test Model (VTM) Reference Software**. 2020. <https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM>. Acesso em: ago. 2025.

LEE, J.; JEONG, J. Deblocking performance analysis of weak filter on versatile video coding. **Electronics Letters**, Wiley Online Library, v. 56, n. 6, p. 289–290, 2020.

LIU, C. et al. Hardware implementation of a high-accuracy and high-throughput rate estimation unit for vvc residual coding. **IEEE Transactions on Circuits and Systems for Video Technology**, IEEE, 2024.

MARPE, D.; Schwarz. H; WIEGAND, T. Context-based adaptive binary arithmetic coding in the H. 264/AVC video compression standard. **IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)**, v. 13, n. 7, p. 620–636, 7 2003.

MARTINS, A. L. D. M. **Projeto da arquitetura de hardware para binarização e modelagem de contextos para o CABAC do padrão de compressão de vídeo H.264/AVC**. Dissertação (Dissertação de Mestrado) — Universidade Federal do Rio Grande do Sul, Instituto de Informática, Porto Alegre, Brasil, 2011.

MENTZER, F.; GOOL, L. V.; TSCHANNEN, M. Learning better lossless compression using lossy compression. In: **Proceedings of the IEEE/CVF conference on computer vision and pattern recognition**. [S.l.: s.n.], 2020. p. 6638–6647.

NAGARAJU, M.; GUPTA, S. K.; BHADAURIA, V. High-throughput, area-efficient hardware architecture of cabac-binarization for uhd applications. **Microelectronics Journal**, Elsevier, v. 123, p. 105425, 2022.

NORKIN, A. et al. Alliance for open media (aomedia) progress report. **SMPTE Motion Imaging Journal**, SMPTE, v. 131, n. 8, p. 88–92, 2022.

PRANGNELL, L. Visually lossless coding in hevc: A high bit depth and 4: 4: 4 capable jnd-based perceptual quantisation technique for hevc. **Signal Processing: Image Communication**, Elsevier, v. 63, p. 125–140, 2018.

RAMACHANDRAN, A. **Decode to Encode: Master Complex Concepts Faster, Bridge Gaps and Be the Expert in Video Coding**. EUA: Avinash Ramachandran, 2018. ISBN 9780998045016. Disponível em: https://www.amazon.com/Decode-Encode-Master-Complex-Concepts/dp/0998045012.

RAMOS, F. L. L. et al. Residual Syntax Elements Analysis and Design Targeting High-Throughput HEVC CABAC. **IEEE Transactions on Circuits and Systems I: Regular Papers**, v. 67, n. 2, p. 475–488, 2020.

RICHARDSON, I. E. **The H.264 Advanced Video Compression Standard**. Second. Chichester, UK: Wiley, 2010. ISBN 978-0-470-74476-4.

ROSA, V. S. d. Arquiteturas de hardware dedicadas para codificadores de vídeo h. 264: filtragem de efeitos de bloco e codificação aritmética binária adaptativa a contexto. 2010.

SANDVINE. **The Global Internet Phenomena Report March 2024**. 2024. Accessed: 2025-04-. Disponível em: https://www.sandvine.com.

SANTOS, C. F. d. Compressão de nuvens de pontos dinâmicas: uma abordagem eficiente para a etapa de predição. Universidade Federal de Pelotas, 2020.

SCHWARZ, H. et al. Quantization and Entropy Coding in the Versatile Video Coding (VVC) Standard. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 31, n. 10, p. 3891–3906, 2021.

Siemens EDA. **ModelSim User's Manual**. [S.l.], 2024. Ferramenta de simulação para VHDL, Verilog e SystemVerilog.

SOARES, J. L. F. Um estudo comparativo das estruturas de codificação de videos dos padrões h. 264/avc e h. 266/vvc. Universidade Federal do Pampa, 2022.

TRAN, D.-L. et al. An efficient hardware implementation of residual data binarization in hevc cabac encoder. **Electronics**, MDPI, v. 9, n. 4, p. 684, 2020.

UHRINA, M. et al. Performance comparison of vvc, av1, hevc, and avc for high resolutions. **Electronics**, MDPI, v. 13, n. 5, p. 953, 2024.

ZHANG, K. et al. Uncompressed high-definition videoconferencing tools for telemedicine and distance learning. **Telemedicine and e-Health**, Mary Ann Liebert, Inc. 140 Huguenot Street, 3rd Floor New Rochelle, NY 10801 USA, v. 19, n. 8, p. 579–584, 2013.

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

Tabela 13 – Elementos sintáticos, representação binária, cMax e formato.

Nº	Elemento Sintático	Binário (9 bits)	cMax	Formato
1	end_of_slice_one_bit	000000001	000001	0001
2	end_of_tile_one_bit	000000010	000001	0001
3	end_of_subset_one_bit	000000011	000001	0001
4	alf_ctb_flag[ ][ ][ ]	000000100	000001	0001
5	alf_use_aps_flag	000000101	000001	0001
6	alf_luma_fixed_filter_idx	000000110	111101	0010
7	alf_luma_prev_filter_idx	000000111	111101	0010
8	alf_ctb_filter_alt_idx[ ][ ][ ]	000001000	000000	0011
9	alf_ctb_cc_cb_idc[ ][ ]	000001001	000000	0011
10	alf_ctb_cc_cr_idc[ ][ ]	000001010	000000	0011
11	sao_merge_left_flag	000001011	000001	0001
12	sao_merge_up_flag	000001100	000001	0001
13	sao_type_idx_luma	000001101	000000	0011
14	sao_type_idx_chroma	000001110	000000	0011
15	sao_offset_abs[ ][ ][ ]	000001111	000000	0011
16	sao_offset_sign_flag[ ][ ][ ]	000010000	000001	0001
17	sao_band_position[ ][ ][ ]	000010001	011111	0001
18	sao_eo_class_luma	000010010	000011	0001
19	sao_eo_class_chroma	000010011	000011	0001
20	split_cu_flag	000010100	000001	0001
21	split_qt_flag	000010101	000001	0001
22	mtt_split_cu_vertical_flag	000010110	000001	0001
23	mtt_split_cu_binary_flag	000010111	000001	0001
24	non_inter_flag	000011000	000001	0001
25	cu_skip_flag[ ][ ]	000011001	000001	0001
26	pred_mode_ibc_flag	000011010	000001	0001
27	pred_mode_plt_flag	000011011	000001	0001
28	cu_act_enabled_flag	000011100	000001	0001

Nº	Elemento Sintático	Binário (9 bits)	cMax	Formato
29	pred_mode_flag	000011101	000001	0001
30	intra_bdpcm_luma_flag	000011110	000001	0001
31	intra_bdpcm_luma_dir_flag	000011111	000001	0001
32	intra_mip_flag	000100000	000001	0001
33	intra_mip_transposed_flag[ ][ ]	000100001	000001	0001
34	intra_mip_mode[ ][ ]	000100010	111101	0010
35	intra_luma_ref_idx	000100011	000000	0011
36	intra_subpartitions_mode_flag	000100100	000001	0001
37	intra_subpartitions_split_flag	000100101	000001	0001
38	intra_luma_mpm_flag[ ][ ]	000100110	000001	0001
39	intra_luma_not_planar_flag[ ][ ]	000100111	000001	0001
40	intra_luma_mpm_idx[ ][ ]	000101000	000000	0011
41	intra_luma_mpm_remainder[ ][ ]	000101001	111101	0010
42	intra_bdpcm_chroma_flag	000101010	000001	0001
43	intra_bdpcm_chroma_dir_flag	000101011	000001	0001
44	cclm_mode_flag	000101100	000001	0001
45	cclm_mode_idx	000101101	000000	0011
46	intra_chroma_pred_mode	000101110	000000	0101
47	general_merge_flag[ ][ ]	000101111	000001	0001
48	inter_pred_idc[ ][ ]	000110000	000000	0110
49	inter_affine_flag[ ][ ]	000110001	000001	0001
50	cu_affine_type_flag[ ][ ]	000110010	000001	0001
51	sym_mvd_flag[ ][ ]	000110011	000001	0001
52	ref_idx_l0[ ][ ]	000110100	000000	0011
53	mvp_l0_flag[ ][ ]	000110101	000001	0001
54	ref_idx_l1[ ][ ]	000110110	000000	0011
55	mvp_l1_flag[ ][ ]	000110111	000001	0001
56	amvr_flag[ ][ ]	000111000	000001	0001
57	amvr_precision_idx[ ][ ]	000111001	000000	0011
58	bcw_idx[ ][ ]	000111010	000000	0011
59	cu_coded_flag	000111011	000001	0001
60	cu_sbt_flag	000111100	000001	0001
61	cu_sbt_quad_flag	000111101	000001	0001

Nº	Elemento Sintático	Binário (9 bits)	cMax	Formato
62	cu_sbt_horizontal_flag	000111110	000001	0001
63	cu_sbt_pos_flag	000111111	000001	0001
64	lfnst_idx	001000000	000000	0011
65	mts_idx	001000001	000000	0011
66	palette_predictor_run	001000010	000000	0100
67	num_signalled_palette_entries	001000011	000000	0100
68	new_palette_entries	001000100	000001	0001
69	palette_escape_val_present_flag	001000101	000001	0001
70	palette_idx_idc	001000110	111101	0010
71	palette_transpose_flag	001000111	000001	0001
72	copy_above_palette_indices_flag	001001000	000001	0001
73	run_copy_flag	001001001	000001	0001
74	palette_escape_val	001001010	000000	0100
75	regular_merge_flag[ ][ ]	001001011	000001	0001
76	mmvd_merge_flag[ ][ ]	001001100	000001	0001
77	mmvd_cand_flag[ ][ ]	001001101	000001	0001
78	mmvd_distance_idx[ ][ ]	001001110	000000	0011
79	mmvd_direction_idx[ ][ ]	001001111	000011	0001
80	ciip_flag[ ][ ]	001010000	000001	0001
81	merge_subblock_flag[ ][ ]	001010001	000001	0001
82	merge_subblock_idx[ ][ ]	001010010	000000	0011
83	merge_gpm_partition_idx[ ][ ]	001010011	111111	0001
84	merge_gpm_idx0[ ][ ]	001010100	000000	0011
85	merge_gpm_idx1[ ][ ]	001010101	000000	0011
86	merge_idx[ ][ ]	001010110	000000	0011
87	abs_mvd_greater0_flag[ ]	001010111	000001	0001
88	abs_mvd_greater1_flag[ ]	001011000	000001	0001
89	abs_mvd_minus2[ ]	001011001	000000	0011
90	mvd_sign_flag[ ]	001011010	000001	0001
91	tu_y_coded_flag[ ][ ]	001011011	000001	0001
92	tu_cb_coded_flag[ ][ ]	001011100	000001	0001
93	tu_cr_coded_flag[ ][ ]	001011101	000001	0001
94	cu_qp_delta_abs	001011110	000000	0111

Nº	Elemento Sintático	Binário (9 bits)	cMax	Formato
95	cu_qp_delta_sign_flag	001011111	000001	0001
96	cu_chroma_qp_offset_flag	001100000	000001	0001
97	cu_chroma_qp_offset_idx	001100001	000000	0011
98	transform_skip_flag[ ][ ]	001100010	000001	0001
99	tu_joint_cbcr_residual_flag[ ]	001100011	000001	0001
100	last_sig_coeff_x_prefix	001100100	000000	0011
101	last_sig_coeff_y_prefix	001100101	000000	0011
102	last_sig_coeff_x_suffix	001100110	000001	0001
103	last_sig_coeff_y_suffix	001100111	000001	0001
104	sb_coded_flag[ ][ ]	001101000	000001	0001
105	sig_coeff_flag[ ][ ]	001101001	000001	0001
106	par_level_flag[ ]	001101010	000001	0001
107	abs_level_gtx_flag[ ][ ]	001101011	000001	0001
108	abs_remainder[ ]	001101100	000000	0011
109	dec_abs_level[ ]	001101101	000000	0011
110	coeff_sign_flag[ ]	001101110	000001	0001