UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação

Dissertação de Mestrado

Simplificações Algorítmicas e Desenvolvimento de Hardware para o *In-loop Filter* do Padrão HEVC

Fabiane Konrad Rediess

Pelotas, 2015

Fabiane Konrad Rediess

Simplificações Algorítmicas e Desenvolvimento de Hardware para o *In-loop Filter* do Padrão HEVC

Dissertação apresentada ao Programa de Pós-graduação em Computação da Universidade Federal de Pelotas, como requisito parcial à obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Luciano Volcan Agostini

Co-orientador: Prof. Dr. Marcelo Schiavon Porto

Pelotas, 2015

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação na Publicação

R317s Rediess, Fabiane Konrad

Simplificações algorítmicas e desenvolvimento de hardware para o In-loop filter do padrão HEVC / Fabiane Konrad Rediess ; Luciano Volcan Agostini, orientador ; Marcelo Schiavon Porto, coorientador. — Pelotas, 2015.

100 f. : il.

Dissertação (Mestrado) — Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2015.

1. Compressão de vídeo. 2. HEVC. 3. In-loop filter. 4. ALF. 5. SAO. I. Agostini, Luciano Volcan, orient. II. Porto, Marcelo Schiavon, coorient. III. Título.

CDD : 005

Elaborada por Aline Herbstrith Batista CRB: 10/1737

Banca Examinadora:

_____
Prof. Dr. Luciano Volcan Agostini (Orientador)

_____
Prof. Dr. Marcelo Schiavon Porto (Co-orientador)

_____
Prof. Dr. Vagner Santos da Rosa (FURG)

_____
Prof. Dr. Leomar Soares da Rosa Jr (UFPel)

_____
Dr. Guilherme Ribeiro Corrêa (UFPel)

AGRADECIMENTOS

Analisando o período em que este trabalho foi desenvolvido, percebo que muitas coisas aconteceram, muitas reviravoltas, muitas boas e algumas ruins. Mas ao fim de tudo, só posso ser grata por tudo que vivi.

Muitos também foram os que passaram deste para outro plano durante este período do meu mestrado. Dona Vera, Dona Terezinha, tio Bruno, Júlia, Zé e Cindy, sou muito grata pelos poucos, mas bons momentos de convivência que tivemos e pelo apoio que sempre me demonstraram.

Primeiramente, gostaria de agradecer aos meus pais, Elmira Konrad Rediess e Wilmar Rediess, por terem dado toda a base moral e por terem forjado todo o caminho para que eu chegasse até aqui.

Agradeço a todos meus amigos e familiares, em especial, minhas amigas Márcia Madeira e Danielle Lisboa, e minha sogra Márcia Froner, que sempre tinham uma palavra certa no momento certo. E pelos bons momentos e boas risadas, que sempre me iluminam a alma.

Sou imensamente grata ao meu marido Diego Froner, por toda sua paciência e atenção dedicadas principalmente nos momentos mais difíceis. Por ser compreensivo nos momentos de esgotamento físico e mental em que o bom humor estava ausente. Pelos momentos felizes que me trouxeram paz de espírito e força para continuar. E principalmente por todo amor e carinho, em especial, nesses dias que antecederam a entrega deste trabalho.

Também não posso deixar de agradecer a Deus e a fé que me manteve de pé durante este período tão conturbado. Durante este período conheci também uma fé que não sabia que tinha. Sou infinitamente grata pela força que esta fé me trouxe e que continua me trazendo, que assim eu possa continuar.

Agradeço aos ex-colegas de GACI, Thaísa Leal da Silva e Guilherme Ribeiro Corrêa, por ter me auxiliado e guiado meus primeiros passos na pesquisa sobre o HEVC

Aos colegas de laboratórios Cássio Cristani, Pargles Dall’Oglio e Victor Junes, que auxiliaram no desenvolvimento deste trabalho, mesmo que, por breves períodos, antes de suas incursões fora do país.

Ao colega de laboratório Ruhan Conceição pelo apoio e pela troca de idéias, que certamente contribuíram significativamente para o desenvolvimento deste trabalho.

Ao meu professor da graduação e primeiro orientador de iniciação científica, José Luís Güntzel, por ter me dado a oportunidade de ter contato com a pesquisa, o que me abriu muitos horizontes e certamente foi determinante para eu chegar até aqui.

Aos meus professores do mestrado, Paulo Ferreira Jr., Lisane Brisolara, Leomar da Rosa Jr., Júlio Mattos, Felipe Marques e Aline Loreto, pelos conhecimentos e incentivos transmitidos durante suas aulas. Momentos que ficarão marcados em minha memória.

Ao Prof. Bruno Zatt que contribuiu muito para o andamento com seu conhecimento e suas sugestões.

Ao meu co-orientador, Prof. Marcelo Porto e ao meu orientador, Prof. Luciano Agostini por terem me aceitado como orientanda. Sei que dei trabalho como orientanda, e sei que vocês mereciam mais, mas foi o possível de acordo com as circunstâncias. Sem vocês este trabalho não existiria, serei eternamente grata pelo apoio, compreensão e incentivo.

Agradeço ainda ao Prof. Luciano por, em 2004, ter me dado atenção e me apresentado o que era a pesquisa científica, pois chegava à Universidade totalmente ignorante neste sentido. Aquele momento, com certeza, foi um momento decisivo e que me guiou por este caminho. Também pela orientação como bolsista de iniciação científica durante a graduação e no Trabalho de Conclusão de Curso da Graduação.

*“**A ciência nunca resolve um problema sem criar pelo menos outros dez”.***

(George Bernard Shaw)

RESUMO

REDIESS, Fabiane. Simplificações Algorítmicas e Desenvolvimento de Hardware para o *In-Loop Filter* do Padrão HEVC. 2015. 100f. Dissertação (Mestrado em Ciência da Computação). Universidade Federal de Pelotas, Pelotas.

O processo de filtragem na codificação de vídeos é uma ferramenta relevante devido ao seu objetivo que é o de suavizar artefatos inseridos pelas demais etapas da compressão qualificando a percepção visual dos vídeos codificados. O padrão *High Efficiency* Video Coding (HEVC) trouxe a proposta de dois novos filtros para o *In-loop Filter*, o *Adaptive Loop Filter* (ALF) e o *Sample Adaptive Offset* (SAO), que são o foco deste trabalho. Apenas o filtro SAO foi inserido na versão final do padrão, mas com o objetivo de melhor explorar as potencialidades do ALF, ele também foi inserido na investigação apresentada neste trabalho. É apresentada inicialmente uma revisão bibliográfica destes dois filtros e após este embasamento teórico, é realizada uma análise dos algoritmos destes filtros buscando simplificações que resultassem em uma redução da complexidade computacional, objetivando a sua implementação em hardware. O filtro ALF envolve uma série de operações matemáticas com dados em ponto flutuante, ponto crítico para uma implementação em hardware. Portanto, a simplificação proposta foi a substituição destas operações em ponto flutuante por operações em ponto fixo. Os resultados dos experimentos mostraram que o impacto desta simplificação é um aumento de apenas 0,05% no *bitrate* para manutenção da mesma qualidade em comparação à aplicação do ALF com dados em ponto flutuante. Entretanto, a simplificação ainda alcança uma redução de 3,38% no *bitrate* quando comparado a não aplicação do ALF. Foram propostas ainda, neste trabalho, arquiteturas para os núcleos do ALF das versões 3 e 5 do HEVC *Test Model* (HM), além de uma versão configurável da versão 3 do HM, em que a arquitetura usa a mesma estrutura para processar qualquer um dos três formatos de filtro. Resultados de síntese para FPGA mostraram que estas arquiteturas alcançaram uma taxa de processamento mínima de 30 quadros UHD 4K por segundo. Para o SAO, além da simplificação baseada na substituição dos dados em ponto flutuante por dados inteiros, propôs-se também a utilização de precisão fracionária com ponto fixo. Outra simplificação proposta para o SAO foi a eliminação de multiplicadores e divisores completos através da aplicação da técnica de *loop unrolling* à função de custo interna do SAO. Os resultados demonstraram que, com a utilização de dados inteiros, há um aumento no *bitrate* de aproximadamente 0,05% e para dados utilizando ponto fixo com precisão fracionária de 8 bits, houve um ganho de 0,0005% no *bitrate* para manutenção da mesma qualidade. Com base nestas simplificações, foi proposta uma arquitetura para a função de custo, a qual alcançou uma taxa de processamento de 1.634 quadros UHD 4K por segundo. Também foi proposta uma arquitetura para a realização das etapas de classificação e levantamento estatístico necessárias para a geração dos *offsets*. Esta arquitetura alcançou uma taxa de processamento de 45 quadros UHD 4K por segundo.

Palavras-chave: Compressão de vídeo, HEVC, *In-Loop Filter*, ALF, SAO, Projeto de Hardware.

ABSTRACT

REDIESS, Fabiane. Simplificações Algorítmicas e Desenvolvimento de Hardware para o *In-Loop Filter* do Padrão HEVC. 2015. 100f. Dissertação (Mestrado em Ciência da Computação). Universidade Federal de Pelotas, Pelotas.

The filtering process has been an important tool to the video coding since it intends to reduce visual artifacts introduced by the other codification steps, increasing the perceptual quality of the encoded videos. The High Efficiency Video Coding (HEVC) brought a proposal of two new filters inside the In-loop Filter, the Adaptive Loop Filter (ALF) and Sample Adaptive Offset (SAO) which are the focus of this work. Only the SAO filter was inserted in the final standard version but intending to better explore the ALF features it was also included in the investigation presented in this work. This text initially brought a theoretical discussion about these filters and after that we analyzed the filters algorithms in order to simplify them focusing on hardware implementations. The ALF filter involves a series of mathematical calculations on floating point data, which is a critical point for hardware implementations. Therefore, the proposed simplification was to replace these data with integer data. The results of the experiments showed that the impact of this simplification represents an increase of only 0.05% in the bitrate to maintain the same quality when compared to the original ALF execution. However, the simplified algorithm achieves a reduction of 3.38% in the bitrate when compared to a video encoding with the ALF filter disabled. This work also proposed architectures to the ALF cores considering the versions 3 and 5 of the HEVC Test Model (HM) software, and also a configurable version considering the version 3 of HM. In this last case, the architecture uses the same structure to process any of the three filter shapes. The architectures were synthesized to Altera FPGAs and the results shown that the hardwares achieved, at least, a processing rate of 30 UHD 4K frames per second. The SAO filter algorithm was also simplified using integer and fixed point data instead of floating point data. Another simplification proposed was the elimination of full multipliers and dividers applying a loop unrolling technique to the SAO internal cost function and converting these operations in shift-adds. The results showed that the use of the simplifications using integer data caused a bitrate increase of 0.05%. The results also showed a bitrate reduction of 0.0005% when using fixed point data with an 8-bit precision. Based on these simplifications, we proposed an architecture for the SAO internal cost function, which reached a processing rate of 1,634 UHD frames per second. It was also proposed an architecture to perform the classification and statistical collection, in order to allow the offsets generation. This architecture was developed to consume only one sample per cycle and it reached a processing rate of 45UHD 4K frames per second.

Keywords: Video compression, HEVC, In-Loop Filter, ALF, SAO, Hardware Design.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ALF	*Adaptive Loop Filter*
ALUT	*Adaptive Look-up Table*
AVC	*Advanced Video Coding*
ASIP	*Application Specific Instruction Set Processor*
BBC	*British Broadcasting Corporation*
BDC	*Band Correction*
BA	*Block-based Adaptation*
BO	*Band Offset*
CB	*Coding Block*
CPB	*Coded Picture Buffer*
CTB	*Coding Tree Block*
CTC	*Common Test Conditions*
CTU	*Coding Tree Unit*
CU	*Coding Unit*
DF	*Deblocking Filter*
EO	*Edge Offset*
EXC	*Extrem Correction*
FPGA	*Field Programmable Gate Array*
HD	*High Definition*
HE	*High Efficiency*
HEVC	*High Efficiency Video Coding*
HM	*HEVC Test Model*
ISO	*International Organization for Standarization*
ITU-T	*International Telecommunication Union - Telecommunication*
JCT-VC	*Joint Collaborative Team on Video Coding*
LCU	*Largest Coding Unit*
MPEG	*Moving Picture Experts Group*
MSE	*Mean Square Error*
PB	*Prediction Block*
PU	*Prediction Unit*
RA	*Region-based Adaptation*
SAO	*Sample Adaptive Offset*
SBTVD	Sistema Brasileiro de Televisão Digital
TB	*Transform Block*
TMuC	*Test Model under Consideration*

TU	*Transform Unit*
UHD	*Ultra High Definition*
VCEG	*Video Coding Experts Group*
VHDL	*VHSIC Hardware Description Language*
WPP	*Wavefront Parallel Processing*

SUMÁRIO

1 INTRODUÇÃO

Os vídeos digitais têm se tornado cada vez mais populares, estando presentes nos mais diversos tipos de dispositivos, desde dispositivos móveis, como celulares, até televisores de altas definições. Entretanto, esta popularização não seria viável sem a compressão de vídeo, pois um vídeo não comprimido necessitaria de uma grande quantidade de bits para ser armazenado e de altas taxas de transmissão para que possa ser transmitido em tempo real. Por exemplo, considerando uma sequência de apenas 10 minutos em resolução *High Definition* 1080p (HD 1080p), 1920x1080 pixels, com uma profundidade de 24 bits por pixel e uma taxa de 30 quadros por segundo, seriam necessários pouco mais de 104 GB de espaço para armazenamento e uma taxa de transmissão de 1,5 Gbps para ser transmitido em tempo real. Para uma resolução ainda maior, como a *Ultra High Definition* 4K (UHD 4K), a qual já vem tomando um espaço de mercado com as TVs 4K, cuja resolução é de 3840x2160 pixels, esta taxa de transmissão aumenta para 6 Gbps e o espaço para armazenamento para pouco mais de 417 GB.

O padrão H.264/AVC (JVT EDITORS, 2003), atualmente utilizado em *Blu-rays* e no Sistema Brasileiro de Televisão Digital (SBTVD), já conquistou posição de destaque na indústria da área, tendo sido o padrão utilizado na maior parte das novas tecnologias que manipulam vídeos digitais. Este padrão conseguiu alcançar taxas de compressão duas vezes maiores do que os padrões anteriores, como o MPEG-2, que é o padrão utilizado em DVDs. No entanto, para alcançar estas taxas, o H.264/AVC tornou-se quatro vezes mais complexo computacionalmente do que os padrões anteriores (RICHARDSON, 2003).

Mesmo com ganhos tão significativos obtidos com o H.264/AVC, os trabalhos e as pesquisas na área de compressão de vídeos não pararam. Tanto a academia quanto a indústria continuaram suas pesquisas, pois há uma demanda contínua e

crescente por resoluções cada vez maiores. E a necessidade do aumento de eficiência de codificação aumenta ainda mais quando o crescimento da resolução vem acompanhado de um sistema *multi-view* ou 3D (MÜLLER *et al.*, 2013), por exemplo. Além disso, há ainda uma demanda contínua e crescente também para um maior volume de transmissão pela rede, em especial, na comunicação com dispositivos móveis, os quais possuem restrições de consumo de energia.

Em 2010, foi formado um grupo, o *Joint Collaborative Team on Video Coding* (JCT-VC), composto por especialistas do ITU-T e da ISO/IEC com objetivo de analisar a possibilidade de um novo padrão de compressão de vídeo. Este padrão foi denominado *High Efficiency Video Coding* (HEVC) (JCT-VC, 2010) e foi oficialmente lançado em 2013 (ITU-T, 2013), sendo o estado da arte na área de codificação de vídeos. O HEVC foi desenvolvido com o objetivo de dobrar a taxa de compressão atingida pelo H.264/AVC, mantendo a mesma qualidade do vídeo codificado, tudo isso mantendo, ou até mesmo reduzindo a complexidade computacional. No entanto, este objetivo não pode ser alcançado devido às diversas técnicas algorítmicas que foram empregadas para que o padrão alcançasse o objetivo de aumentar a taxa de compressão.

O HEVC alcançou, em média, uma redução de bitrate de 37% em comparação ao H.264/AVC, mantendo a mesma qualidade do vídeo, com um aumento de aproximadamente 40% na complexidade computacional (VANNE *et al.*, 2012). Estes dados referem-se ao caso em que somente as ferramentas essenciais são utilizadas. Em um estudo mais recente, os ganhos reportados ao HEVC em comparação ao H.264/AVC alcançam taxas de 52% para resolução UHD 4K (MRAK, BARONCINI e RAMZAN, 2014).

Uma das características da compressão de vídeos que está sendo explorada pelo HEVC é a redução na deterioração da qualidade subjetiva da imagem causada pelas diversas etapas da compressão. Com objetivo que melhorar a qualidade dos vídeos comprimidos, durante o desenvolvimento do HEVC foi proposto um Filtro em Laço, o *In-Loop Filter*, composto por até três filtros: o *Deblocking Filter* (DF), ou Filtro Redutor do Efeito de Bloco, o *Adaptive Loop Filter* (ALF), ou Filtro Adaptativo de Laço e o *Sample Adaptive Offset* (SAO), ou Deslocamento Adaptativo de Amostra. O DF é bastante similar ao filtro já existente no H.264/AVC. Já o ALF e o SAO foram duas inovações propostas pelo HEVC e são o foco deste trabalho. O ALF foi

removido das versões mais recentes do padrão, especialmente por conta da complexidade adicional que foi inserida. Mas como este filtro trouxe alguns ganhos importantes no processo de codificação e como ele estava ainda em discussão pelo JCT-VC quando este trabalho de mestrado teve início, o ALF foi investigado neste trabalho.

1.1 Motivação

A principal motivação para este trabalho encontra-se no contexto onde ele está inserido, pois este está alinhado com o que há de mais recente em pesquisa na área de compressão de vídeos.

Parte deste trabalho possui seu foco no SAO, um dos filtros que foi inserido dentro do mais novo padrão de codificação de vídeo, o HEVC, e parte foca no ALF, um filtro que acabou não sendo utilizado na versão final do padrão, não por ter sido considerado uma ferramenta ineficiente, mas sim por ter sido considerado muito complexo computacionalmente no contexto atual.

Outro fator motivador é a existência de poucos trabalhos relacionados, o que demonstra o quão recente é o tema proposto.

1.2 Objetivos

Este trabalho tem como principal objetivo investigar as soluções algorítmicas para o *In-loop Filter* do novo padrão de codificação de vídeo HEVC, buscando simplificações nesses algoritmos de forma a viabilizar a implementação de arquiteturas eficientes em hardware. Estas arquiteturas em hardware devem apresentar baixa utilização de recursos de hardware e elevadas taxas de processamento, visando a codificação/decodificação de vídeos de alta definição em tempo real.

1.3 Trabalhos Relacionados

Considerando ainda que o HEVC foi lançado recentemente, até onde se tem conhecimento, existem poucos trabalhos focando em implementações em hardware tanto para o SAO quanto para o ALF.

Para o SAO foram identificados poucos trabalhos na literatura. Os trabalhos (ZHU *et al.*, 2013) e (PARK e RYOO, 2013), implementam o filtro em hardware mas

com foco no lado do decodificador. O filtro do lado do codificador é consideravelmente mais complexo, pois necessita realizar a coleta de dados estatísticos e decisão dos tipos e *offsets* a serem aplicados, conforme será melhor explicado na seção 3.2.

O trabalho (ZHU *et al.*, 2014) apresenta uma simplificação na função de custo do SAO e uma implementação em hardware. O trabalho (MODY *et al.*, 2014) implementa uma arquitetura completa do SAO, mas não traz qualquer simplificação ou comparação a outros trabalhos ou mesmo em relação ao padrão.

Para o ALF também são poucos os trabalhos que apresentam soluções em hardware. O trabalho encontrado com o foco mais similar ao deste trabalho foi apresentado em (DU e YU, 2011), entretanto, esta arquitetura implementa uma versão anterior do ALF, combinada com o DF, que era baseada no padrão H.264/AVC e não no HEVC.

Outro trabalho identificado para o ALF foi o (HAUTALA, BOUTELLIER e HANNUKSEDA, 2013) que apresenta uma solução embarcada para a implementação do ALF, aplicando parte em hardware como um processador de uso específico e parte através de programação em linguagem C.

1.4 Organização do Trabalho

Esta dissertação está organizada como segue: no capítulo 2 estão os conceitos relacionados ao padrão HEVC. No capítulo 3 encontra-se a fundamentação teórica do *In-loop Filter*. No capítulo 4 estão as simplificações propostas para o SAO e o ALF e análises de impactos. O capítulo 5 apresenta o hardware proposto para o SAO e o ALF com base nas avaliações efetuadas juntamente com os resultados obtidos. Por fim, o capítulo 6 traz as conclusões e trabalhos futuros.

2 O PADRÃO HEVC

Este capítulo apresenta os principais conceitos relacionados ao mais novo padrão de codificação de vídeos, o *High Efficiency Video Coding* (HEVC), trazendo um histórico dos padrões de compressão de vídeos e as principais características e inovações propostas por este padrão.

2.1 Histórico do padrão HEVC

O padrão que antecedeu o padrão HEVC foi o H.264/AVC, padrão utilizado atualmente no *Blu-Ray* e no Sistema Brasileiro de Televisão Digital (BRASIL, 2003).

O padrão H.264/AVC foi desenvolvido com o objetivo principal de dobrar a taxa de compressão de seu predecessor, o padrão MPEG-2, que é atualmente utilizado em DVDs. Este objetivo foi conquistado através do uso conjunto de diversas técnicas avançadas e inovadoras de compressão e que acarretou em um significativo acréscimo na sua complexidade computacional, que é cerca de quatro vezes maior que a do MPEG-2 (RICHARDSON, 2003).

A taxa de compressão atingida pelo H.264/AVC varia de acordo com a qualidade pretendida para o vídeo codificado. Com uma taxa de 50:1, a redução na qualidade do vídeo é, em geral, pequena e pouco perceptível (ou mesmo imperceptível) ao sistema visual humano (AGOSTINI, 2007). Neste caso, o vídeo gerado possui uma qualidade equivalente a vídeos MPEG-2 usados em DVDs (AGOSTINI, 2007).

Mesmo considerando todos os avanços alcançados pelo H.264/AVC, existe uma crescente demanda por resoluções cada vez maiores nos mais diferentes tipos de dispositivos, não somente em dispositivos como TVs de alta definição, mas também em dispositivos móveis como aparelhos celulares e *tablets*, exigindo uma eficiência de codificação superior ao do H.264/AVC (SULLIVAN *et al.*, 2012), que apesar de ser aplicável, não foi desenvolvido com foco em vídeos de alta definição.

Em função de tais demandas, em janeiro de 2010, foi formado o grupo *Joint Collaborative Team on Video Coding* (JVC-VC), composto por especialistas do ITU-T e ISO/IEC com objetivo de avaliar a possibilidade de um novo padrão de codificação de vídeo ainda mais eficiente que o atual estado da arte, o H.264/AVC. Inicialmente, foi realizada uma chamada de propostas, buscando contribuições da indústria e da academia para tornarem-se parte deste novo padrão de compressão de vídeo.

As propostas apresentadas foram avaliadas pelo JCT-VC e aquelas com resultados mais promissores foram condensadas em um software chamado *Test Model under Consideration* (TMuC), que tinha como objetivo facilitar a avaliação das ferramentas implementadas e servir de base para a continuidade das investigações.

Na terceira reunião do JCT-VC foi definida a primeira versão do *draft* do HEVC (JCT-VC, 2010) o qual foi acompanhado da primeira versão software de referência que trazia a implementação das técnicas definidas no *draft*, o *HEVC Test Model* (HM1) (MCCANN *et al.*, 2010).

O trabalho do JCT-VC continuou nas reuniões seguintes, onde novas propostas eram avaliadas e novas versões do *Working Draft* e do HM foram sendo lançadas, até o lançamento do padrão HEVC, em janeiro de 2013 (ITU-T, 2013).

O HEVC foi desenvolvido com dois objetivos principais: aumentar a taxa de compressão, chegando ao dobro da taxa alcançada pelo H.264/AVC, ou seja, atingindo uma taxa de 100:1 e manter ou até mesmo reduzir a complexidade computacional. Entretanto, o HEVC alcançou, em média, uma redução de *bitrate* de 37% e um aumento de 40% da complexidade computacional (VANNE *et al.*, 2012).

Outro foco do desenvolvimento do HEVC é o maior aproveitamento das arquiteturas dos computadores atuais com multicores através do processamento paralelo (SULLIVAN *et al.*, 2012). Portanto, o HEVC desde sua formação conceitual inicial, foca em uma estrutura inerentemente paralela.

2.2 O Codificador HEVC

A Figura 1 apresenta um diagrama em blocos representando a estrutura de um codificador HEVC típico. Nela estão identificados os blocos de predição intra e interquadros, as transformadas e quantização diretas e inversas, codificação de entropia, bem como os vídeos original e reconstruído e ainda o *buffer* de quadros de referência. (CHEN *et al.*, 2012)

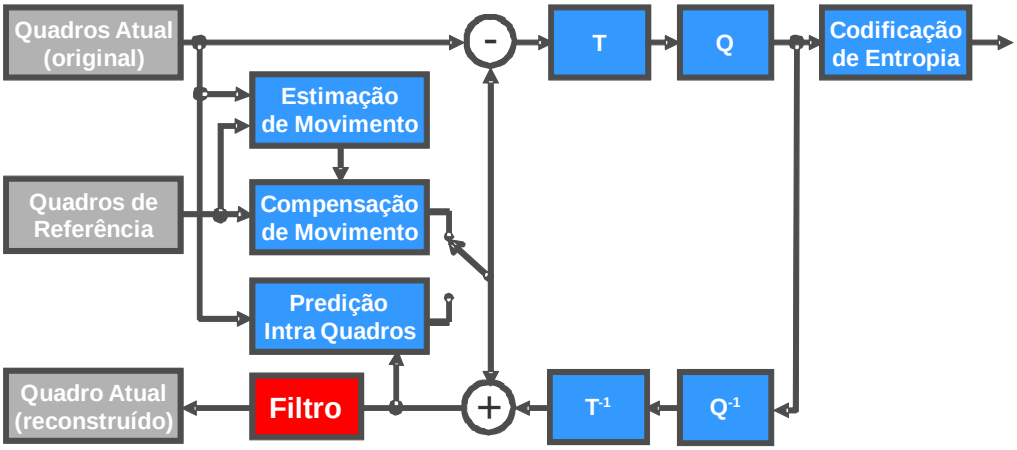


Figura 1– Diagrama de blocos de um codificador HEVC.

De maneira geral, o processo de codificação corresponde a uma sequência de etapas onde cada bloco do quadro alvo da codificação, chamado de atual, passa primeiramente pela predição intraquadros, quando usa informações somente do próprio quadro, e interquadros, quando usa informações de quadros anteriormente codificados. O melhor resultado destas predições é então subtraído do bloco atual original e apenas esta diferença, os resíduos, passam para as etapas seguintes, que tem por objetivo a redução da redundância espacial no domínio das frequências através das transformadas e da quantização. Em seguida, o bloco segue por dois caminhos distintos, um para a codificação de entropia para a redução da redundância na forma como os dados são codificados e outro segue pelas etapas de quantização e transformadas inversas para retorno do bloco ao domínio espacial. Este processo de retorno se torna necessário para permitir que tanto o codificador quanto o decodificador possuam as mesmas referências, pois os processos de transformadas e quantização geram perdas. A estes resíduos reconstituídos é somado o resultado da predição intra ou interquadros e antes de ser armazenado para referências futuras o bloco passa pela etapa de filtragem.

O foco deste trabalho está na unidade de filtro que está localizada exatamente antes do armazenamento do quadro atual reconstruído. Nas primeiras versões do HEVC este bloco era composto pelos três filtros: DF, SAO e ALF, conforme ilustrado na Figura 2. Na versão final do padrão, o ALF foi suprimido.

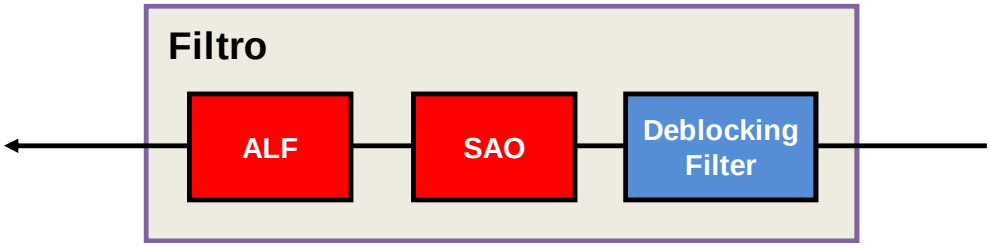


Figura 2 – Unidade *In-loop Filter* das versões iniciais do padrão HEVC.

2.3 Estrutura da Codificação

Esta subseção apresenta os elementos utilizados pelo padrão HEVC como estrutura da codificação, a qual foca na adoção de estruturas de blocos maiores, com mecanismos mais flexíveis e com maiores alternativas de subdivisões (ITU-T, 2013). Outra inovação trazida pelo padrão HEVC quanto à estrutura da codificação foi a adoção de blocos com partições distintas para a etapa de predição e de transformadas, conforme elencados a seguir.

2.3.1 *Coding Tree Units* (CTUs) e *Coding Tree Blocks* (CTBs)

O padrão HEVC traz uma nova visão sobre a forma de particionamento da imagem em unidades básicas de processamento. Um quadro é particionado em grandes blocos, chamados de *Coding Tree Units* (CTUs). E cada CTU é composta por uma *Coding Tree Block* (CTB) de luminância e duas CTBs de crominância, uma para crominância azul e uma para crominância vermelha.

Uma CTB de luminância representa um bloco quadrado, de tamanho LxL amostras, e as duas CTBs de crominância correspondentes representam um segmento quadrado de L/2xL/2, conforme representado na Figura 3, quando se utiliza a subamostragem de cores 4:2:0.O tamanho de L será igual a 16, 32 ou 64, sendo definida pela sintaxe do codificador (SULLIVAN *et al.*, 2012). Este tamanho variável, que é definido de acordo com as necessidades dos codificadores em temos de memória e requisitos computacionais, traz vantagens, especialmente quando se trabalha com vídeos de alta resolução em função dos tamanhos de CTBs maiores (SULLIVAN *et al.*, 2012).

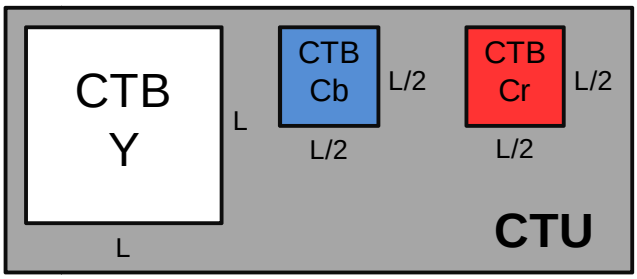


Figura 3 – Estrutura de uma *Coding Tree Unit* (CTU)

As CTBs de luminância ou crominância podem ser utilizadas diretamente como *Coding Blocks* (CBs) ou, ainda, podem ser particionadas recursivamente em múltiplas CBs, utilizando estruturas em árvore quaternária. A Figura 4 apresenta um exemplo de subdivisões em formato de árvore quaternária. Este particionamento da CB de luminância será limitado pelo valor definido na sintaxe do codificador e nunca será menor do que 8x8 amostras.

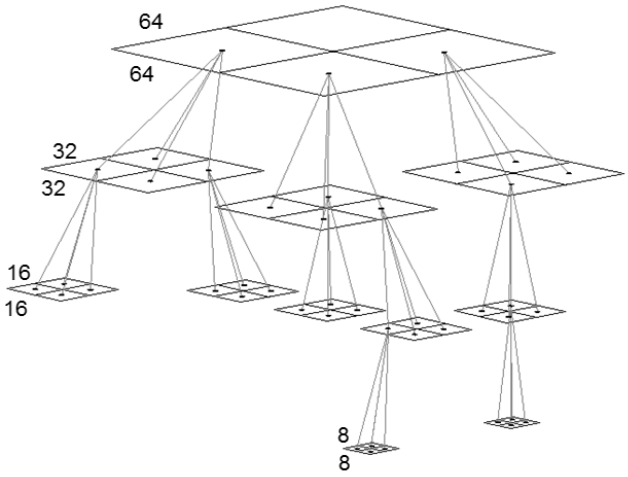


Figura 4 – Exemplo de subdivisão recursiva em árvores quaternárias.
Fonte: (CORRÊA, 2011)

2.3.2 *Coding Units* (CUs) e *Coding Blocks* (CBs)

Uma *Coding Unit* (CU) é a unidade básica de processamento utilizada na codificação e é formada por uma CB de luminância e duas CBs de crominância (SULLIVAN *et al.*, 2012).

Uma CTU pode conter apenas uma *Coding Unit* (CU) ou ser dividida em múltiplas CUs. Cada CU tem um particionamento associado à *Prediction Units* (PUs)

e outro à *Transform Units* (TUs), pois não há vinculação entre o particionamento adotado para predição e para as transformadas (SULLIVAN *et al.*, 2012).

A raiz da árvore corresponde à CTU e o tamanho máximo de um CB de luminância é o tamanho da CTB de luminância. O particionamento em árvore no HEVC é geralmente aplicado na mesma proporção tanto à luminância quanto à crominância, com exceção dos casos em que os CBs de crominância alcancem seus tamanhos mínimos (SULLIVAN *et al.*, 2012).

2.3.3 *Prediction Units* (PUs) e *Prediction Blocks* (PBs)

Uma *Prediction Unit* (PU) é a unidade básica utilizada na predição. Uma PU é formada por um *Prediction Block* (PB) de luminância e dois PBs de crominância, sendo um para crominância azul e um para crominância vermelha.

A decisão sobre a codificação de uma área do quadro como intraquadro ou interquadros é tomada no nível de CU. O modo de predição associado a uma CU é indicado como intra, quando a predição utilizada foi a predição intraquadro (espacial) ou é indicado como inter, quando a predição utilizada foi a interquadros (temporal) (SULLIVAN *et al.*, 2012).

Quando a predição é do tipo intra, o bloco terá o mesmo tamanho do CB para todos os blocos, com exceção do caso em que for atingido o tamanho mínimo estabelecido na sintaxe (nos casos de crominância).

Quando a predição é do tipo inter, será especificado se os CBs de luminância e crominância serão particionados em um, dois ou quatro PBs. O particionamento em quatro PBs somente é permitido quando o tamanho do CB é o mínimo permitido para tamanho de CB.

2.3.4 *Transform Units* (TUs) e *Transform Blocks* (TBs)

A *Transform Unit* (TU) é a unidade básica de processamento utilizada pela etapa de transformadas. Para a codificação residual, um CB pode ser recursivamente particionado em *Transform Blocks* (TBs). O maior tamanho possível de TB é igual ao tamanho do próprio CB (SULLIVAN *et al.*, 2012). Neste caso, somente são especificadas partições quadradas, em que cada bloco é divido em quadrantes.

Para cada CB de tamanho MxM, uma *flag* indica se ele será dividido em quatro blocos com tamanho M/2. Caso mais particionamentos sejam possíveis, em função da máxima profundidade da árvore quaternária residual, conforme definido na sintaxe, a cada quadrante será associado uma nova *flag* que indicará se este será particionado novamente em quatro quadrantes. Os blocos que ficarem localizados nas folhas da árvore residual serão posteriormente processados pelas transformadas.

2.4 *Slices*,*Tiles* e *Wavefronts*

Um *slice* é a estrutura de dados que pode ser decodificada de forma independente dos demais *slices* do mesmo quadro. Um dos maiores objetivos dos *slices* é a ressincronização em caso de perdas de dados. Cada *slice* é formado por um conjunto de CTUs em ordem de varredura do tipo *raster*, sendo que um quadro pode ser composto por apenas um *slice* ou por vários. A Figura 5 (a) ilustra a divisão de um quadro em slices.

Como um *slice* pode ser decodificado de forma independente, isto significa que a predição não usa dados de outros *slices* do mesmo quadro. Entretanto, informações de outros *slices* podem ser necessárias para aplicação do filtro em laço.

Um *tile* é uma nova forma de particionar um quadro, sendo que as partições, neste caso, são retangulares, podem ser decodificadas de forma independente e compartilham alguma informação de cabeçalho. O principal objetivo dos *tiles* é aumentar a capacidade de processamento paralelo. Tipicamente, mas não necessariamente, todos os *tiles* em um quadro contém o mesmo número de CTUs (SULLIVAN *et al.*, 2012). A Figura 5 (b) ilustra um exemplo da divisão de um quadro em *tiles*.

Quando o *Wavefront Parallel Processing* (WPP) está ativado, um *slice* é dividido em linhas de CTUs. A primeira linha é processada de maneira normal, a segunda linha pode começar após algumas decisões tomadas na primeira linha, e assim por diante. O WPP disponibiliza uma forma de processamento paralelo com granularidade bastante fina. Com frequência, um WPP oferece uma maior compressão e evita alguns artefatos visuais introduzidos pelos *tiles* (SULLIVAN *et al.*, 2012). A Figura 5 (c) ilustra o funcionamento do WPP.

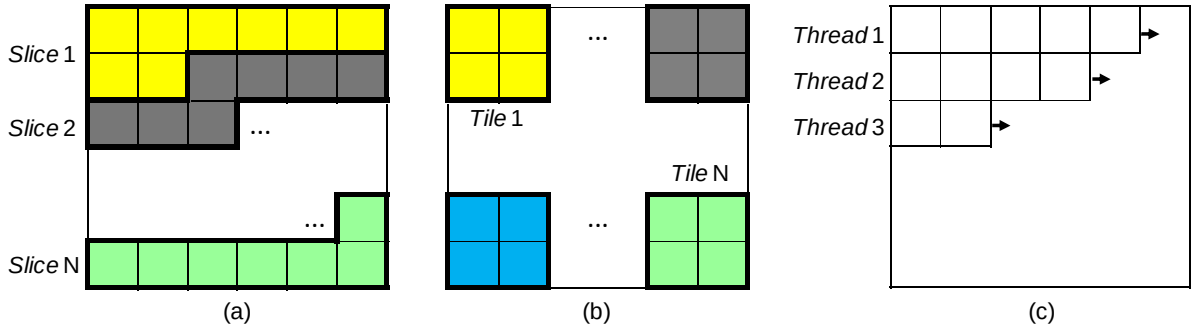


Figura 5 – Exemplos de divisão do quadro em (a) *slices*, (b) *tiles* e (c) WPP.

2.5 Perfis, Camadas e Níveis

Perfis, camadas e níveis especificam restrições nos *bitstreams* e também limites nos requisitos necessários para a decodificação destes *bitstreams*. Além disso, também podem ser usados para indicar os pontos de interoperabilidade entre implementações de decodificadores individuais (ITU-T, 2013).

O padrão HEVC foi desenvolvido para ser genérico, uma vez que serve a uma grande quantidade de aplicações, taxa de dados, resoluções, qualidades e serviços e, desta forma, incluiu uma série de requisitos e elementos algorítmicos. Entretanto, focando em aplicações, estas deverão cobrir ainda outras questões, como mídia de armazenamento, transmissão de TV e comunicação em tempo real. Para atendimento destas demandas de aplicações, foram definidos subconjuntos do padrão, os perfis, camadas e níveis (ITU-T, 2013).

Os perfis definem um conjunto de ferramentas de codificação que podem ser usadas na geração do *bitstream* (ITU-T, 2013). Mesmo com estas restrições, ainda é possível exigir uma grande variação de desempenho de codificadores e decodificadores dependendo de valores assumidos para alguns elementos do *bitstream*, como o tamanho do vídeo. Para muitas aplicações não seria viável implementar um decodificador capaz de lidar com todas estas variações. Para lidar com este problema, foram especificadas camadas e níveis para cada perfil.

As camadas e os perfis impõem restrições a alguns parâmetros chave do *bitstream*, como, por exemplo, máxima taxa de amostras, tamanho máximo do quadro, *bitrate* máximo, taxa de compressão mínima e capacidade do *Coded Picture Buffer* (CPB), que armazena os dados comprimidos antes do fluxo de decodificação (SULLIVAN *et al.*, 2012).

O padrão HEVC definiu três perfis: os perfis *Main* e *Main 10* para vídeos típicos com profundidade de amostra de 8 e 10 bits, respectivamente, e o perfil *Main Still Picture*, para imagens estáticas, quando a codificação fica restrita a ferramentas intraquadro (ITU-T, 2013).

3 *IN-LOOP FILTER* DO HEVC

O *In-Loop Filter*, ou Filtro em Laço, inicialmente proposto pelo HEVC era composto por um conjunto de três filtros: o *Deblocking Filter* (DF), ou Filtro Redutor do Efeito de Bloco, o *Adaptive Loop Filter* (ALF), ou Filtro Adaptativo de Laço e o *Sample Adaptive Offset* (SAO), ou Deslocamento Adaptativo de Amostra. Na versão final do padrão, o ALF foi removido em função de sua complexidade.

O *In-Loop Filter* é aplicado às amostras do quadro reconstruído imediatamente antes do seu armazenamento para referência futura, conforme observado na Figura 1. O DF é o primeiro dos três filtros e é sempre aplicado. Já a aplicação do SAO e do ALF dependem da avaliação da vantagem de sua aplicação, portanto, é possível ter a aplicação sobre o sinal reconstruído apenas do DF, ou então alguma das seguintes configurações: DF + SAO, DF + ALF ou DF + SAO + ALF.

Este capítulo apresenta uma fundamentação teórica sobre os três filtros que compõem o *In-loop Filter*, focando, mais especificamente nos filtros SAO e ALF.

3.1 Deblocking Filter

Em um esquema de codificação que usa predição e transformada baseadas em blocos, as imagens reconstruídas tendem a apresentar uma descontinuidade no sinal reconstruído nas bordas destes blocos. A descontinuidade que se torna visível é chamada de efeito de bloco.

Uma das maiores fontes de efeitos de bloco é a transformada baseada em bloco, seguida pela quantização (NORKIN *et al.*, 2012). Outra fonte geradora do efeito de bloco é a predição interquadros, onde a predição para blocos adjacentes no quadro atual podem não vir de blocos adjacentes nos quadros de referência,

gerando uma descontinuidade nas bordas destes blocos. E ainda, de forma similar, na predição intraquadro, onde blocos adjacentes podem ser diferentes.

Para redução do efeito de blocos, podem ser adotadas duas abordagens, *post-filtering* ou *in-loop filtering*. A abordagem com *post-filtering* não é especificada pelo padrão de codificação de vídeo e pode ser aplicada, por exemplo, no *buffer* do display. O desenvolvedor, neste caso, tem a liberdade de desenvolver algoritmos direcionados para determinadas aplicações. A segunda abordagem, *in-loop filtering*, ou filtragem em laço, trabalha com o laço de codificação/decodificação e, portanto, precisa ser normatizada, para que não haja discrepância entre o codificador e o decodificador.

O *Deblocking Filter* (DF) proposto pelo HEVC foi desenvolvido com o objetivo de melhorar a qualidade subjetiva, mas reduzindo a complexidade computacional em comparação ao *Deblocking Filter* utilizado pelo padrão H.264/AVC. Tal informação é importante, pois o *Deblocking Filter* no H.264/AVC constitui uma parte significativa da complexidade do decodificador. Como resultado, o DF do HEVC é menos complexo comparado ao do H.264/AVC, mas mantém a sua capacidade de melhorar a qualidade subjetiva e objetiva da imagem (NORKIN *et al.*, 2012).

Outro aspecto importante que recebeu atenção no HEVC foi a adequação para processamento paralelo. O DF no HEVC foi desenvolvido de forma a evitar dependências espaciais em toda a imagem o que, em conjunto com outras técnicas, facilita a paralelização.

Os efeitos de bloco são mais facilmente identificados pelo sistema visual humano quando os dois lados da borda são mais suaves e é mais difícil de ser percebido quando o sinal tem uma variação brusca. Além disso, se o sinal que ultrapassa a borda do bloco é caracterizado por variações bruscas, torna-se difícil identificar se os valores presentes no sinal reconstruído são realmente um efeito de bloco ou apenas fazem parte do sinal original.

A Figura 6 apresenta um exemplo da aplicação do *Deblocking Filter* na sequência de vídeo *Race Horses*. A Figura 6 (a) corresponde ao vídeo codificado com o DF desligado, já a Figura 6 (b) representa o vídeo codificado com a aplicação do DF. Na figura da esquerda é possível observar, no recorte, em regiões de maior movimento, o efeito de bloco pronunciado. Já na figura da direita, onde houve a aplicação do filtro, observa-se a suavização destes contornos dos blocos.

(a) (b)

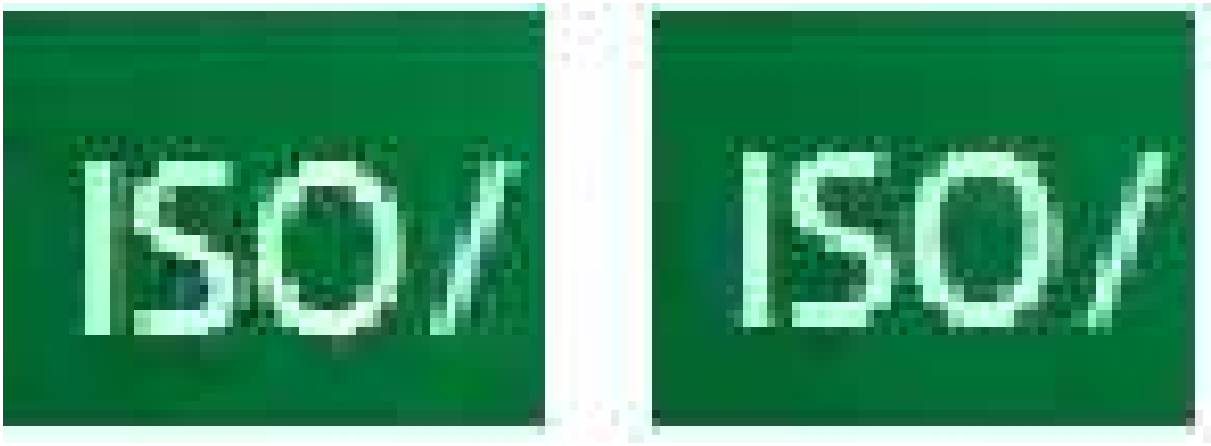
Figura 6 – Exemplo de aplicação do DF para a sequência *Race Horses*, (a) sem DF e (b) com DF.

3.2 *Sample Adaptive Offset*

O *Sample Adaptive Offset* (SAO), ou Deslocamento Adaptativo de Amostra, é uma das inovações trazidas pelo HEVC e é opcionalmente aplicado após o *Deblocking Filter*.

O SAO objetiva reduzir artefatos visuais indesejáveis, incluindo os chamados artefatos *ringing* (FU *et al.*, 2012), os quais aparecem visualmente como “fantasmas”, próximos às bordas de objetos. Os artefatos *ringing* podem tornar-se mais sérios quando são utilizadas transformadas maiores e filtros de interpolação da estimação de movimento fracionária com maior número de passos (FU *et al.*, 2012). O HEVC utiliza transformadas maiores e também filtros de interpolação com maior número de passos quando comparado com seu antecessor H.264/AVC, o que aumenta a relevância do SAO no contexto do HEVC.

A Figura 7 apresenta um exemplo da aplicação do SAO. A imagem à esquerda, Figura 7 (a), representa a imagem sem a aplicação do SAO e a imagem da direita, Figura 7 (b), mostra a mesma imagem após a aplicação do SAO. É possível notar uma melhora na qualidade subjetiva da imagem, onde algumas imperfeições são eliminadas. Esta figura foi gerada através da aplicação do software de referência à sequência *Slide Editing* com QP 37.



(a) (b)

Figura 7 – Exemplo da aplicação do SAO, (a) sem SAO e (b) com SAO.

Outro exemplo da aplicação do SAO encontra-se na Figura 8, para um recorte da sequência de teste *Race Horses*. A Figura 8 (a) representa o recorte da imagem não filtrada com o SAO. Já na Figura 8 (b) consta a imagem correspondente com a aplicação do SAO. Comparando as imagens, com e sem a aplicação do SAO, é possível perceber uma melhora da qualidade subjetiva da imagem através de uma suavização nos contornos dos objetos. T

(a) (b)

Figura 8 – Exemplo da aplicação do SAO, vídeo *RaceHorces*, (a) Sem SAO, (b) Com SAO.

O conceito utilizado pelo SAO é o de classificar as amostras do quadro reconstruído em diferentes categorias e então reduzir a distorção com uma simples

adição de *offset* para cada categoria de amostras. A intensidade da amostra e as propriedades das bordas são informações utilizadas para a classificação. Estas informações estatísticas são coletadas durante a aplicação do DF e então a decisão dos parâmetros do SAO podem ser feitas com a simplificação da taxa de distorção (FU *et al.*, 2011).

A aplicação do SAO divide a imagem em regiões, alinhadas às CTUs para fins de redução de complexidade. A cada região pode ser aplicado um deslocamento de banda, o *Band Offset* (BO), um deslocamento de borda, o *Edge Offset* (EO), ou ainda nenhum deles (FU *et al.*, 2011). O BO classifica as amostras com base em suas intensidades, agrupando amostras próximas. E, então, a este grupo então será aplicado um *offset* objetivando a redução da distorção. Desta forma, o BO acaba por ter maior benefício em regiões planas, onde os artefatos de banda tendem a aparecer (SULLIVAN *et al.*, 2012). Já o EO classifica as amostras de acordo com suas posições buscando identificar picos, vales, bordas côncavas e convexas, a fim de reduzir a distorção nestes pontos.

A Figura 9 apresenta um exemplo de divisão de um quadro em diversas CTUs, onde linhas pontilhadas indicam as bordas das CTUs. A aplicação ou não do SAO com seus métodos de classificação BO e EO, também está indicada. Torna-se importante salientar que nas regiões marcadas como OFF, não são utilizados nenhum dos métodos BO e EO.

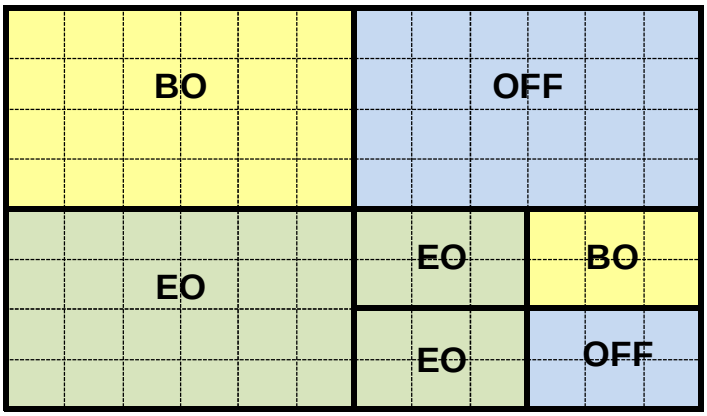


Figura 9 – Exemplo da divisão de um quadro em regiões.(FU *et al.*, 2011)

Nos dois tipos de SAO, EO e BO, para todas as suas categorias, o *offset* inicial é calculado como uma média das diferenças entre as amostras originais e

reconstruídas daquela determinada categoria. Este cálculo implica na necessidade do uso de um divisor completo para efetuar o cálculo da média.

A origem do SAO encontra-se na proposta apresentada pela *Samsung Eletronics* e pela *British Broadcasting Corporation* (BBC) na primeira reunião do JCT-VC (FU *et al.*, 2011). Esta proposta trouxe inicialmente dois métodos de classificação, o *Extrem Correction* (EXC), que é o predecessor do EO, e o *Band Correction* (BDC) que é o predecessor do BO. A principal diferença em comparação ao SAO atual é que os ambos os métodos, EXC e BDC, eram aplicados a todas as amostras e, de forma independente, como se fossem dois filtros diferentes (MCCANN, HAN e KIM, 2010).

3.2.1 *Edge Offset* (EO)

O EO classifica as amostras de uma região em múltiplas categorias, comparando o valor de cada amostra com seus vizinhos. A classificação é feita conforme os padrões mostrados na Figura 10.

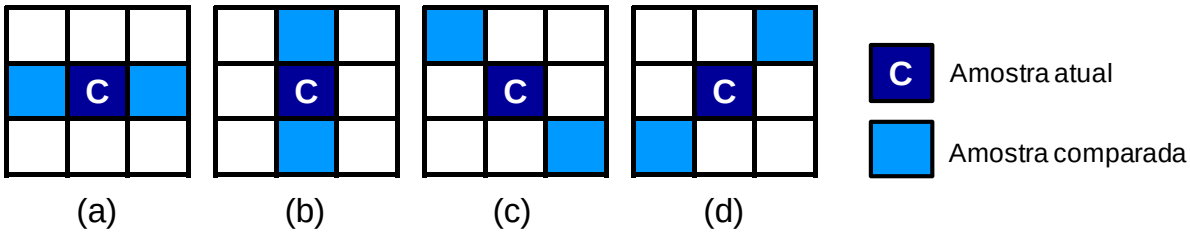


Figura 10 – Padrões de classificação de amostras EO, (a) 0° (b) 90° (c) 135° (d) 45°.

Quando o método EO é utilizado, um dos quatro padrões apresentados na Figura 10 pode ser escolhido para a classificação das amostras. Cada um dos padrões usa duas amostras vizinhas e as regras de classificação destes padrões são as mostrados na Tabela 1.

Tabela 1 – Categorias do EO.

Categoria	Condição
1	c < 2 vizinhos
2	c < 1 vizinho && c == 1 vizinho
3	c > 1 vizinho && c == 1 vizinho
4	c > 2 vizinhos
0	Nenhuma das condições acima

Uma amostra classificada nas categorias 1 ou 4 significa que esta corresponde a um mínimo local ou máximo local, respectivamente. Caso seja classificada nas categorias 2 ou 3, significa que a amostra corresponde a uma borda. Caso a amostra não se enquadre em nenhuma das condições anteriores, isto significa que se trata de uma área homogênea, e o SAO não será aplicado segundo a classificação do EO, mas pode ainda ser aplicado pelo BO.

A aplicação do EO, sem quaisquer restrições aos valores de *offsets*, pode introduzir alguns artefatos visuais, os quais geralmente aparecem como ruído sal e pimenta ou então como uma linha de contorno falso (KIM e KWON, 2013).

Com o objetivo de evitar este tipo de artefatos, foram propostas algumas restrições nos valores dos *offsets* em Kim e Kwon (2013), que posteriormente foram adotadas no padrão HEVC. Os *offsets* das categorias 1 e 2 foram restringidos à valores positivos, enquanto que nas categorias 3 e 4, os valores foram restringidos à faixa negativa. Esta restrição também reduz o número de bits necessários para codificar os *offsets*, visto que não é necessário codificar o sinal do *offset*, pois este pode ser inferido de acordo com o contexto, ou seja, a categoria a qual pertence. Vale destacar que tanto a classe, ou formato, de EO selecionado, quanto o *offset*, serão sinalizados no *bitstream* do vídeo codificado. A categoria selecionada não será sinalizada para o *bitstream*, pois esta classificação será efetuada também no decodificador.

O significado dos sinais utilizados no EO está ilustrado na Figura 11. Sinais positivos nas categorias 1 e 2 resultam em uma suavização, enquanto que valores negativos resultariam em maior nitidez. Já para as categorias 3 e 4, *offsets* com sinais negativos resultam em suavização, e valores positivos resultam em maior nitidez (FU *et al.*, 2012). Com base nisso, o EO utilizado no SAO do HEVC desabilita os valores que aumentam a nitidez e, assim, as categorias 1 e 2 somente podem ter *offsets* com valores positivos, e as categorias 3 e 4 somente podem ter *offsets* negativos. Desta forma, é possível transmitir apenas o valor absoluto do *offset* do EO, sendo o sinal derivado de forma implícita de acordo com a categoria.

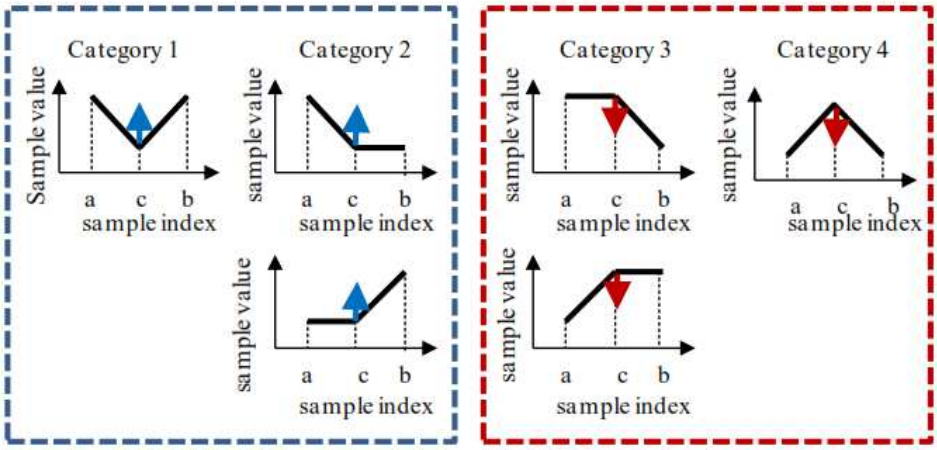


Figura 11 – *Offsets* positivos para categorias 1 e 2 e negativos para categorias 3 e 4.
Fonte: (FU *et al.*, 2012)

3.2.2 *Band Offset* (BO)

O *Band Offset* (BO) classifica as amostras de uma região com base na intensidade destas amostras. Estas amostras são divididas em múltiplas bandas, onde cada banda contém amostras do mesmo intervalo de intensidade. A faixa de intensidade é igualmente dividida em 32 intervalos entre 0 e o valor máximo de intensidade. Para o caso de representação utilizando 8 bits, o valor máximo seria 255 e os 256 valores seriam igualmente distribuídos entre os 32 categorias, ou bandas, onde a primeira categoria englobaria amostras com valores entre 0 e 7, a segunda valores entre 8 e 15 e assim por diante. A cada um destes intervalos será atribuído um *offset*, que será calculado como a média das diferenças entre amostras original e reconstruída daquele intervalo.

Desde sua inserção, na terceira versão do *Draft* do HEVC, até a sexta versão do *Draft*, as 32 bandas eram divididas em dois grupos, conforme ilustrado na Figura 12, e somente os *offsets* de um destes dois grupos, aquele com melhor resultado em termos de distorção, era escolhido para ser transmitido. Um dos grupos consistia das 16 bandas centrais e o outro das 16 bandas restantes.

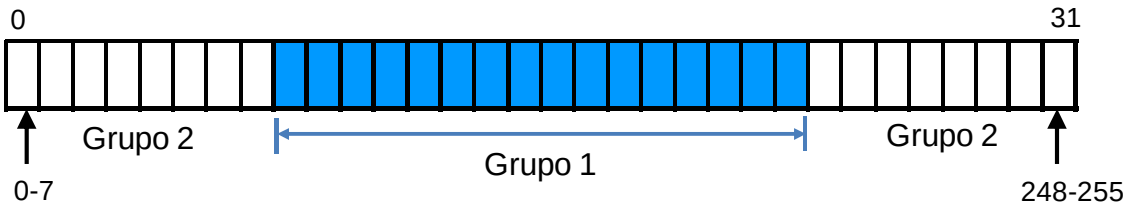


Figura 12 – *Band Offset* em dois grupos.

A partir da sétima versão do *Draft* (JCT-VC, 2012a)do HEVC houve uma mudança na escolha dos grupos. A partir deste ponto, o SAO BO passou a selecionar para transmissão apenas 4bandas adjacentes, dentre as 32 bandas disponíveis, conforme ilustrado na Figura 13.

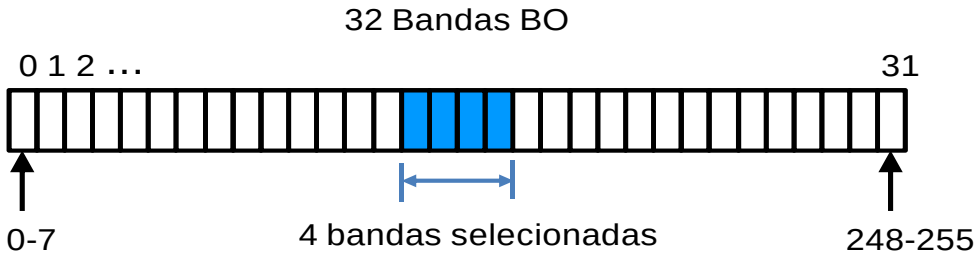


Figura 13 – *Band Offset* com 4 bandas selecionadas.

A principal razão para a redução de 16 para 4 bandas consecutivas reside no fato de que em áreas planas, onde os artefatos de banda tendem a aparecer, a maioria das amplitudes das amostras de uma região tende a estar concentrada em poucas bandas (SULLIVAN *et al.*, 2012). Além disso, a redução no número total de *offsets* a serem transmitidos fica reduzido de 16 para 4, diminuindo o *bitstream* resultante e igualando o número de *offsets* usados no EO.

A Figura 14 apresenta um gráfico que ilustra uma situação em que o BO poderia ser aplicado em uma determinada sequência de amostras. O eixo horizontal corresponde à posição da sequência de amostras e o eixo vertical corresponde à intensidade destas amostras. O eixo vertical encontra-se ainda dividido de acordo com suas intensidades, em 4 bandas do BO, k, k+1, k+2 e k+3. A curva pontilhada representa a variação dos valores das amostras originais, enquanto que a linha tracejada representa os valores das amostras reconstruídas. As amostras reconstruídas normalmente são distintas das amostras originais, pois passaram pelos processos de predição e quantização. No exemplo da Figura 14, as amostras

reconstruídas aparecem deslocadas em relação às amostras originais. Com o BO do SAO é possível aplicar um determinado *offset* a uma determinada banda, por exemplo, para a banda k+3, observa-se em toda sua extensão um deslocamento equivalente entre as amostras, portanto, a aplicação de um *offset* igual a todas as amostras desta banda resultaria em uma aproximação da curva tracejada à curva pontilhada.

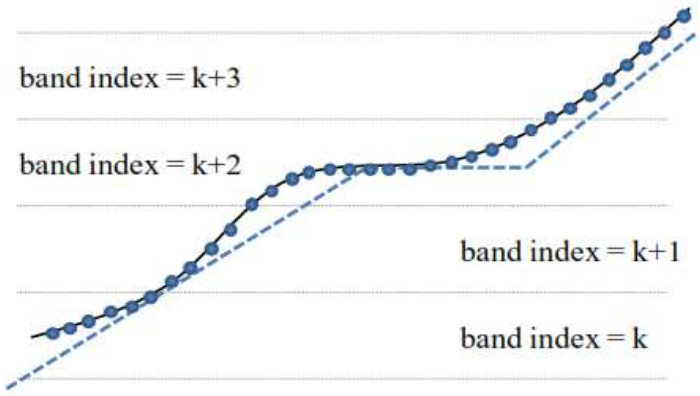


Figura 14 – Exemplo de aplicação do BO.
Fonte: (FU *et al.*, 2012)

3.2.3 Função de Custo

A função de custo utilizada internamente no SAO aplica heurísticas para estimar a melhor relação custo-benefício para decisão entre as variações do SAO. Esta função é aplicada para decidir qual o melhor *offset* para cada tipo e categoria.

Na Figura 15 consta um pseudo-algoritmo que representa a função de custo interna usada no SAO. Previamente ao processo da função interna de custo, serão feitas a classificação das amostras de cada tipo de categoria do SAO, bem como a geração dos *offsets* iniciais para cada tipo e categoria. O processamento da função de custo começa com a limitação do *offset* inicial dentro do intervalo de -7 a 7. Em seguida, o *offset* será incrementado ou decrementado até chegar a zero. O custo de todos estes *offsets* iterados será avaliado e aquele com menor custo para determinado tipo e categoria será escolhido.

```
Para cada tipo_de_SAO
   Para cada categoria_do_tipo
         Se existem amostras neste tipo e categoria
               iniOffset = accumDif / Count
               Clip (ini_offset, -7, 7)
               While iniOffset != 0
                     //Cálculo do custo de iniOffset
                     //Armazena melhor custo e offset
                     //Incrementa/decrementa iniOffset
```

Figura 15 – Pseudoalgoritmo da função de custo do SAO.

A função de custo utilizada internamente no SAO aplica heurísticas para estimar a distorção e o número de bits necessários para representar as informações do filtro, isto é, o *offset*, o tipo e a categoria alvo.

A heurística apresenta dois passos distintos. O primeiro é estimativa da distorção gerada pelo *offset* para cada tipo e categoria do SAO. Este passo está representado pela equação (1), onde '*n*' corresponde ao número de ocorrências de amostras na categoria que está sendo analisada, '*a*' representa à diferença entre amostra original e reconstruída acumulada e '*b*' ao *offset*.

$$d = (nb^2) - (2ab) \tag{1}$$

O segundo passo trata do cálculo do custo propriamente dito, o qual depende da estimativa do número de bits necessários para escrever a informação correspondente. Esta estimativa considera o *offset* atual e o tipo de SAO que esta sendo analisado. O resultado deste passo será igual ao módulo do *offset* atual e, se o tipo de SAO atual for BO, este resultado será acrescido de uma unidade. Isto se deve ao fato de quando o tipo selecionado for BO, o filtro precisa transmitir também o sinal do *offset*, diferentemente do EO que transmite apenas o valor absoluto do *offset* e o sinal é derivado a partir do contexto.

A equação final de custo está representada em (2), onde '*c*' representa o custo resultante, '*d*' a distorção calculada em (1), λ o multiplicador de Lagrange e '*r*' representa o número de bits estimado.

$$\begin{aligned} c &= d + r.\lambda, && \text{se tipo} = \text{EO} \\ c &= d + (r+1).\lambda, && \text{se tipo} = \text{BO} \end{aligned} \tag{2}$$

3.3 Adaptive Loop Filter

O *Adaptive Loop Filter* (ALF), ou Filtro Adaptativo em Laço, foi desenvolvido com base em filtros de Wiener, um modelo matemático cujo objetivo principal é minimizar o Erro Quadrático Médio, ou *Mean Square Error* (MSE), entre dois sinais, no caso, entre a imagem original e a imagem reconstruída (PARK, CHOI e JANG, 2012). Ele objetiva melhorar a qualidade subjetiva da imagem, promovendo uma suavização adicional ao resultado das demais etapas do codificador de vídeo.

A Figura 16 apresenta um exemplo da aplicação do ALF. Em (a) consta a imagem não filtrada pelo ALF e em (b) a imagem filtrada. É possível observar uma suavização geral da imagem.

(a) (b)

Figura 16 – Exemplo de aplicação do ALF no vídeo *riverbed*, (a) sem ALF e (b) com ALF.

A primeira implementação do ALF baseava-se ainda no H.264/AVC e utilizava formatos de filtros quadrados nos tamanhos 5x5, 7x7 e 9x9 (BUDAGAVI, SZE e ZHOU, 2011).

No início do desenvolvimento do HEVC, como resposta à chamada de propostas, o *Call of Proposals* (CfP), foram apresentadas um total de 27 propostas de técnicas ou algoritmos para compor o novo padrão de compressão de vídeo, onde aproximadamente três quartos destas, aplicavam o ALF de alguma forma (CHEN *et al.*, 2012). Assim, o ALF foi adotado já no *Test Model under Consideration* (TMuC) e fez parte do HEVC desde as suas primeiras versões do *Draft*.

Durante o desenvolvimento do HEVC, o ALF foi amplamente estudado pelo JCT-VC e diversas variações e alternativas foram propostas, buscando, em especial,

a redução da complexidade computacional. Neste processo, alcançou-se um significativo progresso em termos de aumento eficiência de codificação e redução de complexidade (JCT-VC, 2012c).

Todavia, na décima primeira reunião do grupo, em Estocolmo, mesmo com os ganhos obtidos, o ALF não foi mantido pelo JCT-VC como uma das ferramentas do perfil *Main* do HEVC (SULLIVAN *et al.*, 2012). Portanto, a partir do HM 8.0, o ALF foi removido.

Com a retirada do ALF do *Working Draft* e do HM surgiram, nas reuniões seguintes, diversas propostas que contemplam a reinserção do ALF no HM, com inúmeras alternativas de simplificações e sugestões de inclusão em um novo perfil (JCT-VC, 2012c).

Apesar dos avanços apresentados nas novas propostas de redução da complexidade computacional, ele ainda não foi considerado satisfatório, em termos custo-benefício, devido à grande quantidade de cálculos e multiplicações que o mesmo demanda. O fator determinante é que o ALF está localizado imediatamente antes do quadro reconstruído ser armazenado como referência e, portanto, precisará estar presente também no decodificador.

Há que se destacar que a remoção não se deu em função do ALF ter sido considerado uma ferramenta ineficiente, mas sim pela sua complexidade computacional elevada. Desta forma, estudos baseados no ALF ainda são de extrema relevância, uma vez que este filtro pode ainda ser integrado tanto em algumas extensões do HEVC, como o 3D-HEVC (MÜLLER *et al.*, 2013), por exemplo, quanto em futuros padrões de codificação de vídeos.

Outra evidência da relevância da continuidade dos estudos com o ALF é o fato deste filtro apresentar maiores reduções em *BD-rate* em sequências de vídeos com resoluções maiores (JCT-VC, 2012c), que tendem a ser cada vez mais utilizadas no mercado.

Este trabalho de mestrado teve início outubro de 2011, utilizando a versão 3 do software de referência, onde o ALF ainda fazia parte do padrão HEVC (em fase de desenvolvimento). Devido a isto e aos argumentos apresentados nos parágrafos anteriores, as investigações com o ALF foram mantidas e os resultados obtidos serão apresentados.

Nas subseções seguintes serão apresentados os principais conceitos relacionados ao ALF e suas variações durante o desenvolvimento do HEVC.

3.3.1 Formatos do ALF

Os núcleos do filtro ALF utilizaram diversos tamanhos e formatos entre as diversas versões do *Working Draft* e do HM. Nas primeiras implementações do ALF, que ainda focavam no H.264/AVC, eram utilizados formatos quadrados de tamanhos 5x5, 7x7 e 9x9.

Na primeira versão do *Working Draft*, o processo de filtragem do ALF fazia uso de múltiplos filtros 2-D em formato de diamante, considerando os mesmos três tamanhos da versão baseada no H.264/AVC, 5x5, 7x7 e 9x9, conforme ilustra a Figura 17. Este processo resultou em uma redução de aproximadamente 50% no volume de dados necessários para o cálculo de cada amostra filtrada em comparação aos formatos quadrados.

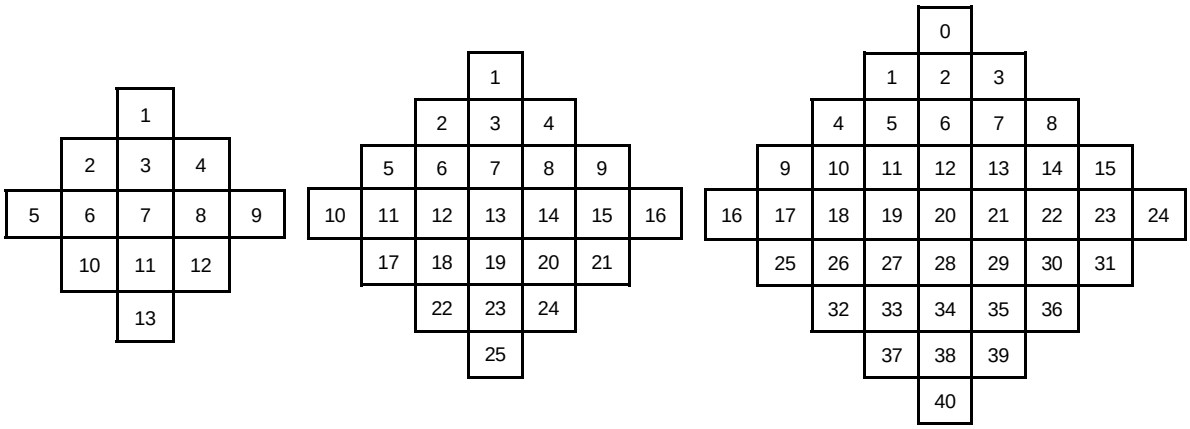
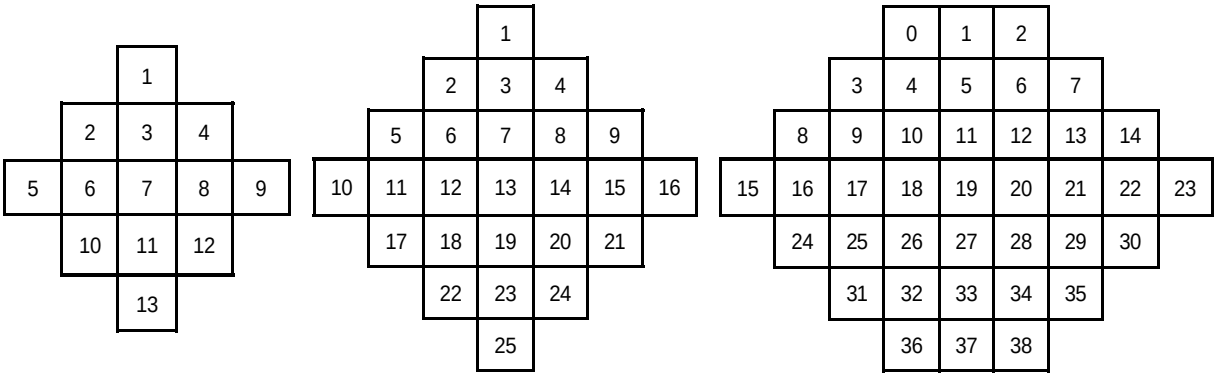


Figura 17 – Formatos do ALF no *Working Draft* 1, (a) 5x5, (b) 7x7 e (c) 9x9.

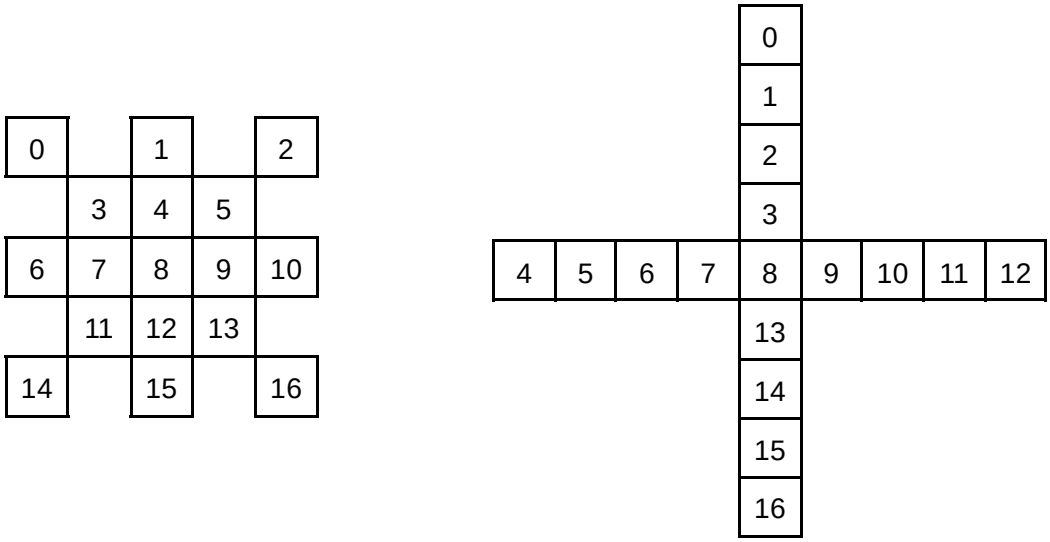
Na versão 3 do *Working Draft* (WIEGAND *et al.*, 2011) são suportados três tamanhos de filtros: 5x5, 7x7 e 9x9, entretanto, a máxima diferença vertical para o pixel atual é restrita a +/- 3, ou seja, o filtro que na versão anterior possui tamanho efetivo de 9x9, agora possui tamanho 9x7, apesar de continuar sendo chamado de 9x9. A Figura 18 apresenta os três formatos dos núcleos do filtro ALF no *Working Draft*3.



(a) (b) (c)

Figura 18 – Formatos do ALF no *Working Draft* 3, (a) 5x5, (b) 7x7 e (c) 9x9.

Na quinta versão do *Working Draft*, foi proposta a utilização de apenas dois formatos, e com um número menor de operações em comparação à versão anterior. Um dos formatos possui tamanho 5x5 e foi chamado *snowflake*, o outro, de tamanho 9x9, foi chamado *cross*. Estes formatos estão apresentados na Figura 19 (a) e (b).



(a) (b)

Figura 19 – Formatos do ALF no *Working Draft* 5, (a) *snowflake* e (b) *cross*.

Na versão 7 do *Working Draft* (JCT-VC, 2012a) e do HM (JCT-VC, 2012b) passou-se a utilizar apenas um formato de filtro, o qual corresponde a uma combinação entre um filtro quadrado 3x3 e um filtro *cross* 9x7, conforme Figura 20.

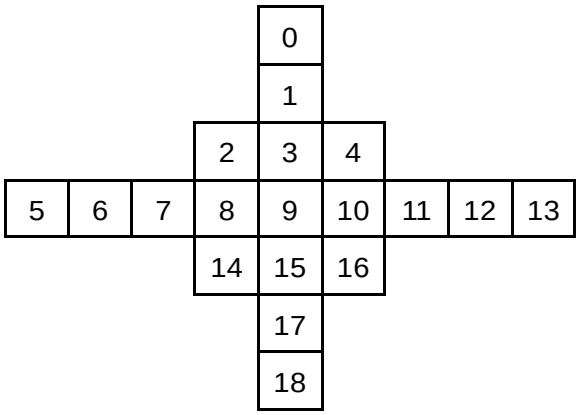


Figura 20 – Formato do ALF no *Working Draft* 7.

Este trabalho de mestrado foca nos formatos de filtro das versões 3 e 5 do *Working Draft*.

3.3.2 Filtro de Wiener no ALF

De maneira geral, o filtro de Wiener objetiva tornar uma amostra de entrada que passou por um processo de ruído, o mais próxima possível da amostra original. Para tanto, é necessário conhecer as características do sinal original e do ruído de forma a permitir a estimação de coeficientes para o filtro de Wiener que atendam ao seu objetivo de reduzir o ruído.

Neste sentido, o filtro de Wiener no ALF objetiva basicamente minimizar o Erro Quadrático Médio, ou *Mean Square Error* (MSE), entre a imagem original e a reconstruída, erro este que é causado pela quantização e pelas transformadas (PARK, CHOI e JANG, 2012).

Para fins de aplicação do filtro, algumas características do sinal e do ruído precisam ser conhecidas. Estas características correspondem à matriz de autocorrelação do quadro reconstruído e ao vetor de correlação cruzada entre o quadro reconstruído e o quadro original.

Considerando como base o filtro 5x5, a matriz de autocorrelação é gerada somando as amostras reconstruídas de forma simétrica, ou seja, as amostras correspondentes aos coeficientes de mesmo índice, para C_6, amostras *l* e *m*, para C_5, *j* e *k* da Figura 21 (b) e assim por diante, gerando um vetor intermediário de nove

posições. A última posição do vetor intermediário será a amostra central do bloco de amostras reconstruídas, pois não há amostra simétrica a esta.

Concluída a geração do vetor intermediário, a próxima etapa a geração da matriz de autocorrelação. Esta é gerada, através da multiplicação entre os valores do vetor intermediário em todas as suas combinações possíveis, sendo a posição na matriz indicada pelos índices de cada combinação. Como este processo é aplicado diversas vezes em regiões do quadro, que compartilham os mesmos coeficientes, o resultado na matriz é acumulado durante o processo de geração dos coeficientes para uma dada região, portanto, além da multiplicação de todas as combinações do vetor intermediário, o resultado será somado ao resultado anterior.

O passo seguinte é a geração do vetor de correlação cruzada, que é um processo semelhante ao da geração da matriz de autocorrelação. A primeira etapa é a geração do vetor intermediário de forma idêntica ao anterior, através da soma das amostras reconstruídas em posições simétricas. O vetor de correlação cruzada é será formado através da multiplicação entre a amostra original e cada elemento do vetor intermediário. Este processo, de maneira análoga ao anterior, também é aplicado a uma região com os mesmos coeficientes e, portanto, o valor precisa ser acumulado.

Após este processo, o filtro de Wiener para o ALF pode ser representado Equação (3),onde R_r é a matriz de autocorrelação, c o vetor de coeficientes e R_{dr} é o vetor de correlação cruzada.

$$R_r \,.\, c = R_{dr} \tag{3}$$

A Equação (3) pode ainda ser representada na forma matricial, conforme a Equação (4) (CONCEIÇÃO, 2014). A primeira parte da esquerda é matriz de autocorrelação do sinal a ser filtrado. Em seguida, está o vetor de coeficientes e, por fim, à direita, está o vetor correlação cruzada, entre o sinal desejado e o sinal a ser filtrado.

$$\begin{bmatrix} R_r[0] & R_r[1] & \cdots & R_r[N] \\ R_r[1] & R_r[0] & \cdots & R_r[N-1] \\ \vdots & \vdots & \ddots & \vdots \\ R_r[N] & R_r[N-1] & \cdots & R_r[0] \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} R_{dr}[0] \\ R_{dr}[1] \\ \vdots \\ R_{dr}[N] \end{bmatrix} \quad (4)$$

A fim de calcular o vetor de coeficientes, é necessário aplicar um método numérico de resolução de sistemas lineares, como o método de eliminação de Gauss ou Cholesky que é o utilizado no software de referência.

3.3.3 Cálculo da Amostra Filtrada

Concluída a etapa de geração dos coeficientes, descrita na seção 3.3.2, é possível proceder a aplicação do filtro sobre o quadro reconstruído.

Entretanto, antes da aplicação efetiva do filtro ALF é necessária uma etapa de preparação da imagem, onde o preenchimento de borda é realizado. Este preenchimento de borda é aplicado no nível de *slice*, com o objetivo de permitir a decodificação dos *slices* de forma independente. Se a borda da região que está sendo processada não coincidir com a borda do *slice*, o preenchimento será feito com as amostras das regiões vizinhas. Por outro lado, caso a borda da região alvo do processamento coincida com a borda do *slice*, o preenchimento será feito com uma extensão das amostras da borda da região através de cópia destes valores (JCT-VC, 2011a).

Após a preparação da imagem, é possível dar início ao processo de filtragem. Para calcular a amostra filtrada considera-se o formato do núcleo do filtro adotado e os coeficientes derivados pelo filtro de Wiener. Por exemplo, considerando as amostras e coeficientes apresentados na Figura 21, a amostra filtrada *a'*, ao centro do diamante, deve ser gerada a partir da multiplicação e acumulação entre as amostras e os coeficientes correspondentes no formato do filtro, isto é, aquelas que estiverem na mesma posição do diamante. A equação resultante desta etapa está descrita em (5).

Considerando a simetria dos coeficientes dos filtros, todos os coeficientes repetem-se, com exceção do coeficiente central. Desta forma, é possível alterar a

ordem das operações e realizar primeiramente a soma das amostras que serão multiplicadas pelo mesmo coeficiente e, só então, proceder a multiplicação. Esta alteração reduz o número de multiplicações necessárias em todos os formatos. No exemplo da Figura 21, a redução é de 13 para 7 multiplicações. A Equação resultante está demonstrada em (6).

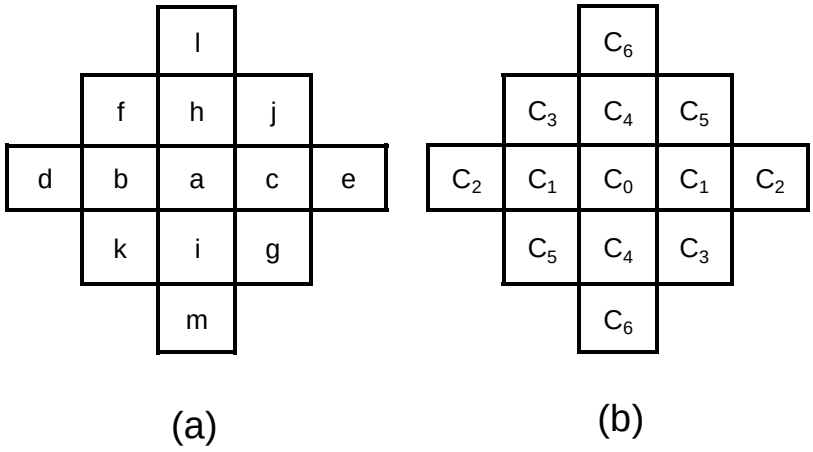


Figura 21 – Exemplo de geração da amostra filtrada pelo ALF.

$$a' = a * c_0 + b * c_1 + c * \ c_1 + d * c_2 + e * c_2 \ldots + l * c_6 + m * c_6 \quad (5)$$

$$a' = a * c_0 + c_1(b + c) + c_2(d + e) + \cdots + c_6(l + m) \quad (6)$$

Para os demais formatos, o cálculo é feito da mesma maneira, ou seja, é feita a multiplicação entre as amostras dos vídeos e os coeficientes do filtro que estiverem na mesma posição. Como a simetria dos coeficientes é mantida em todos os formatos, é possível fazer também a redução do número de multiplicações necessárias.

3.3.4 Método de Adaptação

Considerando que a geração dos coeficientes depende da presença do quadro original e este quadro somente esta presente no lado codificador, estes coeficientes precisarão ser transmitidos junto ao *bitstream* para que o processo de filtragem possa ser realizado no lado do decodificador.

Como o objetivo de reduzir o *overhead* de codificar múltiplos conjuntos de coeficientes, foram definidos dois métodos de adaptação, o baseado em bloco, *Block-based Adaptation* (BA), e o baseado em região, *Region-based Adaptation* (RA) (TSAI *et al.*, 2013). Estes métodos classificam as amostras de um quadro de acordo com características de textura, no caso do BA, e de acordo com a localização, no caso do RA. Para cada método, é possível classificar as amostras em 16 grupos e para cada grupo haverá um conjunto de coeficientes diferentes.

No método BA, as amostras são classificadas em blocos de tamanho 4x4 e de acordo com sua atividade laplaceana e direção. Todos os blocos 4x4 do quadro que estiverem na mesma classe compartilharão o mesmo conjunto de coeficientes (TSAI *et al.*, 2013).

Já no método RA, as amostras são dividas em 16 regiões fixas, retangulares, não sobrepostas e alinhadas às CTUs (TSAI *et al.*, 2013). Cada uma dessas regiões terá seu próprio conjunto de coeficientes.

Estes dois métodos eram usados simultaneamente e podiam ser escolhidos em nível de quadro. Entretanto, no HM7, apenas o método RA foi adotado.

3.3.5 Decisão de Filtro Ligado/Desligado

Como o ALF é aplicado antes do armazenamento do quadro reconstruído, que servirá de referência para os próximos quadros, o resultado do filtro será propagado para os quadros seguintes. Estes quadros seguintes terão referências com maior qualidade, aumentando, por consequência, a eficiência da codificação (YOO *et al.*, 2011). Entretanto, a ideia de aplicar o filtro a todas as amostras de todos os quadros pode não ser a mais adequada, devido à aplicação repetitiva que pode gerar uma super-suavização na imagem. Desta forma, o ideal é adotar uma técnica que controle regiões onde o filtro será ou não aplicado (CHEN *et al.*, 2012).

Nas reuniões dos JCT-VC, diversas alternativas para implementação do ALF foram propostas como, por exemplo, o ALF *Frame-based*, o ALF *Block-based* e o ALF *Quadtree-based*. O método *Frame-based* usa uma *flag* para indicar a decisão sobre se o filtro será aplicado ou não para todo o quadro (VCEG, 2008a). No método *Block-based* é utilizado uma *flag* para cada bloco que indica se o respectivo bloco deve ser filtrado ou não. Neste método, o bloco pode assumir tamanhos fixos de 8x8, 16x16, 24x24, 32x32, 64x64 ou 128x128 amostras. O *Block-based* melhora a

qualidade da imagem em relação ao *Frame-based*, pois a avaliação é feita para unidades menores (blocos) (VCEG, 2008b). No caso do método *Quadtree-based*, o comportamento é similar ao *Block-based*, mas baseado em estrutura de árvore quaternária, o que corresponde à estrutura da CTB.

3.3.6 Algoritmos de Aplicação do ALF

Para aplicar o filtro ALF, primeiramente o codificador precisa decidir quais unidades (bloco, por exemplo) do quadro serão filtradas e quais não serão. Para embasar esta decisão, um filtro simplificado é aplicado a todas as regiões. Então, a distorção obtida é comparada com a distorção da imagem sem aplicação do filtro. Aquelas regiões onde há uma melhora na distorção são marcadas como ON, e as demais como OFF. A filtragem completa aplica os três tamanhos de filtro (no caso do *Draft*3), sendo selecionado o filtro com melhor resultado. Esta etapa é feita apenas para as regiões marcadas como ON. Isto descreve o algoritmo chamado codificação em 16 passos (TSAI *et al.*, 2011).

Uma alternativa ao algoritmo de 16 passos, é o algoritmo em 1 passo (TSAI *et al.*, 2011). O algoritmo de codificação em 1 passo, reaproveita as marcações de regiões ON/OFF feitas nos quadros anteriores. O algoritmo é vantajoso pela tendência de similaridade entre os quadros vizinhos de um vídeo. Nas versões do HM que contemplavam o ALF, a definição de qual dos algoritmos é utilizado é feita pelo usuário no arquivo de configuração, e será sempre o mesmo para a codificação de todo o vídeo (TSAI *et al.*, 2011).

4 SIMPLIFICAÇÕES PROPOSTAS PARA O ALF E O SAO

Este capítulo apresenta as simplificações propostas neste trabalho para os filtros SAO e ALF do HEVC, bem como, as avaliações realizadas no software de referência do HEVC. O foco destas simplificações é reduzir a complexidade computacional envolvida, buscando uma futura implementação em hardware.

O processo de filtragem foi inserido nos padrões de codificação de vídeo com o objetivo principal de melhorar a qualidade subjetiva da imagem, ou seja, a qualidade percebida pelo expectador. Este também é o objetivo principal do *In-Loop Filter* do HEVC. Mesmo com este foco dos filtros na melhora na qualidade subjetiva, a qualidade objetiva, tipicamente medida com o PSNR, segue sendo a única métrica de qualidade utilizada nos processos internos ao codificador. Por conta disso, as simplificações propostas neste capítulo são avaliadas utilizando métricas objetivas, sempre correlacionando a qualidade objetiva (PSNR) com o *bitrate* atingido (taxa de compressão). Para tanto, foi usada a métrica *BD-rate* (BJONTEGAARD, 2001) que indica o percentual de incremento ou decremento do *bitrate* para uma mesma qualidade objetiva, considerando a versão original do algoritmo e a versão com modificações. Deste modo, toda a discussão sobre qualidade que se segue neste capítulo considera a qualidade objetiva (PSNR).

4.1 Simplificações para o SAO

O foco das simplificações propostas para o SAO neste trabalho está em dois pontos críticos para o desenvolvimento de hardware:dados em ponto flutuante e uso de multiplicadores e divisores completos.

O algoritmo do SAO no software de referência considera dados em ponto flutuante e o uso deste tipo de dados resulta em aumento no uso de recursos de hardware bem como reduz a performance. De maneira ilustrativa, é possível

considerar a proporção entre a implementação em hardware de operadores em ponto fixo e em ponto flutuante apresentada em (GOVINDU *et al.*, 2003), onde um somador em ponto fixo representa uma redução no uso de recursos de hardware de 78,5%.

Além disso, o algoritmo do SAO também usa multiplicadores e divisores completos para realizar a estimativa de custo que embasam suas decisões internas, outro fator que aumenta demanda pelo uso de recursos de hardware e reduz a performance.

É importante salientar que, no caso do SAO, estes dois pontos críticos aparecem combinados, ou seja, os multiplicadores completos são aplicados a dados em ponto flutuante, o que torna o desenvolvimento de hardware ainda mais complexo.

Em virtude disto, este trabalho propõe duas simplificações na função interna de custo do algoritmo do SAO. A primeira proposta objetiva usar dados com representação inteira ou com ponto fixo, ao invés de ponto flutuante.

A segunda proposta consiste em eliminar o uso de multiplicadores e divisores completos. Para tanto, é necessário aplicar a técnica de *loop unrolling* e realizar o cálculo do custo para as sete possíveis iterações com equações específicas e independentes. Nesta simplificação, aplica-se cada um dos possíveis *offsets* às equações para estimativa da distorção (1) e do custo (2). Com isso, é possível transformar as multiplicações completas em multiplicações com somas e deslocamentos, o que reduz a complexidade de uma implementação em hardware.

Esta simplificação também permitiu a eliminação do divisor completo que era necessário ao final da etapa de levantamento estatístico de cada tipo e categoria do SAO. Esta divisão era necessária para conhecer o *offset* inicial que seria iterado até que chegasse a 0 e de onde seria escolhido o *offset* com menor custo. Uma vez que o custo de todos os possíveis *offsets* serão calculados, não há necessidade de identificar qual será o primeiro e, assim, também não será necessário realizar a divisão.

O processo de implementação da técnica de *loop unrolling* considerou a substituição nas equações (1) e (2) da variável *offset* por todos os possíveis *offsets* que precisam ser iterados. Como primeiro passo, foi feita a substituição *offset* ‘b’

pelos *offsets* fixos equação de distorção indicada em (1). Desta forma, obtém-se como resultado, as 14 equações apresentadas em (7).

$$\begin{aligned} d_1 &= n - 2a & d_{-1} &= n + 2a \\ d_2 &= 4n - 4a & d_{-2} &= 4n + 4a \\ d_3 &= 9n - 6a & d_{-3} &= 9n + 6a \\ d_4 &= 16n - 8a & d_{-4} &= 16n + 8a \\ d_5 &= 25n - 10a & d_{-5} &= 25n + 10a \\ d_6 &= 36n - 12a & d_{-6} &= 36n + 12a \\ d_7 &= 49n - 14a & d_{-7} &= 49n + 14a \end{aligned} \tag{7}$$

Estas equações apresentadas em (7) podem ainda ser fatoradas de forma a transformar as multiplicações por constantes em multiplicações com somas e deslocamentos. O resultado da fatoração das 14 equações em (7) resulta nas 14 equações apresentadas em (8).

$$\begin{aligned} d_1 &= n - 2a & d_{-1} &= n + 2a \\ d_2 &= 4(n - a) & d_{-2} &= 4(n + a) \\ d_3 &= 2[2(2n - a) - a] + n & d_{-3} &= 2[2(2n + a) + a] + n \\ d_4 &= 8(2n - a) & d_{-4} &= 8(2n + a) \\ d_5 &= 8(2n + n - a) + n - 2a & d_{-5} &= 8(2n + n + a) + n + 2a \\ d_6 &= 8(4n - a) + 4(n - a) & d_{-6} &= 8(4n + a) + 4(n + a) \\ d_7 &= 16(2n + n - a) + n + 2a & d_{-7} &= 16(2n + n + a) + n - 2a \end{aligned} \tag{8}$$

Para a segunda etapa da função de cálculo de custo, que nada mais é do que o custo propriamente dito, representado em (2), seguiu-se o mesmo procedimento

de substituição de valores fixos. Contudo, neste caso, a substituição não é diretamente com o valor do *offset* fixo, mas em valores derivados. A variável substituída em (2) é a variável ‘r’ que diz respeito à estimativa de bits necessários para a escrita dos dados do SAO. Esta estimativa de bits corresponde ao valor do *offset* em módulo, sendo que este valor será acrescido de um, caso o tipo de SAO seja o BO, pois este tipo necessita armazenar também o sinal do *offset* e não somente o seu valor absoluto. Como resultado desta substituição obtém-se as equações em (9).

$$
\begin{aligned}
c_{1\ ou\ -1} &= d_{1\ ou-1} + \lambda, \ \text{se EO} \\
c_{1\ ou\ -1} &= d_{1\ ou-1} + (\lambda \ll 1), \ \text{se BO} \\
c_{2\ ou\ -2} &= d_{2\ ou-2} + (\lambda \ll 1), \ \text{se EO} \\
c_{2\ ou\ -2} &= d_{2\ ou-2} + (\lambda \ll 1) + \lambda, \ \text{se BO} \\
c_{3\ ou\ -3} &= d_{3\ ou-3} + (\lambda \ll 1) + \lambda, \ \text{se EO} \\
c_{3\ ou\ -3} &= d_{3\ ou-3} + (\lambda \ll 2), \ \text{se BO} \\
c_{4\ ou\ -4} &= d_{4\ ou-4} + (\lambda \ll 2), \ \text{se EO} \\
c_{4\ ou\ -4} &= d_{4\ ou-4} + (\lambda \ll 2) + \lambda, \ \text{se BO} \\
c_{5\ ou\ -5} &= d_{5\ ou-5} + (\lambda \ll 2) + \lambda, \ \text{se EO} \\
c_{5\ ou\ -5} &= d_{5\ ou-5} + (\lambda \ll 2) + (\lambda \ll 1), \ \text{se BO} \\
c_{6\ ou\ -6} &= d_{6\ ou-6} + (\lambda \ll 2) + (\lambda \ll 1), \ \text{se EO} \\
c_{6\ ou\ -6} &= d_{6\ ou-6} + (\lambda \ll 3) - \lambda, \ \text{se BO} \\
c_{7\ ou\ -7} &= d_{7\ ou-7} + (\lambda \ll 3) - \lambda, \ \text{se EO} \\
c_{7\ ou\ -7} &= d_{7\ ou-7} + (\lambda \ll 3), \ \text{se BO}
\end{aligned}
\tag{9}
$$

Esta manipulação reduz substancialmente a complexidade de implementação em hardware, uma vez que, o deslocamento pode ser feito apenas com conexões de fios e a implementação de um somador é bem mais simples que um multiplicador. Para avaliar os impactos das simplificações, foram realizados experimentos no software de referência do HEVC, o HM.

4.1.1 Avaliações em Software para as Simplificações no SAO

A fim de verificar o funcionamento e as consequências da aplicação das simplificações propostas ao SAO, foram feitas modificações no software de referência, implementando tais simplificações. Os resultados de *BD-rate* (BJONTEGAARD, 2001) foram comparados com os obtidos na execução do software de referência original. Um aumento no valor do *BD-rate* significa que haverá um aumento no número de bits necessários para representar um vídeo mantendo a mesma qualidade.

4.1.1.1 Avaliação da Simplificação com Inteiros e Ponto Fixo

Esta simplificação considera a substituição de dados em ponto flutuante por dados inteiros e dados em ponto fixo com precisão fracionária variando entre 1 e 8 bits. A avaliação do impacto desta proposta foi feita através de modificações aplicadas ao software de referência, mais especificamente na função de custo interna do SAO.

As avaliações foram feitas no HM 10 (KIM *et al.*, 2013) para os 16 primeiros quadros de 19 sequências de vídeo. Para cada uma destas sequências, foram consideradas as configurações: *Intraonly*, *Randomaccess* e *Lowdelay*, de acordo com o definido nas Condições Comuns de Teste, ou *Common Test Conditions* (CTCs) (BROSSEN, 2013) e os parâmetros de quantização, ou *Quantization Parameter* (QP) de 22, 27, 32 e 37. Para as simulações considerando as precisões de 5, 6 e 7 bits, foram feitas simulações apenas para 3 sequências de vídeo e mantendo as demais condições.

Considerando todas as execuções do software de referência para a realização destas avaliações, incluindo o HM original, foram realizadas 1.416 execuções do software de referência, o que totalizou um volume de 22.656 quadros avaliados.

A Tabela 2 apresenta os resultados médios de *BD-rate* da avaliação usando dados inteiros e usando dados com precisão fracionária em ponto fixo, variando esta precisão entre 1 e 8 bits. Os dados para as precisões fracionárias de 5, 6 e 7 bits foram omitidas de tabela, pois a simulação não foi executada para todas as sequências e suas comparações serão apresentadas em separado.

Tabela 2 – Resultados para a simplificação com dados inteiros e ponto fixo.

	BD-rate			
	Intra Only	***Low Delay***	***Random Access***	**Média**
Inteiros	0,0121	0,0999	0,1156	0,0759
1 bit	0,0011	-0,0089	0,1036	0,0319
2 bits	-0,0000	-0,0171	-0,0026	-0,0066
3 bits	-0,0001	-0,0127	-0,0046	-0,0058
4 bits	0	-0,0129	-0,0036	-0,0055
8 bits	0	0,0021	-0,0035	-0,0005

Nesta simplificação, os resultados mostraram que o uso de dados inteiros em substituição aos dados em ponto flutuante representou um acréscimo de 0,076*BD-rate*, o que significa que utilizando esta proposta, é possível manter a mesma qualidade de um vídeo, com um acréscimo de apenas 0,076% no *bitrate*.

Considerando dados com precisão em ponto fixo, a partir de 1 bit fracionário, observa-se que na configuração *Intra Only* a diferença encontra-se a partir da terceira casa decimal, em média. Já para as demais configurações, há uma variação de valores um pouco mais significativa, mas ficando ainda a partir da segunda casa decimal. Em alguns casos, há até redução em relação ao original. O Apêndice A deste trabalho traz a planilha completa com os resultados por vídeo.

A variação mais significativa que ocorreu no uso de dados inteiros pode ser explicada devido à necessidade de adaptação dos dados em ponto flutuante para dados inteiros, onde é necessário eliminar parte da informação. Entretanto, este acréscimo de apenas 0,08% no *BD-rate* ao eliminar-se a complexidade do uso desses dados em ponto flutuante, é vantajoso e esta solução pode ser considerada em uma futura implementação em hardware.

Contudo, se a demanda da implementação for por uma precisão maior, observa-se que, no momento em que se começa a inserir bits de precisão fracionária

nestes dados, a diferença tende a ser reduzida. Desta forma, uma solução usando precisão de 2 bits fracionários passaria esta diferença para a terceira casa decimal, sendo que esta diferença é para menos, ou seja, representa ainda uma redução em relação original.Caso o objetivo seja uma precisão maior ainda, é possível observar que, em média, com 8 bits esta diferença, ainda que sendo redução, é ainda menor, ficando na quarta casa decimal.

É importante salientar que, em vários casos, a solução aqui proposta apresentou uma redução no *BD-rate* em relação à execução normal do HM. Mesmo considerando dados inteiros, em alguns vídeos, esta redução é identificada. E, em média, esta redução aparece a partir da precisão de 2 bits. Este fato é justificável, pois o SAO usa heurísticas para suas decisões internas, conforme descrito na seção 3.2.3, e estas heurísticas não são perfeitas. Portanto, uma restrição de uma funcionalidade do algoritmo que, localmente não representa redução, pode levar a uma situação não testada e que gera esta redução.

Entretanto, cabe ressaltar que, embora seja um ganho em eficiência do algoritmo, esta diferença representa um erro em relação ao algoritmo original. Por outro lado, observa-se que, conforme se aumenta o número de bits da precisão, a tendência é que este erro seja ainda mais reduzido.

Tendo em vista o volume de simulações, as versões com precisões de 5, 6 e 7 bits fracionários não foram executadas para todas as sequências. Esta decisão baseou-se no fato de que com a execução para 3 vídeos, o comportamento nesta faixa de precisão demonstrou-se mais linear, conforme é possível observar na Tabela 3, onde tanto para a configuração *Intra Only* quanto para a *Random Access* não houve mudança nos resultados deste a versão com precisão 4 bits. E para a configuração *Low Delay* a estabilização deu-se a partir da versão com precisão de 5 bits.

Tabela 3 – Resultados comparativo para todas as precisões em 3 vídeos.

	BD-rate			
	Intra Only	***Low Delay***	***Random Access***	**Média**
Inteiros	-0,0403	-0,0869	0,1517	**0,0082**
1 bit	-0,0003	0,0049	-0,0339	**-0,0098**
2 bits	-0,0006	0,0527	-0,0012	**-0,0182**
3 bits	-0,0002	0,0196	-0,0300	**-0,0166**
4 bits	0	0,0214	-0,0293	**-0,0171**
5 bits	0	0	-0,0293	**-0,0098**
6 bits	0	0	-0,0293	**-0,0098**
7 bits	0	0	-0,0293	**-0,0098**
8 bits	0	0	-0,0293	**-0,0098**

4.1.1.2 Avaliação da Simplificação *Loop Unrolling*

A segunda simplificação proposta foi a aplicação da técnica de *loop unrolling* e a consequente substituição dos multiplicadores completos por somas e deslocamentos. Esta simplificação foi também avaliada através de modificações no software de referência e da comparação desta versão com a execução normal do HM.

A Tabela 4 apresenta os resultados destas avaliações comparando com a versão original do HM e com a primeira proposta de simplificação. A primeira linha corresponde à comparação entre a versão com *loop unrolling* e a execução normal do HM, com a aplicação do SAO. Neste caso, a técnica de *loop unrolling* foi aplicada diretamente ao software de referência, ainda considerando os dados em ponto flutuante usados na versão original. A única alteração foi ao invés de calcular o *offset* inicial e proceder as iterações a partir deste, as iterações passaram a começar sempre em 7 ou -7, conforme o caso.

Observa-se que, para as configurações *Intra Only* e *Low Delay*, em todas as sequências, o resultado das simulações foi igual ao original. Já para a configuração *Random Access*, foi identificado que, em média, o resultado da proposta de *loop unrolling* conseguiu uma redução no *BD-rate*. Esta média deve-se, na verdade, ao resultado em apenas uma das sequências. Nas demais, o comportamento foi igual à versão original.

Tabela 4 – Resultados para a simplificação *loop unrolling*.

	BD-rate			
	Intra Only	***Low Delay***	***Random Access***	**Média**
HM	0	0	-0,0031	-0,0010
Inteiro	0,0005	0,0161	0,1473	0,0546
1 bit	-0,0006	-0,0148	0,1075	0,0311
2 bits	-0,0013	-0,0181	-0,0007	-0,0063
3 bits	-0,0001	-0,0127	-0,0045	-0,0057
4 bits	0	-0,0129	-0,0041	-0,0053
8 bits	0	0,0021	-0,0035	-0,0005

Além desta comparação direta com o HM, foram feitas análises comparativas das versões que combinam a utilização de dados inteiros e ponto fixo com a utilização do *loop unrolling*. Todas estas comparações foram feitas em relação à versão com o algoritmo original do SAO e estes dados estão apresentados nas linhas seguintes da Tabela 4.

Considerando a comparação utilizando dados inteiros e com ponto fixo com precisão fracionária de 1 bit, houve um acréscimo de 0,055% e 0,034% no *BD-rate,* respectivamente. Entretanto, estes valores devem-se a uma única sequência que apresentou um valor mais elevado na configuração *Random Access*. Os valores completos encontram-se no Apêndice B deste trabalho.

A partir do uso de dados em ponto fixo com precisão de 2 bits, os valores médios passam a representar reduções e, da mesma maneira que ocorrido com a primeira simplificação, mesmo representando um ganho na eficiência da codificação, com o aumento da precisão, há uma tendência a ser reduzida a diferença.

Ao analisar a tabela completa, no Apêndice B, verifica-se que, conforme se aumenta a precisão fracionária utilizada, aumenta também o número de sequências e configurações que convergem para o mesmo resultado, ou seja, as simplificações propostas acabam alcançando o mesmo resultado da versão original.

A partir da Tabela 4 observa-se também que nas precisões fracionárias entre 0 e 8 bits, os valores médios são bastante semelhantes aos valores constantes da Tabela 2, que apresenta os resultados para ponto fixo apenas, sem a aplicação do *loop unrolling*. Em sua maioria a equivalência não é perfeita e tal situação pode ser explicada em função de, nestes casos, a avaliação de um número maior de *offsets*

do que o estabelecido pelo HEVC, acaba por resultar em uma pior performance e isto demonstra que a técnica utilizada para a estimativa de custo interna do SAO não se alinha perfeitamente com o módulo externo de decisão.

Percebe-se ainda que, com o aumento da precisão, mais próximos ficam estes valores, até chegar a precisão de 8 bits, onde os valores são iguais. Neste caso, os valores iguais não são coincidência, analisando as tabelas completas nos Apêndices A e B, verifica-se que os valores que geram a média igual, ocorrem nos mesmos vídeos. Desta foram, conclui-se que a introdução da técnica de *loop unrolling* não trouxe qualquer prejuízo à execução do algoritmo do SAO e traz os benefícios de eliminar a necessidade de multiplicadores e divisores completos para uma futura implementação em hardware.

A solução com *loop unrolling,* visto que elimina o uso de multiplicadores e divisores, e com ponto fixo em precisão de 2 bits representa uma solução para uma arquitetura em hardware que, em média, resulta em reduções no *BD-rate* sobre o algoritmo original do SAO. Entretanto, buscando-se uma precisão ainda maior, onde a grande maioria dos casos a execução torna-se equivalente à original, a solução mais adequada seria a versão que utiliza *loop unrolling* e precisão de 8 bits, o que resulta em uma redução de 0,0005% no *bitrate* necessário para representar um vídeo com a mesma qualidade.

4.2 Simplificações para o ALF

Como descrito na seção 3.3, o ALF é baseado em operações matemáticas complexas e, portanto, consome significativo tempo de processamento em relação às demais etapas da codificação e decodificação. O ALF sozinho é responsável por 5% a 18% do tempo de codificação (CHEN *et al.*, 2012).

Com o objetivo de uma futura implementação em hardware do filtro ALF, um dos pontos que precisa ser atacado é novamente a utilização de dados em ponto flutuante, pois este tipo de dado torna o processamento ainda mais complexo e uma implementação em hardware desta solução torna-se mais custosa em termos de consumo de área e de tempo de processamento.

Entretanto, para permitir uma melhor compreensão dos impactos gerados pela simplificação aqui proposta, foi analisado inicialmente o impacto da não aplicação do filtro ALF a uma sequência de vídeos. Desta forma, pretende-se

analisar o quanto a simplificação poderia comprometer o funcionamento do filtro. Para tanto, foram realizados experimentos com o software de referência considerando a diretiva que ativa ou desativa a aplicação do ALF.

Para os experimentos, foi utilizado o HM 5.0 e a execução foi realizada para 13 sequências de vídeos: *Blowing Bubbles*, *BQ Square*, *Race Horses*, *Basketball Drill*, *Basketball Drill Text*, *Vidyo1*, *Vidyo3*, *Vidyo4*, *Cactus*, *Kimono*, *People on Street*, *Steam Locomotive Train* e *Traffic*. Para todos os vídeos foram considerados os parâmetros de quantização 22, 27, 32 e 37, totalizando 52 execuções do software de referência na versão aqui proposta. Foram realizadas ainda as simulações considerando a aplicação do ALF e a não aplicação do ALF, totalizando 156 execuções. Considerando que foram executados 80 quadros em cada simulação, o volume total de quadros analisados em cada versão foi de 4.160 e para as três versões, 12.480 quadros.

A Tabela 5 apresenta os resultados destes experimentos, onde constam os resultados em *BD-rate* individualmente por vídeo e para o grupo de vídeos por resolução. Em todos os casos houve um acréscimo no *BD-rate* quanto o algoritmo do ALF não é aplicado. Analisando individualmente os vídeos, observa-se uma variação de valores entre menos de 1% até pouco mais de 7%, no entanto, a média para vídeos com resolução maior, de 1600p, também é maior, chegando a 5,25%. A média geral entre todos os vídeos simulados chegou a 3,58%, o que significa que para manter a mesma qualidade do vídeo, há necessidade de um aumento 3,58% na quantidade de bits necessários para representá-lo. Isto demonstra que, o ALF tende a ter maior impacto quando se utiliza vídeos com resoluções maiores, uma das demandas crescentes do mercado.

Desta forma, os resultados apresentados na Tabela 5 demonstram que o ALF é uma ferramenta que alcança resultados interessantes em termos de aumento da qualidade objetiva para uma mesma taxa de bits, embora ainda seja considerada computacionalmente complexa. Conforme mencionado na seção 3.3, já existem propostas para a aplicação do ALF em um novo perfil ou em extensões do padrão HEVC e, assim, simplificações neste algoritmo tornarão ainda mais provável a efetivação destas propostas.

Tabela 5 – Impacto da não aplicação do ALF (em *BD-rate*).

Resolução	Vídeo	*BD-rate*	Média
240p	*Blowing Bubbles*	0,6263	**3,2530**
	BQ Square	7,3090	
	Race Horses	1,8238	
480p	*Basketball Drill*	1,5238	**1,4620**
	Basketball Drill Text	1,4001	
720p	*Vidyo1*	2,6534	**3,8824**
	Vidyo 3	5,8468	
	Vidyo 4	3,1471	
1080p	*Cactus*	3,3717	**3,2609**
	Kimono	3,1502	
1600p	*People on Street*	5,0695	**5,2476**
	Steam Locomotive Train	7,3593	
	Traffic	3,3141	
Média		**3,5842**	

Neste trabalho, propõe-se, uma simplificação que considera a utilização de dados inteiros em substituição aos dados em ponto flutuante, originalmente usados pelo software de referência do HEVC. Para avaliar o impacto desta proposta, foram feitos experimentos utilizando o software de referência do HEVC e, assim como no SAO, também foi usado o *BD-rate* como métrica de comparação entre a execução regular do software de referência e a versão com a simplificação. Para estes experimentos, foram levados em conta os mesmos parâmetros utilizados para a construção da Tabela 5.

De acordo resultados dos experimentos, apresentados na Tabela 6, é possível observar que, para a maioria dos vídeos, houve um pequeno aumento no *BD-rate*, ou seja, para que os vídeos possam manter a mesma qualidade, há necessidade de um pequeno acréscimo no número de bits necessários a sua representação. Contudo, também se observa que, em alguns vídeos, houve ainda uma redução deste *BD-rate*. Isto é, mesmo removendo toda a complexidade relacionada aos cálculos com dados em ponto flutuante, a simplificação proposta conseguiu reduzir o número de bits necessários para a representação de um vídeo com a mesma qualidade.

Outro ponto importante a ser observado nestes dados é o resultado médio geral e por grupo de resolução. Percebe-se que, para resoluções menores, entre 240p e 1080p, o acréscimo médio no *BD-rate* fica entre 0,04% e 0,08%, entretanto, para resoluções maiores, 1600p, o impacto foi ainda menor, alcançando apenas 0,02%. Já para a média geral, os resultados demonstram que a substituição de dados em ponto flutuante por dados inteiros no ALF ocasiona um aumento no *BD-rate* de aproximadamente 0,05%. Considerando o custo computacional para a realização dos cálculos em ponto flutuante, o prejuízo para a taxa de bits é bastante reduzido.

Tabela 6 – Impacto da utilização de dados inteiros no ALF.

Resolução	Vídeo	BD-rate	Média
240p	*Blowing Bubbles*	0,0155	**0,0397**
	BQ Square	-0,0033	
	Race Horses	0,1068	
480p	*Basketball Drill*	0,0109	**0,0515**
	Basketball Drill Text	0,0920	
720p	*Vidyo1*	0,1309	**0,0790**
	Vidyo 3	0,0301	
	Vidyo 4	0,0760	
1080p	*Cactus*	-0,0327	**0,0518**
	Kimono	0,1364	
1600p	*People on Street*	-0,0376	**0,0191**
	Steam Locomotive Train	0,0277	
	Traffic	0,0671	
Média		0,0477	

Observou-se ainda que, embora a utilização de dados inteiros, em média, apresente um aumento do número de bits necessários para representar um vídeo mantendo sua qualidade, este aumento ainda é muito menor do que o aumento quando o ALF não é aplicado.

Para melhor avaliar a comparação entre a solução proposta neste trabalho, ALF com dados inteiros, e a utilização do codificador do HEVC sem o ALF, foi elaborada a Tabela 7, que demonstra redução no *BD-rate* em todos os vídeos avaliados, alcançando a média de redução de aproximadamente 3,38%, o que

significa que para manter a mesma qualidade do vídeo codificado sem a utilização do ALF, a solução aqui proposta consegue reduzir o número de bits necessários para representá-lo em 3,38%. Observa-se, ainda, uma redução acima da média nos vídeos de resolução mais elevada, 1600p, onde se alcançou a redução de 4,94%.

Tabela 7 – Comparação entre não aplicação do ALF e solução com inteiros.

Resolução	Vídeo	BD-rate	Média
240p	*Blowing Bubbles*	-0,6070	**-3,0378**
	BQ Square	-6,8213	
	Race Horses	-1,6852	
480p	*Basketball Drill*	-1,4901	**-1,4279**
	Basketball Drill Text	-1,3656	
720p	*Vidyo1*	-2,4538	**-3,6399**
	Vidyo 3	-5,4881	
	Vidyo 4	-2,9780	
1080p	*Cactus*	-3,2708	**-3,1078**
	Kimono	-2,9447	
1600p	*People on Street*	-4,8608	**-4,9430**
	Steam Locomotive Train	-6,8608	
	Traffic	-3,1417	
Média		-3,3795	

Em resumo, a simplificação proposta neste trabalho para o algoritmo do filtro ALF do HEVC, que consiste na substituição de dados em ponto flutuante por dados inteiros, resulta em impactos mínimos em termos de eficiência da codificação. Este fato é relevante, pois considerando uma futura implementação em hardware, é possível reduzir significativamente o custo em termos consumo de área e de energia com um impacto reduzido na eficiência. Esta simplificação pode ainda ser considerada para futuras implementações em software do algoritmo.

5 ARQUITETURAS DE HARDWARE DESENVOLVIDAS

Este capítulo apresenta as arquiteturas propostas para os filtros SAO e ALF do HEVC. Para o SAO foram propostas arquiteturas para a análise estatística, para os classificadores, para a função de custo e para armazenamento temporário de amostras, os *buffers*. Já para o ALF foram desenvolvidas arquiteturas para o núcleo das versões 3 e 5 do *Draft* e do HM do HEVC. Ainda para o ALF, foi proposta também uma versão configurável do ALF da versão 3 do *Draft*.

5.1 Arquiteturas para o SAO

Para o *Sample Adaptive Offset* (SAO) este trabalho propõe uma arquitetura que engloba as etapas de coleta estatística de dados, de classificação em ambos os tipos de SAO: EO e BO e também para a função de custo utilizada internamente no algoritmo do SAO para estimar a distorção gerada pelos *offsets* de acordo com as simplificações proposta na seção 4.1.

A Figura 22 apresenta uma visão em alto nível da arquitetura proposta para o SAO neste trabalho. A arquitetura recebe como entrada as amostras originais e reconstruídas, uma de cada por ciclo de *clock*. São utilizados dois *buffers*, um para armazenamento das amostras reconstruídas e outro para armazenamento da diferença entre amostras originais e reconstruídas. Não é necessário armazenar a amostra original, pois somente a diferença será utilizada para determinar o *offset*. Ligadas aos *buffers* estão as estruturas utilizadas para classificação e coleta dos dados estatísticos necessários para a definição dos *offsets* a serem utilizados em cada tipo e categoria do SAO. Em seguida, encontra-se a arquitetura responsável por realizar a estimativa de custo, que considera uma relação entre a distorção resultante da aplicação de determinado *offset* e o *overhead* necessário para representar o *offset* no *bitstream*.

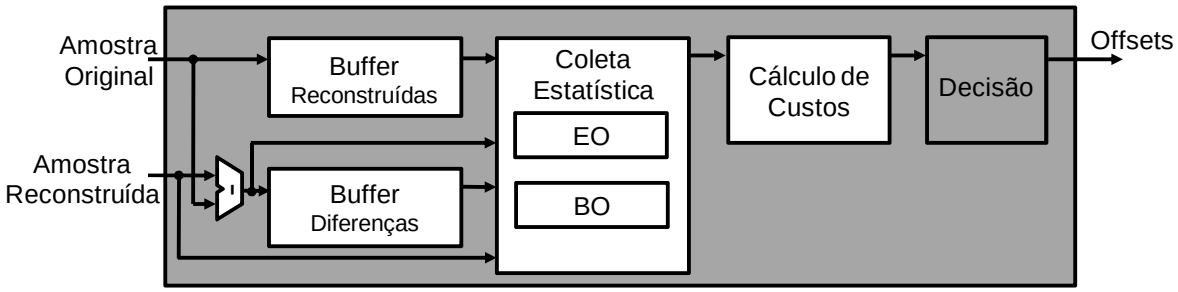


Figura 22 – Arquitetura do SAO em alto nível.

O processo do BO começa imediatamente após a chegada da primeira amostra da região que está sendo processada e não utiliza os *buffers* para sua classificação, pois sua classificação depende somente da intensidade da amostra atual. Por isso, a arquitetura apresenta caminhos diretos entre amostra reconstruída de entrada e a diferença. Portanto, a latência para que o BO conclua sua classificação e coleta estatística é apenas o tempo de carga da LCU, de 64x64 amostras, ou seja, 4.096 amostras.

Entretanto, o início do processo do EO precisa aguardar que o *buffer* de amostras reconstruídas carregue a terceira linha. Isto se deve ao fato do formato utilizado nas comparações do EO. Após o *buffer* de amostras reconstruídas estar cheio, ou seja, a terceira linha da LCU foi carregada para o *buffer*, as comparações e classificações serão realizadas e seu resultado será salvo em registradores de barreira temporal, a fim de liberar o *buffer* e as estruturas para a carga da nova linha. Enquanto a nova linha é carregada, começa o processo de coleta de dados estatísticos do EO, através do acesso aos dados da barreira temporal. Esta leitura é realizada a uma taxa de uma amostra por ciclo de *clock*, onde o EO faz a atualização de suas diferenças acumuladas e dos números de ocorrência em cada categoria. Este processo se repete até a conclusão das 64 linhas da CTU. Desta forma, a latência para a entrega dos dados estatísticos do EO de uma CTU é de 65 ciclos após a conclusão de sua carga.

O módulo de decisão não foi implementado neste trabalho, pois objetiva-se futuramente analisar o impacto de simplificações na função externa, bem como a inter-relação entre as duas funções de custo.

5.1.1 *Buffers*

Este trabalho propõe uma arquitetura para o SAO que utiliza dois *buffers*, um para armazenar as amostras reconstruídas e um para armazenar as diferenças entre amostras originais e reconstruídas.

A Figura 23 representa o *buffer* de amostras reconstruídas proposto neste trabalho. O *buffer* foi desenvolvido para receber as amostras em ordem *raster scan* e para armazenar três linhas de 64 amostras. Esta estrutura foi escolhida devido ao formato da comparação necessária no SAO do tipo EO, que compara uma amostra central com uma amostra da linha anterior e uma amostra da linha posterior. A saída não se enconstra identificada nesta figura, pois este *buffer* foi implementado em conjunto com a unidade responsável pela coleta de dados estatísticos para o SAO EO, onde cada posição do *buffer* está conectado aos componentes responsáveis pelas comparações.

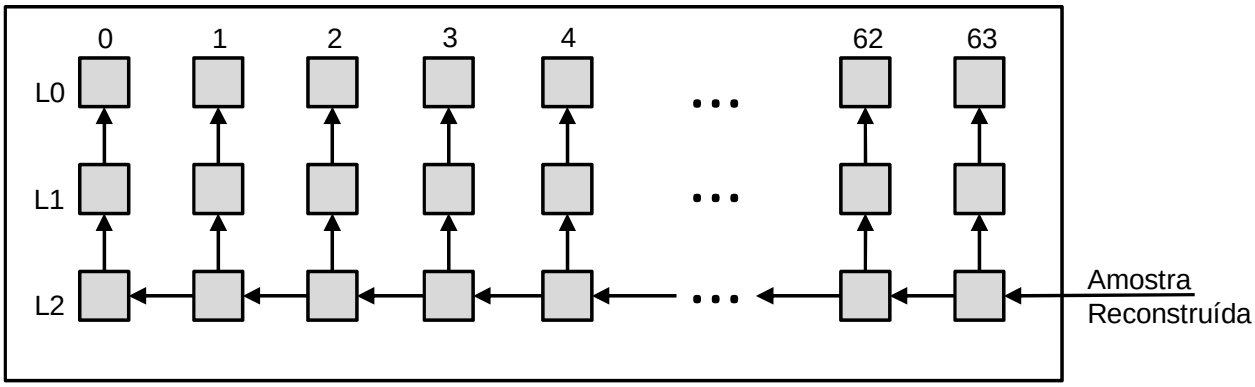


Figura 23 – Arquitetura proposta para o *buffer* de amostras reconstruídas.

Cada linha armazena 64 amostras (colunas entre 0 e 63 na Figura 23) com o objetivo de trabalhar com CTUs de 64x64. A cada ciclo de *clock*, uma nova amostra entra na linha L2, posição 63, e todas as demais amostras da linha são deslocadas para a esquerda. Estando a linha L2 completa, todas as amostras são deslocadas para a posição correspondente na linha L1. No mesmo ciclo de *clock*, todas as amostras na linha L1 são deslocadas para a linha L0.

O objetivo deste procedimento é aumentar o reaproveitamento de dados, reduzindo o acesso à memória. Uma vez que estas três linhas do *buffer* correspondem a três linhas da CTU e uma amostra considerada na comparação em

uma determinada linha será reaproveitada na próxima linha, não será necessário acessar a memória mais de uma vez para buscar a mesma amostra.

Além das amostras reconstruídas, o SAO também recebe como entrada as amostras originais. Entretanto, a informação necessária nos demais passos do SAO é somente a diferença entre as amostras originais e as amostras reconstruídas. Portanto, decidiu-se pela implementação de um *buffer* que armazenasse a diferença, ao invés de armazenar a amostra original. Tendo em vista que há a necessidade de acessar esta informação duas vezes durante o processamento do SAO, uma para a classificação do BO e uma para a do EO, optou-se por armazenar a diferença calculada ao invés de armazenar a amostra original, para reduzir o número de vezes em que o cálculo é realizado.

A Figura 24 representa a estrutura da arquitetura proposta para o *buffer* das diferenças. Esse *buffer* precisa armazenar apenas duas linhas, pois o acesso à informação da diferença entre as amostras será necessário apenas para a amostra da linha central no buffer de amostras reconstruídas. Portanto, ao passo que estão sendo classificadas as amostras da linha L1 do *buffer* de amostras reconstruídas, o *buffer* de diferenças também será acessado pela linha L1. De maneira análoga ao que acontece com os classificadores quando a carga de uma linha é concluída, as diferenças serão carregadas para a barreira temporal.

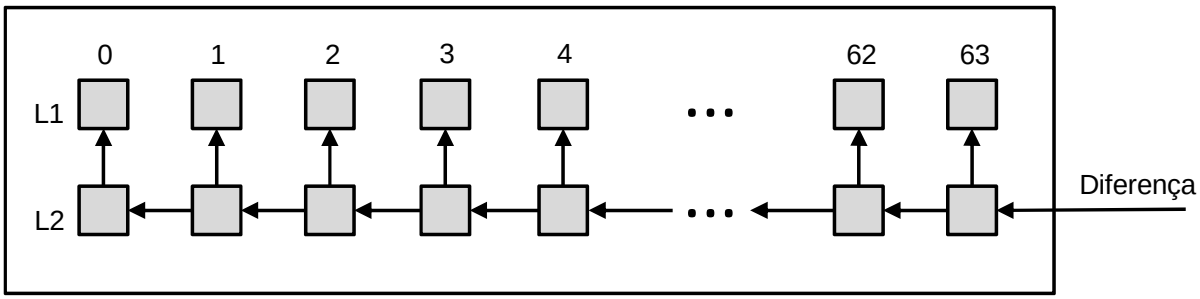


Figura 24 – Arquitetura proposta para o *buffer* dasdiferenças.

Neste trabalho, as amostras reconstruídas foram consideradas com largura de 8 bits e as diferenças foram consideradas com 9 bits.

5.1.2 *Edge Offset* (EO)

A unidade *Edge Offset* (EO) gera os dados estatísticos para cada categoria e cada classe de EO, os quais são necessários para a definição do *offset* correspondente.A arquitetura proposta, neste trabalho, para este módulo é composta por duas etapas: a etapa dos comparadores e a etapa da definição da categoria. A Figura 25 representa a estrutura de comparadores proposta (os círculos). Os comparadores estão conectados ao *buffer* de amostras reconstruídas e cada componente comparador, recebe duas amostras como entrada, retornando o valor -1 caso a primeira seja menor do que a segunda, o valor 1 caso a primeira seja maior do que a segunda e o valor 0 caso ambas sejam iguais.

A amostra atual, na Figura 25, corresponde ao quadrado na linha L1, posição 1, e os círculos representam as comparações que estão sendo feitas em relação a esta amostra. Os comparadores representados como círculos em branco, formando uma linha de comparação horizontal, correspondem aos dois comparadores necessários para identificar a Classe 0 do EO. Os círculos pretos, correspondem aos comparados relacionados à Classe 1, ou seja, formato vertical. E os comparadores relacionados às Classes 2 (135º) e 3 (45º) correspondem aos círculos pontilhados e listrados, respectivamente. Cada conjunto de comparadores correspondentes a uma mesma classe são conectados a um componente chamado de classificador. Este classificador, analisando os resultados das duas comparações, gera a identificação da categoria daquela classe, conforme a Tabela 1.

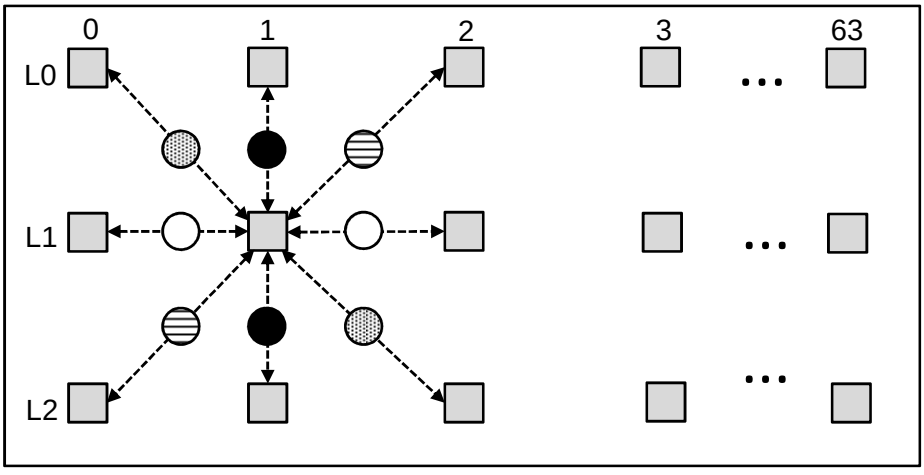


Figura 25 – Estrutura de comparação e classificação do EO.

A Figura 26 apresenta a arquitetura proposta para realizar a classificação das amostras nas 4 classes e 4 categorias em cada classe, totalizando 16 categorias. O objetivo desta arquitetura é proceder a coleta de dados estatísticos e a classificação para embasar a decisão do *offset* a ser aplicado às amostras reconstruídas de cada uma dessas categorias.

De acordo com o resultado das comparações e classificações realizadas, a unidade correspondente a cada classe e categoria será ativada. Esta unidade é responsável por acumular o valor da diferença entre a amostra original e a reconstruída e contar o número de ocorrências para a categoria correspondente. Este o processo permite que seja efetuado o cálculo do *offset* ao final do processamento da CTU. Para cada classe de EO (formato), apenas uma categoria será selecionada para cada amostra, e todas as classes terão uma das 5 categorias da Tabela 1, mas apenas as categorias válidas serão consideradas para os cálculos. Se a categoria de uma classe for igual a 0, nenhuma unidade de categoria será atualizada, pois a ocorrência de categoria igual a 0 corresponde a não aplicação do SAO EO.

Esta arquitetura recebe como entrada além da diferença entre as amostras original e reconstruída, os resultados dos classificadores para cada classe de EO. A cada ciclo de *clock*, uma amostra da CTU é analisada para os quatro formatos de EO possíveis e o resultado desta análise será usado como entrada para os demultiplexadores que selecionarão a categoria adequada em cada classe. Após a entrada da útima amostra da CTU no *buffer*, a etapa de coleta estatístico do SAO EO estará concluída e os dados necessários ao cálculo do custo, diferença acumulada e número de ocorrência em cada categoria, estarão disponíveis.

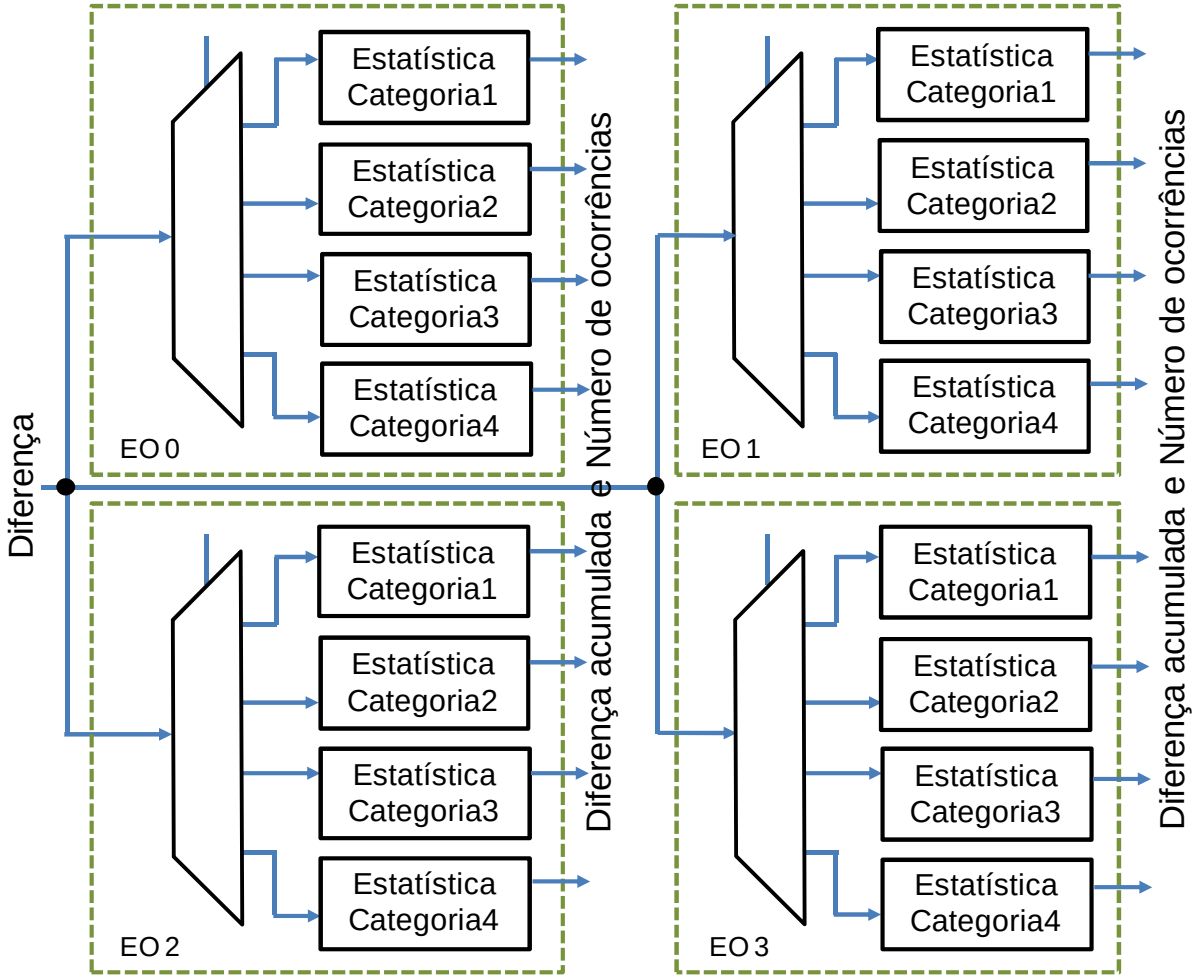


Figura 26 – Arquitetura de classificação do EO.

5.1.3 *Band Offset* (BO)

A unidade *Band Offset* (BO) é responsável pela classificação das amostras da região em 32 bandas e por realizar os a coleta estatística necessária para a geração dos *offsets* correspondentes.

O processo para realizar a classificação das amostras pelo método BO é semelhante ao usado no EO, entretanto, a etapa de classificação não depende de comparadores como no EO, uma vez que o BO considera a intensidade das amostras e não seus vizinhos. A classificação é feita por faixas de intensidade, considerando 32 bandas, portanto, a classificação pode ser feita analisando-se os 5 bits mais significativos desta amostra. A Figura 27 apresenta a arquitetura proposta neste trabalho para a realização da classificação segundo o BO. O funcionamento do módulo para cada banda, de BO0 a BO31, é semelhante àquele utilizado para as

categorias de EO, ou seja, acumular a soma das amostras identificadas naquela banda e contar o número de ocorrências.

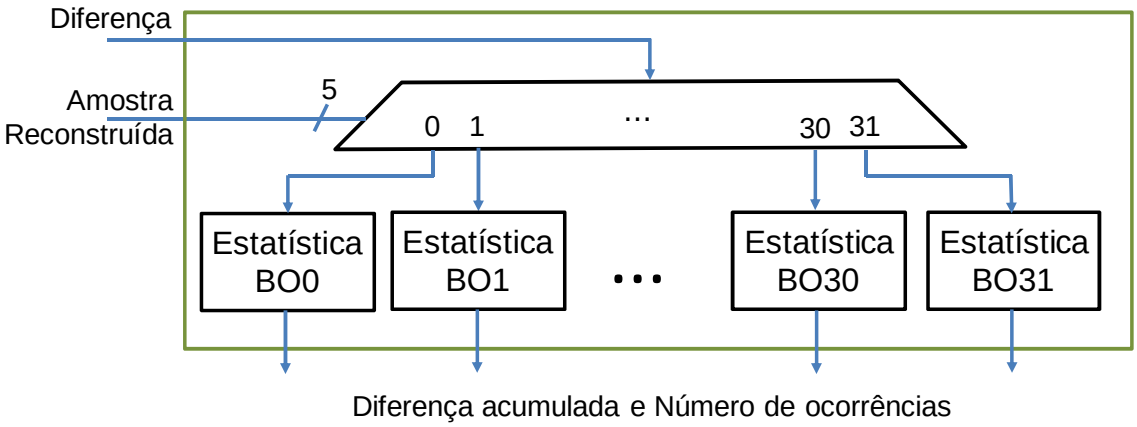


Figura 27 – Arquitetura para classificação do *Band Offset* (BO).

Esta arquitetura recebe como entrada, a cada ciclo, uma amostra reconstruída e a correspondente amostra original. Os 5 bits mais significativos da amostra reconstruída são considerados para realizar a classificação da amostra em uma das 32 bandas. Após o processamento da CTU, ou 4096 amostras, a etapa de coleta estatística estará concluída e os dados nas saídas de cada unidade de BO estarão corretos. Portanto, neste ponto, a arquitetura disponibiliza a diferença acumulada e o número de ocorrências em cada banda de BO.

5.1.4 Função de Custo

A função de custo interna utilizada pelo SAO para embasar a decisão de qual *offset* será escolhido utiliza as equações descritas em (1) e (2) para estimar a distorção e o custo de cada um dos tipos e categorias.

Focando em uma implementação em hardware das equações de custo em (1) e (2) que alcance elevadas taxas de processamento, todos os possíveis *offsets*, resultados das iterações, bem como seus respectivos custos, precisam ser gerados em apenas um ciclo de *clock*. Alternativamente, uma estrutura *pipeline* onde cada iteração do laço de *offsets* seja representada por apenas um ou por uma sequência de estágios de *pipeline* pode ser usada. Entretanto, para manter elevada a taxa de processamento, a estrutura de *pipeline* precisa ser multiplicada por 7, número máximo de iterações. Caso a estrutura não seja multiplicada, a performance seria dividida por 7.

As equações em (1) e (2) foram simplificadas e fatoradas focando em uma implementação em hardware de acordo com a proposta apresentada na seção 4.1.A Figura 28 representa um diagrama em alto nível da arquitetura proposta neste trabalho para implementação da função interna de custo do SAO. Esta arquitetura recebe como entrada a diferença acumulada entre as amostras originais e reconstruídas, o número de ocorrências e o multiplicador de Lagrange. E como saída, apresenta o custo dos 7 possíveis *offsets* positivos ou negativos. Cada uma das saídas implementa uma das equações de (9). Por exemplo, a saída “Custo1” corresponde ao custo c_1 ou c_{-1} da equação (9).

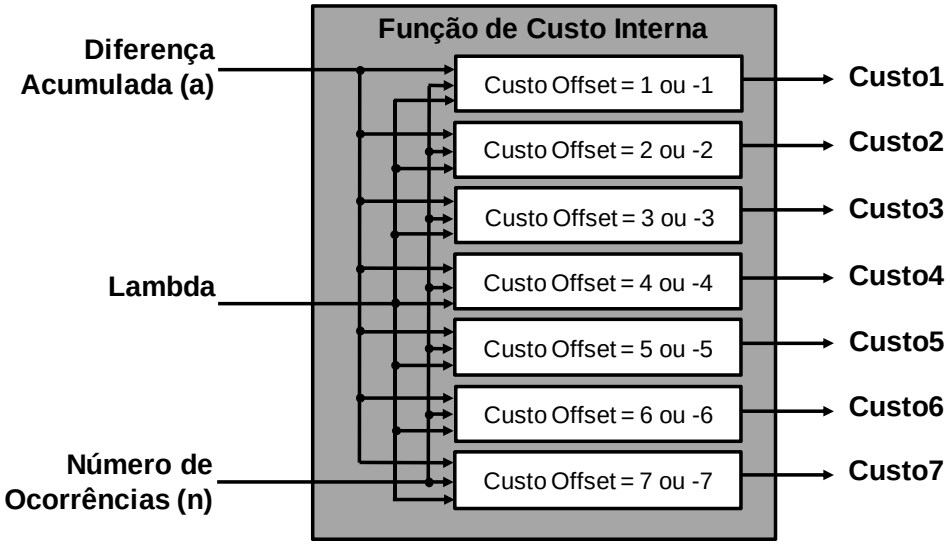


Figura 28 – Diagrama em alto nível da função interna de custo do SAO.

A função de custo foi implementada em uma versão totalmente combinacional e para um conjunto de 4.096 amostras de entrada, são necessários 48 ciclos de *clock* para gerar todos os custos de todas as categorias, classes e tipos do SAO. Uma implementação com pipeline aumentaria substancialmente a quantidade de registradores envolvidos, além de dificultar o reaproveitamento de subexpressões realizado pela ferramenta de síntese.

5.1.5 Resultados de Síntese

Esta seção apresenta os resultados de síntese para as arquiteturas de classificação e coleta estatística e da função de custo propostas para o filtro SAO.

5.1.5.1 Resultados de Síntese para a Arquitetura de Classificação

As arquiteturas propostas para a realização das etapas de classificação e de geração de dados para geração de *offsets* do SAO foram descritas em VHDL e sintetizadas para um dispositivo FPGA da família Stratix IV da Altera. Para tanto, foi utilizada a ferramenta Quartus II também da Altera e o dispositivo alvo foi o EP4SGX180HF35C2.

A Tabela 8 apresenta os resultados desta síntese. Os resultados em termos de uso de recursos do dispositivo demonstram que a arquitetura proposta para a estrutura de classificação e geração dos *offsets* utilizou 6.874 *Adaptive Look-up Tables* (ALUTs), o que corresponde a 5% das disponíveis, e 5.603registradores lógicos dedicados, o que corresponde a menos de 4% dos recursos disponíveis no dispositivo alvo.

Tabela 8 – Resultados de síntese para a arquitetura de classificação do SAO.

Indicador	BO	EO	*Buffer*	SAO
ALUTs	1.441	689	4.714	6.874
Registradores	1.024	544	4.035	5.603
Frequência (MHz)	800	607	383	373

Considerando os resultados de frequência máxima de operação gerada pelo Quartus II, é possível estimar a taxa de processamento, em quadros por segundo, considerando que todas as CTUs do vídeo são filtradas, o que sempre ocorre no codificador, pois o filtro é inicialmente aplicado para depois ser realizada a decisão se o dado filtrado será usado ou não. Considera-se que a cada novo ciclo de *clock*, uma nova amostra será consumida pela arquitetura e assim que a última amostra da CTU tiver sido processada, a arquitetura apresentará os resultados de classificação e dados para geração dos *offsets*. A estimativa de taxa de processamento em quadros por segundo para a arquitetura consta na Tabela 9.

Tabela 9 – Estimativa de taxa de processamento para a arquitetura de classificação do SAO.

Resolução	Quadros por Segundo
720p (1280x720)	405
1080p (1920x1080)	180
UHD 4K (3840x2160)	45

A arquitetura proposta neste trabalho alcançou uma frequência máxima de operação de 373 MHz e, assim, é capaz de processar 405 quadros por segundo na resolução de 720p (1280x720 pixels), 180 quadros por segundo na resolução 1080p (1920x1080 pixels) e 45 quadros por segundo na resolução UHD 4K (3840x2160 pixels).

Estes resultados demonstram que a arquitetura aqui proposta é capaz de processar vídeos de alta definição, até UHD 4K, em tempo real.

5.1.5.2 Resultados de Síntese para a Função de Custo

A arquitetura proposta neste trabalho para a função de custo interna do SAO foi descrita em VHDL e sintetizada para o mesmo dispositivo FPGA utilizado para a arquitetura de classificação, ou seja, o EP4SGX180HF35C2 da família Stratix IV da Altera. A síntese usou o software Quartus II da Altera.

A Tabela 10 apresenta os resultados extraídos desta síntese. Estes resultados demonstram um baixíssimo uso de recursos do dispositivo, utilizando apenas 896 ALUTs e 252 registradores e alcançando uma frequência de operação de 160MHz.

Tabela 10 – Resultados de síntese para a função de custo do SAO.

Indicador	Resultados
ALUTs	896
Registradores	252
Frequência (MHz)	160

Com os dados da Tabela 10, é possível estimar a taxa de processamento alcançada por esta arquitetura em quadros por segundo. Esta estimativa de taxa de

processamento está demonstrada na Tabela 11. Considerando a frequência de operação alcançada e que são necessários apenas 48 ciclos de *clock* para gerar o custo para todas as possibilidades de SAO para uma CTU (4.096 amostras), a arquitetura proposta alcança uma taxa de 1.634 quadros UHD 4K por segundo.

Tabela 11 – Estimativa de taxa de processamento para a função de custo do SAO.

Resolução	Quadros por Segundo
720p (1280x720)	13.889
1080p (1920x1080)	6,536
UHD 4K (3840x2160)	1.634

Devido a esta elevada taxa de processamento, a arquitetura proposta poderia também ser usada em frequência mais baixa, objetivando uma redução no consumo de energia. Desta forma, para processar vídeos de resolução UHD 4K a 120 quadros por segundo, a arquitetura proposta neste trabalho pode operar a apenas 12 MHz.

5.1.6 Validação das Arquiteturas do SAO

O processo de validação das arquiteturas propostas para o SAO considerou dados retirados do software de referência, HM10.

Para a validação das arquiteturas de classificação e coleta estatística, foram inseridos métodos no código para extração dos dados para arquivos de texto.

Para a validação da função de custo, os dados foram retirados da versão do software de referência com as otimizações de *loop unrolling* e dados com precisão fracionária de 8 bits.

Em ambos os casos, foi utilizada a ferramenta Modelsim da Mentor Graphics para aplicação do estímulos e extração dos resultados gerados pelas arquiteturas. Os arquivos com resultados gerados pelo software de referência e extraídos pelo Modelsim foram comparados e, assim, as arquiteturas foram consideradas validadas.

5.1.7 Comparação com Trabalhos Relacionados para o SAO

Esta seção busca comparar o trabalho proposto nesta dissertação de mestrado com os demais trabalhos relacionados presentes na literatura.

Os dois primeiros trabalhos relacionados ao SAO propõem arquiteturas em hardware, mas com foco apenas no decodificador. Esta condição inviabiliza uma comparação justa e direta, pois tais arquiteturas não implementam as estruturas necessárias à etapa de coleta estatística e a decisão dos *offsets* a serem usados, uma vez que os *offsets* são enviados no *bitstream*.

É importante salientar ainda que nosso trabalho foca em vídeos de alta definição com redução de uso de recursos de hardware. Desta forma, foi adotado o consumo de uma amostra por ciclo de *clock*, por outro lado, o trabalho de (ZHU *et al.*, 2013) propõe o consumo de 16 amostras por ciclo, mesmo focando apenas no decodificador, o que leva a um alto consumo de hardware e um possível gargalo na memória.

Outro trabalho relacionado ao SAO, (ZHU *et al.*, 2014), foca em uma implementação em hardware do codificador e ataca também a função de custo utilizada para decisão. Este trabalho propõe três simplificações principais na forma de cálculo do custo do SAO.

A primeira delas é a eliminação da função de custo interna, estabelecendo que o *offset* inicial é o adotado e não há iteração, conforme define o algoritmo original do SAO. Contudo, não traz qualquer avaliação em do impacto desta simplificação.

Outra simplificação proposta é que a decisão externa do SAO, que escolhe qual o tipo e categoria será aplicado, será tomada com base somente na distorção e não considerará a taxa de bits necessária para representar esta informação. Mais uma vez, não é apresentada justificativa ou análise de impacto.

A terceira simplificação trata da decisão de aplicar ou não o SAO, que considera tanto a equação de distorção quanto a equação de custo, mas propõe valores fixos para a taxa de bits, considerando custo 3 para não aplicação SAO e custo 10 para aplicação. Novamente, sem justificativas ou análise de impacto.

Conforme afirmação do autor, esta modificação faz com que a fórmula de custo elimine a necessidade de divisores, mas mantém multiplicadores. Ressalta-se ainda que, como há o cálculo do *offset* inicial, a implementação em hardware demandará um divisor completo para geração deste valor.

Os resultados apresentados neste trabalho relacionado foram obtidos mediante a simulação para 5 sequências de vídeo na configuração *Low Delay*. Em

média, em comparação à versão com SAO, houve um aumento de 0,38% no *BD-rate*. Analisando os resultados deste trabalho de mestrado, para os mesmos vídeos e configurações, considerando ainda dados inteiros, o aumento no *BD-rate* foi de apenas 0,005%, enquanto que, analisando todo o conjunto de vídeos simulados, este aumento foi um pouco maior, de 0,076%.

Ainda assim, nosso trabalho trouxe a proposta de duas simplificações no algoritmo original do SAO, ambas baseadas em questões críticas para a implementação em hardware, quais sejam: dados em ponto flutuante e uso de multiplicadores e divisores, além de analisar o impacto dessas simplificações. Por outro lado, o trabalho de (ZHU *et al.*, 2014) propôs simplificações que não eliminaram a necessidade de multiplicadores e divisores e também não trouxe justificativas e análises dos impactos de cada simplificação. O artigo não menciona ainda o tipo de dados utilizados e apresenta seus resultados para o conjunto de simplificações apresentadas.

O artigo (ZHU *et al.*, 2014) que também propôs simplificação na função de custo, necessita receber como entrada, a cada ciclo de *clock*, um conjunto de amostras correspondente a um bloco 4x4, ou seja, 16 amostras, e mais 20 amostras do contorno deste bloco, totalizando 36 amostras por ciclo. Este trabalho também não traz qualquer estrutura de *buffer* para reaproveitamento de amostras já lidas. e foi desenvolvido considerando o uso de multiplicadores e divisores, enquanto nosso trabalho eliminou tal necessidade, além de consumir apenas uma amostra por ciclo ao invés de 36 amostras.

Foi identificado ainda o trabalho (MODY *et al.*, 2014) que também foca em uma implementação em hardware para o SAO, abordando o codificador, entretanto, este trabalho apresenta um modo muito próprio de discussão e de apresentação de resultados. O artigo traz poucas referências aos documentos e artigos relacionados ao HEVC e faz toda a discussão sobre o funcionamento do SAO como sendo característica de sua arquitetura e não técnicas propostas pelo padrão. O trabalho não traz qualquer simplificação em relação ao padrão. A arquitetura alcança alto desempenho, conseguindo processar vídeos UHD 4K em tempo real. O trabalho traz ainda uma série simulações com diversos vídeos, sendo que nenhum deles utilizado pelas condições comuns de teste do HEVC e mostra que apresenta uma economia em *bitrate* em comparação ao HM, embora não apresente qualquer parâmetro para

esta comparação. O trabalho afirma também ser eficiente em uso de área de hardware, entretanto, não há qualquer comparação ou referência a trabalhos relacionados ou mesmo, técnicas que pudessem justificar uma redução de área.

5.2 Arquiteturas para o ALF

Conforme explicado na seção 3.3, a aplicação do filtro ALF consiste na realização de multiplicações de amostras reconstruídas pelos coeficientes que estiverem nas mesmas posições do formato do filtro alvo (diamante, *snowflake*, etc.) e da soma destes resultados intermediários.

Considerando ainda a simetria dos coeficientes dos filtros, que é característica independente do formato, é possível reduzir o número de multiplicações a metade, apenas alterando a ordem de processamento.

Esta seção apresenta as arquiteturas propostas, neste trabalho, para o núcleo do ALF utilizado no HEVC, nas versões 3 e 5 do *Draft* do HEVC e do HM.

5.2.1 Arquitetura para os Núcleos do ALF no HM 3

A versão 3 do software de referência e do *Working Draft* adotava o formato diamante para o núcleo do ALF e três tamanhos: 5x5, 7x7 e 9x9, conforme Figura 18.

O núcleo ALF de tamanho 5x5 possui 13 posições e, portanto, acessa 13 amostras do vídeo. Entretanto, não é necessário realizar as 13 multiplicações. Este número pode ser reduzido para 7, que é o número de coeficientes.

A arquitetura proposta o núcleo do filtro ALF tamanho 5x5 está representada na Figura 29. Para este tamanho de núcleo, o hardware proposto foi desenvolvido com 4 estágios de *pipeline*, onde a operação crítica para o atraso do circuito, a multiplicação, foi isolada no segundo estágio do *pipeline*. No primeiro estágio são feitas as somas das amostras das posições correspondentes aos coeficientes simétricos. No segundo estágio é feita a multiplicação deste resultado pelo coeficiente correspondente. Os valores que precisam ser operados pelos multiplicadores são dinâmicos, portanto, é necessária a utilização de multiplicadores completos. As demais operações necessárias, ou seja, os demais somadores necessários para acumulação dos resultados parciais e operação *clipping*, necessária para que a amostra resultante retorne à faixa de representação de

amostras considerada foram distribuídas nos estágios 3 e 4 de forma a melhor balancear o *pipeline* e focando na linearidade com os demais tamanhos que também foram implementados com 4 estágios. Foi incluída, ainda, no terceiro estágio do *pipeline,* a soma com a constante 128, cujo objetivo é o arredondamento parcial do valor que será descartado na operação seguinte. Esta operação seria a última a ser realizada antes do *clipping*, entretanto, foi deslocada para reduzir o número de operadores em cascata.

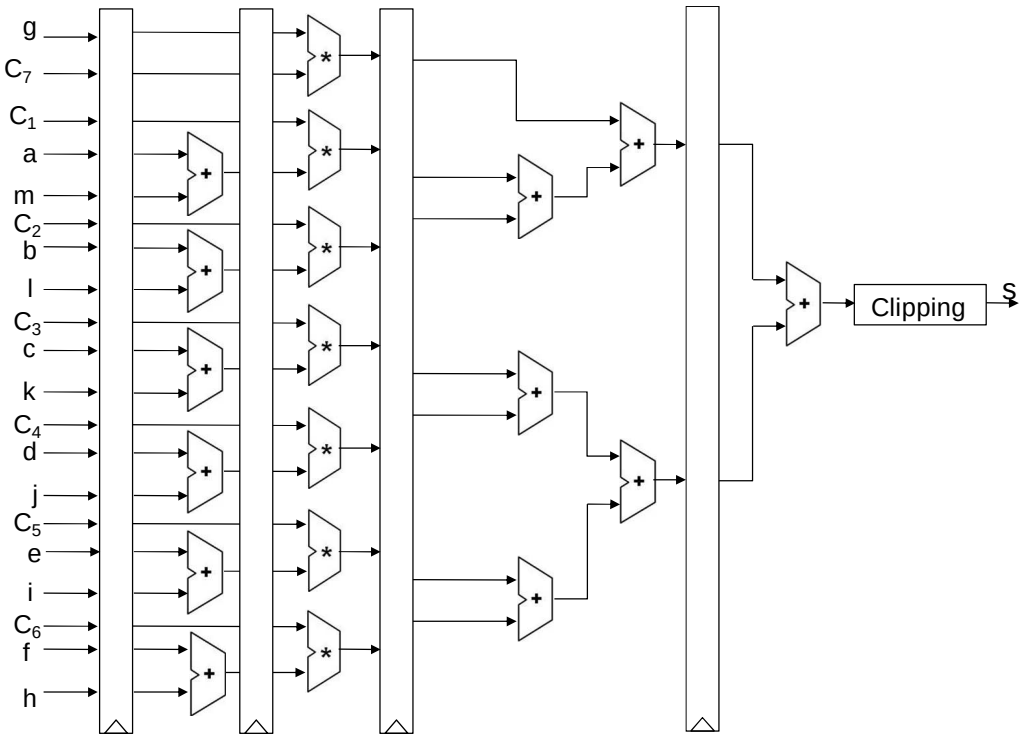


Figura 29 – Arquitetura proposta para o núcleo do ALF do HM3, tamanho 5x5.

Para os demais tamanhos do núcleo do ALF do HM3, o 7x7 e 9x9, a arquitetura é semelhante à implementada para o tamanho 5x5 e demonstrada na Figura 29. A principal diferença para os demais tamanhos está no volume de dados processados. A Tabela 12 apresenta uma comparação entre os três tamanhos de filtro desenvolvidos, apresentando número de estágios de *pipeline*, de operadores e de dados de entrada.

Tabela 12 – Comparação entre os três tamanhos do ALF do HM3.

Indicador	5x5	7x7	9x9
Estágios de *pipeline*	4	4	4
Somadores	13	25	39
Multiplicadores	7	13	20
Clipping	1	1	1
Coeficientes	7	13	20
Amostras	13	25	39

A arquitetura proposta para o núcleo do filtro ALF de tamanho 7x7 possui um total de 25 posições e, de acordo com a simetria, totaliza 13 coeficientes e, por consequência, 13 multiplicações. De maneira análoga à arquitetura do ALF 5x5, no primeiro estágio de *pipeline* da arquitetura do ALF 7x7 são feitas as somas entre as amostras em posições simétricas, totalizando 12 somadores operando em paralelo neste estágio. No segundo estágio são feitas as multiplicações, totalizando 13 multiplicadores, também operando em paralelo. No estágio seguinte é feita a acumulação dos valores intermediários, o arredondamento e o *clipping*.

Para o tamanho 9x9 do núcleo do filtro ALF, seguiu-se o mesmo procedimento. Este tamanho apresenta um total de 39 posições, que resultarão em um total de 19 somadores operando em paralelo no primeiro estágio de *pipeline*. Já no segundo estágio de *pipeline*, nesta versão, serão 20 multiplicadores operando em paralelo. Nos demais estágios, assim como nos demais tamanhos, serão feitos a acumulação dos valores parciais, o arredondamento e o *clipping*.

5.2.1.1 Resultados de Síntese ALF do HM3

As arquiteturas propostas neste trabalho para o núcleo do filtro ALF do HM3 foram descritas em VHDL e sintetizadas para um dispositivo FPGA da família Stratix IV da Altera. A síntese foi direcionada para o EP4SGX180HF35C2 e utilizou o software Quartus II também da Altera

A Tabela 13 apresenta os resultados de síntese obtidos para os três tamanhos de núcleo do ALF. Em termos de uso de recursos do dispositivo, todas as arquiteturas utilizam um baixo número de ALUTs e registradores em comparação ao disponível no dispositivo, usando sempre menos de 1% da capacidade. A arquitetura

do filtro de tamanho 9x9 representa a arquitetura com a menor frequência de operação, ainda alcançando 253 MHz. Este bom desempenho, mesmo com a utilização de multiplicadores completos, deve-se ao fato do dispositivo disponibilizar blocos DSP que implementam a multiplicação de maneira eficiente.

Tabela 13 – Resultados de síntese para os núcleos do ALF do HM3.

Indicador	5x5	7x7	9x9
ALUTs	207(<1%)	369(<1%)	587 (<1%)
Registradores	226(<1%)	425(<1%)	630 (<1%)
Blocos DSP 18-bit	10 (1%)	18(2%)	27(3%)
Frequência	299	266	253

Considerando as frequências de operação geradas pelos Quartus II, foi realizada uma estimativa da taxa de processamento em quadros por segundo para algumas resoluções. Esta estimativa foi realizada levando em conta o nível de paralelismo e a frequência atingida por cada arquitetura e considerando também o pior caso, ou seja, o caso em que todas as amostras do vídeo são filtradas. Os resultados desta estimativa constam na Tabela 14.

Tabela 14 – Estimativa de taxa de processamento para o ALF do HM3.

Resolução	5x5	7x7	9x9
720p (1280x720)	324	288	274
1080p (1920x1080)	144	128	122
UHD 4K (3840x2160)	36	32	30

A partir desta estimativa, verifica-se que, mesmo para a arquitetura 9x9, que apresentou o pior resultado em frequência de operação, ainda é capaz de processar 30 quadros UHD 4K por segundo.

5.2.2 Arquitetura para os Núcleos do ALF no HM 5

A versão 5 do software de referência e do *Working Draft* passou a adotar dois novos formatos para o filtro ALF, em substituição ao filtro em formato diamante. Foram adotados os formatos *snowflake* 5x5 e *cross* 9x9, conforme já apresentado na Figura 19.

Estes dois formatos adotados no HM5 trabalham com o mesmo número de amostras e, por consequência, também com o mesmo número de coeficientes. São 17 amostras e 9 coeficientes processados.

A arquitetura proposta para os núcleos do filtro ALF do HM5 segue a mesma estrutura da arquitetura apresentada na seção 5.2.1 para o HM3, pois a regra para o cálculo da amostra filtrada é exatamente a mesma, a diferença restará apenas na posição das amostras reconstruídas que serão consideradas na filtragem.

Tendo em vista que o núcleo da arquitetura aqui proposta apenas recebe as amostras para processamento e não está vinculada às suas posições; e que a quantidade de amostras a ser processada nos dois formatos é rigorosamente a mesma, uma única arquitetura poderá atender aos dois novos formatos.

Assim, o hardware proposto para o filtro ALF do HM5 foi desenvolvido com 4 estágios de *pipeline*, sendo que, no primeiro estágio são feitas as somas das amostras que possuem o mesmo coeficiente em função da simetria do filtro, utilizando um total de 8 somadores. No segundo estágio da *pipeline* serão realizadas as multiplicações pelos coeficientes correspondentes, totalizando 9 multiplicadores. No terceiro e no quarto estágios, é feita a acumulação dos valores intermediários, o arredondamento e o *clipping* para que o valor retorne à faixa de representação original das amostras.

5.2.2.1 Resultados de Síntese ALF do HM5

A arquitetura aqui proposta foi descrita em VHDL e sintetizada utilizando a ferramenta Quartus II da Altera. A síntese foi direcionada para o dispositivo EP4SGX180HF35C2 da família Stratix IV também da Altera e os resultados desta síntese, bem como a estimativa da taxa de processamento, encontram-se na Tabela 15.

A arquitetura desenvolvida apresenta baixo uso de recursos do dispositivo considerando que é capaz de processar vídeos UHD 4K (3840x2160) em tempo real e usa menos de 1% das ALUTs e dos registradores disponíveis no dispositivo.

Tabela 15 – Resultados de síntese para os núcleos do ALF do HM5.

Indicador	*SnowflakelCross*
ALUTs	270 (<1%)
Registradores	299 (<1%)
Blocos DSP	12 (1%)
Frequência	292

Conforme demonstrado na Tabela 15, a arquitetura proposta alcançou uma frequência de operação de 292 MHz e, com isso, foi possível estimar a taxa de processamento em quadros por segundo que esta arquitetura seria capaz de processar. Estes resultados encontram-se na Tabela 16, Chegou-se então à capacidade de processamento de 316 quadros 720p (1280x720) por segundo, 140 quadros 1080p (1920x1080) por segundo e 35 quadros UHD 4K (3840x2160) por segundo.

Tabela 16 – Taxa de processamento para o ALF do HM5.

Resolução	Quadros por Segundo
720p (1280x720)	316
1080p (1920x1080)	140
UHD 4K (3840x2160)	35

5.2.3 Arquitetura Configurável para o HM3

Conforme apresentado na seção 5.2.1, este trabalho propôs três arquiteturas distintas para implementação dos três tamanhos de núcleo do filtro ALF do HEVC: 5x5, 7x7 e 9x9.

Entretanto, é possível notar que um filtro encontra-se inserido no filtro seguinte, isto é, no filtro 7x7 é possível localizar o filtro 5x5 e dentro do filtro 9x9 é possível localizar o filtro 7x7. A Figura 30 representa o casamento entre os três tamanhos do filtro ALF no HM3.

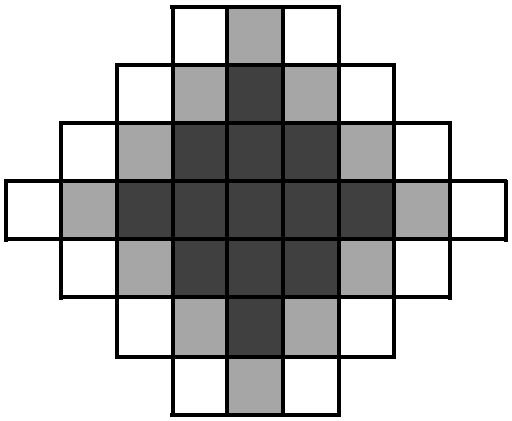


Figura 30 – Casamento dos três tamanhos do filtro ALF do HM3.

Então foi desenvolvida uma arquitetura configurável capaz de tirar proveito do fato dos filtros maiores incluírem os filtros menores e, assim, promover o reuso dos elementos de hardware para a implementação deste filtro. Desse modo, pretende-se reduzir o uso de recursos de hardware necessários em comparação a uma implementação dos três filtros de maneira independente. Outro objetivo desta arquitetura é reduzir tanto o número de elementos ociosos como o tempo em que permanecem ociosos, também em comparação a uma arquitetura que implemente os três formatos de filtro de maneira independente. Desta forma, busca-se reduzir, o consumo de potência de *leakage*. Esta redução em uso de recursos de hardware e de consumo de potência torna-se importante quando o foco está em dispositivos embarcados que possuem restrições de tamanho de bateria.

Na Figura 31 está representada a arquitetura configurável proposta para os três tamanhos do filtro ALF no HM3. O bloco chamado “Núcleo 5x5” representa o filtro ALF de tamanho 5x5, conforme apresentado anteriormente na Figura 29, excluindo-se a etapa de normalização, ou seja, sem o arredondamento e o *clipping.* Estas duas etapas foram transpostas para o final da nova arquitetura. Este bloco corresponde às amostras mais escuras na Figura 30. O bloco chamado “Borda 7x7” representa o processamento adicional necessário para completar o filtro 7x7 considerando que toda a etapa central já está coberta pelo bloco “Núcleo 5x5”. Este bloco seria correspondente às amostras em cinza na Figura 30. E, por fim, o bloco “Borda 9x9” representa o cálculo necessário para completar o filtro 9x9, considerando que já está coberto o filtro 7x7. Estas amostras correspondem às amostras em branco na Figura 30.

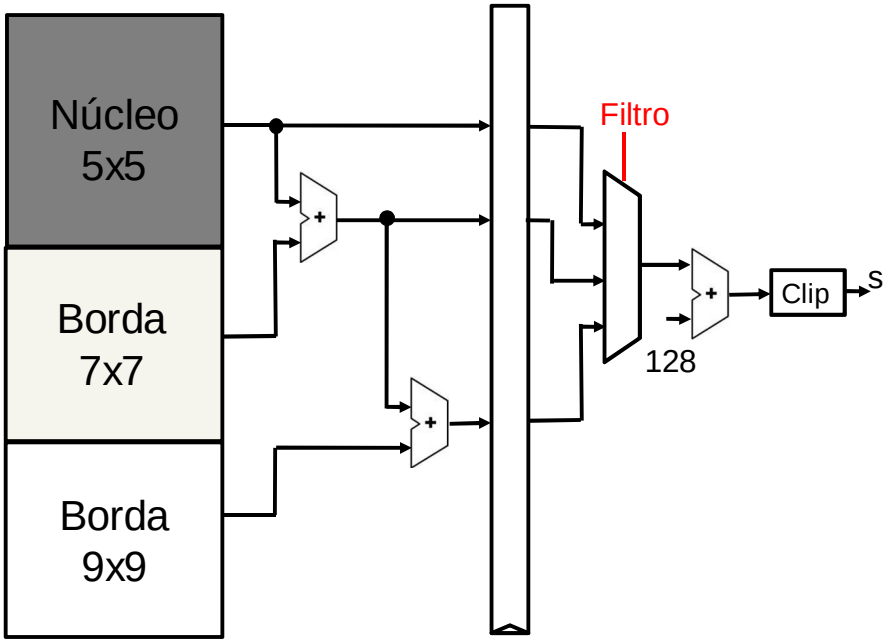


Figura 31 – Arquitetura configurável proposta para o HM3.

As etapas seguintes da arquitetura representada na Figura 31 correspondem aos somatórios necessários à conclusão de cada filtro, de um multiplexador que faz seleção de qual o filtro válido em cada momento, de acordo com um sinal de controle, e por fim o arredondamento e o *clipping*.

5.2.3.1 Resultados de Síntese para a Arquitetura Configurável

A arquitetura foi descrita em VHDL e sintetizada para o dispositivo FPGA EP4SGX180HF35C2 da família Stratix IV da Altera, utilizando a ferramenta Quartus II também da Altera.

Os resultados em termos de consumo de hardware estão representados como ALUTs, número de registradores e de blocos DSP. Constam ainda os ganhos obtidos pela versão configurável em comparação à versão com os três tamanhos de forma independente. Por fim, estão apresentados também os resultados de frequência máxima alcançada e a estimativa de taxa de processamento em quadros por segundo para algumas resoluções.

Como é possível verificar na Tabela 17, a arquitetura configurável proposta apresenta ganhos expressivos em termos de consumo de hardware, chegando a um ganho de 50% em comparação a uma versão que implemente os três formatos do filtro de maneira independente.

Tabela 17 – Resultados de síntese para a arquitetura configurável.

Indicador	5x5	7x7	9x9	Total	Configurável	Ganhos
ALUTs	207	369	587	1.163	527	45%
Registradores	226	425	630	1.281	660	51%
Blocos DSP	10	18	27	55	27	50%
Frequência(MHz)	299	266	253	-	250	-

A frequência máxima de operação obtida pela arquitetura configurável foi de 250 MHz e, com este valor, chegou-se a estimativa de taxa de processamento, conforme demonstrado na Tabela 18, de 270 quadros 720p, 120 quadros 1080p e 30 quadros UHD 4K no pior caso, quando todas as amostras são filtradas.

Tabela 18 –Taxa de processamento para a arquitetura configurável.

Resolução	Quadros por Segundo
720p (1280x720)	270
1080p (1920x1080)	120
UHD 4K(3840x2160)	30

Estes resultados demonstram que, mesmo com uma redução de 45%, 51% e 50% no uso de recursos do dispositivo, ALUTs, registradores e blocos DSP, respectivamente, a arquitetura ainda consegue processar quadros UHD 4K em tempo real.

5.2.4 Validação das Arquiteturas do ALF

Para validação das arquiteturas propostas nesta dissertação de mestrado para os núcleos do filtro ALF também foi utilizada a ferramenta ModelSim da Mentor Graphics e os dados utilizados para validação foram extraídos do software de referência em suas versões 3 e 5.

De maneira análoga ao processo executado para a validação das arquiteturas do SAO, os dados foram extraídos do software de referência imediatamente antes da aplicação do núcleo do filtro e logo após sua aplicação. Os dados de entrada foram aplicados às arquiteturas e os resultados extraídos foram comparados com os de referência. Desta forma, as arquiteturas foram consideradas validadas.

5.2.5 Comparação com Trabalhos Relacionados para o ALF

Esta seção apresenta alguns comentários e comparações com os principais trabalhos relacionados identificados quanto ao ALF.

Durante as reuniões do JCT-VC, várias propostas para o ALF foram apresentadas, algumas em alto nível, propondo novos algoritmos e métodos e adaptação e outras com foco no núcleo do filtro, propondo novos formatos e tamanhos.

O primeiro trabalho relacionado ao ALF identificado foi publicado por (DU e YU, 2011) e apresentou uma implementação em hardware para um filtro ALF, entretanto, seu foco era ainda no H.264/AVC. A arquitetura proposta por (DU e YU, 2011) considerava diferentes formatos de núcleo e implementava uma versão combinada com o DF. Estes fatores impedem uma comparação direta.

Outro trabalho identificado com foco no ALF, (HAUTALA, BOUTELLIER e HANNUKSEDA, 2013), entretanto, este trabalho propõe uma solução hardware-software, utilizando um processador dedicado a aplicações específicas, ou um *Application Specific Instruction-Set Processor* (ASIP) e um programa usando linguagem C pura.

Este trabalho sintetizado para um dispositivo Stratix III da Altera, para fins de validação, e para ASIC 90nm, alcançando uma frequência de operação de 290 MHz, sendo capaz de processar vídeos 1080p em tempo real.

Como resultado da síntese para o dispositivo Stratix III, a solução apresentada consumiu 12.598 elementos lógicos, aproximadamente 5% do disponível no dispositivo, 5.723 registradores, 16 elementos DSP e 21 multiplicadores. Comparando com as arquiteturas propostas nesta dissertação de mestrado para o núcleo do ALF, mesmo considerando a arquitetura com os três tamanhos de núcleo do HM3, a versão que demanda mais amostras e, portanto, mais cálculos, utilizou apenas 1.163 elementos lógicos e 1.281 registradores.

Este resultado demonstra um elevado consumo de recursos de hardware do trabalho de (HAUTALA, BOUTELLIER e HANNUKSEDA, 2013), o que parece contrapor-se ao objetivo do trabalho, mencionado em seu título, que é uma solução com baixo consumo de potência.

Assim, uma solução embarcada para o problema do núcleo do ALF mostra-se bastante custosa em comparação às arquiteturas propostas em nosso trabalho, que podem ser facilmente adaptadas para outros formatos e tamanhos, apenas adicionando ou removendo operações em paralelo nos estágios do *pipeline*.

6 CONCLUSÕES

Esta dissertação apresentou um estudo sobre os algoritmos utilizados pelo mais novo padrão de compressão de vídeos, o HEVC, em seu *In-loop Filter*. Este estudo focou em duas inovações trazidas pelo HEVC, os filtros ALF e SAO. Em ambos os casos, este estudo tinha como objetivo o desenvolvimento de arquiteturas em hardware capazes de processar vídeos de alta definição. Para tanto, foram propostas simplificações no software de referência a fim de permitir o desenvolvimento de arquiteturas mais eficientes e menos custosas.

Inicialmente, buscou-se trazer uma visão geral sobre a codificação de vídeos e o padrão HEVC para fins de contextualização do trabalho. Em seguida, passou-se a apresentar a fundamentação teórica para o *In-loop Filter* do HEVC.

Posteriormente, passou-se à etapa de investigação algorítmica dos filtros ALF e SAO, buscando possíveis simplificações que pudessem viabilizar o desenvolvimento de arquiteturas em hardware buscando maior eficiência e menos custo.

Para o filtro SAO, as investigações algorítmicas focaram na função de custo interna usada na decisão do *offset* a ser utilizado. Esta função de custo é baseada em multiplicadores e divisores completos, além de utilizar dados em ponto flutuante. Ambos são pontos críticos para o desenvolvimento em hardware. Desta forma, foram propostas duas simplificações: a primeira delas propõe a substituição dos dados em ponto flutuante por dados inteiros ou então com precisão em ponto fixo. A segunda simplificação foca na eliminação dos multiplicadores e divisores, através da aplicação a técnica de *loop unrolling* às equações de distorção e custo usadas pelo SAO.

O impacto destas simplificações foi avaliado através de uma série de experimentos com o software de referência adaptado, de onde, concluiu-se que para utilização de dados inteiros há necessidade de um aumento de 0,055% no *bitrate* para a manutenção da qualidade de um vídeo comparado à codificação utilizando o

SAO original. Considerando ainda a utilização de dados em ponto fixo com precisão de 8 bits, o resultado médio é ainda uma redução de 0,0005% no *bitrate* para manutenção da mesma qualidade do SAO original.

Para o ALF, também foi realizada a investigação algorítmica que identificou como mais complexa a etapa de geração dos coeficientes com a utilização de dados em ponto flutuante combinados com uma série de operações matemáticas, que envolvem multiplicações e divisões. Com o objetivo de reduzir esta complexidade, foi proposta a utilização de dados inteiros em substituição aos dados em ponto flutuante.

O impacto desta simplificação foi avaliado através de uma série de experimentos considerando uma versão adaptada do software de referência. Os resultados demonstraram que com a utilização de dados inteiros há necessidade de um aumento no *bitrate* de 0,0477% em relação à versão que aplicação o ALF com dados em ponto flutuante. Ao comparar esta simplificação com uma versão que não aplica o ALF, ainda representa um ganho significativo, pois reduz o *bitrate* em 3,3795%.

A partir das simplificações propostas para a função de custo interna do SAO, optou-se pelo desenvolvimento de uma arquitetura que utiliza na função de custo uma precisão em ponto fixo de 8 bits, tendo em vista que, este se equivale à execução do software de referência original na maioria das sequências testadas. Esta arquitetura de custo, alcançou uma frequência de operação de 160 MHz e assim é capaz de processar até 1.634 quadros UHD 4K por segundo.

Também foi implementada a arquitetura responsável pelas demais etapas do SAO, a classificação das amostras de acordo com os tipos e categorias e a coleta de dados estatísticos para permitir a geração do *offset* a ser aplicado. A síntese desta arquitetura alcançou a frequência de 373 MHz e assim pode processar até 45 quadros UHD 4K por segundo.

Para o ALF, foram desenvolvidas arquiteturas para o núcleo deste filtro para as versões 3 e 5 do HM e do *Draft*, além de uma versão configurável para o HM 3. Na implementação dos núcleos dos filtros usados no HM 3, a arquitetura do formato 9x9 (pior caso) alcançou a frequência de operação de 253 MHz e assim é capaz de processar 30 quadros UHD 4K por segundo. Para o núcleo do filtro do HM5 também foi proposta um arquitetura, sendo que a mesma arquitetura é capaz de operar nos

dois formatos utilizados pelo software de referência. A frequência alcançada pela síntese foi de 292 MHz e, com isso, é possível processar até 35 quadros UHD 4K por segundo. A última versão desenvolvida neste trabalho para o núcleo do filtro ALF tira proveito da coexistência dos filtros menores dentro dos filtros maiores no ALF do HM3 e, desta forma, propõe uma arquitetura configurável, capaz de operar nos três formatos. Esta arquitetura reduziu o uso de recursos do dispositivo em mais de 30% se comparado à soma dos recursos das arquiteturas independentes e ainda assim alcançou a frequência de operação de 250 MHz, sendo capaz de processar 30 quadros UHD 4K por segundo, reduzindo a utilização de recursos do dispositivo em aproximadamente 50% em comparação a uma versão que implemente os três formatos de maneira independente.

Como trabalho futuro pretende-se continuar as investigações na função de custo do SAO, focando também na parte externa, buscando avaliar como alternativa a inserção da mesma função de custo interna para a decisão externa e como estas duas funções de custo funcionam em conjunto. Em termos de hardware, como trabalhos futuros propõe a avaliação da conexão entre as arquiteturas de classificação e coleta estatística com as funções de englobando as decisões internas e externas do SAO.

Para o ALF, pretende-se continuar as avaliações das etapas de geração dos coeficientes, buscando viabilizar uma arquitetura capaz de processar vídeos de alta definição em tempo real.

7 REFERÊNCIAS BIBLIOGRÁFICAS

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

BHASKARAN, V.; KONSTANTINIDES, K. **Image and Video Compression Standards:** Algorithms and Architectures. 2. ed. Boston: Kluwer Academic Publishers, 1997.

BJONTEGAARD, G. **Calculation of Average PSNR differences between RD-curves**. VCEG, VCEG-M33. Austin. 2001.

BRASIL. **Decreto n.º 4.901, Institui o Sistema Brasileiro de Televisão Digital - SBTVD, e dá outras providências**. [S.l.]. 2003.

BROSSEN, F. **Common Test Conditions and Software Reference Configurations**. JCT-VC, JCTVC-L1100. Genebra. 2013.

BUDAGAVI, M.; SZE, V.; ZHOU, M. **HEVC ALF Decode Complexity Analysis and Reduction**. IEEE International Conference on Image Processing. Brussels: [s.n.]. 2011.

CHEN, C.-Y. et al. The Adaptive Loop Filtering Techniques in the HEVC Standard. **Proceedings of the SPIE Applications of Digital Image Processing XXXV**, San Diego, Califórnia, EUA, v. 8499, Outubro 2012.

CHEN, H.-H. et al. **Fast Adaptive Loop Filter Algorithm for High Efficiency Video Coding**. IEEE International Conference on Consumer Eletronics - Berlin (ICCE-Berlin). Berlin: IEEE. 2012. p. 24-25.

CONCEIÇÃO, R. **Avaliação algorítmica e desenvolvimento de arquiteturas de hardware dedicadas para o filtro ALF do padrão HEVC**. Trabalho de Conclusão de Curso (Engenharia da Computação). Universidade Federal de Pelotas. Pelotas, p. 73. 2014.

CORRÊA, G. **Controle de Complexidade Computacional em Codificadores de Vídeo de Alta Eficiência**. Projeto de Tese (Doutorado em Engenharia Eletrotécnica e de Computadores) - Faculdade de Ciências e Tecnologia, Universidade de Coimbra. Coimbra, Portugal, p. 96. 2011.

DU, J.; YU, L. **A parallel and area-efficient architecture for deblocking filter and Adaptive Loop Filter**. IEEE International Symposium on Circuits and Systems (ISCAS). Rio de Janeiro: IEEE. 2011. p. 945-948.

FU, C.-M. et al. Sample Adaptive Offset for HEVC. **IEEE 13th International Workshop on Multimedia Signal Processing**, Hangzhou, 2011.

FU, C.-M. et al. Sample Adaptive Offset in the HEVC Standard. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 22, n. 12, p. 1755-1764, 2012.

GHANBARI, M. **Standards Codecs:** Image Compression to Advanced Video Coding. United Kingdom: The Institution of Electrical Engineers, 2003.

GONZALES, R.; WOODS, R. **Processamento de Imagens Digitais**. São Paulo: Edgard Blücher, 2003.

GOVINDU, G. et al. **Area, and Power Performance Analysis of a Floating-point based Application on FPGAs**. Proceedings of High Performance Embedded Computing Workshop (HPEC). Lexington: [s.n.]. 2003.

HAUTALA, I.; BOUTELLIER, J.; HANNUKSEDA, J. **Programmable Low Power Implementation of the HEVC Adaptive Loop Filter**. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). Vancouver: IEEE. 2013. p. 2664-2668.

ITU-T. **High Efficiency Video Coding: Recommendation ITU-T H.265**. [S.l.]. 2013.

JCT-VC. **High Efficiency Video Coding (HEVC) Text Specification Working Draft 1**. JCTVC-C403, JCT-VC Meeting. Guangzhou. 2010.

JCT-VC. **WD4: Working Draft 4 of High-Efficiency Video Coding**. JCTVC-F803, JCT-VC Meeting. Torino. 2011a.

JCT-VC. **HM4: High Efficiency Video Coding (HEVC) Test Model 4 Encoder Description**. JCTVC-F802, JCT-VC Meeting. Torino. 2011b.

JCT-VC. **High Efficiency Video Coding (HEVC) Text Specification Working Draft 7**. JCTVC-I1003, JCT-VC Meeting. Genebra. 2012a.

JCT-VC. **High Efficiency Video Coding (HEVC) Test Model 7 (HM 7) Encoder Description**. JCTVC-I1002, JCT-VC Meeting. Genebra. 2012b.

JCT-VC. **Inclusion of Adaptive Loop Filtering (ALF) in a New Profile or Main**. JCTVC-J0307, JCT-VC Meeting. Estocolmo. 2012c.

JCT-VC. **High Efficiency Video Coding (HEVC) Text Specification Draft 10**. JCTVC-L1003, JCT-VC Meeting. Genebra. 2013.

JVT EDITORS. **Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification**. (ITU-T REc.H.264|ISO/IEC 14496-10 AVC). [S.l.]. 2003.

KIM, I.-K. et al. **High Efficiency Video Coding (HEVC) Test Model 10 (HM10) Encoder Description**. JCT-VC, JCTVC-L1002_v3. Genebra. 2013.

KIM, W.; KWON, D. **Improved Sample Adaptive Offset for HEVC**. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). Vancouver: [s.n.]. 2013. p. 1700-1703.

MCCANN, K. et al. **Encoder-side description of HEVC Test Model (HM)**. JCT-VC, JCTVC-C402. Guangzhou. 2010.

MCCANN, K.; HAN, W.-J.; KIM, I.-K. **Samsung's Response to the Call for Proposals on Video Compression**. JCT-VC-A124. Dresda: JCT-VC. 2010.

MODY, M. et al. **High Throughput VLSI Architecture for HEVC SAO Encoding for Ultra HDTV**. IEEE International Symposium on Circuits and Systemas (ISCAS). Melbourne: IEEE. 2014. p. 2620-2623.

MRAK, M.; BARONCINI, V.; RAMZAN, N. **Report on HEVC compression performance**. JCT-VC, JCTVC-L1100. Valencia: [s.n.]. 2014.

MÜLLER, K. et al. 3D High-Efficiency Video Coding for Multi-View Video and Depth Data. **IEEE Transactions on Image Processing**, v. 22, n. 9, p. 3366-3378, 23 maio 2013.

NORKIN, A. et al. HEVC Deblocking Filter. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 22, n. 12, p. 1746-1754, 2012.

PARK, S.; CHOI, K.; JANG, E. CU Depth-based ALF Decision for Fast HEVC Encoding. **IEEE 16th International Symposium**, 2012.

PARK, S.; RYOO, K. **The hardware desing of effective SAO for HEVC decoder**. IEEE Global Conference on Consumer Electronics. Tokyo: IEEE. 2013. p. 303-304.

REDIESS, F. et al. **High Throughput Hardware Desing for the Adaptive Loop Filter of the Emerging HEVC Video Coding**. Symposium on Integrated Circuits and Systems Design. Brasília: IEEE. 2012.

REDIESS, F. et al. Cost Function Optimization and its Hardware Design for the Sample Adaptive Offset of HEVC Standar. **IEEE European Signal Processing Conference (EUSIPCO)**, Lisboa, 2014. 206-210.

REDIESS, F. et al. **Sample Adaptive Offset Filter Hardware Design for HEVC Encoder**. IEEE Visual Communications and Image Processing (VCIP). Valetta: IEEE. 2014.

RICHARDSON, I. **Video Codec Design - Developing Image and Video Compression Systems**. Chichester: Jon Wiley & Sons Publishers, 2002.

RICHARDSON, I. **H.264 and MPEG-4 Video Compression:** Video Coding for Next-generation Multimedia. USA: John Wiley & Sons Publishers, 2003.

SHI, Y.; SUN, H. **Image and Video Compression for Multimedia Engineering:** Fundamentals, Algorithms and Standards. Boca Raton: CRC Press, 1999.

SULLIVAN, G. et al. Overview of the High Efficiency Video Coding (HEVC) Standard. **IEEE Transactions on Circuits and Systems for Video Technology**, v. 22, n. 12, p. 1649-1668, 2012.

TSAI, C.-Y. et al. One-Pass Encoding Algorithm for Adaptive Loop Filter in High-Efficiency Video Coding. **IEEE Visual Communications on Image Processing**, Tainan, 2011.

TSAI, C.-Y. et al. Adaptive Loop Filtering for Video Coding. **IEEE Journal of Selected Topics on Signal Processing**, v. 7, n. 6, p. 934-945, 2013.

VANNE, J. et al. Comparative Rate-Distortion-Complexity Analysis of HEVC and AVC Video Codecs. **IEEE TRansactions on Circuits and Systems for Video Technology**, v. 22, n. 12, p. 1885-1898, dez 2012.

VCEG. **Adaptive (Wiener) Filter for Video Compression**. VCEG-AI14, VCEG Meeting. Berlin. 2008a.

VCEG. **Block-based Adaptive Loop Filter**. VCEG-AI18, VCEG Meeting. Berlin. 2008b.

VCEG. **Specification and experimental results of Quadtree-based Adaptive Loop Filter**. VCEG-AK22, VCEG Meeting. Yokohama. 2009.

WIEGAND, T. et al. **Working Draft 3 of High-Efficiency Video Coding**. JCTVC-E603. Genebra. 2011.

YOO, Y.-J. et al. Enhanced Adaptive Loop Filter for Motion Compensated Frame. **IEEE Transaction on Image Processing**, v. 20, n. 8, p. 2177-2188, 2011.

ZHU, J. et al. **A Combined SAO and De-blocking Filter Architecture for HEVC Video Decoder**. IEEE International Conference on Image Processing (ICIP). Melbourn: IEEE. 2013. p. 1967-1971.

ZHU, J. et al. **Fast SAO Estimation Algorithm and Its VLSI Architecture**. IEEE International Conference on Image Processing (ICIP). Paris: [s.n.]. 2014.

APÊNDICE A – RESULTADOS DA SIMPLIFICAÇÃO PARA DADOS INTEIROS E PONTO FIXO

	BD-rate (%)																	
	INTEIRO			**1 BIT**			**2 BITS**			**3 BITS**			**4 BITS**			**8 BITS**		
	I	L	R	I	L	R	I	L	R	I	L	R	I	L	R	I	L	R
Traffic 2560x1600	0,0239		0,1631	- 0,0010		- 0,0387	- 0,0009		-	- 0,0009		-	-		-	-		-
PeopleOnStreet 2560x1600	- 0,0384		0,0736	0,0007		- 0,0023	- 0,0002		- 0,0023	0,0001		- 0,0050	-		-	-		-
Kimono 1920x1080	0,0616	0,1682	- 0,0034	0,0003	- 0,0015	-	-	0,0006	-	-	-	-	-	-	-	-	-	-
ParkScene 1920x1080	0,0217	0,1991	0,0198	0,0001	0,0076	0,0055	0,0001	0,0122	- 0,0356	0,0003	0,0004	- 0,0019	-	0,0004	-	-	-	-
Cactus 1920x1080	0,0004	- 0,3508	- 0,4202	- 0,0004	- 0,0008	- 0,0262	- 0,0002	- 0,0101	- 0,0224	0,0003	0,0061	- 0,0014	-	0,0061	-	-	-	-
BQTerrace 1920x1080	0,0057	0,3919	0,2026	0,0106	0,0106	0,0083	0,0004	0,0104	- 0,0250	0,0000	-	- 0,0085	-	-	-	-	-	-
BasketballDrive 1920x1080	- 0,0367	0,1287	- 0,0551	-	- 0,1108	- 0,0077	0,0001	- 0,1512	- 0,0077	- 0,0001	- 0,1108	- 0,0077	-	- 0,1108	- 0,0077	-	-	- 0,0077
RaceHorses 832x480	0,0445	0,2073	0,0187	- 0,0014	0,0137	- 0,0332	- 0,0001	0,0180	- 0,0017	- 0,0003	-	-	-	-	-	-	-	-
BQMall 832x480	- 0,0635	0,1356	0,1446	- 0,0008	-	-	- 0,0015	0,0233	-	0,0007	-	-	-	-	-	-	-	-
PartyScene 832x480	0,0046	0,1301	0,0351	0,0003	0,0278	0,0624	0,0002	0,0109	0,0090	-	-	- 0,0018	-	-	-	-	-	-
BasketballDrill 832x480	- 0,1158	- 0,2020	0,1642	- 0,0005	- 0,0084	- 0,0498	- 0,0012	- 0,1026	- 0,0586	-	-	- 0,0600	-	-	- 0,0600	-	-	- 0,0586
RaceHorses 416x240	0,1054	0,0556	0,0887	- 0,0010	- 0,0748	0,0670	- 0,0010	- 0,0887	-	- 0,0003	- 0,1036	-	-	- 0,1036	-	-	-	-
BQSquare 416x240	- 0,0662	0,0841	- 0,1572	-	- 0,0136	- 0,0011	-	- 0,0069	- 0,0011	-	-	- 0,0011	-	-	-	-	-	-
BlowingBubbles 416x240	0,0955	0,1250	- 0,1078	0,0006	- 0,0150	0,0107	-	0,0029	0,0107	-	-	-	-	-	-	-	-	-
BasketballPass 416x240	0,0086	0,4275	- 0,0820	- 0,0002	-	0,0089	- 0,0005	-	-	- 0,0005	-	-	-	-	-	-	-	-
BaskeballDrillText 832x480	0,1296	0,1663	- 0,3152	0,0049	0,1097	- 0,1201	- 0,0017	0,0122	- 0,0049	- 0,0001	0,0310	-	-	0,0310	-	-	0,0350	-
ChinaSpeed 1024x768	0,0353	0,0281	0,1392	0,0000	0,0182	- 0,0181	- 0,0000	- 0,0028	0,0562	- 0,0003	- 0,0391	-	-	- 0,0429	-	-	-	-
SlideEditing 1280x720	0,0486	0,2212	1,6020	0,0002	- 0,1110	2,1063	0,0001	- 0,0214	0,0352	- 0,0001	-	-	-	- 0,0003	-	-	-	-
SlideShow 1280x720	- 0,0343	- 0,2174	0,6854	0,0084	- 0,0031	- 0,0029	0,0056	0,0020	- 0,0012	-	-	-	-	-	-	-	-	-
Médias	0,0121	0,0999	0,1156	0,0011	- 0,0089	0,1036	- 0,0000	- 0,0171	- 0,0026	- 0,0001	- 0,0127	- 0,0046	-	- 0,0129	- 0,0036	-	0,0021	- 0,0035

I: *Intra Only* L: *Low Delay* R: *Random Access*

APÊNDICE B – RESULTADOS DA SIMPLIFICAÇÃO DE *LOOP UNROLLING*

	BD-rate (%)																				
	LOOP HM NORMAL			**0 bit - inteiro**			**1 BIT**			**2 BITS**			**3 BITS**			**4 BITS**			**8 BITS**		
	I	L	R	I	L	R	I	L	R	I	L	R	I	L	R	I	L	R	I	L	R
Traffic 2560x1600	-		-	- 0,0014		0,0079	- 0,0010		- 0,0387	- 0,0007		0,0229	- 0,0009		-	-		-	-		-
PeopleOnStreet 2560x1600	-		-	0,0004		0,0293	0,0007		- 0,0023	- 0,0002		- 0,0113	0,0001		- 0,0050	-		-	-		-
Kimono 1920x1080	-	-	-	0,0001	0,0018	0,0250	0,0003	- 0,0015	-	-	0,0006	-	-	-	-	-	-	-	-	-	-
ParkScene 1920x1080	-	-	-	0,0003	0,0108	0,0757	0,0001	0,0076	0,0055	0,0001	0,0122	- 0,0356	0,0003	0,0004	- 0,0019	-	0,0004	-	-	-	-
Cactus 1920x1080	-	-	-	- 0,0004	- 0,0286	- 0,0093	- 0,0004	- 0,0262	- 0,0008	- 0,0004	- 0,0262	- 0,0008	0,0003	0,0061	- 0,0014	-	0,0061	-	-	-	-
BQTerrace 1920x1080	-	-	-	0,0010	0,0141	- 0,0521	0,0010	0,0106	0,0083	0,0004	0,0104	- 0,0250	0,0000	-	- 0,0085	-	-	-	-	-	-
BasketballDrive 1920x1080	-	-	-	0,0008	- 0,1713	0,0057	0,0013	- 0,1611	0,0532	0,0001	- 0,1512	- 0,0077	- 0,0001	- 0,1108	- 0,0077	-	- 0,1108	- 0,0077	-	-	- 0,0077
RaceHorses 832x480	-	-	-	- 0,0015	0,0683	- 0,0690	- 0,0014	0,0137	- 0,0332	- 0,0001	0,0180	- 0,0017	- 0,0003	-	-	-	-	-	-	-	-
BQMall 832x480	-	-	-	- 0,0008	- 0,0156	- 0,0369	- 0,0008	0,0177	- 0,0549	- 0,0015	0,0233	-	0,0007	-	-	-	-	-	-	-	-
PartyScene 832x480	-	-	-	0,0001	0,1400	0,0800	0,0003	0,0278	0,0624	0,0002	0,0109	0,0090	-	-	- 0,0018	-	-	-	-	-	-
BasketballDrill 832x480	-	-	- 0,0586	- 0,0012	- 0,0353	0,0049	- 0,0005	- 0,0498	- 0,0084	- 0,0012	- 0,1026	- 0,0586	-	-	- 0,0586	-	-	- 0,0586	-	-	- 0,0586
RaceHorses 416x240	-	-	-	- 0,0012	0,0180	0,0737	- 0,0010	- 0,0748	0,0670	- 0,0010	- 0,0887	-	- 0,0003	- 0,1036	-	-	- 0,1036	-	-	-	-
BQSquare 416x240	-	-	-	- 0,0003	- 0,0136	0,0298	-	- 0,0136	- 0,0011	-	- 0,0069	- 0,0011	-	-	- 0,0011	-	-	-	-	-	-
BlowingBubbles 416x240	-	-	-	0,0004	0,1706	0,0094	0,0006	- 0,0150	0,0107	-	0,0029	0,0107	-	-	-	-	-	-	-	-	-
BasketballPass 416x240	-	-	-	- 0,0001	0,1086	- 0,0063	- 0,0002	-	0,0089	- 0,0005	-	-	- 0,0005	-	-	-	-	-	-	-	-
BaskeballDrillText 832x480	-	-	-	0,0045	0,0557	- 0,1490	0,0049	0,1097	- 0,1201	- 0,0017	0,0122	- 0,0049	- 0,0001	0,0310	-	-	0,0310	-	-	0,0350	-
ChinaSpeed 1024x768	-	-	-	- 0,0002	0,0368	- 0,0168	0,0000	0,0182	- 0,0181	- 0,0000	- 0,0028	0,0562	- 0,0003	- 0,0391	-	-	- 0,0429	-	-	-	-
SlideEditing 1280x720	-	-	-	- 0,0001	- 0,0883	2,9288	0,0002	- 0,1110	2,1063	0,0001	- 0,0214	0,0352	- 0,0001	-	-	-	- 0,0003	-	-	-	-
SlideShow 1280x720	-	-	-	0,0094	0,0011	- 0,1319	0,0084	- 0,0031	- 0,0029	0,0056	0,0020	- 0,0012	-	0,0007	-	-	-	-	-	-	-
Médias	-	-	- 0,0031	0,0005	0,0161	0,1473	0,0007	- 0,0148	0,1075	- 0,0000	- 0,0181	- 0,0007	- 0,0001	- 0,0127	- 0,0045	-	- 0,0129	- 0,0035	-	0,0021	- 0,0035

I: *Intra Only* L: *Low Delay* R: *Random Access*

APÊNDICE C – PUBLICAÇÕES DERIVADAS DESTE TRABALHO

A.1) Trabalhos completos publicados em anais de congressos

REDIESS, Fabiane; CONCEICAO, Ruhan; ZATT, Bruno; PORTO, Marcelo; AGOSTINI, Luciano. Cost Function Optimization and its Hardware Design for the Sample Adaptive Offset of HEVC Standard. In: 22th European Signal Processing Conference (EUSIPCO), 2014, Lisboa. 22th European Signal Processing Conference (EUSIPCO).Piscataway: IEEE, 2014. p. 206-210.

REDIESS, Fabiane; CONCEICAO, Ruhan; ZATT, Bruno; PORTO, Marcelo; AGOSTINI, Luciano. Sample Adaptive Offset Filter Hardware Design for HEVC Encoder. In: 2014 IEEE Visual Communications and Image Processing (VCIP), 2014, Valleta. 2014 IEEE Visual Communications and Image Processing (VCIP). Piscataway: IEEE, 2014. p. 299-302.

CONCEICAO, Ruhan; REDIESS, Fabiane; ZATT, Bruno; PORTO, Marcelo; AGOSTINI, Luciano. Configurable hardware design for the HEVC-based Adaptive Loop Filter. In: 2014 IEEE 5th Latin American Symposium on Circuits and Systems (LASCAS), 2014, Santiago. 2014 IEEE 5th Latin American Symposium on Circuits and Systems. p. 1-4.

REDIESS, Fabiane; AGOSTINI, Luciano; CRISTANI, Cássio; DALLOGLIO, Pargles; PORTO, Marcelo. High Throughput Hardware Design for the Adaptive Loop Filter of the Emerging HEVC Video Coding. In: SBCCI 2012 25th Annual Symposium on Integrated Circuits and System Design, 2012, Brasília. SBCCI 2012 25th Annual Symposium on Integrated Circuits and System Design.Los Alamitos: IEEE, 2012. p. 1-5.

JUNES, Victor; DALLOGLIO, Pargles; REDIESS, Fabiane; PORTO, Marcelo; AGOSTINI, Luciano. An architecture for the new Adaptive Loop Filter of the High Efficiency Video Coding. In: Simpósio Sul de Microeletrônica, 2013, Porto Alegre. 28º Simpósio Sul de Microeletrônica, 2013.

REDIESS, Fabiane; CRISTANI, Cássio; DALLOGLIO, Pargles; PORTO, Marcelo; AGOSTINI, Luciano. Architectural Design for the Adaptive Loop Filter of the Emerging High Efficiency Video Coding Standard.In: South Symposium on

Microelectronics - SIM, 2012, São Miguel das Missões. 27th South Symposium on Microelectronics - SIM, 2012.

A.2) Resumos publicados em anais de congressos

REDIESS, Fabiane; CONCEIÇÃO, Ruhan; ZATT, Bruno; PORTO, Marcelo; AGOSTINI, Luciano. Otimizações na Função de Custo Interna do Filtro SAO do Padrão de Compressão de Vídeo HEVC. In:XVI ENPOS - Encontro de Pós-graduação da Universidade Federal de Pelotas. Pelotas, 2014.

REDIESS, Fabiane; ZATT, Bruno; PORTO, Marcelo; AGOSTINI, Luciano. Arquitetura Configurável para o Filtro Adaptativo de Laço do Padrão de Compressão de Vídeo HEVC. In: XV ENPOS - Encontro de Pós-graduação da Universidade Federal de Pelotas. Pelotas, 2013.

REDIESS, Fabiane; CRISTANI, Cássio; DALLOGLIO, Pargles; PORTO, Marcelo; AGOSTINI, Luciano. Desenvolvimento de Hardware para o Filtro Adaptativo de Laço do Padrão Emergente HEVC de Codificação de Vídeo. In: XIV ENPOS - Encontro de Pós-graduação da Universidade Federal de Pelotas. Pelotas, 2012.