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

Dissertação

DESENVOLVIMENTO DE UMA ARQUITETURA PARA ESTIMAÇÃO DE MOVIMENTO FRACIONÁRIA SEGUNDO O PADRÃO EMERGENTE HEVC

Vladimir Afonso

Pelotas, 2013

VLADIMIR AFONSO

DESENVOLVIMENTO DE UMA ARQUITETURA PARA ESTIMAÇÃO DE MOVIMENTO FRACIONÁRIA SEGUNDO O PADRÃO EMERGENTE HEVC

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

Orientador: Prof. Dr. Denis T. Franco
Co-Orientador: Prof. Dr. Luciano V. Agostini

PELOTAS
2013

Dados Internacionais de Catalogação na Publicação (CIP)

A257d Afonso, Vladimir
Desenvolvimento de uma arquitetura para estimação de movimento fracionária segundo o padrão emergente HEVC / Vladimir Afonso ; Denis Teixeira Franco, orientador ; Luciano Volcan Agostini, co-orientador. - Pelotas, 2013.
89 p. : il.

Dissertação (Mestrado em Computação) – Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2013.

1. Vídeo digital. 2. Padrão HEVC. 3. Estimação de movimento fracionária. I. Franco, Denis Teixeira, orient. II. Agostini, Luciano Volcan, co-orient. III. Título.

CDD: 621.381

Catalogação na Fonte: Leda Lopes CRB 10/ 2064

Banca examinadora:

.....
Prof. Dr. Denis Teixeira Franco (Orientador)

.....
Prof. Dr. Luciano Volcan Agostini (Co-orientador)

.....
Prof. Dr. Vagner Santos da Rosa

.....
Prof. Dr. Júlio Carlos Balzano de Mattos

.....
Prof. Dr. Marcelo Schiavon Porto

RESUMO

AFONSO, Vladimir. **Desenvolvimento de uma arquitetura para estimação de movimento fracionária segundo o padrão emergente HEVC**. 2013. 89f. Dissertação (Mestrado) – Programa de Pós-Graduação em Computação. Universidade Federal de Pelotas, Pelotas.

O processamento em tempo real de vídeos digitais de alta resolução está associado a uma elevada complexidade computacional, principalmente devido à necessidade do uso de técnicas de compressão de dados. Dessa forma, o desenvolvimento de circuitos integrados específicos para processamento de vídeo é uma atividade importante na área de pesquisa de sistemas digitais, uma vez que soluções em software geralmente não atingem os desempenhos necessários para diversas aplicações, em especial para dispositivos móveis. Os codificadores de vídeo apresentam diversas etapas distintas, como transformadas, quantização, codificação de entropia e estimação de movimento (ME – *Motion Estimation*), entre outras. A etapa ME é a que mais contribui para a redução na quantidade de dados a serem transmitidos, sendo que a mesma ainda pode utilizar uma técnica de refinamento chamada estimação de movimento fracionária (FME – *Fractional Motion Estimation*), a qual contribui para melhorar os resultados obtidos. Inúmeros artigos científicos podem ser encontrados na literatura propondo arquiteturas para a FME do padrão de codificação de vídeo H.264/AVC (*Advanced Video Coding*). Porém, existem poucos trabalhos relacionados com a FME do padrão *High Efficiency Video Coding* (HEVC), que ainda está em desenvolvimento e será o sucessor do padrão H.264/AVC. Portanto, se faz necessário o desenvolvimento de arquiteturas eficientes para a etapa de FME do padrão HEVC. Este trabalho apresenta o estudo algorítmico e o desenvolvimento de sistemas de hardware para a implementação da FME segundo o padrão de codificação de vídeo HEVC. Os resultados de síntese mostram que o hardware desenvolvido é capaz de processar vídeos *Full* HD (1920x1080 pixels) e QFHD (3840x2160 pixels) em tempo real.

Palavras-chave: Vídeo Digital, Padrão HEVC, Estimação de Movimento Fracionária.

ABSTRACT

AFONSO, Vladimir. **Desenvolvimento de uma arquitetura para estimação de movimento fracionária segundo o padrão emergente HEVC**. 2013. 89f. Dissertação (Mestrado) – Programa de Pós-Graduação em Computação. Universidade Federal de Pelotas, Pelotas.

The real time processing of high-resolution digital videos is associated with a high computational complexity, mainly due to the necessity of using data compression techniques. Because of this, the development of specific integrated circuits for video processing is an important area of research in digital systems, since software solutions do not allow the required performance for several applications, especially mobile devices. The video encoders have several distinct stages, as transforms, quantization, entropy coding and motion estimation (ME), among others. The step of ME is the most important for the reduction of information to be transmitted, and can use a refinement technique called fractional motion estimation (FME), which helps to improve the results. Many articles can be found in the scientific literature proposing FME architectures for the H.264/AVC (Advanced Video Coding) standard. However, there are few works with FME for the High Efficiency Video Coding (HEVC) standard, which is under development and will be the successor to the H.264/AVC standard. Therefore, the development of efficient architectures for FME according to the HEVC standard is necessary. This work presents the algorithmic study and development of hardware implementations developed for the FME defined in the HEVC standard. The synthesis results show that the developed hardware is able to process Full HD (1920x1080 pixels) and QFHD (3840x2160 pixels) videos in real time.

Keywords: Digital Video, HEVC Standard, Fractional Motion Estimation.

AGRADECIMENTOS

Em primeiro lugar, agradeço aos meus pais, Eledir Lobo Afonso e Neli Afonso, que apesar de todas as adversidades imagináveis, tomaram a decisão de permitir que eu deixasse a pequena cidade de Herval, ainda adolescente, para cursar um curso técnico em Eletrônica na antiga ETFPel em Pelotas. Por sabedoria, eles me deram uma das coisas mais importantes, senão a mais importante, que pais podem dar aos seus filhos, a oportunidade de estudar e ter uma profissão. Me deram a oportunidade a qual eles mesmos não puderam gozar plenamente durante suas vidas. Agradeço a eles também, bem como aos meus demais familiares e meus amigos, pela compreensão da ausência em diversos momentos durante o desenvolvimento desse trabalho.

Faço um agradecimento especial ao meu orientador, Prof. Dr. Denis Franco e ao meu co-orientador, Prof. Dr. Luciano Agostini, que além de me oportunizarem a pesquisa de um tema extremamente relevante, disponibilizaram parte do seu tempo, já tão exíguo, para me auxiliar de maneira ativa no desenvolvimento de cada etapa desse trabalho. Foram de fato orientadores, sou grato pelas discussões, revisões e todas outras contribuições dadas por vocês.

A todos os professores e bolsistas do Grupo de Arquiteturas e Circuitos Integrados, o GACI, fica o meu agradecimento pela oportunidade de aprendizado e pela boa convivência estabelecida durante esses dois anos de pesquisa. Em especial, agradeço ao colega mestrando Ricardo Jeske e aos bolsistas de Iniciação Científica Henrique Maich, Marcel Corrêa e Mateus Grellert, que colaboraram ou participaram de forma importante nesse trabalho, em diferentes momentos.

Por fim, agradeço a todos aqueles, que de alguma forma, seja participando da minha formação profissional ou como ser humano, contribuíram para que mais este objetivo fosse alcançado.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ALF	*Adaptive Loop Filter*
ALUT	*Adaptive Look-Up Table*
AVC	*Advanced Video Coding*
BMA	*Block Matching Algorithm*
CABAC	*Context-Based Adaptive Binary Arithmetic Coding*
CAVLC	*Context-Based Adaptive Variable Length Coding*
CTU	*Coding Tree Unit*
CU	*Coding Unit*
DCT	*Discrete Cosine Transform*
DS	*Diamond Search*
DVD	*Digital Versatile Disk*
EPZS	*Enhanced Predictive Zonal Search*
FIR	*Finite Impulse Response*
FME	*Fractional Motion Estimation*
FPGA	*Field Programmable Gate Array*
FS	*Full Search*
GACI	Grupo de Arquiteturas e Circuitos Integrados
GPB	*Generalized P and B picture*
HD	*High Definition*
HE10	*High Efficiency* 10
HEVC	*High Efficiency Video Coding*
HM	*High Efficiency Video Coding Test Model*
HM1	*High Efficiency Video Coding Test Model 1 Encoder Description*
HM9	*High Efficiency Video Coding Test Model 9 Encoder Description*
IDR	*Instantaneous Decoding Refresh*
IEC	*International Electrotechnical Commission*
ISO	*International Organization for Standardization*
ITU-T	*International Telecommunication Union – Telecommunication Standardization Sector*

JCT-VC	*Joint Collaborative Team on Video Coding*
LCU	*Largest Coding Unit*
MC	*Motion Compensation*
ME	*Motion Estimation*
MHz	Mega-Hertz, 10^6 hertz, unidade de frequência
MPEG	*Moving Picture Experts Group*
MSB	*Most Significant Bit*
MSE	*Mean Squared Error*
PIXEL	*Picture Element*
PSNR	*Peak-to-Signal Noise Ratio*
PU	*Prediction Unit*
QFHD	*Quad Full* HD
QP	*Quantization Parameter*
qps	quadros por segundo
ROM	*Read-Only-Memory*
SAD	*Sum of Absolute Differences*
TMuC	*Test Model under Consideration*
TU	*Transform Unit*
TV	Televisão
UFPel	Universidade Federal de Pelotas
UP	Unidade de Processamento
VCEG	*Video Coding Experts Group*
VLSI	*Very-Large-Scale Integration*
VHDL	VHSIC *Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuit*
WQVGA	*Wide Quarter Video Graphics Array*
WQXGA	*Wide Quad eXtended Graphics Array*
WVGA	*Wide Video Graphics Array*
XGA	*eXtended Graphics Array*

SUMÁRIO

1 INTRODUÇÃO

Existem, atualmente, diversas aplicações envolvendo vídeo digital, como aparelhos de DVD e Blu-Ray, TV digital, compartilhamento de vídeos online, vídeo-telefonia, entre outras. A elevada complexidade computacional associada ao processamento em tempo real de vídeos digitais de alta definição pode ser tratada de forma mais eficiente através do desenvolvimento de circuitos integrados específicos, já que de outra forma, com soluções em software, seriam necessários processadores de alto desempenho. A utilização deste tipo de processadores resultaria em um aumento de consumo energético e inviabilizaria a execução de vídeos digitais em algumas aplicações, como dispositivos portáteis.

O algoritmo de codificação, responsável pela compressão das sequências de vídeo originais (não processadas), apresenta diversas etapas, cada uma das quais com o objetivo de explorar algum tipo de redundância nos dados existentes nessas sequências. Desta forma, é possível reduzir a quantidade de dados a ser armazenada e transferida, com mínimas perdas na qualidade da imagem ou até sem perda alguma.

Dentre os padrões de compressão disponíveis atualmente, o padrão H.264/AVC (*Advanced Video Coding*) é o mais eficiente em termos de desempenho na compressão de dados, ao custo de uma maior complexidade computacional em relação aos padrões concorrentes (AGOSTINI, 2007). Desde a sua aprovação em 2003, o padrão H.264/AVC vem sendo investigado exaustivamente pelos pesquisadores. Nos últimos anos, com a evolução da capacidade computacional dos equipamentos eletrônicos, foram possíveis inúmeras contribuições para melhorar a eficiência do padrão. Contudo, a demanda por resoluções cada vez maiores, inclusive no que se refere a aplicações destinadas aos dispositivos com recursos

computacionais ou energéticos limitados, como é o caso dos dispositivos portáteis, estimulou o desenvolvimento de um novo padrão.

Com o objetivo principal de dobrar a taxa de compressão permitida pelo padrão H.264/AVC, vem sendo desenvolvido o padrão *High Efficiency Video Coding* (HEVC) (SULLIVAN; WIEGAND, 2009). O padrão HEVC está em desenvolvimento desde Janeiro de 2010 para ser o sucessor do padrão H.264/AVC e está sendo definido através da colaboração de especialistas reunidos em um grupo intitulado *Joint Collaborative Team on Video Coding* (JCT-VC). Mesmo com o padrão HEVC ainda em desenvolvimento, já é possível verificar que novas características foram introduzidas a fim de aumentar as taxas de compressão (KIM *et al.,* 2012) (BROSS *et al.,* 2012). Algumas destas características estão relacionadas à predição inter-quadros, mais especificamente à etapa de estimação de movimento (ME – *Motion Estimation*).

Considerando um codificador de vídeo genérico, a etapa ME visa à identificação e redução, ou até à eliminação, das redundâncias do tipo temporal, que consistem na grande semelhança que quadros temporalmente próximos costumam apresentar em uma sequência de vídeo. Dessa forma, é possível transmitir somente as informações referentes à diferença entre estes quadros. Esta é a etapa que apresenta a maior complexidade computacional de todo o codificador, mas também é a etapa que contribui para a maior parte dos ganhos em termos de taxa de compressão (PURI, 2004).

Os movimentos existentes entre quadros temporalmente vizinhos normalmente não se limitam a posições inteiras de pixel (*picture element*) e, por isso, a ME emprega a técnica de estimação de movimento fracionária (FME – *Fractional Motion Estimation*) com o objetivo de obter maior eficiência na codificação utilizando posições de sub-pixel, além das posições inteiras (RICHARDSON, 2003). Para tal, a etapa de FME é composta, basicamente, de duas partes: (a) uma etapa de interpolação, na qual são gerados os valores das amostras em posições de sub-pixel, e (b) uma etapa de busca e comparação, onde os valores gerados são comparados com o melhor resultado da ME em posições inteiras. O desenvolvimento de arquiteturas eficientes para FME é muito importante, uma vez que uma parte significativa da complexidade e dos ganhos de compressão obtidos se deve a esta etapa.

1.1 Motivação

Diversas são as motivações para o desenvolvimento de arquiteturas de hardware para FME do padrão HEVC. Sem dúvida, o fato de que o padrão HEVC ainda se encontra em desenvolvimento e, por isso, existem poucos trabalhos relacionados na literatura científica, é extremamente motivador para os pesquisadores da área de codificação de vídeo. Considerando que foram encontrados, até o momento, poucos trabalhos relacionados à FME como (GUO *et al.,* 2012), a motivação se torna ainda maior. Além disso, os dispositivos eletrônicos que permitem a execução de vídeos digitais são cada vez mais exigidos, pois, sistematicamente, estes equipamentos têm seus recursos ampliados, como por exemplo, a execução de vídeos digitais com resoluções crescentes em dispositivos portáteis, o que já é uma realidade. Por isso, se faz necessário o desenvolvimento de arquiteturas de hardware cada vez mais eficientes, seja em termos de velocidade, consumo energético ou quantidade de hardware utilizado.

1.2 Objetivos

Este trabalho tem como objetivos principais o estudo e o desenvolvimento de uma arquitetura de hardware com elevada taxa de processamento para a FME do padrão de codificação de vídeo HEVC.

1.3 Organização do Trabalho

Este trabalho está estruturado em sete capítulos, de forma que nos capítulos iniciais são abordados alguns conceitos introdutórios para a compreensão dos capítulos seguintes, enquanto que nos capítulos finais são apresentadas a arquitetura de hardware desenvolvida, os esquemas propostos e resultados obtidos, assim como as conclusões deste trabalho.

O segundo capítulo apresenta alguns conceitos importantes sobre compressão de vídeos digitais, como as métricas utilizadas para comparação, a estimação de movimento e a estimação de movimento fracionária, além de outros conceitos básicos relacionados ao assunto.

Em seguida, no terceiro capítulo, são apresentados alguns conceitos referentes ao padrão de codificação de vídeo HEVC. Além disso, é apresentado o estado da arte em termos de arquiteturas para FME e são realizadas algumas comparações entre os padrões de codificação de vídeo atuais.

No quarto capítulo são apresentadas as avaliações realizadas utilizando o software de referência do HEVC. O quinto capítulo, por sua vez, apresenta a metodologia empregada no desenvolvimento da arquitetura de hardware, bem como os diagramas do hardware proposto.

A seguir, no sexto capítulo, são descritos os resultados obtidos com a arquitetura desenvolvida.

Enfim, no sétimo e último capítulo são apresentadas as conclusões deste trabalho, bem como as oportunidades de estudos em trabalhos futuros.

2 CONCEITOS DE COMPRESSÃO DE VÍDEOS DIGITAIS

Este capítulo tem por objetivo apresentar alguns conceitos importantes sobre compressão de dados em vídeos digitais. Os conceitos apresentados neste capítulo estão estreitamente relacionados com a arquitetura de hardware desenvolvida neste trabalho, a qual será apresentada nos próximos capítulos.

2.1 Conceitos Fundamentais de Vídeos Digitais

Um vídeo digital consiste em uma sequência de imagens independentes, captadas com um determinado intervalo de tempo entre as mesmas. Estas imagens independentes, que formam uma sequência de vídeo, são chamadas de quadros (*frames)* e são compostas por pixels. Os pixels são os pontos que formam a imagem e são representados através de três amostras, que correspondem às componentes de brilho ou cor, conforme o sistema utilizado para representação das cores. Os quadros de uma sequência de vídeo, compostos pelos pixels, também podem ser divididos em blocos pelos codificadores. Estes blocos podem apresentar diferentes tamanhos de acordo com o padrão de codificação e, inclusive, em um mesmo padrão o tamanho do bloco pode ser variável, permitindo a aplicação mais eficiente das técnicas de compressão. Na Figura 2.1 pode ser observada uma sequência de quadros temporalmente próximos e um destes quadros divididos em blocos.

A representação digital de uma cena envolve, basicamente, dois tipos de amostragem: espacial e temporal. A amostragem espacial consiste em uma matriz de pontos chamada de resolução do vídeo, que tem o objetivo de formar os quadros da sequência de vídeo. Considerando que determinadas características das amostras sejam mantidas, como o número de bits, a qualidade da imagem será melhor, quanto maior for a resolução do vídeo, uma vez que mais pixels serão

usados na representação dos quadros. Já a amostragem temporal está relacionada aos intervalos de tempo entre a captura das imagens em uma sequência de vídeo. A percepção da movimentação no vídeo será melhor, quanto maior for a taxa de amostragem temporal. Para que seja obtida uma sensação de tempo real, a taxa de captura das imagens deve ser de, no mínimo, 24 a 30 imagens a cada segundo (GONZALEZ, 2003), sendo que algumas aplicações usam 60 imagens por segundo ou mais.

Figura 2.1 – Sequência de quadros e representação em blocos de um quadro.

2.2 Codificadores de Vídeo Digital

Na Figura 2.2 pode ser observado o diagrama em blocos de um modelo de codificador de vídeo de acordo com a maioria dos padrões de compressão de vídeo atuais (AGOSTINI, 2007), incluindo os padrões H.264/AVC e HEVC.

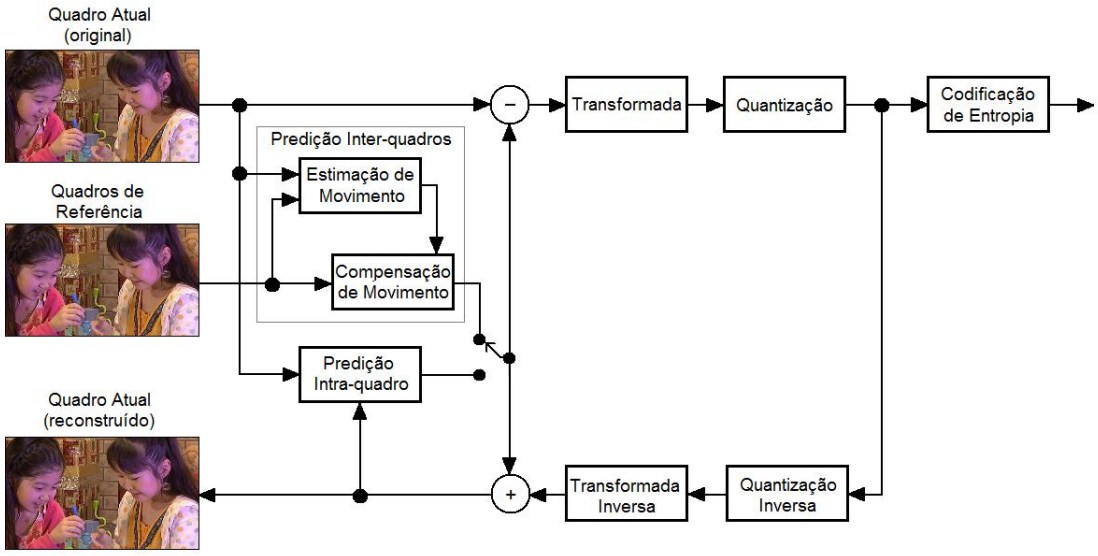


Figura 2.2 – Modelo de codificador de vídeo.

Como pode ser observado na Figura 2.2, na predição inter-quadros estão localizadas a ME e a compensação de movimento (MC - *Motion Compensation*). A ME tem como objetivo identificar a redundância existente entre quadros temporalmente próximos e, para tal, utiliza quadros processados anteriormente, chamados quadros de referência. A ME será mais bem explicada nas seções seguintes. Na etapa MC são construídos os quadros estimados a partir das informações de movimento geradas pela ME.

A etapa de predição intra-quadro permite a redução de redundâncias espaciais, que estão relacionadas com a tendência que pixels vizinhos de um mesmo quadro têm de apresentar valores semelhantes. Para aplicação da técnica, são necessárias apenas as informações do quadro atual. Este quadro é dividido em blocos, permitindo a comparação do bloco que está sendo codificado com outros blocos do quadro.

A chave seletora representada na Figura 2.2, corresponde a uma etapa de controle no codificador, a qual é responsável por escolher entre as predições inter-quadros ou intra-quadro, a que será utilizada em cada bloco, de acordo com as características do vídeo a ser codificado. Em seguida, uma operação de subtração entre os valores do quadro atual e do quadro reconstruído é realizada, permitindo a obtenção de um resíduo.

Após a obtenção do resíduo, existe a etapa chamada de transformada, cujo objetivo é transformar as informações do domínio espacial para o domínio das frequências. Esta transformação é necessária para que a operação chamada de quantização possa ser utilizada. O princípio de funcionamento da quantização está baseado na eliminação das frequências menos relevantes ao sistema visual humano. A quantização gera perdas no processo de codificação. Porém, em geral, estas perdas podem ser controladas e interferem de forma pouco significativa na qualidade da imagem.

Em seguida, após a quantização, é realizada a codificação de entropia. A codificação de entropia consiste de uma técnica de compressão que visa à alteração na representação dos símbolos com o uso de codificação de comprimento de palavra variável, sendo que diversos tipos de algoritmos podem ser utilizados.

Ainda se fazem necessárias, no codificador, as etapas de transformada e quantização inversas para geração do quadro atual reconstruído. Este quadro é

obtido através da soma do resíduo com a saída da transformada inversa e é muito importante, pois é utilizado como quadro de referência para a predição inter-quadros ou intra-quadro, na codificação dos blocos ou quadro subsequentes. Estas operações existem para permitir que codificadores e decodificadores utilizem as mesmas referências, já que a quantização gera perdas irreversíveis no processo de codificação.

2.3 Métricas de Comparação Aplicadas a Vídeos Digitais

Respeitada a especificação do decodificador, inúmeras técnicas podem ser empregadas nos codificadores para a compressão de vídeos digitais. Para tal, é fundamental a utilização de critérios para comparação entre o vídeo original e o vídeo comprimido, a fim de verificar a eficiência da compressão, ou seja, a redução na quantidade de dados a serem transmitidos e armazenados e os efeitos resultantes da compressão na qualidade da imagem. Além disso, algumas métricas são necessárias durante a codificação propriamente dita, uma vez que o codificador, ao empregar as diversas técnicas de compressão disponíveis, deve fazê-lo de maneira que alcance os melhores resultados possíveis em termos de qualidade da imagem e compressão.

A avaliação da eficiência em termos de taxas de compressão é relativamente simples, uma vez que uma medição do *bit-rate* (taxa de bits) sem a utilização de determinada técnica de compressão, e outra medição, com a utilização dessa técnica, permite verificar a redução obtida na quantidade dos dados. Uma redução no valor do *bit-rate* produzido para um mesmo vídeo fonte está sempre relacionada a um aumento na taxa de compressão.

A avaliação de qualidade da imagem dos vídeos digitais pode ser feita utilizando métricas objetivas e subjetivas (RICHARDSON, 2003). As métricas objetivas permitem verificar as alterações na qualidade do vídeo com a utilização de equações matemáticas. Estas equações são aplicadas aos valores dos pixels, antes e depois da compressão. Um problema relacionado a este tipo de métrica é que alguns fatores subjetivos que poderiam ser observados por um ser humano não são levados em consideração. Mesmo assim, o critério objetivo PSNR (*Peak Signal-to-Noise Ratio*), é o critério mais utilizado (GHANBARI, 2003).

O PSNR pode ser aplicado a um bloco, um quadro ou até mesmo ao vídeo todo, e o valor é fornecido em decibéis (dB). O cálculo do PSNR utiliza o critério de similaridade erro médio quadrático (MSE – *Mean Squared Error*), como pode ser observado nas equações (1) e (2). Nestas equações, *MSE* representa o erro médio quadrático dos pixels de um quadro, *R* e *O* representam os quadros reconstruído e original, respectivamente, *m* e *n* são as dimensões do quadro e *MAX* é o valor máximo das amostras. Em geral, quanto maior for o valor resultante do PSNR, maior será a qualidade da imagem (RICHARDSON, 2002). Porém, como mencionado anteriormente, existe a possibilidade de um bom resultado de PSNR, com valor elevado, ser de um quadro com uma qualidade visual subjetiva inferior a um resultado de PSNR de valor menor, uma vez que determinadas partes da imagem podem ser mais importantes que outras, quando observadas por um ser humano (RICHARDSON, 2003).

$$PSNR_{dB} = 20 \cdot \log_{10}\left(\frac{MAX}{\sqrt{MSE}}\right) \quad (1)$$

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\left(R_{i,j} - O_{i,j}\right)^2 \quad (2)$$

Além do MSE, existem outros critérios de similaridade que podem ser utilizados durante a codificação propriamente dita, permitindo ao codificador tomar a decisão de qual o melhor caminho a ser seguido na codificação. Entre estes critérios está a soma das diferenças absolutas (SAD – *Sum of Absolute Differences*) (KUHN, 1999), bastante conhecido e utilizado. A grande vantagem do SAD em relação ao MSE está na simplicidade do critério para o desenvolvimento em hardware dos algoritmos de codificação. Apesar de ser mais simples, e gerar resultados um pouco inferiores aos de outros critérios como o MSE, o SAD pode ser implementado utilizando apenas operações de soma e subtração, o que não é possível com o MSE, que utiliza operações como divisão e exponenciação. A equação (3) mostra como o SAD pode ser calculado. As grandezas *R* e *O* representam os quadros reconstruído e original, respectivamente, e *m* e *n* são as dimensões do quadro.

$$SAD = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} \left| R_{i,j} - O_{i,j} \right| \quad (3)$$

As métricas de comparação apresentadas nessa seção foram as utilizadas nas avaliações e no desenvolvimento da arquitetura deste trabalho. O SAD, em especial, será melhor explicado nas seções seguintes, uma vez que é muito importante na ME desenvolvida neste trabalho.

2.4 Estimação de Movimento

Considerando um modelo de codificador de vídeo atual, a ME é a etapa que visa à identificação e redução das redundâncias do tipo temporal. As redundâncias temporais podem ser observadas de diferentes formas. Por exemplo, o fundo de uma cena sem movimentação de câmera tende a manter os mesmos valores de pixel durante vários quadros. Por outro lado, durante a movimentação de um objeto em uma cena, muitas vezes os valores de blocos de pixels deste objeto pouco se alteram, apenas sofrem um deslocamento da sua posição em relação aos demais quadros. Dessa forma, a ME é empregada para explorar estas similaridades, predizendo o quadro atual, que está sendo codificado, utilizando quadros de referência.

O quadro atual é dividido em blocos e estes são comparados aos blocos do quadro de referência. O critério de similaridade utilizado para esta comparação é aplicado aos blocos de uma área de busca (ou área de pesquisa), no quadro de referência, correspondente à área ao redor da posição original do bloco a ser codificado. Desta forma, próximo à posição original, a probabilidade de se obter um bom resultado é maior (KUO, 2009) (LAI, 2010). Além disso, a aplicação do critério em uma área limitada permite uma redução significativa da complexidade computacional quando comparado à área de um quadro inteiro. Em seguida, é gerado um vetor de movimento que irá permitir a localização do bloco que gerou o melhor casamento com o bloco atual, ou seja, a maior semelhança entre o bloco do quadro atual e o bloco de referência. Com isso, além do vetor, só é necessário o

envio do resíduo resultante da diferença entre esses blocos. Esta visão geral do processo de ME pode ser observada na Figura 2.3 (PORTO, 2012, p.37).

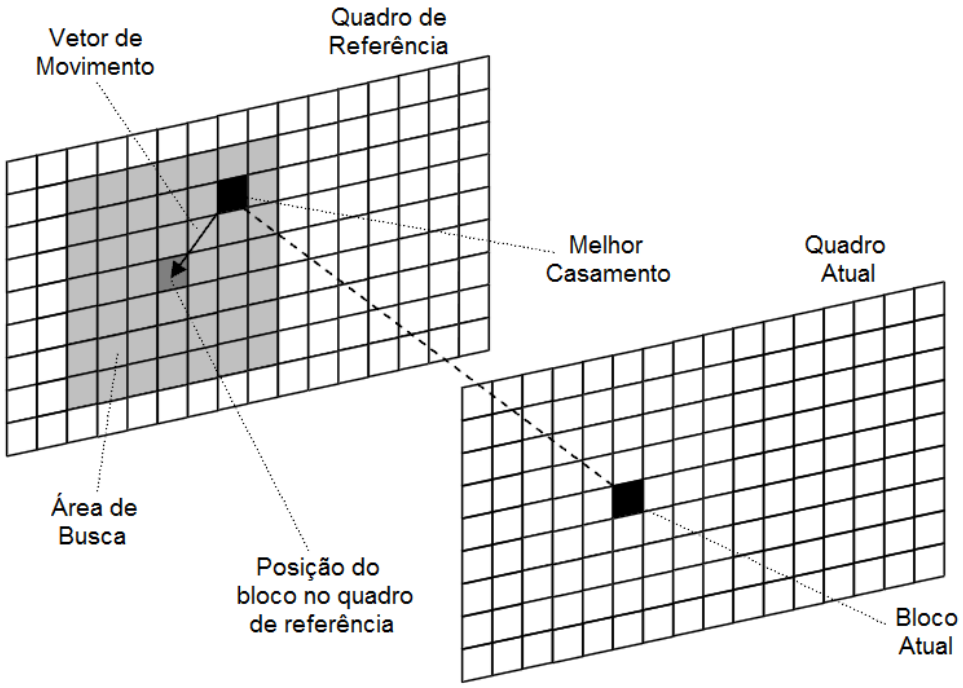


Figura 2.3 – Elementos da estimação de movimento.

Na etapa seguinte, a MC, é construído o quadro predito, baseado nas informações dos vetores de movimento gerados pela ME. Quanto menor for o resíduo gerado na codificação, melhores serão os resultados de compressão, sendo que o resíduo ideal tem valor nulo, ou seja, representa um bloco de vídeo candidato exatamente igual ao bloco a ser codificado. No HEVC, estes resultados de resíduo são gerados para todos os tamanhos de blocos possíveis, mas apenas o particionamento que apresentar o melhor resultado global (avaliando taxa de compressão e qualidade da imagem) é selecionado para a codificação. Durante a codificação no padrão emergente HEVC os resultados do resíduo são gerados utilizando-se basicamente o SAD como critério de similaridade, embora outros critérios também possam ser usados, sem prejuízo de compatibilidade com as definições do padrão.

A busca pelo bloco de melhor casamento dentro da área de pesquisa é realizada de acordo com algum algoritmo de busca. O tipo de algoritmo de busca utilizado está diretamente relacionado à complexidade computacional e à qualidade dos vetores de movimento gerados. Os algoritmos mais utilizados são chamados de *Block Matching Algorithm* (BMA), nos quais os quadros são divididos em blocos. Os

algoritmos de busca podem ser de dois tipos: ótimos, quando verificam todos os blocos dentro da área de busca gerando um resíduo ótimo; e subótimos ou rápidos, quando são utilizadas heurísticas para acelerar a busca, as quais reduzem a quantidade de blocos a serem comparados. Algoritmos subótimos permitem a redução da complexidade computacional podendo apresentar igualmente bons resultados de resíduo (PORTO, 2012).

Existem inúmeros algoritmos de busca publicados na literatura. Entre estes algoritmos pode-se destacar o Busca Completa (FS – *Full Search*) (BHASKARAN, 1999) (LIN, 2005) e o *Diamond Search* (DS) (KUHN, 1999) (ZHU, 2000) (YI, 2005), bastante conhecidos e utilizados.

O algoritmo FS é um algoritmo ótimo, pois é realizada a avaliação de todos os blocos candidatos da área de busca. Já a maneira como o bloco é deslocado na comparação pode ocorrer de diversas formas, como por exemplo, de um canto superior ou inferior da área de busca ao canto diagonal oposto, ou em espiral a partir do centro da área de busca (RICHARDSON, 2002). Apesar de o FS apresentar uma elevada complexidade computacional, o algoritmo não apresenta dependência de dados, e possibilita uma extensiva exploração do paralelismo, o que é bastante interessante visando uma implementação em hardware.

No software de referência HM (HEVC *Test Model*) do padrão HEVC, além do algoritmo FS, está disponível o algoritmo *Enhanced Predictive Zonal Search* (EPZS) (TOURAPIS, 2002). Como neste trabalho o algoritmo utilizado foi o FS, não serão apresentados maiores detalhes do EPZS.

2.5 Estimação de Movimento Fracionária

A FME é uma técnica importante que pode ser empregada na etapa ME, com o objetivo de gerar resíduos com menor amplitude e, consequentemente, ampliar a eficiência na codificação. Basicamente, é realizada uma interpolação entre posições inteiras de amostras no quadro de referência, permitindo a busca em posições interpoladas de sub-pixel, além das posições inteiras de pixel, como pode ser observado na Figura 2.4. A FME é utilizada por padrões atuais, como o HEVC e o H.264/AVC. O padrão H.264/AVC prevê a utilização de vetores de movimento com precisão de ½ pixel e ¼ de pixel. No padrão HEVC, o processo de FME sofreu

alterações que permitem melhorias na qualidade da codificação, mas manteve a utilização de vetores de movimento com precisão de ½ pixel e ¼ de pixel.

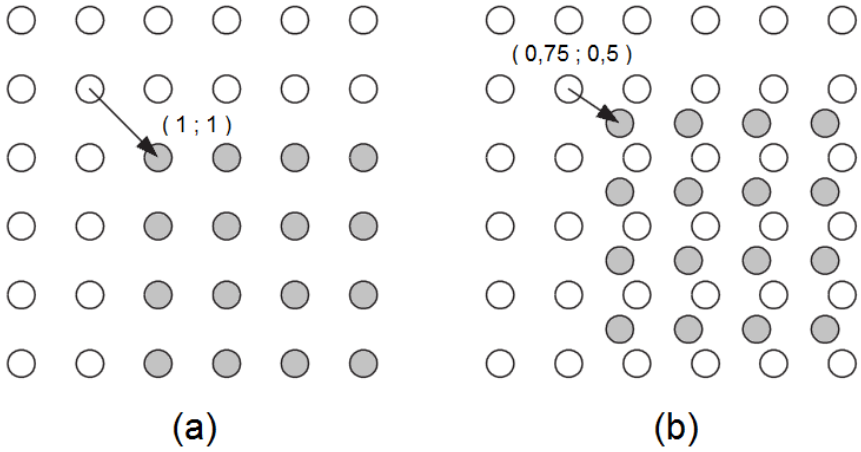


Figura 2.4 – Vetores de movimento: (a) ME em posições inteiras e (b) FME.

A técnica de FME permite uma maior eficiência na codificação, pois possibilita bons casamentos entre os blocos dos quadros atual e de referência, reduzindo os resíduos e, consequentemente, permitindo o armazenamento de um número menor de informações. Isto é possível porque os movimentos existentes entre quadros temporalmente vizinhos normalmente não se limitam a posições inteiras de pixel. Contudo, à medida que a precisão aumenta na FME, os ganhos obtidos em termos de compressão se tornam cada vez menos significativos. Isto acontece em decorrência da necessidade de mais vetores de movimento e, consequentemente, mais bits na representação da FME (CARVALHO, 2007). Nos primeiros documentos do HEVC foi cogitada a utilização de precisão de 1/8 de pixel para as amostras de luminância, o que não se confirmou.

O processo de FME é composto, basicamente, de duas etapas: (a) uma etapa de interpolação, na qual são gerados os valores das amostras em posições de sub-pixel e (b) uma etapa de busca, onde os valores gerados são comparados com o melhor casamento em posições inteiras (CORRÊA, 2011a) (CORRÊA, 2011b).

3 PADRÃO EMERGENTE HEVC DE CODIFICAÇÃO DE VÍDEO

Neste capítulo é apresentada uma visão geral do padrão emergente de codificação de vídeo HEVC. Algumas características importantes são introduzidas, como o funcionamento da FME no HEVC, foco deste trabalho. Também é apresentado o estado da arte em termos de arquiteturas para FME dos padrões de codificação de vídeo atuais e são realizadas algumas comparações entre estes padrões de codificação de vídeo.

3.1 Histórico

O padrão HEVC está em desenvolvimento desde Janeiro de 2010 para ser o sucessor do padrão H.264/AVC. O HEVC está sendo desenvolvido através da colaboração de especialistas da ITU-T e da ISO/IEC, reunidos no grupo JCT-VC (BROSS *et al.,* 2012). O objetivo principal do HEVC é dobrar a taxa de compressão alcançada pelo padrão H.264/AVC, mantendo a mesma qualidade subjetiva da imagem. Regularmente, ocorrem reuniões do JCT-VC em diferentes locais do mundo e, após estas reuniões, são disponibilizados os resultados obtidos pelo HEVC em termos de melhorias nas taxas de compressão, comparado ao padrão H.264/AVC (LI *et al.,* 2012).

A chamada à apresentação de propostas que iniciou o projeto do padrão HEVC foi emitida conjuntamente pelo ITU-T (VCEG) e ISO/IEC (MPEG) em Janeiro de 2010 (ITU-T/ISO/IEC, 2010), e os trabalhos de normatização subsequentes estão sendo conduzidos pelo JCT-VC. Em Abril do mesmo ano, na primeira reunião realizada, o JCT-VC definiu a primeira versão do software de referência, chamada *Test Model under Consideration* (TMuC), documentada posteriormente (JCT-VC, 2010). O objetivo desta documentação era começar a preparação de um modelo de

teste formal e a descrição do software de referência, incluindo métodos de codificação de referência para serem compartilhados pelos pesquisadores. Estes métodos de referência permitem avaliações justas do impacto das novas ferramentas de codificação propostas durante o processo de normatização do HEVC.

A partir da terceira reunião, em Outubro de 2010, foram estabelecidos os documentos HEVC *Test Model* (HM) (KIM *et al.,* 2012), modelos de referência que permanecem até o presente momento. O primeiro foi intitulado HEVC *Test Model* 1 (HM1) e a cada nova reunião, uma nova versão do HM é disponibilizada. Atualmente, no documento HM9 – *High Efficiency Video Coding Test Model 9 Encoder Description* (KIM *et al.,* 2012), estão definidos dois perfis para o padrão HEVC, o perfil *Main* , chamado de *Low Complexity* em versões anteriores, e o perfil *High Efficiency* 10 (HE10). O foco de cada um dos perfis é diferente, enquanto um perfil está focado em uma elevada eficiência na compressão em detrimento da complexidade computacional, o outro, o perfil *Main*, tem o objetivo de manter uma baixa complexidade, com um desempenho de compressão razoavelmente elevado. Na última versão do *draft* do HEVC (*Draft* 9), a principal diferença observada entre os perfis está no fato do perfil HE10 utilizar um *bit depth* de 10 bits, enquanto que no perfil *Main*, o *bit depth* é 8 bits (BROSS *et al.,* 2012). Em versões anteriores do *draft* do HEVC as diferenças dos perfis eram bem mais evidentes, mas aos poucos as ferramentas de codificação utilizadas, bem como outras características relacionadas a codificação foram se alinhando. Por exemplo, o perfil HE10 utilizava a ferramenta ALF (*Adaptive Loop Filter*) no processo de *Loop Filtering* e o algoritmo CABAC (*Context-Based Adaptive Binary Arithmetic Coding*) no codificador de entropia, enquanto que o perfil *Main* não utilizava ALF e a codificação de entropia utiliza CAVLC (*Context-Based Adaptive Variable Length Coding*). Atualmente, nenhum dos perfis usa ALF e ambos utilizam o algoritmo CABAC na codificação de entropia. As informações fornecidas na seção seguinte, sobre a estrutura geral de codificação do HEVC, são baseadas no perfil *Main* e estão de acordo com o *Draft* 9 do HEVC (BROSS *et al.,* 2012).

3.2 Estrutura Geral de Codificação

O padrão HEVC utiliza um esquema de codificação híbrido bem conhecido, baseado em blocos, que apresenta codificação com predições intra e compensação de movimento, transformadas e codificação de entropia. Em contraste com esquemas convencionais, o HEVC emprega um esquema de compressão de vídeo baseado no particionamento dos blocos codificados em uma hierarquia altamente flexível, que permite o uso de blocos grandes e múltiplos níveis para blocos de predição e transformadas, além de novas ferramentas de codificação. Estes novos aperfeiçoamentos presentes no HEVC melhoram significativamente sua eficiência na codificação.

Uma das principais inovações do padrão HEVC está no novo esquema de compressão de vídeo baseado em uma hierarquia flexível de representação unitária que inclui três conceitos de bloco: Unidade de Codificação (CU – *Coding Unit*), Unidade de Predição (PU – *Prediction Unit*) e Unidade de Transformada e Quantização (TU – *Transform Unit*) (KIM *et al.,* 2012). A separação da estrutura do bloco em três diferentes conceitos permite a cada um ser otimizado de acordo com sua função.

Em padrões de codificação anteriores como MPEG-1 e MPEG-2 é utilizada compensação de movimento com tamanho de bloco fixo, com bloco 16 × 16 amostras. O Padrão H.264/AVC, por sua vez, utiliza um tamanho de macrobloco de 16 × 16 que pode ser dividido em tamanhos menores, até 4 x 4 amostras. Mesmo assim, o tamanho de bloco máximo de 16 × 16 pode ser pequeno quando aplicado a vídeos de alta resolução. Para resolver este problema, o tamanho máximo permitido no HEVC passou a ser 64 × 64. No HEVC, a estrutura de compressão de vídeo foi projetada de forma que todos os blocos do codificador podem explorar o tamanho de bloco flexível.

3.2.1 Estrutura de Predição Temporal

O codificador HM trabalha com três tipos de estruturas de predição temporal, dependendo das condições experimentais, tal como definido no documento de

condições de teste do JCT-VC (BOSSEN, 2012). Estas estruturas de predição temporal são chamadas de *intra-only*, *low-delay* e *random-access.*

Na configuração de teste *intra-only*, cada quadro da sequência de vídeo é codificado como quadro IDR (*Instantaneous Decoding Refresh*), que são quadros Intra, ou seja, que utilizam predição intra-quadro apenas. Nenhum quadro de referência é utilizado e, portanto, não é empregada a predição temporal. O valor do parâmetro de quantização (QP – *Quantization Parameter*), que define a intensidade do processo de quantização, não é alterado durante a codificação. A Figura 3.1 apresenta uma representação gráfica da configuração *intra-only*. O número associado com cada quadro representa a ordem de codificação.

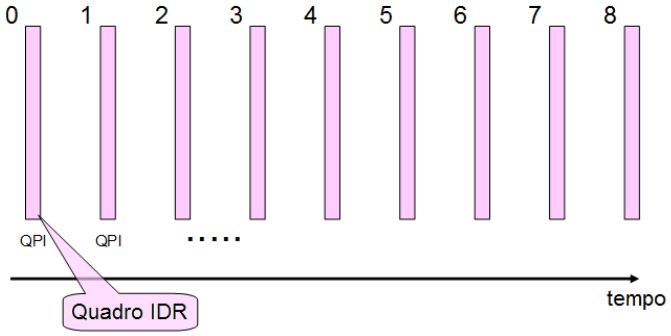


Figura 3.1 – Representação gráfica da configuração *intra-only*.

Dois tipos de configurações *low-delay* estão definidas para teste, tipo P ou tipo B. Em ambas condições, apenas o primeiro quadro de uma sequência de vídeo é codificado como quadro IDR. Na condição indispensável de teste *low-delay B*, todos os quadros sucessivos serão codificados como *Generalized P and B picture* (GPB). O quadro GPB é capaz de usar apenas quadros de referência temporalmente anteriores com relação ao quadro atual, sendo que quadros B podem utilizar quadros de referência de duas listas distintas, e quadros P utilizam apenas uma lista de quadros de referência. O QP de cada quadro codificado como Inter será obtido ao adicionar um *off-set* ao QP do quadro codificado como Intra. O *off-set* depende da camada temporal. Na condição *low-delay P* (opcional), todos os quadros Inter deverão ser codificados como quadros do tipo P. Na configuração *low-delay* a ordem de codificação se dá de acordo com a ordem de exibição. A Figura 3.2 mostra uma representação gráfica da configuração *low-delay*. O número associado com cada quadro representa a ordem de codificação.

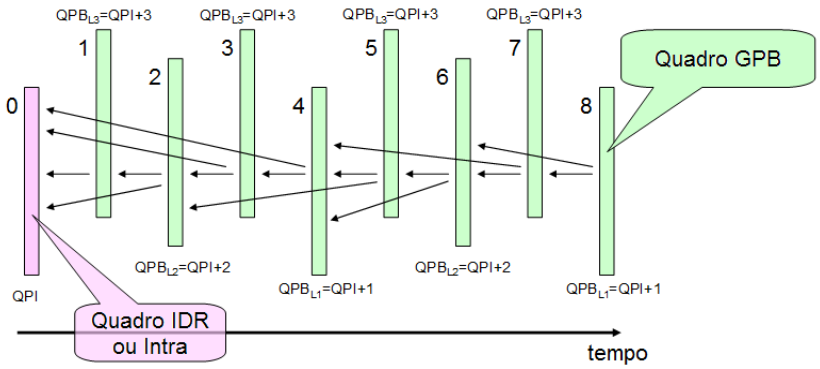


Figura 3.2 – Representação gráfica da configuração *low-delay*.

Para a condição de teste *random-access*, uma estrutura hierárquica de quadros B pode ser utilizada para codificação. Um quadro Intra será inserido ciclicamente após 1 segundo aproximadamente. Os quadros localizados entre os quadros Intra na ordem de exibição serão codificados como quadros B. O quadro GPB é utilizado na camada temporal de nível mais baixo. As segunda e terceira camadas temporais consistem de quadros B referenciados, e a mais alta camada temporal contém quadros B não-referenciados. O QP de cada quadro Inter codificado será obtido ao adicionar um *off-set* ao QP do quadro codificado como Intra. O *off-set* depende da camada temporal. Para a configuração *random-access* a ordem de codificação não é igual à ordem de exibição. A Figura 3.3 mostra a representação gráfica da configuração *random-access*. O número associado com cada quadro representa a ordem de codificação.

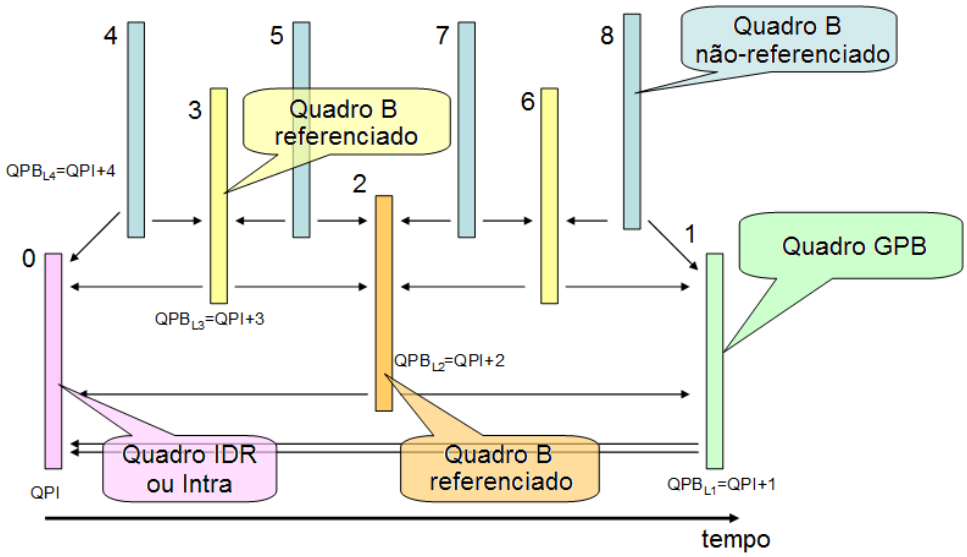


Figura 3.3 – Representação gráfica da configuração *random-access*.

3.2.2 Estrutura de Particionamento

A estrutura geral de codificação do Padrão HEVC permite o particionamento da imagem em CTUs (*Coding Tree Units*). Uma CTU é composta por um bloco de amostras de luminância juntamente com outros dois blocos correspondentes às amostras de crominância. Os tamanhos dos blocos de crominância, em relação ao bloco de luminância, dependem de qual é a amostragem de cores adotada. O conceito de CTU é análogo ao conceito de macrobloco utilizado em padrões anteriores, como o padrão H.264/AVC. Considerando a versão mais atual do HEVC até o término deste trabalho, o tamanho máximo permitido para uma CTU é 64x64 amostras de luminância, o que corresponde também ao maior tamanho de CU permitido (KIM *et al.,* 2012). Em versões anteriores dos *drafts* do HEVC, as CTUs já foram chamadas de LCUs (*Largest Coding Units*) e de *treeblocks*.

Uma CTU é composta por uma ou mais CUs. As CUs são utilizadas para predição inter-quadros/intra-quadro, são sempre quadradas, com tamanho 2Nx2N, onde N pode apresentar os valores 4, 8, 16 e 32. Portanto, as CUs assumem tamanhos a partir de 8x8 amostras de luminância até o tamanho da CTU. O conceito de CU permite a divisão recursiva em quatro blocos de tamanho igual, partindo da CTU. Este processo com divisões recursivas permite tamanhos pequenos ou grandes de unidades e forma uma estrutura de codificação em forma de árvore quadrática composta por blocos de CU. As figuras 3.4 e 3.5 mostram uma CTU dividida em CUs e a árvore quadrática que corresponde a esta divisão, respectivamente.

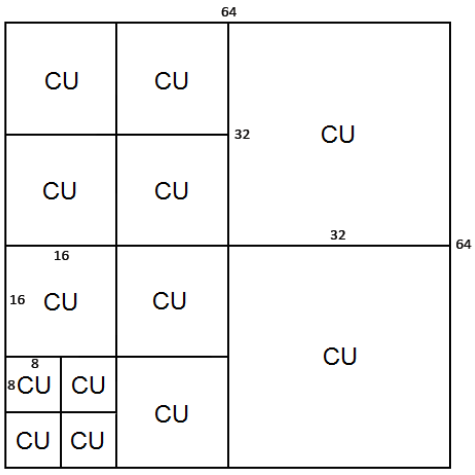


Figura 3.4 – Exemplo de uma CTU dividida em CUs.

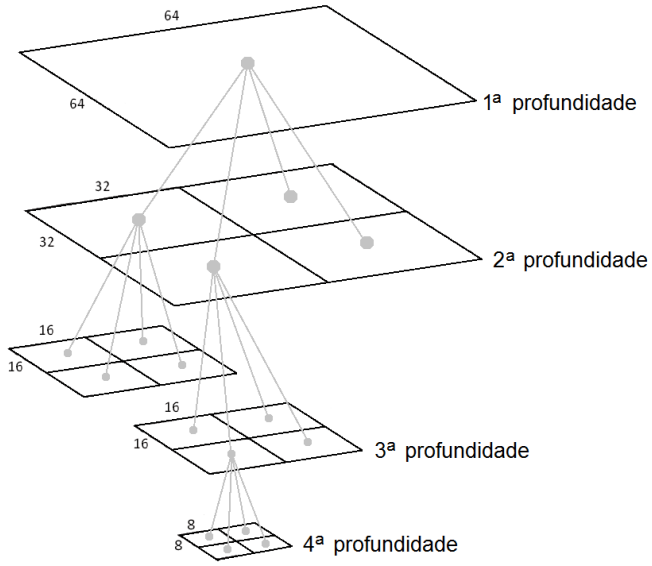


Figura 3.5 – Exemplo de árvore quadrática de uma CTU.

Na Figura 3.5 pode-se observar uma árvore quadrática de uma CTU. Primeiramente a CTU de dimensões 64x64 (1ª profundidade) é dividida em quatro blocos 32x32 (2ª profundidade). Destes blocos 32x32, dois são divididos em blocos 16x16 (3ª profundidade). Os blocos 32x32 não divididos são codificados como CUs 32x32. Na 3ª profundidade acontece o mesmo processo, os blocos 16x16 que não são divididos são codificados como CUs 16x16. Um dos blocos 16x16 é dividido em blocos 8x8 (4ª profundidade), chegando à menor dimensão possível para CU, sendo assim codificado.

Cada CU pode conter uma ou mais PUs. A PU é a unidade básica utilizada para os processos de predição, o que contempla a etapa de ME. A PU não se restringe às formas quadradas, ou seja, são permitidas também formas retangulares. Isto é feito com o objetivo de facilitar o particionamento que corresponde a limites de objetos reais na imagem. A Figura 3.6 mostra os diferentes tipos de divisão de uma CU em PUs (KIM *et al.,* 2012, p. 9). O tipo de divisão NxN só é permitido quando N = 4. Mesmo assim, no caso da predição inter-quadros o tipo de divisão NxN está desabilitado por padrão. Desta forma, pode-se concluir que o menor tamanho possível para PU é 4x4, no caso da predição intra-quadro e o maior tamanho é 64x64, o qual pode ocorrer apenas quando a CU é 64x64. Na predição intra-quadro, apenas os particionamentos 2Nx2N e NxN são utilizados. Os quatro particionamentos assimétricos (2NxnU, 2NxnD, nLx2N e nRx2N), que podem ser utilizados na predição inter-quadros, só estão disponíveis quando N > 4. Em versões

anteriores do *draft* do HEVC, os particionamentos assimétricos só estavam disponibilizados para o perfil *High Efficiency* do HEVC.

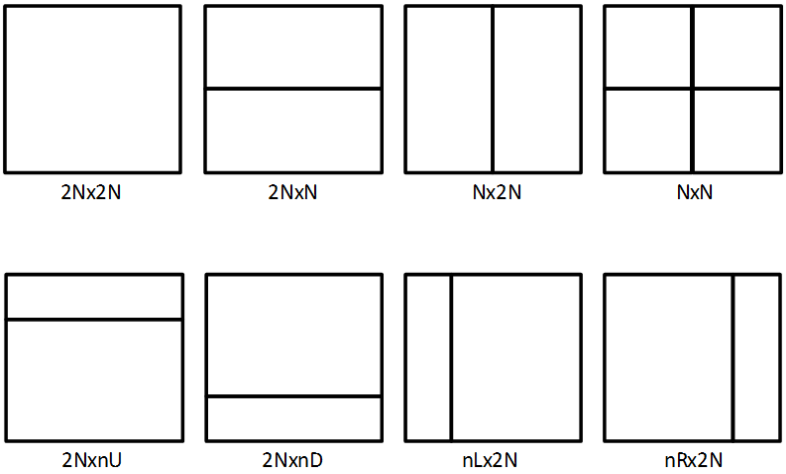


Figura 3.6 – Oito tipos possíveis de divisão de uma CU em PUs.

A TU é a unidade básica para os processos de transformada e quantização e é sempre formada por quadrados. A TU pode apresentar tamanhos de 4x4 até 32x32 amostras. Cada CU pode conter uma ou mais TUs, onde várias TUs podem estar dispostas em uma estrutura de *quadtree*, como apresentado na Figura 3.7 (KIM *et al.,* 2012). Em versões anteriores do *draft* do HEVC, as TUs também podiam ter particionamentos não-quadráticos no perfil *High Efficiency* do HEVC.

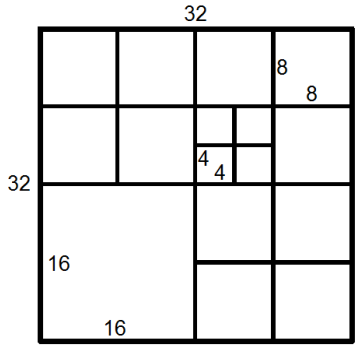


Figura 3.7 – Exemplo de CU 32x32 dividida em TUs.

3.3 FME no Padrão HEVC

No padrão HEVC é proposta a utilização de um filtro de interpolação separável de 8 *taps*, baseado em DCT (*Discrete Cosine Transform)*, que interpola diretamente as amostras de luminância das posições inteiras de pixel, antes da

busca fracionária por um melhor casamento em ¼ de pixel (KIM *et al.,* 2012). A Figura 3.8 representa as amostras em posições inteiras (quadrados sombreados e com letras maiúsculas), bem como as amostras em posições fracionárias (quadrados não sombreados e com letras minúsculas) para interpolação das amostras de luminância com precisão de ¼ de pixel do padrão HEVC, considerando um bloco de tamanho 8x8.

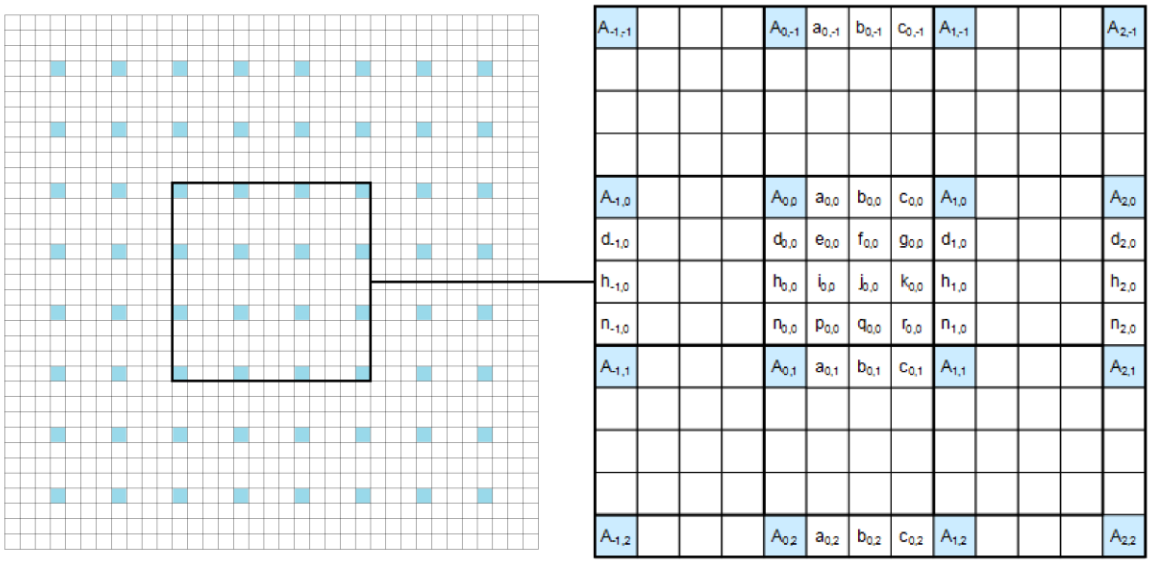


Figura 3.8 – Amostras em posições inteiras e amostras em posições fracionárias.

A partir das amostras de luminância em posições inteiras e de filtros FIR (*Finite Impulse Response)* de 8 *taps*, são obtidos os valores das posições fracionárias $a_{0,0}$, $b_{0,0}$, $c_{0,0}$, $d_{0,0}$, $h_{0,0}$ e $n_{0,0}$, como pode ser observado nas equações (4) a (9). Estas equações foram obtidas através do documento HEVC *draft 9* (BROSS *et al.,* 2012, p. 135). O valor da variável *shift1*, utilizada no deslocamento a direita das equações, pode ser encontrado através de uma análise no software de referência e em documentos do HEVC, como será mais bem explicado nos capítulos seguintes.

$$a_{0,0} = (-A_{-3,0} + 4*A_{-2,0} - 10*A_{-1,0} + 58*A_{0,0} + 17*A_{1,0} - 5*A_{2,0} + A_{3,0}) >> shift1 \quad (4)$$

$$b_{0,0} = (-A_{-3,0} + 4*A_{-2,0} - 11*A_{-1,0} + 40*A_{0,0} + 40*A_{1,0} - 11*A_{2,0} + 4*A_{3,0} - A_{4,0}) >> shift1 \quad (5)$$

$$c_{0,0} = (A_{-2,0} - 5*A_{-1,0} + 17*A_{0,0} + 58*A_{1,0} - 10*A_{2,0} + 4*A_{3,0} - A_{4,0}) >> shift1 \quad (6)$$

$$d_{0,0} = (-A_{0,-3} + 4*A_{0,-2} - 10*A_{0,-1} + 58*A_{0,0} + 17*A_{0,1} - 5*A_{0,2} + A_{0,3}) >> shift1 \quad (7)$$

$$h_{0,0} = (-A_{0,-3} + 4*A_{0,-2} - 11*A_{0,-1} + 40*A_{0,0} + 40*A_{0,1} - 11*A_{0,2} + 4*A_{0,3} - A_{0,4}) >> shift1 \quad (8)$$

$$n_{0,0} = (A_{0,-2} - 5*A_{0,-1} + 17*A_{0,0} + 58*A_{0,1} - 10*A_{0,2} + 4*A_{0,3} - A_{0,4}) >> shift1 \quad (9)$$

O cálculo dos valores das posições fracionárias $e_{0,0}$, $f_{0,0}$, $g_{0,0}$, $i_{0,0}$, $j_{0,0}$, $k_{0,0}$, $p_{0,0}$, $q_{0,0}$ e $r_{0,0}$ requer primeiramente o cálculo dos valores das posições fracionárias $a_{0,i}$, $b_{0,i}$ e $c_{0,i}$, onde i varia de -3 a 4 na direção vertical. Em seguida, através de filtros FIR de 8 *taps*, são obtidos os valores das demais posições fracionárias, como pode ser observado nas equações (10) a (18). Assim como no caso da variável *shift1*, o valor da variável *shift2* pode ser encontrado através de uma análise no software de referência e em documentos do HEVC, como será explicado nos próximos capítulos.

$$e_{0,0} = (-a_{0,-3} + 4*a_{0,-2} - 10*a_{0,-1} + 58*a_{0,0} + 17*a_{0,1} - 5*a_{0,2} + a_{0,3}) >> shift2 \quad (10)$$

$$f_{0,0} = (-b_{0,-3} + 4*b_{0,-2} - 10*b_{0,-1} + 58*b_{0,0} + 17*b_{0,1} - 5*b_{0,2} + b_{0,3}) >> shift2 \quad (11)$$

$$g_{0,0} = (-c_{0,-3} + 4*c_{0,-2} - 10*c_{0,-1} + 58*c_{0,0} + 17*c_{0,1} - 5*c_{0,2} + c_{0,3}) >> shift2 \quad (12)$$

$$i_{0,0} = (-a_{0,-3} + 4*a_{0,-2} - 11*a_{0,-1} + 40*a_{0,0} + 40*a_{0,1} - 11*a_{0,2} + 4*a_{0,3} - a_{0,4}) >> shift2 \quad (13)$$

$$j_{0,0} = (-b_{0,-3} + 4*b_{0,-2} - 11*b_{0,-1} + 40*b_{0,0} + 40*b_{0,1} - 11*b_{0,2} + 4*b_{0,3} - b_{0,4}) >> shift2 \quad (14)$$

$$k_{0,0} = (-c_{0,-3} + 4*c_{0,-2} - 11*c_{0,-1} + 40*c_{0,0} + 40*c_{0,1} - 11*c_{0,2} + 4*c_{0,3} - c_{0,4}) >> shift2 \quad (15)$$

$$p_{0,0} = (a_{0,-2} - 5*a_{0,-1} + 17*a_{0,0} + 58*a_{0,1} - 10*a_{0,2} + 4*a_{0,3} - a_{0,4}) >> shift2 \quad (16)$$

$$q_{0,0} = (b_{0,-2} - 5*b_{0,-1} + 17*b_{0,0} + 58*b_{0,1} - 10*b_{0,2} + 4*b_{0,3} - b_{0,4}) >> shift2 \quad (17)$$

$$r_{0,0} = (c_{0,-2} - 5*c_{0,-1} + 17*c_{0,0} + 58*c_{0,1} - 10*c_{0,2} + 4*c_{0,3} - c_{0,4}) >> shift2 \quad (18)$$

3.3.1 Comparação da FME no HEVC com o Padrão H.264/AVC

A interpolação de ¼ de pixel das amostras de luminância no padrão HEVC é realizada de forma diferente da empregada no padrão H.264/AVC. No padrão H.264/AVC, a interpolação de ¼ de pixel das amostras de luminância é realizada em duas etapas, com uma dependência em relação às amostras de ½ pixel. Na primeira destas etapas, as amostras em posições com precisão de ½ pixel são geradas a partir das amostras de luminância em posições inteiras através de um filtro FIR de 6 *taps*. Ainda nesta etapa, após a interpolação, é realizada uma busca em posições fracionárias de ½ pixel que permita um casamento melhor do que o obtido através da busca em posições inteiras. Só então, caso seja encontrado um melhor casamento em precisão de ½ pixel, é realizada uma segunda etapa. A segunda etapa consiste em gerar as amostras de luminância em posições de ¼ de pixel através de uma média simples entre uma posição de ½ pixel obtida na etapa anterior e uma posição inteira. Também é feita nesta etapa a busca fracionária por um melhor casamento em ¼ de pixel (AGOSTINI, 2007).

3.3.2 Estado da Arte das Arquiteturas para FME

Uma vez que o padrão HEVC ainda se encontra em desenvolvimento, se torna fundamental um acompanhamento dos documentos sobre o padrão disponibilizados a cada reunião do JCT-VC, principalmente das versões do HEVC *Draft* e HEVC *Test Model*, os quais permitem identificar qualquer alteração que aconteça nas ferramentas de codificação e no software de referência do padrão. Estes documentos são essenciais para o desenvolvimento de qualquer trabalho envolvendo o padrão HEVC e podem ser encontrados em (JCT-VC Document Management System, 2012**)**.

Para permitir um estudo sobre o estado da arte das arquiteturas de FME segundo o padrão HEVC, bem como realizar futuras comparações de resultados, pesquisou-se na literatura científica alguns trabalhos relacionados. Contudo, até a conclusão deste trabalho foi encontrado apenas um artigo científico que envolve o desenvolvimento de arquiteturas para a FME com precisão de sub-pixel do HEVC.

A arquitetura encontrada em (GUO *et al.,* 2012) foi desenvolvida para uma versão anterior do HM, com diferenças no algoritmo utilizado para o cálculo das posições fracionárias em relação à versão utilizada neste trabalho. Os resultados apresentados no artigo, por sua vez, são referentes apenas à etapa de interpolação das amostras de luminância, e consideram sub-blocos de tamanho 4 x 4, não abordando a busca e a comparação com blocos fracionários. Basicamente, a arquitetura proposta foi desenvolvida com foco em circuitos VLSI (*Very-Large-Scale Integration*) e utiliza filtros reconfiguráveis, *pipeline* e um esquema de reuso de filtros com o objetivo de elevar as taxas de processamento e diminuir a utilização dos recursos de hardware. Como resultados, este trabalho atingiu uma taxa de processamento de 60 quadros QFHD (3840x2160 pixels) por segundo e reduziu a utilização dos recursos de hardware em 30%, quando comparado com uma versão sem otimizações. Contudo, em função das diferenças com relação a este trabalho, expostas anteriormente, não foram possíveis comparações diretas com os resultados apresentados no artigo.

Com o objetivo de identificar estratégias interessantes que poderiam ser utilizadas no projeto da arquitetura, também foram considerados para estudo aqueles trabalhos que apresentam arquiteturas para FME segundo o padrão H.264/AVC, do qual é possível encontrar inúmeros trabalhos relacionados na

literatura científica como (CORRÊA, 2011a) (CORRÊA, 2011b) (OKTEM, 2007) (KAO, 2006) (YALCIN, 2006). Alguns trabalhos para o H.264/AVC abordam o desenvolvimento de arquiteturas para precisão de ¼ de pixel e outros para ½ pixel. Os resultados obtidos, também são bem variados dependendo do foco dos projetos e quais requisitos são relevantes, como taxa de processamento, consumo de energia, qualidade resultante do vídeo, entre outros.

Corrêa (2011a) (2011b) apresenta em dois de seus trabalhos, arquiteturas para FME com abordagem focada em blocos de tamanho fixo 8x8. Em um dos trabalhos, a arquitetura proposta trabalha com precisão de ½ pixel, enquanto que no outro, a arquitetura foi desenvolvida para precisão de ¼ de pixel. Para adotar a decisão de utilizar tamanho de bloco fixo 8x8, Corrêa (2011a) (2011b) realizou avaliações com base no *bit-rate* e no PSNR de cinco vídeos codificados utilizando o software de referência do padrão H.264/AVC. As duas abordagens de Corrêa (2011a) (2011b), as quais consideram tamanho de bloco fixo 8x8, resultam em aumento do *bit-rate* e redução do PSNR. Estas alterações têm efeitos importantes em dois aspectos relacionados a vídeo digital: a diminuição da qualidade da imagem e a diminuição do custo de implementação, uma vez que a complexidade computacional é menor e menos hardware é necessário. Contudo, conforme as avaliações de qualidade de imagem, a queda de qualidade não foi muito expressiva, considerando o tamanho de bloco utilizado.

Kao (2006) propõe uma arquitetura que emprega um modelo matemático para estimar SADs em posições de ¼ de pixel ao invés de realizar em sequência etapas de interpolação e busca, conforme o método convencional. Desta forma, o tempo de computação e os requisitos de acesso à memória são reduzidos, sem prejuízo significativo na qualidade da imagem.

Oktem (2007) apresenta um hardware capaz de realizar a interpolação de ¼ de pixel da FME do padrão H.264/AVC de forma dinâmica, ou seja, apenas são calculadas as posições de ¼ de pixel necessárias para a realização da busca no local apontado pelo vetor de movimento de ½ pixel. Desta forma pode ser reduzida a quantidade de cálculos realizados para a interpolação de ¼ de pixel da FME e, portanto, há uma redução no consumo de energia do hardware.

Como pode ser observado, com escolhas adequadas é possível conseguir um bom equilíbrio entre qualidade de imagem, custo de hardware e desempenho. Na fase de desenvolvimento da arquitetura para FME deste trabalho foram realizadas

avaliações com base no *bit-rate* e no PSNR de vídeos codificados, utilizando o software de referência do padrão HEVC e considerando alguns tamanhos fixos de bloco. Com base nos resultados, foram analisadas as variações de qualidade de imagem com diferentes tamanhos de bloco. Estas ações permitiram encontrar um bom equilíbrio entre qualidade de imagem, custo de hardware e taxa de processamento na especificação do projeto de hardware, como será apresentado nos capítulos seguintes.

4 AVALIAÇÕES USANDO O SOFTWARE DE REFERÊNCIA

O software de referência do padrão HEVC é muito importante, pois fornece as condições para a realização de experimentos que possibilitem avaliar o quanto o desempenho de cada ferramenta de codificação é satisfatório. Todas as versões do software de referência do HEVC podem ser encontradas em (HEVC Reference Software, 2012), sendo possível o download de qualquer versão através do *Apache Subversion* (Subversion, 2012), um sistema de controle de versão. Para fazer o download, utilizando o sistema operacional Linux, basta utilizar o comando *svn checkout* no terminal do Linux. Após executar o comando, uma pasta HM, contendo diversas outras subpastas, é disponibilizada. De posse dos arquivos, a execução do software de referência ainda pode ser feita de diversas formas. Pode-se, por exemplo, executar o bloco codificador ou o bloco decodificador. Se for escolhido o bloco codificador, diversas configurações são possíveis, e cada uma delas conduz a experimentos em um ambiente de teste bem definido. Para executar o codificador deve-se primeiramente compilar o código, o que pode ser simplificado através de um arquivo *Makefile* fornecido na pasta *HM/build/linux/app/TAppEncoder/* e um comando *make* no terminal do Linux. Após ser gerado o arquivo executável, que ficará na pasta *HM/bin/TAppEncoderStatic*, é necessário executá-lo fornecendo dois conjuntos de informações: as configurações do vídeo a ser codificado e as configurações relativas ao funcionamento do codificador. A seguinte linha de comando pode ser utilizada no terminal do Linux, a partir da pasta HM, para a execução do codificador considerando a sequência *SlideShow*, a configuração *low-delay* e o perfil *Main*: *./bin/TAppEncoderStatic –c cfg/per-sequence/SlideShow.cfg –c cfg/encoder_lowdelay_main.cfg*.

Os arquivos de configuração das sequências de vídeo de teste definidas pelo JCT-VC (BOSSEN, 2012) podem ser encontrados na pasta *HM/cfg/per-*

sequence/. Já os arquivos de configuração do codificador ficam na pasta *HM/cfg/*. Os arquivos de configuração das sequências fornecem, entre outras informações, o endereço do arquivo de vídeo a ser codificado, a taxa de quadros por segundo, a resolução do vídeo, o *bit depth* de entrada e o número de quadros a serem codificados. Os arquivos de configuração do codificador fornecem inúmeras informações, relativas a todas as ferramentas de codificação utilizadas, como predição inter/intra, transformadas, quantização e codificação de entropia. Entre as informações do arquivo, estão o tamanho máximo das unidades de codificação, a área de busca para a estimação de movimento e o número de quadros de referência.

Alguns experimentos foram realizados com o auxílio do software de referência, com o objetivo de realizar avaliações importantes relacionadas com este trabalho. Através da variação de configurações e alterações no código do software, foi possível extrair informações como: o impacto da ME e da FME nos resultados de compressão; os tamanhos de PU mais selecionados, ou seja, que apresentam os melhores resultados na codificação; resultados de qualidade de imagem e *bit-rate* fixando o tamanho dos blocos.

Conhecendo-se os tamanhos de blocos mais importantes para a predição inter-quadros, foram adotadas estratégias para o desenvolvimento da arquitetura de hardware que permitiram um equilíbrio entre desempenho, qualidade resultante de imagem e custo de hardware. Outro importante papel do software de referência é a contribuição para validação da arquitetura proposta. Com alterações no código de referência foi possível obter, por exemplo, os valores das amostras de luminância antes e após a interpolação realizada na FME, possibilitando a validação do hardware proposto com o auxílio da ferramenta ModelSim.

4.1 Condições e Configurações de Teste Utilizadas nas Avaliações

O documento mais recente, até o término deste trabalho, que define condições e configurações de teste que devem ser utilizadas em experiências com o software de referência do padrão HEVC pelo JCT-VC é o documento JCTVC-K1100 da 11ª reunião (BOSSEN, 2012). Estas condições e configurações de teste foram recomendadas e utilizadas para contribuições técnicas durante todo o processo de normatização do HEVC. O JCT-VC define, nesse documento, oito condições de

teste, refletindo uma combinação de alta eficiência (perfil HE10) e baixa complexidade (perfil *Main*) com configurações denominadas *intra-only*, *random-access* e *low-delay* (BOSSEN, 2012). Deste conjunto de oito condições, nem todas precisam ser utilizadas, apenas aquelas relacionadas ao experimento a ser realizado. O documento também fornece uma lista de 24 sequências de teste, definidas pelo JCT-VC, separadas em classes de acordo com suas resoluções e características, de forma que 4 vídeos na resolução WQXGA (2560x1600 pixels), 5 vídeos na resolução *Full* HD (1920x1080 pixels), 5 vídeos na resolução HD (1280x720 pixels), 1 vídeo na resolução XGA (1024x768 pixels), 5 vídeos na resolução WVGA (832x480 pixels) e 4 vídeos na resolução WQVGA (416x240 pixels) são utilizados. A Tabela 4.1 apresenta todas as sequências de vídeo utilizadas nos testes, bem como algumas de suas características e as configurações nas quais devem ser utilizadas. O documento ainda define que todos os quadros de todas as sequências devem ser codificados nos experimentos utilizando estas sequências.

Tabela 4.1 – Lista das sequências fornecidas para teste pelo JCT-VC.

Classificação das Sequências	**Nome da Sequência**	**Quadros**	***Frame rate (qps)***	***Bit depth***	**Configuração Temporal**		
					Intra-only	***Random-access***	***Low-delay***
Classe A (WQXGA)	Traffic	150	30	8	Main/HE10	Main/HE10	–
	PeopleOnStreet	150	30	8	Main/HE10	Main/HE10	–
	Nebuta	300	60	10	Main/HE10	Main/HE10	–
	SteamLocomotive	300	60	10	Main/HE10	Main/HE10	–
Classe B (*Full* HD)	Kimono	240	24	8	Main/HE10	Main/HE10	Main/HE10
	ParkScene	240	24	8	Main/HE10	Main/HE10	Main/HE10
	Cactus	500	50	8	Main/HE10	Main/HE10	Main/HE10
	BQTerrace	600	60	8	Main/HE10	Main/HE10	Main/HE10
	BasketballDrive	500	50	8	Main/HE10	Main/HE10	Main/HE10
Classe C (WVGA)	RaceHorses	300	30	8	Main/HE10	Main/HE10	Main/HE10
	BQMall	600	60	8	Main/HE10	Main/HE10	Main/HE10
	PartyScene	500	50	8	Main/HE10	Main/HE10	Main/HE10
	BasketballDrill	500	50	8	Main/HE10	Main/HE10	Main/HE10
Classe D (WQVGA)	RaceHorses	300	30	8	Main/HE10	Main/HE10	Main/HE10
	BQSquare	600	60	8	Main/HE10	Main/HE10	Main/HE10
	BlowingBubbles	500	50	8	Main/HE10	Main/HE10	Main/HE10
	BasketballPass	500	50	8	Main/HE10	Main/HE10	Main/HE10
Classe E (HD)	FourPeople	600	60	8	Main/HE10	–	Main/HE10
	Johnny	600	60	8	Main/HE10	–	Main/HE10
	KristenAndSara	600	60	8	Main/HE10	–	Main/HE10
Classe F (Diversas Resoluções)	BaskeballDrillText	500	50	8	Main/HE10	Main/HE10	Main/HE10
	ChinaSpeed	500	30	8	Main/HE10	Main/HE10	Main/HE10
	SlideEditing	300	30	8	Main/HE10	Main/HE10	Main/HE10
	SlideShow	500	20	8	Main/HE10	Main/HE10	Main/HE10

As quatro sequências da classe A se caracterizam por estarem na resolução WQXGA (2560x1600 pixels), sendo que duas das sequências têm *bit depth* de 10

bits. Os cinco vídeos da classe B estão na resolução *Full HD* (1920x1080 pixels). A característica comum das quatro sequências da classe C é que se encontram na resolução WVGA (832x480 pixels). Já os quatro vídeos da classe D se caracterizam por estarem na resolução WQVGA (416x240 pixels). Na classe E são utilizados três vídeos na resolução HD (1280x720 pixels). A única classe que reúne vídeos de diferentes resoluções é a classe F, apresentando quatro vídeos em três resoluções distintas, um em WVGA (832x480 pixels), dois em HD (1280x720 pixels) e um em XGA (1024x768 pixels). Apesar de estarem em resoluções diferentes, os vídeos da classe F apresentam características muito interessantes envolvendo textos, que os vídeos das demais classes não apresentam, como: uma apresentação de *slides* com diversas animações, mudanças de fundo e transição de janelas, aparecimento e desaparecimento gradual de textos e figuras e textos com diferentes fontes e cores; uma edição de *slides* com alternância de janelas e rolagem, seleção e digitação de textos; uma cena de um jogo de corrida de carros com muitas mudanças de iluminação e textos com caracteres chineses; uma cena de um treino de basquete em que um texto animado se movimenta da direita para a esquerda sobreposto à imagem dos jogadores. Outra classe que apresenta vídeos bem característicos é a classe E. Os três vídeos desta classe apresentam grande repetição entre os quadros da sequência, uma vez que apresentam fundos sem qualquer tipo de movimentação de câmera e que ocupam a maior parte da imagem. Em todos os vídeos da classe E a movimentação se limita, basicamente, a conversação entre os personagens das sequências.

Outra característica interessante apresentada nas sequências descritas na Tabela 4.1 é a variação na taxa de quadros por segundo, que pode ir de 20 a 60 quadros por segundo nas diferentes sequências.

A Figura 4.1 apresenta o primeiro quadro de todas as sequências de teste definidas pelo JCT-VC, sendo que a sequência *RaceHorses* é fornecida em duas resoluções diferentes.


Traffic


PeopleOnStreet


Nebuta


SteamLocomotive


Kimono


ParkScene


Cactus


BQTerrace


BasketballDrive


RaceHorses


BQMall

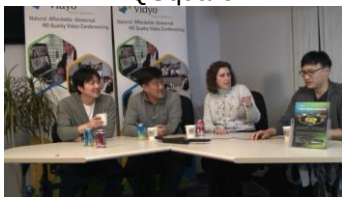

PartyScene


BasketballDrill


BQSquare


BlowingBubbles


BasketballPass

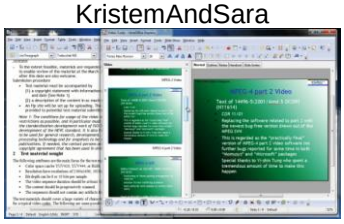

FourPeople


Johnny


KristemAndSara


BasketballDrillText


ChinaSpeed


SlideEditing

SlideShow

Figura 4.1 – Sequências de vídeo para teste definidas pelo JCT-VC.

Em todas as avaliações realizadas neste trabalho foi utilizado o software de referência HM, na versão *HM-8.0rc2* (HEVC Reference Software, 2012). As condições e configurações de teste utilizadas nos experimentos estão de acordo com o documento JCTVC-K1100 (BOSSEN, 2012) e todas as avaliações realizadas, descritas nas próximas seções, consideraram o perfil *Main*. Outra informação importante é que todas simulações utilizaram o valor *padrão* de QP, que é 32. O documento de teste define quatro valores para serem utilizados: 22, 27, 32 e 37. Esses valores se referem ao valor de QP do primeiro quadro Intra da codificação e que, como já explicado, é usado como referência para os demais quadros da sequência. O valor de QP consiste de uma entrada externa da etapa de quantização que determina o quanto as informações (resíduos) serão cortadas. Valores maiores de QP implicam em resíduos de menor valor. Dessa forma, o valor de QP tem papel importante nos resultados de qualidade e no *bit-rate* de saída. O uso de um único QP nas avaliações desenvolvidas neste trabalho foi considerado suficiente para avaliar as características desejadas relativas ao uso de distintos tamanhos de PUs. Essa conclusão foi obtida através de um experimento realizado com os vídeos *Kimono* e *PartyScene*, usando os quatro QPs definidos. Os resultados deste experimento mostraram-se coerentes com os resultados dos experimentos realizados utilizando QP 32. Dessa forma, visando à simplificação dos experimentos, apenas um valor de QP foi utilizado.

4.2 Avaliação do Impacto da Predição Inter-Quadros e da FME na Compressão do HEVC

A primeira avaliação realizada com o software de referência teve o objetivo de verificar o impacto da utilização da predição inter-quadros e da técnica de FME nos resultados de compressão obtidos com o padrão emergente HEVC.

Primeiramente, para identificar o impacto da predição inter-quadros, foram realizadas simulações utilizando as configurações *intra-only* e *low-delay*, uma vez que a configuração *intra-only* não utiliza a predição inter-quadros e a configuração *low-delay* a utiliza. Desta forma, comparando os resultados de *bit-rate* e qualidade de imagem (através de valores PSNR) obtidos com essas configurações, é possível verificar o impacto da predição inter-quadros na codificação. Em seguida, através de

alterações no software de referência, a FME foi desabilitada e foram realizadas novas simulações na configuração *low-delay*, permitindo verificar os ganhos obtidos com a FME na predição inter-quadros.

O impacto da utilização da predição inter-quadros nos resultados de compressão obtidos com o padrão HEVC podem ser observados na Tabela 4.2. Esta tabela apresenta os valores médios de redução no *bit-rate* e na qualidade da imagem de acordo com as classes das sequências (BOSSEN, 2012). O valor Y-PSNR se refere às amostras de luminância, enquanto que os valores U-PSNR e V-PSNR se referem às amostras de crominância. É possível observar que a redução no *bit-rate* é de no mínimo 84%. A queda na qualidade de imagem é, no máximo, de 2,42 dB.

Tabela 4.2 – Impacto da predição inter-quadros na codificação de vídeos.

Classe de Sequências	**Redução do *bitrate* e variação na qualidade de imagem (PSNR)**			
	***Bit-rate* (%)**	**Y-PSNR (dB)**	**U-PSNR (dB)**	**V-PSNR (dB)**
B	87,0978	-1,4547	-0,5224	-0,5178
C	85,5108	-2,0941	-0,7893	-0,7876
D	84,0668	-2,4182	-0,7953	-0,9800
E	96,2607	-0,8242	-0,0743	-0,0974
F	90,6511	-1,4447	-0,7843	-0,7207
Média	88,7175	-1,6472	-0,5931	-0,6207

O impacto da FME nos resultados da predição inter-quadros pode ser observado na Tabela 4.3. A FME é responsável por uma redução de, no mínimo, 5% no *bit-rate*. A qualidade de imagem apresenta ganho em todos os casos.

Tabela 4.3 – Impacto da FME nos resultados da predição inter-quadros.

Classe de Sequências	**Redução do *bitrate* e variação na qualidade de imagem (PSNR)**			
	***Bit-rate* (%)**	**Y-PSNR (dB)**	**U-PSNR (dB)**	**V-PSNR (dB)**
B	6,9532	0,1205	0,0313	0,0337
C	8,4648	0,2267	0,0308	0,0523
D	11,4743	0,3103	0,1722	0,1840
E	6,0546	0,1174	0,0844	0,1402
F	5,0387	0,0784	0,0821	0,0038
Média	7,5971	0,1707	0,0802	0,0828

As avaliações realizadas permitiram verificar os ganhos na compressão possibilitados pela predição inter-quadros e pela FME no padrão HEVC. Apesar de uma perda de qualidade considerável devido à utilização da predição inter-quadros, a redução no *bit-rate* é tão elevada que justifica esta perda na qualidade (já que a decisão de qual modo de codificação será usado é função da relação entre *bit-rate* e PSNR, seguindo o RDO – *Rate Distortion Optimization* (KIM *et al.,* 2012)). No caso

da FME, a utilização da ferramenta é extremamente importante, pois permite ganhos nas duas frentes, ou seja, aumenta as taxas de compressão, melhorando a qualidade da imagem. Estes resultados foram o ponto de partida para verificar a pertinência deste trabalho, e do desenvolvimento de uma arquitetura de hardware para FME do padrão HEVC.

4.3 Avaliação dos Tamanhos de PUs Mais Selecionados

O software de referência HM define duas funções diferentes para a manipulação dos blocos de vídeo na codificação. Em um primeiro momento, é chamada uma função de compressão, onde os blocos CTU de tamanho 64x64 são particionados e testados. Os testes de cada CTU são realizados com o objetivo de obter o melhor resultado de particionamento de acordo com os métodos de predição empregados. Para tal, é analisado se o melhor método de codificação do bloco é a predição intra-quadro, inter-quadros ou *Skip* (KIM *et al.,* 2012), além de determinar o tipo de particionamento e a profundidade a serem utilizadas. Os blocos *Skip* são utilizados quando o bloco que está sendo codificado é muito similar ao bloco de mesma posição no quadro de referência, e se caracteriza pelo codificador não enviar nenhuma informação para a representação do bloco, nem o resíduo e nem o vetor de movimento (RICHARDSON, 2003). Quanto mais movimentação tiver o vídeo, menor será a ocorrência de blocos *Skip.*

No caso da predição inter-quadros sete tipos diferentes de particionamento são testados em cada profundidade permitida pela estrutura de codificação do HEVC. Após as comparações realizadas no primeiro nível de profundidade será armazenada a informação de qual método de predição e tipo de particionamento resultou no melhor resultado. Essas comparações são realizadas em quatro níveis, ou profundidades. Dessa forma, as três últimas comparações acontecerão a partir de CUs de tamanhos 32x32, 16x16 e 8x8. Na última profundidade, quando a CU é 8x8, apenas três tipos de particionamentos para predição inter-quadros são testados e, na predição intra-quadro, passam a ser dois os tamanhos avaliados. No final desses testes, a codificação de um bloco poderá ser escolhida como *Skip*, como inter-quadros ou como intra-quadro. Para *Skip*, são possíveis quatro tamanhos diferentes, para inter-quadros, 24 tamanhos diferentes e, para intra-quadro, são cinco tamanhos possíveis. Essa etapa serve para identificar a melhor opção de codificação para

cada bloco. Posteriormente, em outra etapa, chamada de codificação, é que os blocos serão codificados diretamente de acordo com os resultados pré-armazenados. A Tabela 4.4 apresenta todos os tamanhos de PU, com o tipo de particionamento e profundidade possíveis para todos os métodos de predição.

Tabela 4.4 – Tamanhos de PU permitidos de acordo com o método de predição.

Profundidade	Tipo de Particionamento	Tamanho da PU	Predição		
			Inter-quadros	Intra-quadro	*Skip*
0 (N=32)	2Nx2N	64x64	X	X	X
	2NxN	64x32	X		
	Nx2N	32x64	X		
	2NxnU (2NxN/2)	64x16	X		
	2NxnD (2Nx3N/2)	64x48	X		
	nLx2N (N/2x2N)	16x64	X		
	nRx2N (3N/2x2N)	48x64	X		
1 (N=16)	2Nx2N	32x32	X	X	X
	2NxN	32x16	X		
	Nx2N	16x32	X		
	2NxnU (2NxN/2)	32x8	X		
	2NxnD (2Nx3N/2)	32x24	X		
	nLx2N (N/2x2N)	8x32	X		
	nRx2N (3N/2x2N)	24x32	X		
2 (N=8)	2Nx2N	16x16	X	X	X
	2NxN	16x8	X		
	Nx2N	8x16	X		
	2NxnU (2NxN/2)	16x4	X		
	2NxnD (2Nx3N/2)	16x12	X		
	nLx2N (N/2x2N)	4x16	X		
	nRx2N (3N/2x2N)	12x16	X		
3 (N=4)	2Nx2N	8x8	X	X	X
	2NxN	8x4	X		
	Nx2N	4x8	X		
	NxN	4x4		X	

Grande parte da complexidade computacional associada à predição inter-quadros, se deve a tomada de decisão de quais são os tamanhos de PU que geram os melhores resultados na codificação, visto a quantidade de comparações que devem ser realizadas. Na versão mais recente do *draft* do padrão HEVC estão definidos 24 tamanhos de PU para serem testados na predição inter-quadros (KIM *et al.,* 2012). Todos estes diferentes tamanhos de PUs devem ser codificados em todas as etapas de compressão para definir qual deles apresenta a melhor relação entre compressão e qualidade e este processo é extremamente custoso. Reduzir a complexidade na predição inter-quadros, causando a menor perda possível na qualidade do vídeo codificado e na taxa de compressão, é um objetivo altamente desejável, principalmente para aplicações com restrições de processamento e de consumo de energia, como dispositivos portáteis.

Com o objetivo de conhecer quais são os tamanhos de PUs mais selecionados no processo de codificação, em especial na predição inter-quadros, foram realizadas, no escopo deste trabalho, algumas avaliações utilizando o software de referência HM. Todas as sequências definidas no documento (BOSSEN, 2012) foram codificadas para todas as classes das configurações *random-access* e *low-delay*. A configuração *intra-only* não foi utilizada, uma vez que a ME não é empregada nesta configuração. Portanto, foram utilizadas 24 sequências de teste separadas em classes de acordo com suas resoluções e características (BOSSEN, 2012).

A Tabela 4.5 apresenta os resultados obtidos quanto ao número de PUs selecionados na predição inter-quadros, desconsiderando blocos *Skip*, na configuração *low-delay.* Os resultados estão separados de acordo com as classes das sequências (BOSSEN, 2012). É possível perceber que nas classes C, D e F o tamanho de bloco mais selecionado foi 8x8, e que nas classes B e E, o tamanho mais selecionado foi 16x16. O resultado obtido nas classes B e E é esperado, uma vez que resoluções maiores tendem a causar a seleção de blocos maiores. Realizando uma média entre os resultados das classes percebe-se que os blocos 8x8 são os mais selecionados.

Tabela 4.5 – Tamanhos de PUs mais selecionados na configuração *low-delay* de acordo com as classes.

Tamanho da PU	**Quantidade de seleções de acordo com a classificação das sequências de vídeo (%)**					
	Classe B (1920x1080)	**Classe C (832x480)**	**Classe D (416x240)**	**Classe E (1280x720)**	**Classe F (832x480,1024x768,1280x720)**	**Média das Classes**
64x64	3,0383	1,2744	0,3135	4,0347	1,8262	2,0974
64x32	1,3047	0,4492	0,3724	1,7975	0,4771	0,8802
32x64	1,5723	0,6273	0,4107	4,0426	0,5175	1,4341
64x16	0,3170	0,0615	0,0270	0,3429	0,1758	0,1848
64x48	0,3170	0,0615	0,0270	0,3429	0,1758	0,1848
16x64	0,4076	0,1010	0,0435	0,9977	0,0972	0,3294
48x64	0,4076	0,1010	0,0435	0,9977	0,0972	0,3294
32x32	11,7579	5,4415	3,2790	10,7596	5,9764	7,4429
32x16	3,6059	2,1059	1,7708	4,1224	2,3267	2,7863
16x32	4,5268	3,2234	2,6814	7,6992	2,0100	4,0282
32x8	2,1564	1,1542	0,9747	2,3034	1,2637	1,5705
32x24	2,1564	1,1542	0,9747	2,3034	1,2637	1,5705
8x32	2,6462	1,8344	1,5337	4,1303	1,0857	2,2461
24x32	2,6462	1,8344	1,5337	4,1303	1,0857	2,2461
16x16	**16,8730**	13,8173	11,6039	**14,0307**	12,3268	13,7303
16x8	4,1158	4,4626	4,6078	3,5976	4,8437	4,3255
8x16	5,7227	7,4293	7,3940	7,1682	5,7458	6,6920
16x4	2,3790	2,5463	2,7824	1,9977	2,9773	2,5365
16x12	2,3790	2,5463	2,7824	1,9977	2,9773	2,5365
4x16	3,1703	4,3715	4,5779	3,8279	3,0161	3,7927
12x16	3,1703	4,3715	4,5779	3,8279	3,0161	3,7927
8x8	14,9225	**21,1278**	**21,7653**	8,0789	**22,2261**	**17,6241**
8x4	4,7500	7,6908	10,5938	2,5646	11,4229	7,4044
4x8	5,6573	12,2130	15,3292	4,9044	13,0693	10,2346
Total	100,0000	100,0000	100,0000	100,0000	100,0000	100,0000

Outro arranjo dos dados foi realizado com os resultados na configuração *low-delay*, separando-os por resolução, e não mais por classes, como pode ser observado na Tabela 4.6. Percebeu-se, nesse caso, que o tamanho 8x8 é o mais selecionado, exceto na resolução 1920x1080, no qual o tamanho 16x16 é mais selecionado. Ainda assim, nos vídeos da resolução 1920x1080, verificou-se um resultado próximo entre blocos 8x8 e 16x16, e que, na média de todas as resoluções, o tamanho 8x8 é o mais selecionado.

Tabela 4.6 – Tamanhos de PUs mais selecionados na configuração *low-delay* de acordo com as resoluções.

Tamanho da PU	Quantidade de seleções de acordo com a resolução das sequências de vídeo (%)					
	1920x1080	1280x720	1024x768	832x480	416x240	Média
64x64	3,0383	2,7344	1,4678	1,7300	0,3135	1,8568
64x32	1,3047	0,9192	0,6616	0,4423	0,3724	0,7400
32x64	1,5723	2,0693	0,4552	0,5373	0,4107	1,0090
64x16	0,3170	0,1755	0,1686	0,1704	0,0270	0,1717
64x48	0,3170	0,1755	0,1686	0,1704	0,0270	0,1717
16x64	0,4076	0,4620	0,1074	0,0990	0,0435	0,2239
48x64	0,4076	0,4620	0,1074	0,0990	0,0435	0,2239
32x32	11,7579	7,1705	6,7564	5,8613	3,2790	6,9650
32x16	3,6059	2,2559	2,5518	2,6773	1,7708	2,5723
16x32	4,5268	4,4641	1,6290	2,7510	2,6814	3,2105
32x8	2,1564	1,3510	1,8605	1,1465	0,9747	1,4978
32x24	2,1564	1,3510	1,8605	1,1465	0,9747	1,4978
8x32	2,6462	2,3503	0,9084	1,5541	1,5337	1,7985
24x32	2,6462	2,3503	0,9084	1,5541	1,5337	1,7985
16x16	**16,8730**	11,0620	10,2789	15,5355	11,6039	13,0707
16x8	4,1158	3,8513	5,4031	4,7032	4,6078	4,5362
8x16	5,7227	7,6531	3,7407	6,6909	7,3940	6,2403
16x4	2,3790	2,0620	3,3881	2,9107	2,7824	2,7044
16x12	2,3790	2,0620	3,3881	2,9107	2,7824	2,7044
4x16	3,1703	3,7253	2,3208	3,7978	4,5779	3,5184
12x16	3,1703	3,7253	2,3208	3,7978	4,5779	3,5184
8x8	14,9225	**16,7876**	**21,2694**	**21,8641**	**21,7653**	**19,3218**
8x4	4,7500	7,9944	14,7276	7,6633	10,5938	9,1458
4x8	5,6573	12,7860	13,5507	10,1866	15,3292	11,5020
Total	100,0000	100,0000	100,0000	100,0000	100,0000	100,0000

Considerando a configuração *random-access* e as mesmas condições do experimento anterior, os resultados são semelhantes, como pode ser observado na Tabela 4.7, que agrupa os resultados por classes. Para essa configuração, o tamanho de bloco 8x8 é o mais selecionado em três das classes avaliadas: C, D e F. Nas outras duas classes dessa configuração, A e B, os tamanhos mais selecionados são 32x32 e 16x16, respectivamente. Quando é feita uma média de todas as classes, verifica-se que o tamanho mais selecionado é o 8x8, assim como na configuração *low-delay*.

Quando os resultados na configuração *random-access* são separados por resolução, e não por classes, percebe-se que o tamanho 8x8 é o mais selecionado nas três resoluções menores. Nas três resoluções maiores, os blocos mais

selecionados são 32x32, 16x16 e 4x8, como pode ser observado na Tabela 4.8. Mesmo assim, verificou-se um resultado próximo entre blocos 8x8 e os tamanhos de blocos mais selecionados nas resoluções maiores, e que, na média de todas as resoluções, o tamanho 8x8 é o mais selecionado. Cabe salientar que em ambas as configurações, tanto *low-delay*, como *random-access*, mesmo quando o tamanho 8x8 não é o mais selecionado em uma classe ou resolução, o percentual é bastante significativo quando comparado ao percentual do tamanho mais selecionado.

Tabela 4.7 – Tamanhos de PUs mais selecionados na configuração *random-access* de acordo com as classes.

Tamanho da PU	**Quantidade de seleções de acordo com a classificação das sequências de vídeo (%)**					
	Classe A (2560x1600)	**Classe B (1920x1080)**	**Classe C (832x480)**	**Classe D (416x240)**	**Classe F (832x480,1024x768,1280x720)**	**Média das Classes**
64x64	3,7481	3,1223	1,1248	0,4798	2,2373	2,1425
64x32	2,3838	1,5186	0,4776	0,5580	0,8092	1,1494
32x64	2,6174	2,4139	1,0918	0,7995	1,1767	1,6198
64x16	0,3109	0,2712	0,0631	0,0440	0,3276	0,2033
64x48	0,3109	0,2712	0,0631	0,0440	0,3276	0,2033
16x64	0,3910	0,5337	0,2112	0,1166	0,1527	0,2810
48x64	0,3910	0,5337	0,2112	0,1166	0,1527	0,2810
32x32	**11,5337**	9,6140	4,5907	2,5499	5,9061	6,8389
32x16	6,9838	3,4304	2,1263	1,5001	2,7830	3,3647
16x32	7,2570	5,5148	4,5516	2,9636	3,1411	4,6856
32x8	3,5571	1,9312	1,1598	0,8678	1,3921	1,7816
32x24	3,5571	1,9312	1,1598	0,8678	1,3921	1,7816
8x32	3,4785	2,8226	2,4964	1,6177	1,5092	2,3849
24x32	3,4785	2,8226	2,4964	1,6177	1,5092	2,3849
16x16	10,2983	**14,6069**	11,8038	9,5212	10,8170	11,4094
16x8	5,7040	4,2938	4,5733	4,9935	5,2530	4,9635
8x16	5,4647	6,8495	9,2716	8,6988	6,3225	7,3214
16x4	2,8094	2,1995	2,3720	2,8195	2,9839	2,6368
16x12	2,8094	2,1995	2,3720	2,8195	2,9839	2,6368
4x16	2,4103	3,4341	5,1648	4,9925	3,1647	3,8333
12x16	2,4103	3,4341	5,1648	4,9925	3,1647	3,8333
8x8	9,5981	13,7018	**17,6332**	**20,4681**	**18,3920**	**15,9586**
8x4	4,5897	5,6328	6,8077	10,3395	11,0397	7,6819
4x8	3,9074	6,9165	13,0129	16,2120	13,0621	10,6222
Total	100,0000	100,0000	100,0000	100,0000	100,0000	100,0000

Tabela 4.8 – Tamanhos de PUs mais selecionados na configuração *random-access* de acordo com as resoluções.

Tamanho da PU	Quantidade de seleções de acordo com a resolução das sequências de vídeo (%)						
	2560x1600	1920x1080	1280x720	1024x768	832x480	416x240	Média
64x64	3,7481	3,1223	3,1990	1,8513	1,5146	0,4798	2,3192
64x32	2,3838	1,5186	0,6324	1,1416	0,5113	0,5580	1,1243
32x64	2,6174	2,4139	2,1920	0,8199	0,9542	0,7995	1,6328
64x16	0,3109	0,2712	0,0745	0,1920	0,3628	0,0440	0,2092
64x48	0,3109	0,2712	0,0745	0,1920	0,3628	0,0440	0,2092
16x64	0,3910	0,5337	0,1722	0,1181	0,1959	0,1166	0,2546
48x64	0,3910	0,5337	0,1722	0,1181	0,1959	0,1166	0,2546
32x32	**11,5337**	9,6140	5,2001	6,1384	5,3511	2,5499	6,7312
32x16	6,9838	3,4304	1,3155	3,0810	2,8221	1,5001	3,1888
16x32	7,2570	5,5148	4,3128	2,2322	3,9593	2,9636	4,3733
32x8	3,5571	1,9312	0,8556	1,8180	1,2281	0,8678	1,7096
32x24	3,5571	1,9312	0,8556	1,8180	1,2281	0,8678	1,7096
8x32	3,4785	2,8226	1,6964	1,0448	2,2220	1,6177	2,1470
24x32	3,4785	2,8226	1,6964	1,0448	2,2220	1,6177	2,1470
16x16	10,2983	**14,6069**	8,3660	9,1527	13,2158	9,5212	10,8601
16x8	5,7040	4,2938	4,9383	5,7971	4,7034	4,9935	5,0717
8x16	5,4647	6,8495	9,1898	4,1347	8,0410	8,6988	7,0631
16x4	2,8094	2,1995	2,0269	3,1530	2,9279	2,8195	2,6560
16x12	2,8094	2,1995	2,0269	3,1530	2,9279	2,8195	2,6560
4x16	2,4103	3,4341	3,8334	2,3326	4,4369	4,9925	3,5733
12x16	2,4103	3,4341	3,8334	2,3326	4,4369	4,9925	3,5733
8x8	9,5981	13,7018	15,3068	**19,3521**	**18,6074**	**20,4681**	**16,1724**
8x4	4,5897	5,6328	9,9999	14,8821	7,0087	10,3395	8,7421
4x8	3,9074	6,9165	**18,0295**	14,1001	10,5641	16,2120	11,6216
Total	100,0000	100,0000	100,0000	100,0000	100,0000	100,0000	100,0000

4.4 Avaliação de *Bit-rate* e PSNR com Tamanho de Bloco Fixo 8x8

Na seção anterior, foram apresentados os resultados da avaliação realizada com o objetivo de se conhecer quais são os tamanhos de PUs mais selecionados durante a codificação no HEVC. Verificou-se que tanto para a configuração *low-delay*, como para a configuração *random-access*, o tamanho de bloco mais selecionado é o 8x8. De posse desse resultado, uma nova experiência foi realizada. Através de alterações do código, o tamanho das PUs foi fixado em 8x8 para todos os blocos da predição inter-quadros, exceto blocos *Skip*. Dessa forma, foi possível avaliar as perdas em termos de qualidade de imagem e *bit-rate* utilizando tamanho de bloco fixo na ME do HEVC. Os resultados dessa avaliação podem ser observados na Tabela 4.9 e descrevem as perdas ocasionadas utilizando tamanho de bloco fixo 8x8 em relação à codificação com todos os tamanhos de bloco. É possível observar um aumento de 13,18% e 9,18% no *bit-rate* médio para as configurações *low-delay* e *random-access*, respectivamente. Também é possível observar os valores referentes à perda na qualidade de imagem, em média, sempre menores que 0,45dB.

Tabela 4.9 – Resultados de *bit-rate* e PSNR utilizando bloco de tamanho fixo 8x8.

Classes de Sequências	Configuração *low-delay*				Configuração *random-access*			
	***Bit-rate* (%)**	**Y-PSNR (dB)**	**U-PSNR (dB)**	**V-PSNR (dB)**	***Bit-rate* (%)**	**Y-PSNR (dB)**	**U-PSNR (dB)**	**V-PSNR (dB)**
Classe A (2560x1600)	-	-	-	-	11,1280	-0,6320	-0,2433	-0,1894
Classe B (1920x1080)	13,3201	-0,5135	-0,3791	-0,4131	10,7575	-0,4596	-0,2080	-0,2945
Classe C (832x480)	10,9450	-0,3783	-0,3769	-0,4150	8,3435	-0,3797	-0,2737	-0,3117
Classe D (416x240)	8,4443	-0,3467	-0,3457	-0,3292	5,3075	-0,3268	-0,2113	-0,2162
Classe E (1280x720)	17,4339	-0,5484	-0,3915	-0,4231	-	-	-	-
Classe F (832x480,1024x768,1280x720)	15,7493	-0,4112	-0,5831	-0,5350	10,3769	-0,4030	-0,3988	-0,3741
Média das Classes	13,1785	-0,4396	-0,4153	-0,4231	9,1827	-0,4402	-0,2670	-0,2772

Finalmente, observando os resultados com bloco fixo 8x8, foi realizado outro experimento, com o objetivo de identificar o impacto da FME dentro da ME em um cenário com bloco de tamanho fixo. Os resultados para essa avaliação podem ser observados na Tabela 4.10. Esta tabela mostra as perdas ocasionadas pela retirada da FME considerando tamanho de bloco fixo 8x8, em relação à codificação com todos os tamanhos de bloco usando a FME. Através dos resultados, pode-se observar um aumento no *bit-rate* médio de 22,91% e 14,65% para as configurações *low-delay* e *random-access*, respectivamente. Na Tabela 4.10 é possível verificar, também, a queda na qualidade da imagem para este caso, atingindo 0,6172 dB no pior caso.

Tabela 4.10 – Resultados de *bit-rate* e PSNR utilizando bloco de tamanho fixo 8x8 e sem FME.

Classes de Sequências	Configuração *low-delay*				Configuração *random-access*			
	***Bit-rate* (%)**	**Y-PSNR (dB)**	**U-PSNR (dB)**	**V-PSNR (dB)**	***Bit-rate* (%)**	**Y-PSNR (dB)**	**U-PSNR (dB)**	**V-PSNR (dB)**
Classe A (2560x1600)	-	-	-	-	15,2305	-0,7576	-0,2957	-0,2282
Classe B (1920x1080)	22,9825	-0,6340	-0,4003	-0,4548	15,5102	-0,5960	-0,2448	-0,3369
Classe C (832x480)	21,3375	-0,5861	-0,4234	-0,4944	15,0014	-0,6415	-0,3419	-0,3938
Classe D (416x240)	23,4056	-0,6357	-0,5290	-0,5382	13,8654	-0,5987	-0,2898	-0,3032
Classe E (1280x720)	25,7817	-0,6357	-0,4294	-0,5222	-	-	-	-
Classe F (832x480,1024x768,1280x720)	21,0369	-0,4627	-0,6557	-0,6191	13,6315	-0,4921	-0,4186	-0,3995
Média das Classes	22,9088	-0,5908	-0,4876	-0,5257	14,6478	-0,6172	-0,3181	-0,3323

Apesar de ocorrerem perdas significativas adotando tamanho de bloco fixo 8x8, tanto em termos de *bit-rate* como na qualidade da imagem, deve-se levar em

consideração que a complexidade computacional é reduzida drasticamente quando comparado com um cenário com 24 tamanhos de blocos diferentes a serem avaliados, que teriam que passar por todas as etapas de codificação para posterior decisão. Então, a decisão pelo uso de um único tamanho de bloco reduz a complexidade da ME em mais de 24 vezes. Como a ME é o módulo mais complexo do codificador, é possível imaginar que este ganho de complexidade será de grande relevância para o codificador como um todo. Assim, as perdas em qualidade e taxa de compressão se justificam, especialmente para aplicações focadas em dispositivos móveis, que possuem elevada restrição de capacidade de processamento e de consumo de energia.

Por outro lado, o uso da FME se justifica pelo fato de melhorar os resultados de *bit-rate* e qualidade de imagem com um custo computacional muito pequeno, já que a FME pode ser aplicada depois da ME para blocos de tamanho fixo, em um processo que pode ser realizado em paralelo com a ME do próximo bloco a ser processado. Além disso, o número de operações necessárias na FME é muito inferior ao utilizado na ME inteira e em outras operações do codificador.

Após a análise dos resultados apresentados nesta seção, optou-se, neste trabalho, pelo desenvolvimento de uma arquitetura para FME utilizando blocos de tamanho fixo 8x8.

5 ARQUITETURA DE HARDWARE DESENVOLVIDA

Basicamente, a FME pode ser dividida em duas etapas: interpolação e busca. Este trabalho apresenta o desenvolvimento de arquiteturas de hardware para FME capazes de realizar tanto a interpolação das amostras de luminância com precisão de ¼ de pixel do padrão HEVC, como a posterior busca e comparação dos blocos de vídeo gerados com o bloco que apresentou o melhor resultado na ME inteira. Para o projeto foi considerado o perfil *Main* do HEVC, com amostras de luminância com 8 bits do tipo sem sinal, em que a FME será realizada a partir de blocos de PU com tamanho fixo 8x8, o que reduz a complexidade computacional consideravelmente.

Desde a chamada de propostas para o padrão HEVC, a normatização da FME tem sofrido algumas alterações na medida em que avançam as contribuições com relação à ferramenta, como pode ser observado nos *drafts* do padrão ou em documentos específicos do JCT-VC (ALSHINA *et al.*, 2011). A arquitetura de hardware desenvolvida está de acordo com dois dos mais recentes documentos elaborados pelo JCT-VC até o término deste trabalho: HM9 – *High Efficiency Video Coding Test Model 9 Encoder Description* (KIM *et al.,* 2012) e *High Efficiency Video Coding text specification draft* 9 (BROSS *et al.,* 2012). Nestes documentos, é possível encontrar em detalhes o processo para a geração das posições de sub-pixel segundo o padrão HEVC.

5.1 Etapa de Interpolação

A etapa de interpolação é responsável por gerar os valores das amostras de luminância em posições de sub-pixel, e pode ser dividida, basicamente, em: (a) uma memória de entrada, para armazenar os valores de luminância das posições inteiras; (b) filtros de interpolação, para gerar os valores das amostras em posições

fracionárias e (c) uma memória de saída, para armazenar os valores das amostras de sub-pixel. Os filtros que geram as amostras em posições de sub-pixel têm papel fundamental e, portanto, o desenvolvimento da etapa de interpolação passou primeiramente pelo desenvolvimento dos filtros.

O cálculo dos valores das amostras em posições de sub-pixel podem ser realizados utilizando os algoritmos apresentados no capítulo 3, através das equações (4) a (18).

Com o objetivo de reduzir a quantidade de hardware necessária para implementação dos filtros de interpolação do padrão HEVC, foi feita uma análise dos cálculos a serem realizados para determinação de cada um dos valores das posições fracionárias. Em um primeiro momento se observou uma similaridade entre algumas posições, como pode ser observado na Tabela 5.1. Inicialmente, foram identificados quatro grupos diferentes de hardware para a construção da arquitetura dos filtros de interpolação. As posições $a_{0,0}$, $c_{0,0}$, $d_{0,0}$, e $n_{0,0}$, por exemplo, podem utilizar o mesmo hardware, uma vez que apesar das posições inteiras utilizadas para os cálculos serem diferentes, as constantes utilizadas nas multiplicações são as mesmas para as 4 posições.

Tabela 5.1 – Posições fracionárias e similaridades entre as posições nos algoritmos.

Posições Fracionárias	**Similaridade**
$a_{0,0}$, $c_{0,0}$, $d_{0,0}$ e $n_{0,0}$	Utilizam as mesmas constantes nas multiplicações (–1, –5, –10, 1, 4, 17 e 58) e *shift1*.
$b_{0,0}$ e $h_{0,0}$	Utilizam as mesmas constantes nas multiplicações (–1, –11, 4 e 40) e *shift1*.
$e_{0,0}$, $f_{0,0}$, $g_{0,0}$, $p_{0,0}$, $q_{0,0}$ e $r_{0,0}$	Utilizam as mesmas constantes nas multiplicações (–1, –5, –10, 1, 4, 17 e 58) e *shift2*.
$i_{0,0}$, $j_{0,0}$ e $k_{0,0}$	Utilizam as mesmas constantes nas multiplicações (–1, –11, 4 e 40) e *shift2*.

Contudo, uma análise mais detalhada a respeito das variáveis utilizadas para o cálculo dos valores de luminância com precisão de sub-pixel, permitiu uma redução na quantidade de filtros diferentes a serem desenvolvidos em hardware. As informações sobre essas variáveis são definidas segundo o documento HEVC *draft 9* (BROSS *et al.,* 2012) pelas seguintes expressões.

- *bit_depth_luma_minus8* pode ter valores de 0 a 6
- $BitDepth_Y = 8 + bit\_depth\_luma\_minus8$
- $shift1 = BitDepth_Y - 8$
- $shift2 = 6$

Analisando o software de referência HM e o *draft*, observou-se que a variável $BitDepth_Y$ está definida como uma constante cujo valor é 14. De posse do valor de $BitDepth_Y$ e de acordo com as expressões definidas no documento HEVC *draft 9* (BROSS *et al.,* 2012), é possível calcular os valores das demais variáveis, como segue abaixo.

- *bit_depth_luma_minus8* = 6
- $BitDepth_Y = 14$
- *shift1* = 6
- *shift2* = 6

Através dos resultados, é possível observar que os valores de *shift1* e *shift2* são idênticos. Portanto, a fim de diminuir o número de arquiteturas de filtros a serem desenvolvidas, as posições de sub-pixel foram separadas em dois grupos, de acordo com as constantes utilizadas nas multiplicações. A Tabela 5.2 apresenta os dois tipos de filtros que foram implementados de acordo com as constantes utilizadas nas multiplicações. Foi chamado de filtro tipo *Up/Down* o filtro utilizado para o cálculo das posições fracionárias $a_{0,0}$, $c_{0,0}$, $d_{0,0}$, $e_{0,0}$, $f_{0,0}$, $g_{0,0}$, $n_{0,0}$, $p_{0,0}$, $q_{0,0}$ e $r_{0,0}$. Por sua vez, o filtro tipo *Middle* é utilizado para o cálculo das posições fracionárias $b_{0,0}$, $h_{0,0}$, $i_{0,0}$, $j_{0,0}$ e $k_{0,0}$.

Tabela 5.2 – Arranjo final das posições fracionárias para o desenvolvimento das arquiteturas dos filtros.

Filtro	**Posições Fracionárias**	**Similaridade**
Tipo *Up/Down*	$a_{0,0}$, $c_{0,0}$, $d_{0,0}$, $e_{0,0}$, $f_{0,0}$, $g_{0,0}$, $n_{0,0}$, $p_{0,0}$, $q_{0,0}$ e $r_{0,0}$	Utilizam as mesmas constantes nas multiplicações (–1, –5, –10, 1, 4, 17 e 58).
Tipo *Middle*	$b_{0,0}$, $h_{0,0}$, $i_{0,0}$, $j_{0,0}$ e $k_{0,0}$	Utilizam as mesmas constantes nas multiplicações (–1, –11, 4 e 40).

Para permitir o paralelismo desejado, como será explicado nos próximos parágrafos, o filtro *Up/Down* foi replicado, transformando-se em dois filtros: *Up* e *Down*. A decisão de chamar os filtros de tipo *Up*, *Middle* e *Down* se deve à localização das posições fracionárias calculadas por cada tipo de filtro, como pode ser observado na Figura 5.1. Na Figura 5.1 são destacados em (a), as posições fracionárias para o filtro tipo *Up*, em (b), as posições fracionárias para o filtro tipo *Middle* e em (c), as posições fracionárias para o filtro tipo *Down*. Como é possível perceber a Figura 5.1, cada filtro passa a ser responsável pela geração de cinco amostras fracionárias.

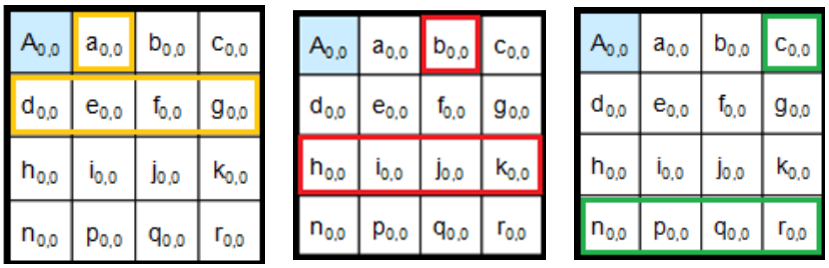


Figura 5.1 – Localização das posições fracionárias de acordo com o tipo do filtro.

Antes da implementação em hardware, cada tipo de filtro passou por uma análise a fim de se localizar outras possibilidades de otimização. Para o filtro tipo *Up/Down*, por exemplo, foram realizadas várias otimizações. Com o objetivo de dar maior clareza às otimizações realizadas, as representações das amostras inteiras e fracionárias no documento HEVC *draft 9* (BROSS *et al.,* 2012) foram substituídas por $\mathbf{a_0}$-$\mathbf{a_7}$ e **s** para entradas e saída, respectivamente. Primeiramente, as multiplicações e as constantes utilizadas na equação (19) foram reescritas na forma de soma/subtração de multiplicações cujas constantes são números na base dois, como pode ser observado na equação (20). Esta estratégia permite a realização de multiplicações utilizando somente operações de soma/subtração e deslocamentos, diminuindo consideravelmente o custo de hardware dos multiplicadores. Uma vez que cada constante poderia ser substituída por mais de um conjunto de soma/subtração de multiplicações de constantes na base dois, foram escolhidos os conjuntos que utilizaram um número menor de operações de soma/subtração, e consequentemente, menor custo de hardware. Por exemplo, $-10*a_2$ pode ser reescrito como $-(8*a_2 + 2*a_2)$, mas também poderia ser reescrito como $-(16*a_2$ -

$4*a_2 - 2*a_2$). Porém, neste último caso a escolha acarretaria em um maior custo de hardware, com um desempenho inferior. Em seguida, foram feitos agrupamentos de forma que as multiplicações por constantes na base dois fossem realizadas apenas uma vez, como pode ser observado na equação (21). Isto equivale a dizer que os deslocamentos necessários podem ser realizados apenas uma vez, além das operações de soma e subtração utilizarem menos bits, pois são realizadas antes destas operações. Finalmente, pode ser visto na equação (22) o resultado final das otimizações, já com os deslocamentos a serem realizados. Por exemplo, multiplicar por dois é equivalente a um deslocamento à esquerda de um bit, o que apresenta um custo muito menor do que uma multiplicação convencional.

$$s = [-a_0 + 4*a_1 - 10*a_2 + 58*a_3 + 17*a_4 - 5*a_5 + a_6] >> 6 \quad (19)$$

$$s = [-a_0 + 4*a_1 - (8*a_2 + 2*a_2) + (64*a_3 - 4*a_3 - 2*a_3) + (16*a_4 + a_4) - (4*a_5 + a_5) + a_6] >> 6 \quad (20)$$

$$s = [1*(-a_0 + a_4 - a_5 + a_6) - 2*(a_2 + a_3) + 4*(a_1 - a_3 - a_5) - 8*(a_2) + 16*(a_4) + 64*(a_3)] >> 6 \quad (21)$$

$$s = \{(-a_0 + a_4 - a_5 + a_6) - [(a_2 + a_3) << 1] + [(a_1 - a_3 - a_5) << 2] - (a_2 << 3) + (a_4 << 4) + (a_3 << 6)\} >> 6 \quad (22)$$

No filtro tipo *Middle* também foram realizadas várias otimizações similares às do filtro anterior. Foram utilizadas estratégias de reaproveitamento de hardware e multiplicações baseadas em soma/subtração e deslocamentos. Nas equações (23) a (28) pode ser observado todo o processo de otimização para o segundo filtro. Uma vantagem que pode ser observada na otimização do filtro tipo *Middle* é a possibilidade de reaproveitar hardware nas operações de soma, como pode ser observado na equação (26). Os valores R_1 e R_2 representam duas sub-expressões de hardware que podem ser reaproveitadas, conforme a equação (27). Na equação (28) é possível verificar o resultado final das otimizações, já com os deslocamentos a serem realizados.

$$s = [-a_0 + 4*a_1 - 11*a_2 + 40*a_3 + 40*a_4 - 11*a_5 + 4*a_6 - a_7] >> 6 \quad (23)$$

$$s = [-a_0 + 4*a_1 - (8*a_2 + 2*a_2 + a_2) + (32*a_3 + 8*a_3) + (32*a_4 + 8*a_4) - (8*a_5 + 2*a_5 + a_5) + 4*a_6 - a_7] >> 6 \quad (24)$$

$$s = [1*(-a_0 - a_2 - a_5 - a_7) + 2*(-a_2 - a_5) + 4*(a_1 + a_6) + 8*(-a_2 + a_3 + a_4 - a_5) + 32*(a_3 + a_4)] >> 6 \quad (25)$$

$$R_1 = a_2 + a_5 \quad \text{e} \quad R_2 = a_3 + a_4 \quad (26)$$

$$s = [1*(-a_0 - a_7 - R_1) - 2*R_1 + 4*(a_1 + a_6) + 8*(R_2 - R_1) + 32*R_2] >> 6 \quad (27)$$

$$s = \{(-a_0 - a_7 - R_1) - (R_1 << 1) + [(a_1 + a_6) << 2] + [(R_2 - R_1) << 3] + (R_2 << 5)\} >> 6 \quad (28)$$

Após a realização das otimizações, os filtros de interpolação foram descritos em VHDL através da ferramenta Quartus II da Altera (Altera, 2012). No

desenvolvimento do hardware, primeiramente foram descritas em VHDL duas arquiteturas puramente combinacionais, uma para cada tipo de filtro. Com a finalidade de obter melhores resultados em termos de desempenho, na sequência, foram desenvolvidas versões com dois e quatro estágios de *pipeline* destes filtros. Os tamanhos das entradas a_0-a_7 utilizadas nas equações (22) e (28) poderiam ser de 8 bits, considerando como entradas valores de amostras em posições inteiras, uma vez que o tamanho adotado no projeto para as amostras de luminância foi de 8 bits, sem sinal. Nesse caso, as saídas dos filtros apresentariam valores entre -64 e 319 ou -96 e 351 para algumas das posições fracionárias, dependendo do tipo de filtro. Contudo, os filtros das equações (22) e (28) podem ser utilizados considerando como entradas valores de amostras de posições fracionárias, requerendo que o tamanho das entradas seja de 10 bits, como pode se observado nas tabelas 5.3 e 5.4. Dessa forma, como o filtro funcionará com *pipeline* e um mesmo filtro poderá receber na entrada 8 ou 10 bits, dependendo das posições a serem calculadas, o tamanho adotado para as entradas dos filtros foi de 10 bits.

Tabela 5.3 – Faixa de valores possíveis nas entradas e saída do filtro tipo *Up/Down*.

Filtro Tipo *Up/Down*		
Posições Fracionárias	$a_{0,0}$, $c_{0,0}$, $d_{0,0}$ e $n_{0,0}$	$e_{0,0}$, $f_{0,0}$, $g_{0,0}$, $p_{0,0}$, $q_{0,0}$ e $r_{0,0}$
Valores possíveis nas entradas a_0 a a_6	0 a 255	-64 a 319
Número de bits necessários nas entradas	8	10
Menor valor possível na saída	-64	-160
Maior valor possível na saída	319	415
Número de bits necessários na saída	10	10

Tabela 5.4 – Faixa de valores possíveis nas entradas e saída do filtro tipo *Middle*.

Filtro Tipo *Middle*		
Posições Fracionárias	$b_{0,0}$ e $h_{0,0}$	$i_{0,0}$, $j_{0,0}$ e $k_{0,0}$
Valores possíveis nas entradas a_0 a a_7	0 a 255	-96 a 351
Número de bits necessários nas entradas	8	10
Menor valor possível na saída	-96	-264
Maior valor possível na saída	351	519
Número de bits necessários na saída	10	11

As arquiteturas desenvolvidas para os filtros tipo *Up/Down* e tipo *Middle* podem ser observadas nas figuras 5.2 e 5.3, respectivamente. As versões apresentadas são as versões com quatro estágios de *pipeline.* Contudo, a diferença para as versões com dois estágios de *pipeline* é que nesse caso não são usadas as barreiras temporais 1 e 3. No caso das versões combinacionais dos filtros, nenhuma das três barreiras temporais é utilizada.

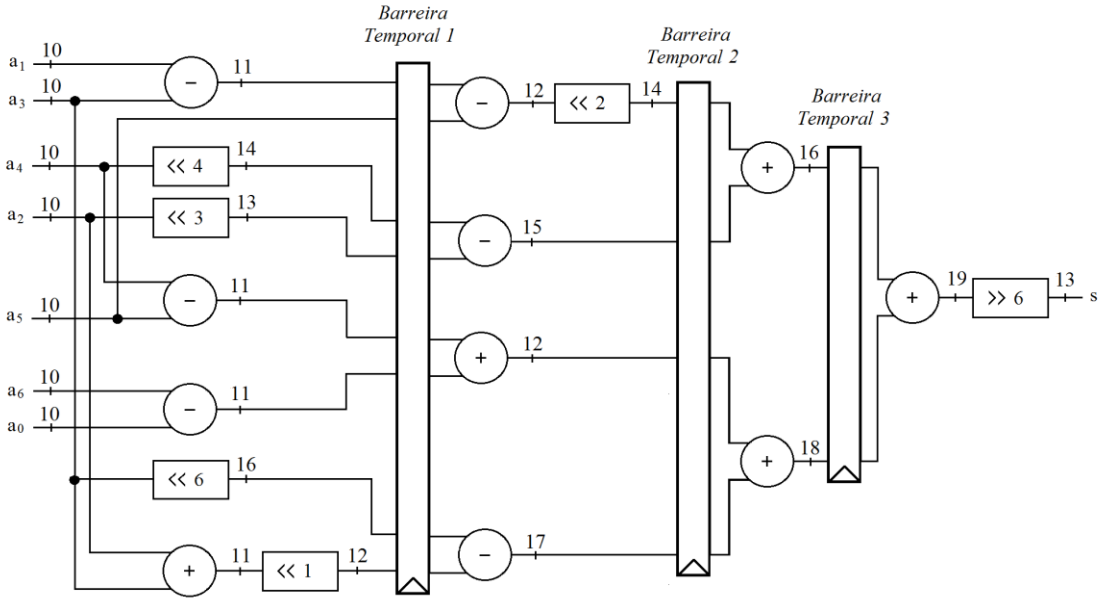


Figura 5.2 – Arquitetura de hardware para o filtro tipo *Up/Down*.

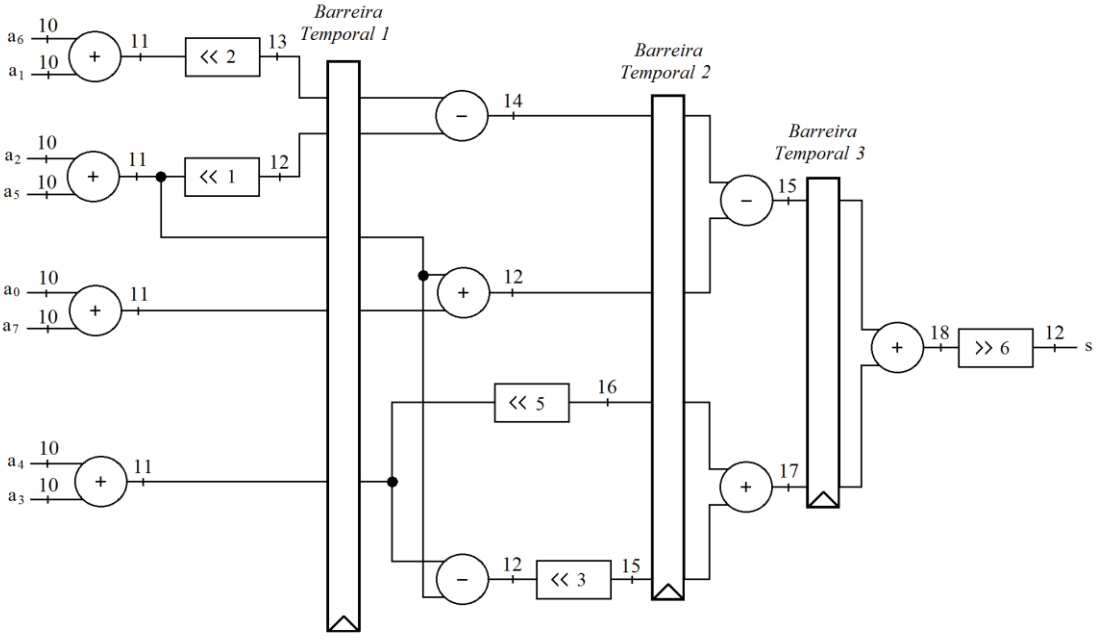


Figura 5.3 – Arquitetura de hardware para o filtro tipo *Middle*.

Como pode ser observado nos esquemas das figuras 5.2 e 5.3, os somadores e subtratores podem apresentar diferentes tamanhos em cada uma de suas entradas. Dessa forma, é necessária a extensão de sinal a fim de que as entradas de um mesmo somador ou subtrator tenham o mesmo número de bits. A extensão de sinal consiste em copiar o bit mais significativo (MSB – *Most Significant Bit*), quantas vezes sejam necessárias à esquerda até que as entradas tenham o mesmo tamanho. As operações de extensão de sinal não foram representadas nos esquemas. Contudo, estas operações estão presentes em todos os somadores e subtratores cujas entradas não apresentam o mesmo tamanho. Outra característica das operações de soma e subtração é que a saída apresenta um bit a mais em relação às entradas devido ao *carry-out* de cada somador e subtrator.

Outra consideração importante relacionada com as arquiteturas dos filtros é que no último deslocamento, o de seis bits à direita (equivalente a uma divisão por 64), é necessário ainda verificar se o bit MSB dos 6 bits descartados é igual a 1. Se for, ele deve ser somado ao resultado final do deslocamento, evitando, assim, erros de arredondamento nos valores das amostras de luminância.

Para o desenvolvimento da arquitetura completa da etapa de interpolação ainda são necessárias uma memória de entrada, para o armazenamento das amostras de luminância em posições inteiras, e uma memória de saída, para conter os valores das amostras em posições fracionárias. Estas memórias foram implementadas utilizando registradores. As informações armazenadas por estes registradores serão necessárias na busca e comparação de blocos fracionários com o bloco que apresentou o melhor resultado na ME inteira. A Figura 5.4 mostra a arquitetura proposta para toda a etapa de interpolação.

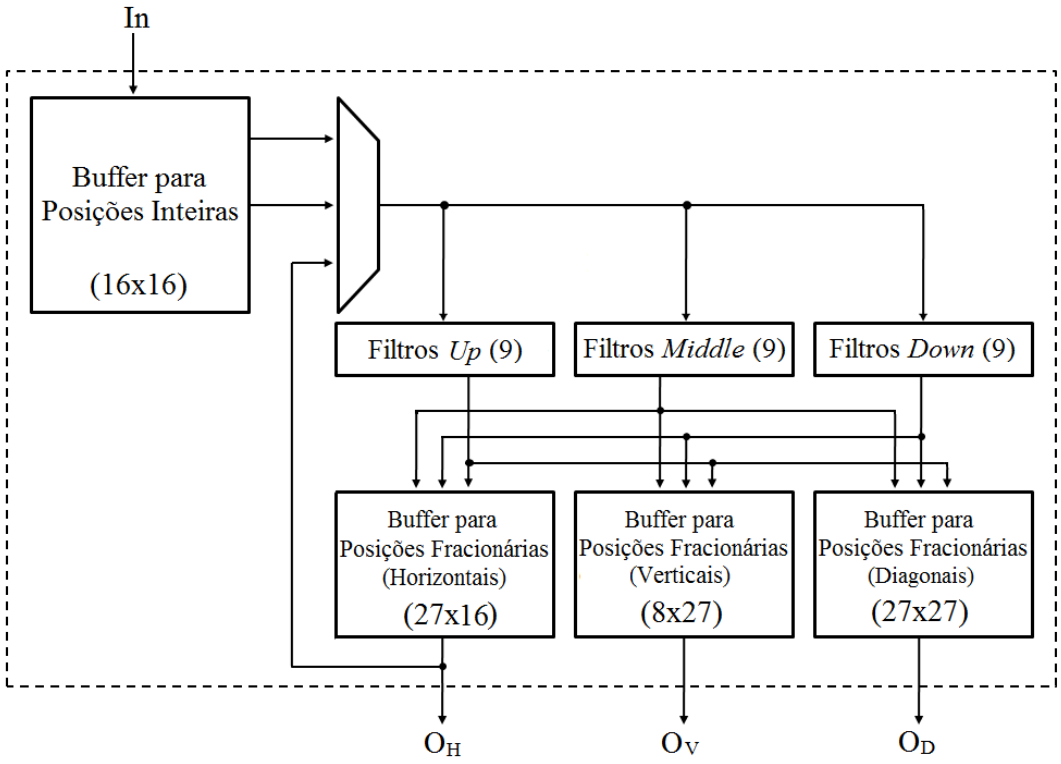


Figura 5.4 – Arquitetura de hardware para a etapa de interpolação.

A arquitetura completa para a etapa de interpolação apresenta um *buffer* para posições inteiras que deve armazenar 256 posições de 8 bits, uma vez que devem ser armazenadas, além das amostras em posições inteiras do bloco 8x8, uma borda de mais 4 amostras ao redor das amostras do bloco. A borda é necessária para o cálculo das posições fracionárias que não estão no centro do bloco. A Figura 5.5 ilustra como a borda é utilizada, onde as amostras inteiras ou fracionárias dentro do quadro vermelho representam as amostras do bloco 8x8, e as amostras fora do quadrado vermelho, a borda. Em verde, é possível observar a posição fracionária ‘b’ cujo valor deve ser calculado, e em azul, as amostras necessárias para o cálculo de acordo com o algoritmo da posição ‘b’. No esquema da Figura 5.4, *In* representa a entrada das amostras de luminância em posições inteiras, cuja atualização no *buffer* ocorre sempre em linhas.

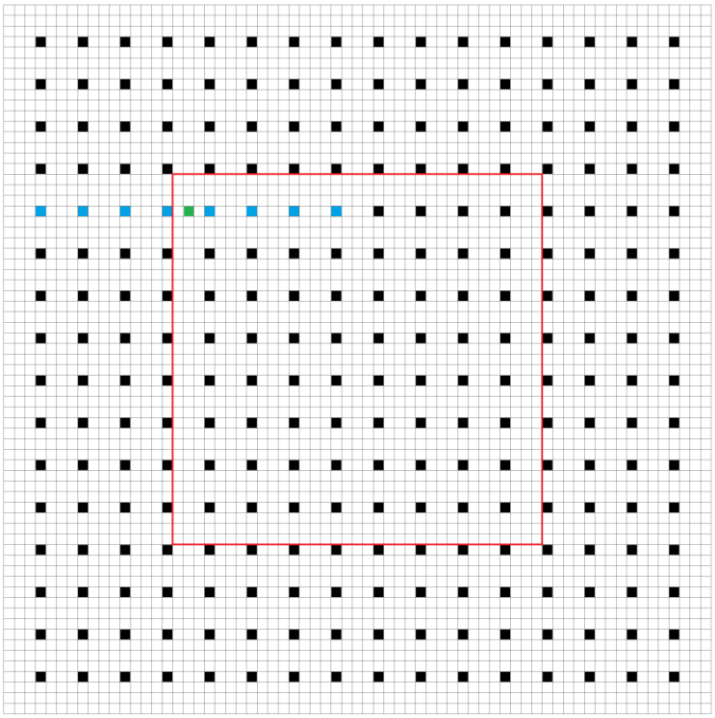


Figura 5.5 – Representação da borda de amostras necessária para a interpolação.

Para a interpolação das amostras em posições inteiras, optou-se por um esquema capaz de realizar o cálculo de uma linha ou coluna inteira de amostras em posições fracionárias a cada ciclo. Portanto, foram necessários três conjuntos de filtros (*Up*, *Middle* e *Down*), cada um com 9 unidades, permitindo o cálculo de 9 posições fracionárias por conjunto, ou 27 posições por ciclo.

Na Figura 5.4 o multiplexador tem a função de selecionar as amostras de entrada dos conjuntos de filtros, vindas do *buffer* de posições inteiras ou de posições fracionárias horizontais, de acordo com uma etapa de controle. No total, para um bloco de tamanho 8x8 ,devem ser calculadas 432 posições fracionárias horizontais em relação as posições inteiras, 216 posições fracionárias verticais e 729 posições fracionárias diagonais, como pode ser observado em verde na Figura 5.6.

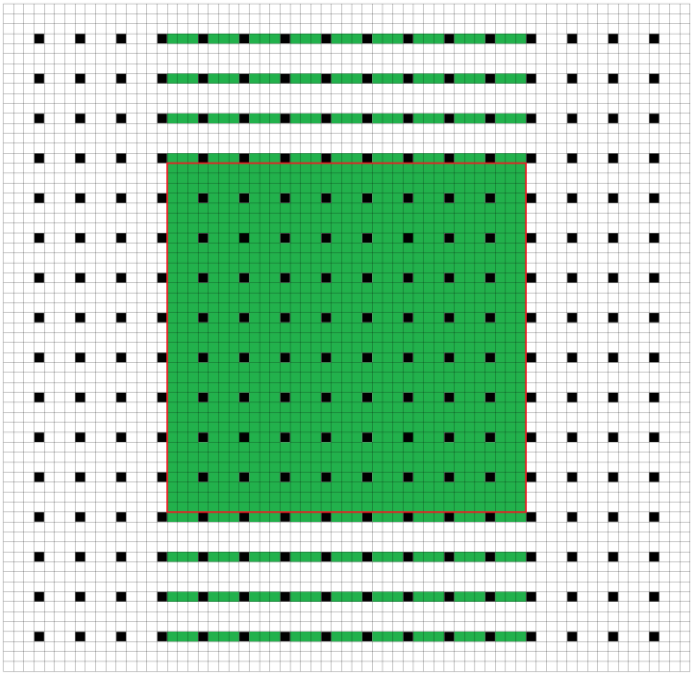


Figura 5.6 – Posições fracionárias que devem ser calculadas.

Na Figura 5.7 são destacados os três tipos de posições fracionárias de acordo com os *buffers* da seguinte forma: em (a), as posições fracionárias horizontais, em (b), as posições fracionárias verticais e em (c), as posições fracionárias diagonais. Na Figura 5.4, O_H, O_V e O_D representam as saídas dos *buffers* que armazenam as amostras de luminância nas posições fracionárias.

Figura 5.7 – Relação das posições fracionárias de acordo com os *buffers*.

As posições fracionárias na borda que devem ser calculadas são necessárias devido ao fato de algumas posições fracionárias requererem outras posições fracionárias (horizontais) como entrada, o que pode ser observado na Tabela 5.5.

Tabela 5.5 – Posições de entrada para o cálculo das posições fracionárias.

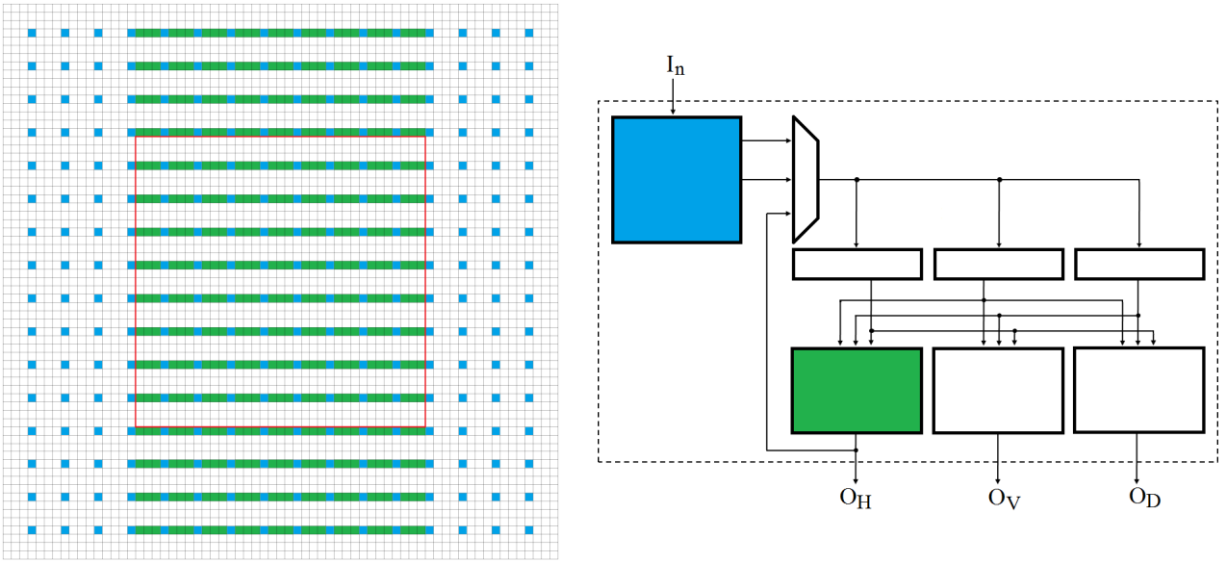
Posições Fracionárias	Posições utilizadas para o cálculo	
$a_{0,0}$	$A_{-3,0}$ a $A_{3,0}$	$A_{-3,0}$ a $A_{4,0}$
$b_{0,0}$	$A_{-3,0}$ a $A_{4,0}$	
$c_{0,0}$	$A_{-2,0}$ a $A_{4,0}$	
$d_{0,0}$	$A_{0,-3}$ a $A_{0,3}$	$A_{0,-3}$ a $A_{0,4}$
$h_{0,0}$	$A_{0,-3}$ a $A_{0,4}$	
$n_{0,0}$	$A_{0,-2}$ a $A_{0,4}$	
$e_{0,0}$	$a_{0,-3}$ a $a_{0,3}$	$a_{0,-3}$ a $a_{0,4}$
$i_{0,0}$	$a_{0,-3}$ a $a_{0,4}$	
$p_{0,0}$	$a_{0,-2}$ a $a_{0,4}$	
$f_{0,0}$	$b_{0,-3}$ a $b_{0,3}$	$b_{0,-3}$ a $b_{0,4}$
$j_{0,0}$	$b_{0,-3}$ a $b_{0,4}$	
$q_{0,0}$	$b_{0,-2}$ a $b_{0,4}$	
$g_{0,0}$	$c_{0,-3}$ a $c_{0,3}$	$c_{0,-3}$ a $c_{0,4}$
$k_{0,0}$	$c_{0,-3}$ a $c_{0,4}$	
$r_{0,0}$	$c_{0,-2}$ a $c_{0,4}$	

Os *buffers* para armazenamento de posições horizontais e verticais deveriam trabalhar com amostras de 10 bits, enquanto que o *buffer* para posições diagonais trabalharia com amostras de 11 bits, conforme explicado anteriormente. Contudo, como a próxima etapa da FME, a etapa de busca e comparação requer entradas de 8 bits, as amostras devem passar por uma operação chamada de clipagem. A clipagem consiste de tornar zero os valores menores que zero e tornar 255 valores maiores que 255. Assim, as amostras ficam com 8 bits de tamanho. Os valores entre 0 e 255 não são alterados nesse processo. As amostras fracionárias verticais e diagonais podem ter a clipagem realizada antes mesmo de ter seus valores armazenados nos respectivos *buffers*. Contudo, as amostras horizontais só podem ser clipadas já no envio para a etapa de busca e comparação, ou seja, o *buffer* para amostras horizontais tem 10 bits, enquanto os outros têm o tamanho reduzido para 8 bits. Isto é necessário porque as posições horizontais calculadas devem ser utilizadas nos cálculos de outras posições fracionárias.

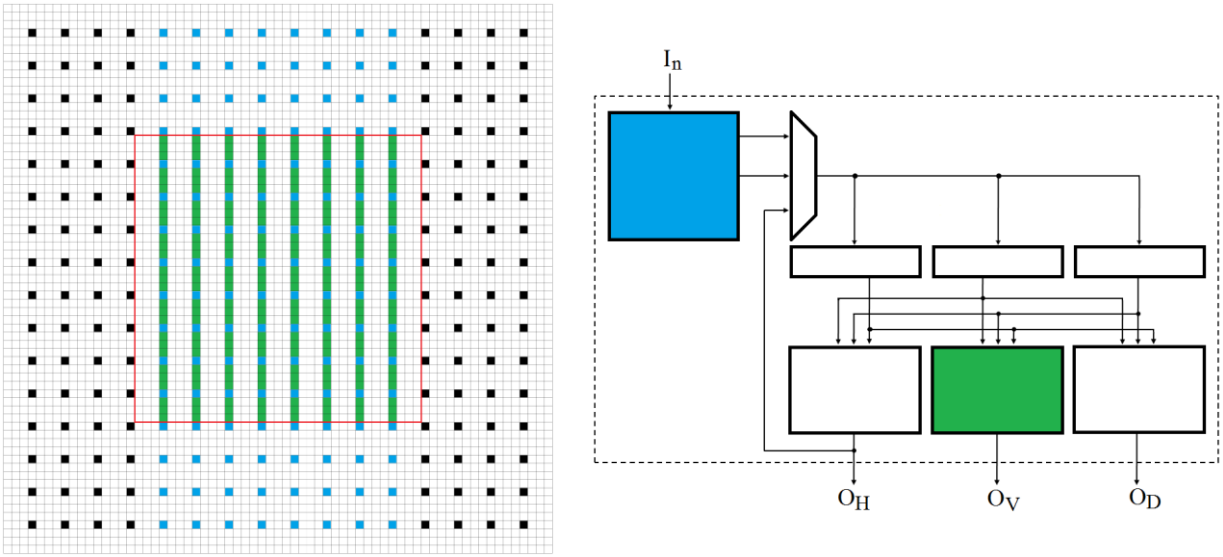
A Tabela 5.5 e a Figura 5.8 também permitem entender como se dão os cálculos das amostras em posições fracionárias. Na Figura 5.8, os esquemas da arquitetura consistem em versões simplificadas da arquitetura apresentada na Figura 5.4, sendo que as amostras em azul representam as entradas e as amostras em verde, as saídas. O mesmo padrão de cores também permite identificar o *buffer* que armazena as amostras de entrada e o *buffer* que armazena as amostras de saída. Primeiramente, a partir das amostras em posições inteiras horizontais de uma linha são calculadas todas as posições fracionárias horizontais dessa linha, ou seja, todas as posições (a, b, c). Esse processo é feito para as 16 linhas. Em seguida, utilizando amostras em posições verticais inteiras de uma coluna são calculadas as amostras em posições fracionárias verticais da coluna, ou seja, (d, h, n). Este processo deve ser repetido 8 vezes, pois apenas 8 das colunas precisam ser processadas. Por fim, a partir das amostras em posições fracionárias (a, b, c) recém-calculadas são calculadas as amostras das demais posições fracionárias. São calculadas todas as colunas que contém as posições (e, i, p) utilizando posições **a**, todas as colunas de posições (f, j, q) a partir das posições **b** e todas as colunas de posições (g, k, r) usando posições **c**, o que totaliza mais 27 colunas.

De acordo com o número de linhas e colunas a serem processadas pode-se estimar um total de 51 ciclos para a completa interpolação de um bloco 8x8, considerando que o *pipeline* dos filtros de interpolação esteja preenchido. Isso vale para todas as versões de filtros desenvolvidas, dado que, para todas elas, é idêntico o número de amostras consumidas por ciclo de *clock*. As amostras em posições inteiras são atualizadas durante o cálculo das amostras fracionárias diagonais, já que não são utilizadas durante os 27 ciclos necessários para esse cálculo. Contudo, na inicialização, as amostras em posições inteiras são carregadas antes do restante da arquitetura entrar em operação (16 ciclos de *clock*). Portanto, a arquitetura precisa de 67 a 71 ciclos para interpolar por completo o primeiro bloco, dependendo da versão dos filtros. Após essa latência, 51 ciclos de *clock* são necessários para a interpolação dos blocos subsequentes.

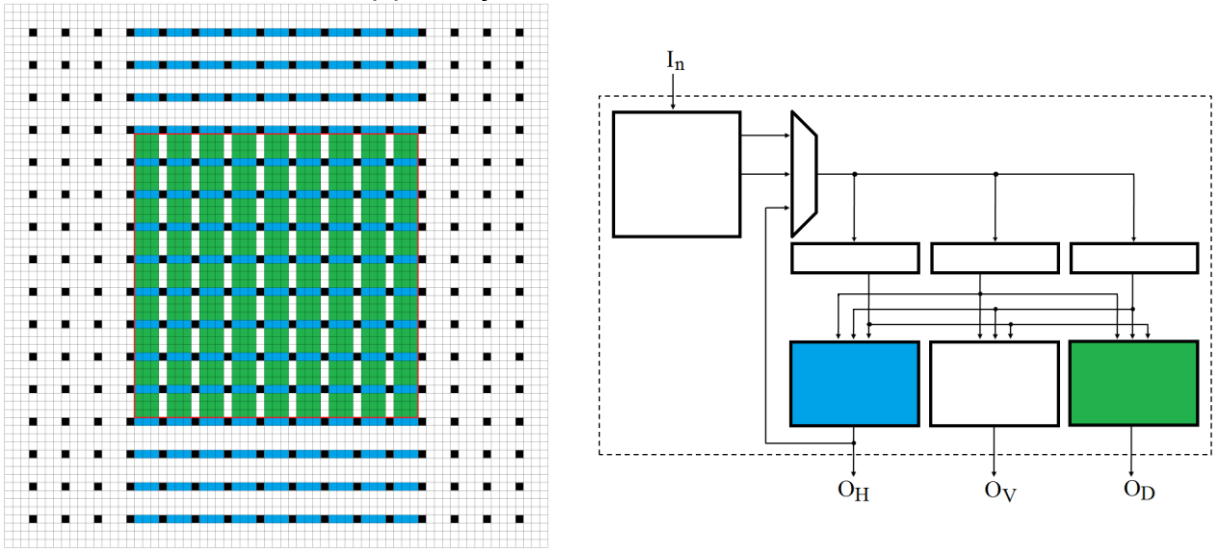
Dado que o principal objetivo deste trabalho é o desenvolvimento de uma FME com elevada taxa de processamento capaz de processar vídeos de alta resolução em tempo real, a arquitetura de filtro com quatro estágios de *pipeline* será a utilizada na arquitetura completa da FME.



(a) Posições fracionárias horizontais.



(b) Posições fracionárias verticais.



(c) Posições fracionárias diagonais.

Figura 5.8 – Cálculo das posições fracionárias.

5.2 Etapa de Busca e Comparação

Primeiramente, cabe salientar que a etapa de busca e comparação foi desenvolvida após a etapa de interpolação estar completamente implementada e, portanto, foi adequada ao que já estava desenvolvido para a etapa de interpolação, de forma que mantivesse o mesmo número de ciclos para o processamento de um bloco 8x8.

Como pode ser observado na Figura 5.9, o bloco 8x8 que apresentou o melhor resultado após a busca da ME inteira (representado pelos quadrados pretos) pode ser comparado com outros 48 blocos novos, em posições fracionárias, gerados pela etapa de interpolação (7 x 7 –1). Na Figura 5.9, cada quadrado identifica uma amostra da imagem após a interpolação. Assim, os números dentro dos quadrados identificam a primeira amostra de cada bloco a ser comparado. Neste trabalho optou-se por utilizar o algoritmo FS, ou seja, em que todos os blocos interpolados são comparados. Este algoritmo foi usado porque o número de blocos a serem comparados é reduzido e, dessa forma, serão obtidos resultados ótimos dentro da área de busca.

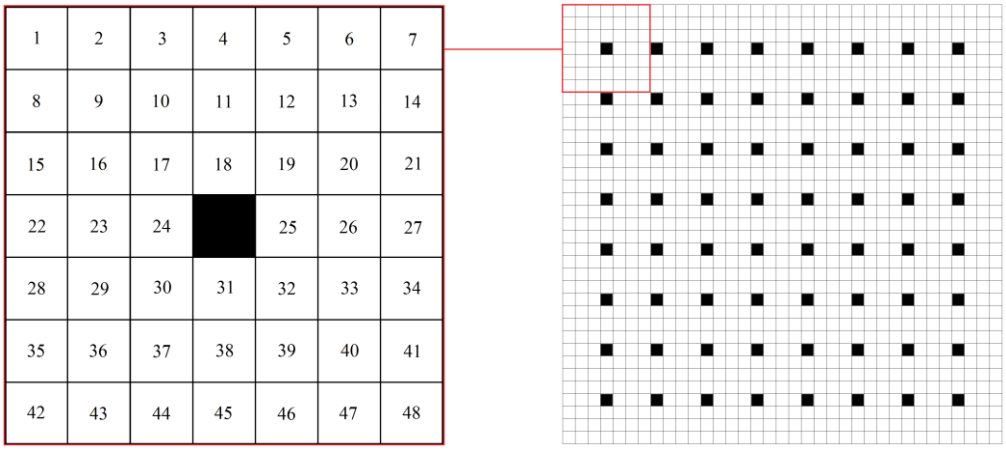


Figura 5.9 – Blocos em posições fracionárias gerados pela etapa de interpolação.

Basicamente, a etapa de busca e comparação é formada por árvores de SAD, um comparador de SADs e *buffers* de armazenamento. As árvores de SAD permitem o cálculo do SAD de todos os blocos fracionários gerados pela etapa de interpolação. O comparador de SADs tem por função comparar o SAD dos blocos fracionários com o SAD do bloco que obteve o melhor resultado na ME inteira. Ainda

são necessários um *buffer* para armazenar as amostras em posições inteiras do bloco atual e uma memória para os vetores de movimento fracionários. Uma visão geral da etapa de busca e comparação pode ser observada na Figura 5.10.

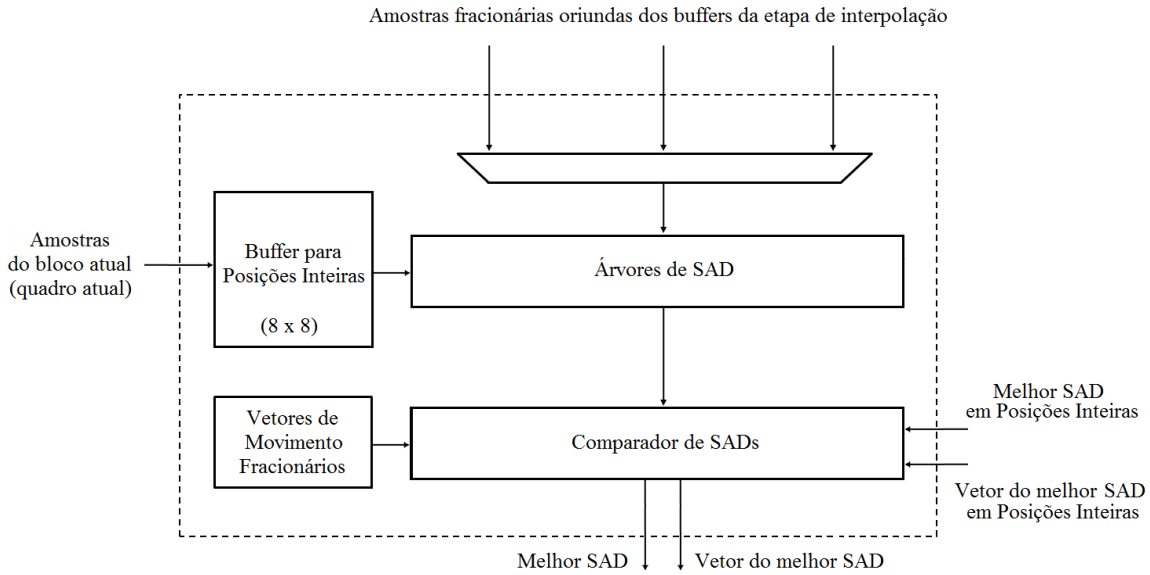


Figura 5.10 – Arquitetura de hardware para a etapa de busca e comparação.

Na Figura 5.10, o módulo com as árvores de SAD na verdade consiste de 12 unidades de processamento (UPs) menores capazes de calcular o SAD de 12 blocos de posições fracionárias simultaneamente. Cada UP foi desenvolvida em um *pipeline* de cinco estágios. A cada oito ciclos de *clock*, após a latência, são entregues pelas árvores de SAD para o módulo comparador os resultados de SAD de 12 blocos. Desta forma, como 48 blocos fracionários precisam ser comparados, são necessários 32 ciclos de *clock* após o preenchimento do *pipeline*.

O *buffer* para posições inteiras da Figura 5.10 fornece as amostras em posições inteiras do bloco atual, necessárias nas árvores de SAD.

Na Figura 5.11, pode-se observar em detalhe a arquitetura de uma UP das árvores de SAD, com seus cinco estágios de *pipeline*. Uma UP tem a função de calcular o SAD a partir de uma linha (ou coluna) de um bloco fracionário e uma linha (ou coluna) do bloco atual. As entradas B representam as amostras do bloco atual e as entradas R, representam as amostras do bloco fracionário, a partir de um quadro de referência. Desta forma, como são 12 unidades, são calculados 12 blocos. Logo na entrada da UP, existem módulos *abs*, os quais têm a função de transformar o resultado das subtrações utilizadas no primeiro estágio de *pipeline* nas árvores de SAD em um número absoluto. Para tal, é realizado um teste no MSB do resultado da

operação de subtração. Caso o MSB seja um, indica que o resultado da operação resultou em número negativo. Então, para se obter o resultado absoluto, é necessário a realização de um complemento de 2 (o valor do resultado é invertido e acrescido de 1). No caso do resultado ser positivo, este não precisa ser modificado. Em ambos os casos, o módulo *abs* elimina o bit de sinal para a próxima etapa de *pipeline*. A partir deste resultado, os valores absolutos de cada posição vão sendo somados até se obter o valor do SAD de uma linha ou coluna. O acumulador no final da arquitetura irá acumular o resultado das oito linhas ou colunas do bloco, obtendo-se, após oito ciclos, o SAD para o bloco. A saída dos acumuladores das árvores de SAD precisam de 14 bits na representação (255 x 8 x 8) e a saída do último somador, antes do acumulador, deve ser de 11 bits (255 x 8). O multiplexador utilizado no acumulador tem a função de inicializar e reinicializar o valor do acumulador a cada oito ciclos de *clock*, quando os resultados finais de SAD de cada bloco ficam prontos. Os estágios de *pipeline* foram balanceados de forma a não haver operações de soma ou subtração em série em um mesmo estágio, visando atingir uma elevada frequência de operação para essa primeira versão da UP. Apenas no primeiro estágio podem acontecer duas operações em série (subtração e geração do valor absoluto), mas estas duas operações foram alocadas neste estágio porque os operadores possuem menor largura de bits se comparado com os operadores dos outros estágios e, portanto, possuem os menores atrasos. A arquitetura de uma unidade de processamento das árvores de SAD necessita de 13 ciclos de *clock* para entregar o resultado de SAD do primeiro bloco. Após essa latência, necessária para o preenchimento do *pipeline*, a unidade entrega o resultado de SAD de um novo bloco a cada 8 ciclos, conforme explicado anteriormente.

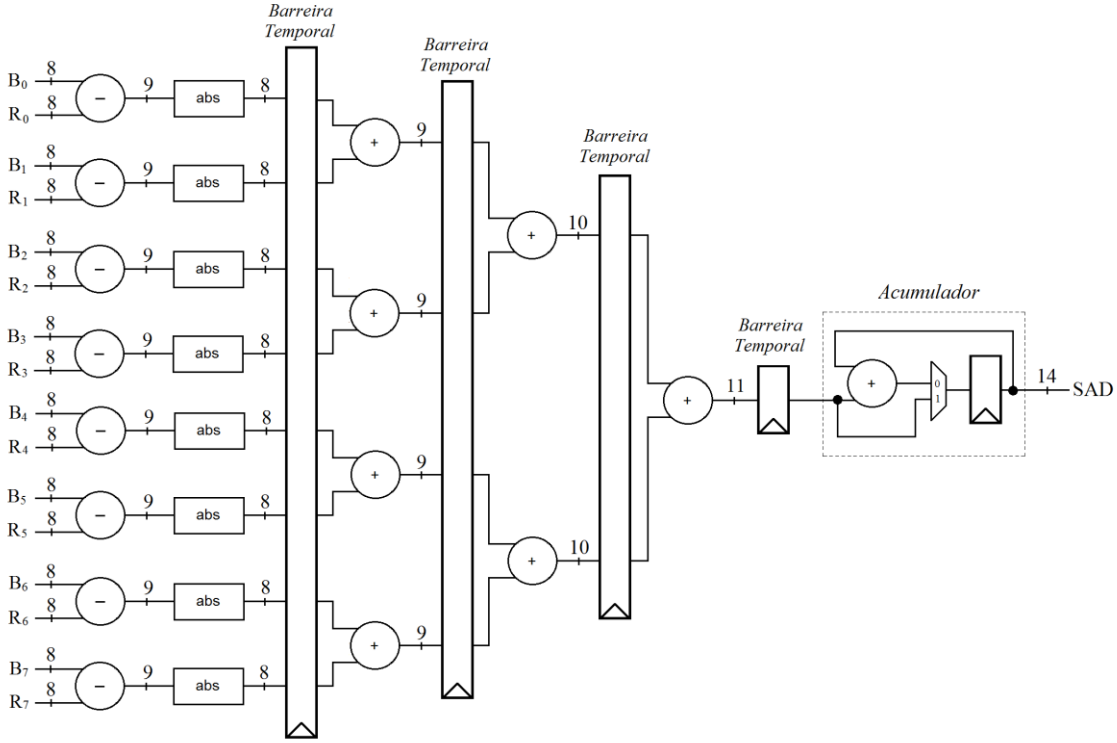


Figura 5.11 – Arquitetura de uma unidade de processamento das árvores de SAD.

Na Tabela 5.6 se pode observar em detalhes como se dá o cálculo do SAD de todos os blocos, desconsiderando os ciclos necessários para o preenchimento do *pipeline*. Nesta tabela, o *buffer* utilizado para o cálculo de cada bloco está representado pelas letras H (horizontal), V (vertical) e D (diagonal), antecedendo o número do bloco. Primeiramente, são calculados os blocos oriundos dos *buffers* de posições fracionárias horizontais e verticais e, posteriormente, os blocos oriundos de posições fracionárias diagonais.

Tabela 5.6 – Blocos calculados por cada unidade de processamento das árvores de SAD.

Unidade de Processamento	Ciclos de *clock* utilizados e os respectivos blocos calculados			
	1 a 8	9 a 16	17 a 24	25 a 32
UP01	H 22	D 01	D 03	D 06
UP02	H 23	D 08	D 10	D 13
UP03	H 24	D 15	D 17	D 20
UP04	H 25	D 28	D 30	D 33
UP05	H 26	D 35	D 37	D 40
UP06	H 27	D 42	D 44	D 47
UP07	V 04	D 02	D 05	D 07
UP08	V 11	D 09	D 12	D 14
UP09	V 18	D 16	D 19	D 21
UP10	V 31	D 29	D 32	D 34
UP11	V 38	D 36	D 39	D 41
UP12	V 45	D 43	D 46	D 48

Após o cálculo do SAD dos 12 primeiros blocos, os valores dos SADs são enviados, juntamente com os respectivos vetores de movimento, ao comparador. A Figura 5.12 apresenta a arquitetura do comparador de SAD, que foi desenvolvida com cinco estágios de *pipeline*. A função do comparador é armazenar o SAD e o vetor do bloco que apresentou o melhor resultado entre os blocos fracionários e comparar com o melhor resultado da ME inteira, para entregar, na saída, o vetor do bloco com o melhor resultado entre todas as possibilidades. Os dois multiplexadores permitem a comparação do primeiro conjunto de resultados com o melhor resultado da ME em posições inteiras. A identificação e o valor do SAD do bloco de melhor resultado na ME inteira são usados para inicializar o valor armazenado na saída do comparador durante a comparação dos primeiros 12 blocos. Em seguida, após a comparação dos 12 primeiros blocos com o melhor resultado da ME inteira, os outros 36 blocos serão comparados com esse primeiro resultado, sempre de 12 em 12. Como pode ser observado na Figura 5.12, internamente no comparador de SADs, as comparações são realizadas de dois em dois blocos.

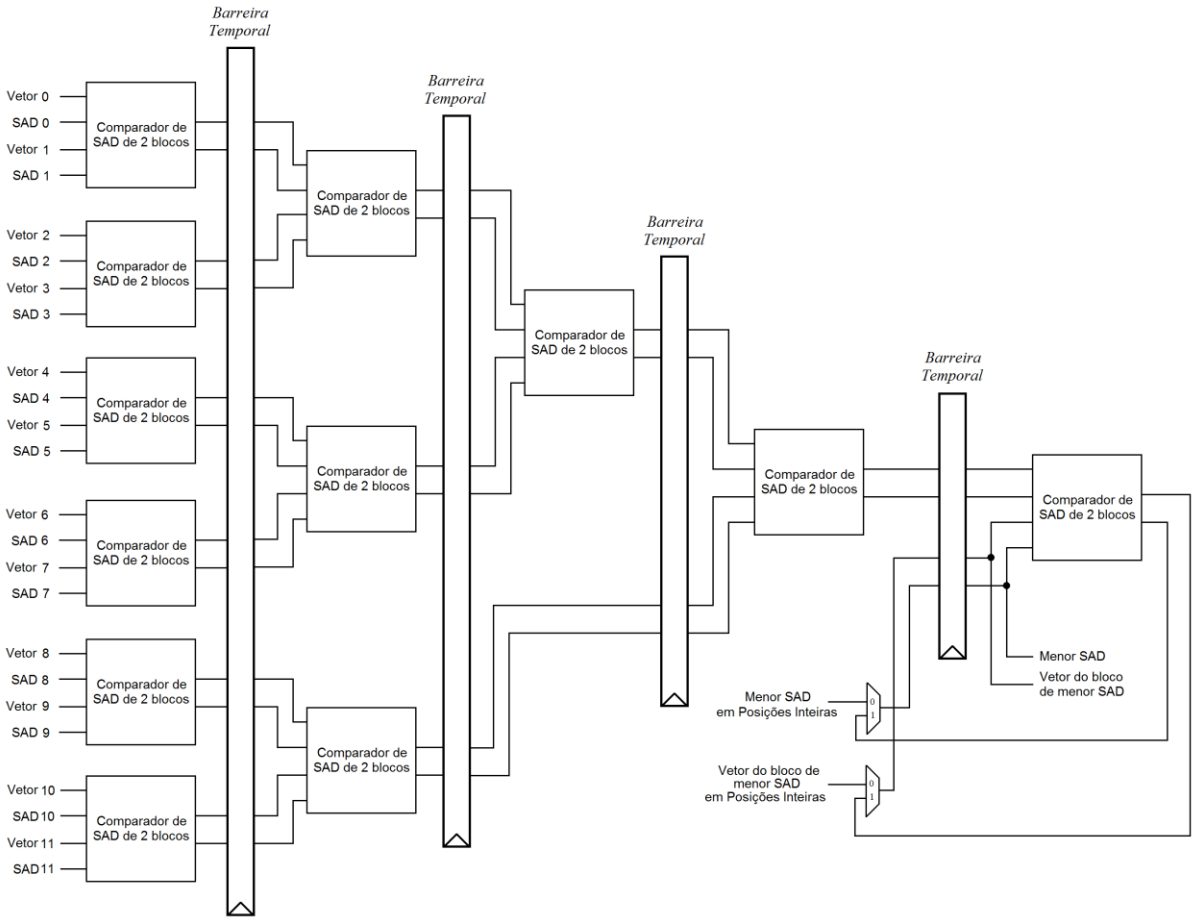


Figura 5.12 – Arquitetura do comparador de SADs para 12 blocos.

Na Figura 5.13, pode-se observar a arquitetura dos comparadores de dois blocos utilizados no comparador de SADs que compara 12 blocos. A arquitetura do comparador de 2 blocos é bem simples, e consiste basicamente em realizar uma subtração entre dois valores de SAD referentes a dois blocos. De acordo com o resultado do MSB do resultado da operação é possível saber qual dos blocos apresentava o maior SAD. Como esse bit está ligado diretamente ao controle de dois multiplexadores, cujas entradas estão ligadas aos SADs e vetores de movimento, serão entregues na saída do comparador apenas os valores referentes ao melhor resultado de SAD. Como pode ser observado nas figuras 5.12 e 5.13, os estágios de *pipeline* foram utilizados de forma a não haver operações de subtração em série. O comparador de SADs para 12 blocos entrega o resultado da comparação após 5 ciclos de *clock*, necessários para o preenchimento do *pipeline*. Como o comparador tem dependência em relação ao módulo das árvores de SAD, e as árvores de SAD levam 13 ou 8 ciclos para entregar os resultados de SAD de 12 blocos ao comparador, o comparador funciona perfeitamente.

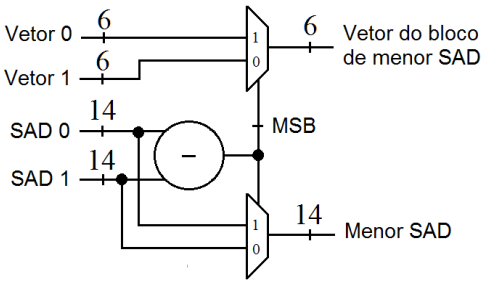


Figura 5.13 – Arquitetura do comparador de SADs para 2 blocos.

Como são 48 blocos para comparação na FME, são necessários seis bits para a representação dos vetores de movimento fracionários, mais um *flag* para diferenciar o vetor fracionário do vetor da ME inteira. Todos os vetores possíveis estão armazenados em uma ROM (*Read-Only-Memory*) e estes vetores são selecionados de acordo com a posição do bloco com o menor SAD.

5.3 Integração das Etapas de Interpolação e de Busca e Comparação

As etapas de interpolação e de busca e comparação foram integradas a partir do desenvolvimento de uma unidade de controle. Uma máquina de estados foi implementada para permitir o controle adequado da arquitetura completa.

Como a etapa de interpolação requer 51 ciclos para gerar os valores de sub-pixel referentes a um bloco 8x8, foi necessário um estudo para que não ocorresse falta de sincronismo, e para que valores dos *buffers* não fossem perdidos. Basicamente, optou-se pela realização da busca em posições horizontais e verticais enquanto as posições diagonais estão sendo interpoladas, e a busca em posições diagonais durante a interpolação das posições horizontais e verticais de um bloco seguinte. Já o *buffer* para as amostras do bloco atual é atualizado durante a interpolação das amostras fracionárias diagonais, exatamente no momento em que essas informações não são utilizadas. A Tabela 5.7 ilustra como é realizado o sincronismo.

Tabela 5.7 – Sincronismo das etapas de interpolação e de busca e comparação.

Número de ciclos de *clock*	8	8	4	16	8	14	2	3	8	5	3	5	3	5	3	5	3	6	2	3	8
Atualização do *buffer* do bloco de referência	16					16										16					
Atualização do *buffer* do bloco atual		8														8					
Etapa de Interpolação				51						51											
Preenchimento do *pipeline* dos filtros de interpolação			4																		
Interpolação – posições fracionárias horizontais				16						16											
Interpolação – posições fracionárias verticais					8									8							
Interpolação – posições fracionárias diagonais						27										27					
Etapa de Busca e Comparação							42														
Preenchimento do *pipeline* das árvores de SAD							5												5		
Cálculo do SAD dos blocos - posições fracionárias horizontais e verticais									8												8
Cálculo do SAD dos blocos - posições fracionárias diagonais										24											
Preenchimento do *pipeline* do comparador de SADs de 12 blocos										5		5		5		5					
Latência da arquitetura completa	100																				

A arquitetura completa para FME pode ser observada na Figura 5.14. Os primeiros resultados válidos na saída da arquitetura de FME dependem, além dos ciclos gastos na interpolação e na busca, também dos ciclos necessários para o

armazenamento dos valores das amostras em posições inteiras do bloco de referência (16 ciclos de *clock*), para o preenchimento do *pipeline* dos filtros de interpolação (4 ciclos de *clock*) e para o preenchimento do *pipeline* do comparador de SADs de 12 blocos (5 ciclos de *clock*). Dessa forma, considerando que a arquitetura utiliza filtros com 4 estágios de *pipeline*, estes resultados estão disponíveis após 100 ciclos. Após esse tempo de latência, a cada 51 ciclos serão disponibilizados os valores do melhor SAD e do vetor de movimento de acordo com a FME para um bloco de tamanho 8x8. O sincronismo entre as arquiteturas de interpolação e de busca ocorre perfeitamente, pois o preenchimento do *pipeline* dos módulos da etapa de busca e a atualização dos valores do *buffer* para amostras do bloco atual ocorrem de forma a não afetar o total de ciclos necessários na interpolação, uma vez que a quantidade total de ciclos necessários na etapa de busca é menor. Portanto, a arquitetura completa atinge um número de ciclos para o seu funcionamento, compatível com a etapa de interpolação.

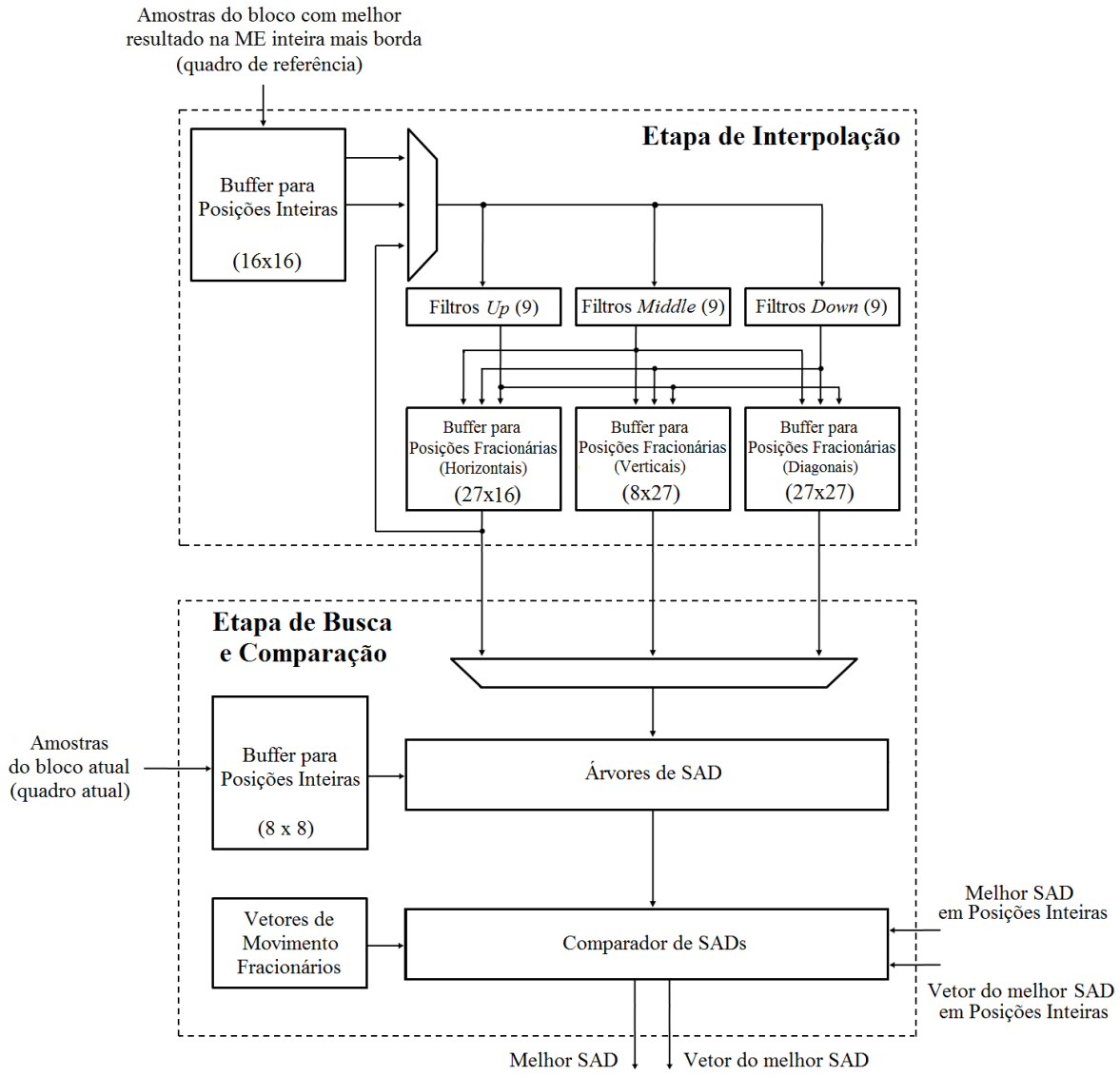


Figura 5.14 – Arquitetura completa para FME segundo o padrão HEVC.

6 RESULTADOS E DISCUSSÕES

Neste capítulo são apresentados os resultados obtidos com a arquitetura de hardware desenvolvida, bem como o processo de validação realizado com os dados de saída obtidos.

6.1 Resultados de Síntese

Os resultados obtidos com a síntese dos circuitos descritos consideram, primeiramente, os módulos principais que compõe a arquitetura. No caso dos filtros de interpolação, diferentes versões foram implementadas. As arquiteturas de hardware desenvolvidas para cada módulo, bem como para a FME completa, foram descritas em VHDL através da ferramenta Quartus II da Altera (Altera, 2012). Todos os resultados foram obtidos considerando o dispositivo FPGA Stratix III, modelo EP3SE50F484C2. Este dispositivo foi escolhido por se tratar de um FPGA amplamente utilizado na literatura científica e por ter uma quantidade de recursos de hardware disponível que é compatível com a quantidade utilizada pela arquitetura completa para a FME.

6.1.1 Filtros de Interpolação

Os resultados obtidos para os filtros de interpolação, através de síntese com a ferramenta Quartus II, podem ser observados nas tabelas 6.1 e 6.2. A Tabela 6.1 apresenta os resultados referentes às versões do filtro tipo *Up/Down*, enquanto que os resultados das versões do filtro tipo *Middle* podem ser vistos na Tabela 6.2. Com o objetivo de verificar os ganhos obtidos com as otimizações propostas, foram descritas versões diretamente no ambiente do Quartus, utilizando macrofunções

fornecidas pelo software. Estas versões para comparação foram descritas para cada tipo dos filtros, apenas na versão combinacional.

Tabela 6.1 – Resultados obtidos com as versões do filtro tipo *Up/Down*.

Arquitetura do Filtro *Up/Down*	**ALUTs Combinacionais**	**Total de registradores**	**Frequência Máxima (MHz)**
Versão do Quartus	144	0	166,42
Combinacional	163	0	186,85
Pipeline e 2 estágios	162	48	308,83
Pipeline e 4 estágios	162	157	502,77
Dispositivo FPGA Altera Stratix III EP3SE50F484C2			

Tabela 6.2 – Resultados obtidos com as versões do filtro tipo *Middle*.

Arquitetura do Filtro *Middle*	**ALUTs Combinacionais**	**Total de registradores**	**Frequência Máxima (MHz)**
Versão do Quartus	160	0	135,12
Combinacional	161	0	174,22
Pipeline e 2 estágios	160	46	261,99
Pipeline e 4 estágios	160	115	448,23
Dispositivo FPGA Altera Stratix III EP3SE50F484C2			

Os resultados apresentados indicam um baixo uso de recursos de hardware para todas as implementações. É importante destacar que a versão gerada diretamente pelo Quartus usa um número menor de ALUTs, em função das otimizações internas da ferramenta de síntese. Por outro lado, os ganhos em frequência de operação ficam claros com o uso das otimizações realizadas, sendo superiores a 39MHz no filtro tipo *Middle*. O uso de *pipeline*, como esperado, ampliou muito a frequência de operação, com impacto no número de registradores utilizados.

6.1.2 Demais Módulos da Parte Operativa

Os resultados de síntese obtidos para as árvores de SAD podem ser observados na Tabela 6.3. Como o *pipeline* usado nas árvores de SAD foi bem

balanceado, a frequência atingida foi elevada, mas o custo em termos do uso de registradores também foi significativo. O custo de ALUTs é função do número de somadores presentes nas árvores de SAD.

Os resultados obtidos para o comparador de SADs também podem ser observados na Tabela 6.3. Novamente, a frequência de operação atingida foi elevada, em função do *pipeline* usado.

A Tabela 6.3 também apresenta os resultados obtidos para a unidade de controle. As ALUTs e os registradores utilizados foram necessários para implementar a máquina de estados finitos projetada para o controle. Como não há grande complexidade nestas operações, a frequência máxima atingida chegou a 500 MHz.

Tabela 6.3 – Resultados obtidos com diferentes módulos da parte operativa.

Arquitetura	**ALUTs Combinacionais**	**Total de registradores**	**Frequência Máxima (MHz)**
Árvores de SAD	2.604	1.740	659,63
Comparador de SADs	248	254	445,63
Unidade de controle	162	79	500,00
Dispositivo FPGA Altera Stratix III EP3SE50F484C2			

6.1.3 Resultados Gerais da Arquitetura

Na Tabela 6.4 se pode observar os resultados de hardware e frequência de operação para a arquitetura da FME completa, utilizando as versões de filtros de interpolação com 4 estágios de *pipeline*. A frequência de operação chegou a 320,82MHz, sendo bastante inferior àquelas frequências obtidas nos módulos individualmente. Essa queda na frequência era esperada, em função do tamanho final do hardware gerado. O uso de recursos de hardware foi elevado, tanto em termos de ALUTs, quanto em termos de uso de registradores, utilizando cerca da metade dos recursos de hardware do dispositivo FPGA. Este custo é função do elevado grau de paralelismo empregado no projeto da arquitetura desenvolvida, já que o objetivo era o processamento de vídeos de alta resolução em tempo real.

Tabela 6.4 – Resultados obtidos com a arquitetura completa da FME.

ALUTs Combinacionais	Total de registradores	Frequência Máxima (MHz)
14.527 (38,23% FPGA)	22.199 (58,42%FPGA)	320,82
Dispositivo FPGA Altera Stratix III EP3SE50F484C2		

Considerando que é necessário o processamento de pelo menos 30 quadros por segundo para se obter a sensação de movimento contínuo, vídeos *Full* HD (1920x1080 pixels) necessitam de uma taxa de processamento de 62,208 milhões de amostras de luminância por segundo. Como na especificação do projeto foram considerados blocos de tamanho fixo 8x8, devem ser processados 972 mil blocos por segundo. Sabendo que a arquitetura desenvolvida necessita de 51 ciclos para processar por completo um bloco 8x8, após a latência, a frequência mínima para atingir 30 quadros por segundo é de 49,572 MHz. Se fossem usados 60 quadros por segundo, a frequência mínima da arquitetura subiria para 99,144MHz. Então, para os dois casos, a arquitetura desenvolvida apresenta frequência suficiente para atingir tempo real. Com a frequência máxima, a arquitetura da FME é capaz de processar 194 quadros *Full* HD (1920x1080 pixels) por segundo.

Para vídeos QFHD (3840x2160 pixels) é necessário uma taxa de processamento de 248,832 milhões de amostras de luminância por segundo, ou 3,888 milhões de blocos 8x8 por segundo. Então, a frequência mínima para atingir 30 quadros por segundo é de 198,288 MHz. Dessa forma, a arquitetura desenvolvida apresenta frequência suficiente para atingir tempo real para vídeos QFHD (3840x2160 pixels) a 30 quadros por segundo. Com a frequência máxima, a arquitetura da FME é capaz de processar 48 quadros QFHD (3840x2160 pixels) por segundo.

6.2 Validação

O processo de validação da arquitetura de hardware desenvolvida foi feito utilizando-se a ferramenta ModelSim (ModelSim, 2012). A partir do software de referência do padrão HEVC na versão *HM-8.0rc2* (HEVC Reference Software, 2012),

e também do ModelSim, foram gerados diferentes arquivos de saída para os módulos que compõem a FME e para a arquitetura completa, considerando uma mesma entrada no processo de codificação. Foi desenvolvido um programa para comparar os arquivos de texto gerados pelo software de referência e pelo ModelSim. O programa desenvolvido é capaz de detectar qualquer diferença entre os resultados obtidos com a arquitetura e com o software de referência, permitindo a validação do hardware proposto. Para esta validação foi utilizada a sequência *BasketballPass* na geração dos valores e mais de 1000 blocos foram comparados.

7 CONCLUSÕES

Este trabalho apresentou o desenvolvimento em hardware de uma arquitetura para FME do padrão HEVC. Um estudo detalhado dos *drafts* e do software de referência do padrão HEVC também foi realizado, permitindo um embasamento teórico fundamental para o desenvolvimento de arquiteturas de hardware eficientes.

Como o padrão HEVC ainda se encontra em desenvolvimento, foram encontrados poucos trabalhos na literatura científica relacionados com as arquiteturas de FME desenvolvidas que permitissem comparações justas. Contudo, baseado também em um estudo sobre o estado da arte das arquiteturas de FME para o padrão H.264/AVC, foi possível analisar o impacto da utilização de algumas estratégias durante a especificação da arquitetura.

Uma série de avaliações foi realizada utilizando o software de referência do HEVC, as quais permitiram saber, entre outras informações, o impacto da ME e da FME nos resultados de compressão obtidos pelo padrão, os tamanhos de blocos PU mais selecionados durante o processo de codificação e as implicações na qualidade da imagem e no *bit-rate*, fixando o tamanho dos blocos. Estas avaliações foram fundamentais para o desenvolvimento de uma arquitetura eficiente para a FME segundo o padrão HEVC.

Os resultados de síntese da arquitetura completa indicam que a frequência de operação atingida chega a 320MHz, permitindo o processamento de 194 quadros *Full* HD (1920x1080 pixels) ou 48 quadros QFHD (3840x2160 pixels) por segundo. Com o paralelismo aplicado à arquitetura, com cerca de 50 MHz é possível atingir 30 quadros por segundo para vídeos *Full* HD e com 100MHz é possível atingir até 60 quadros por segundo para esta resolução. Assim, é possível perceber a elevada taxa de processamento da arquitetura desenvolvida, onde com frequências baixas é possível atingir tempo real mesmo para resoluções elevadas.

Como trabalhos futuros, pretende-se realizar o desenvolvimento de uma arquitetura para FME do padrão HEVC com suporte a vários tamanhos de blocos, incluindo blocos de dimensões maiores que a utilizada neste trabalho.

REFERÊNCIAS

AGOSTINI, L. **Desenvolvimento de Arquiteturas de Alto Desempenho Dedicadas a Compressão de Vídeo Segundo o Padrão H.264/AVC.** 2007. 172f. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

ALSHINA, E., ALSHIN, A. e PARK, JH. **CE3: 7 Taps Interpolation Filters for Quarter Pel Position MC from Samsung and Motorola Mobility (JCTVC-G778)**, Geneva, 2011.

ALTERA. FPGA CPLD and ASIC from Altera. **Altera Web Site**. Disponivel em: <www.altera.com> Acesso em: Jun. 2012.

BHASKARAN, V. e KONSTANTINIDES, K. **Image and Video Compression Standards**: Algorithms and Architectures. 2nd ed. Boston: Kluwer Academic Publishers, 1999.

BOSSEN, F. **Common Test Conditions and Software Reference Configurations (JCTVC-K1100)**, Shangai, 2012.

BROSS, B., HAN, W., OHM, J., SULLIVAN, G. e WIEGAND, T. **High Efficiency Video Coding text specification draft 9 (JCTVC-K1003)**, Shangai, 2012.

CARVALHO, M. G. **Implementação Hardware/Software da Estimação de Movimento Segundo o Padrão H.264.** 2007. 102f. Dissertação (Mestrado em Sistemas e Computação) – Departamento de Informática e Matemática Aplicada, UFRN, Natal.

CORRÊA, M. M. e SCHOENKNECHT, M. T. A H.264/AVC quarter-pixel motion estimation refinement architecture targeting high resolution videos. In: VII Southern Conference on Programmable Logic, SPL**,** 2011. **Proceedings...** Cordoba: IEEE, 2011a. p. 131-136.

CORRÊA, M. M., SCHOENKNECHT, M. T. e DORNELLES, R. S. A high-throughput hardware architecture for the H.264/AVC half-pixel motion estimation targeting high-definition videos. **International Journal of Reconfigurable Computing**, [S.l.], v. 2011, 2011b.

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

GONZALEZ, R. e WOODS, R. **Processamento de Imagens Digitais.** São Paulo: Edgard Blücher, 2003.

GUO Z., ZHOU D. e GOTO S. An optimized MC interpolation architecture for HEVC. **IEEE International Conference on Acoustics, Speech and Signal Processing**, Mar. 2012.

HEVC Reference Software. Disponível em: <https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/> Acesso em: Jun. 2012.

ITU-T; ISO/IEC. **Joint Call for Proposals on Video Compression Technology (VCEG-AM91)**, Kyoto, 2010.

JCT-VC. **Encoder-side Description of Test Model under Consideration (JCTVC-B204)**, Geneva, 2010.

JCT-VC Document Management System. Disponível em: <http://phenix.int-evry.fr/jct/> Acesso em: Jun. 2012.

KAO, C., KUO, H. e LIN, Y. High performance fractional motion estimation and mode decision for the H.264/AVC. In: IEEE International Conference on Multimedia and Expo, ICME, 2006. **Proceedings...** [S.l.]: IEEE, 2006. p. 1241-1244.

KIM, I., McCANN, K., SUGIMOTO, K., BROSS, B. e HAN, W. **HM9: High Efficiency Video Coding Test Model 9 Encoder Description (JCTVC-K1002)**, Shangai, 2012.

KUHN, P. **Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation**. Boston: Kluwer Academic Publishers, 1999.

KUO, C. et al. A Novel Prediction-Based Directional Asymmetric Search Algorithm for Fast Block-Matching Motion Estimation. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 19, n. 6, p. 893-899, Jun. 2009.

LAI, Y., CHEN, L. e HUANG, S. Hybrid Parallel Motion Estimation Architecture Based on Fast Top-Winners Search Algorithm. **IEEE Transactions on Consumer Electronics**, [S.l.], v. 56, n. 3, p. 1837-1842, Ago. 2010.

LI, B., SULLIVAN, G. e XU, J. **Comparison of Compression Performance of HEVC Draft 8 with AVC High Profile (JCTVC-K0279)**, Shangai, 2012.

LIN, C. e LEOU, J. An Adaptative Fast Full Search Motion Estimation Algorithm for H.264. In: IEEE INTERNATIONAL SYMPOSIUM CIRCUITS AND SYSTEMS, ISCAS, 2005. **Proceedings…** [S.l.]: IEEE, 2005. p. 1493-1496.

ModelSim. Advanced Simulation and Debugging. **ModelSim Web Site**. Disponível em: <www.model.com> Acesso em: Jun. 2012.

OKTEM, S., e HAMZAOGLU, I. An efficient hardware architecture for quarter-pixel accurate H.264 motion estimation. In: 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools, DSD '07**,** 2007. **Proceedings...** Lubeck: IEEE, 2007. p. 444–447.

PORTO, M. **Desenvolvimento Algorítmico e Arquitetural para a Estimação de Movimento na Compressão de Vídeo de Alta Definição.** 2012. 166f. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

PURI, A. et al. Video Coding Using the H.264/MPEG-4 AVC Compression Standard. **Elsevier Signal Processing: Image Communication.** [S.l.], n. 19, p.793–849, 2004.

RICHARDSON, I. **Video Codec Design: Developing Image and Video Compression Systems.** Chichester: John Wiley and Sons, 2002.

RICHARDSON, I. **H.264 and MPEG-4 Video Compression: Video Coding for Next-Generation Multimedia.** Chichester: John Wiley and Sons, 2003.

Subversion. Apache Subversion. **Subversion Web Site**. Disponível em: < http://subversion.apache.org/> Acesso em: Jun. 2012.

SULLIVAN, G. e WIEGAND, T. **Draft requirements for next-generation video coding project (VCEG-AL96)**, [S.l.], 2009.

TOURAPIS, A. **Enhanced Predictive Zonal Search for Single and Multiple Frame Motion Estimation**. Visual Communications and Image Processing 2002, San Jose, CA, Jan. 2002.

YALCIN, S. e HAMZAOGLU, I. A high performance hardware architecture for half-pixel accurate H.264 motion estimation. In: 14th International Conference on Very Large Scale Integration and System-on-Chip, VLSI-SoC '06, 2006. **Proceedings...** Nice: IEEE, 2006. p. 63–67.

YI, X. e LING, N. Rapid Block-matching motion estimation using modified diamond search algorithm. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2005. **Proceedings…** [S.l.]: IEEE, 2005. p. 5489 – 5492.

ZHU, S. e MA, K. A New Diamond Search Algorithm for Fast Block-Matching Motion Estimation. **IEEE Transactions on Image Processing**, [S.l.], v. 9, n. 2, p. 287-290, Fev. 2000.

APÊNDICE

Lista de Publicações

A) Trabalhos completos publicados em anais de congressos:

Evento: LASCAS 2013 – *4th IEEE Latin American Symposium on Circuits and Systems*
Título: *Low Cost and High Throughput FME Interpolation for the HEVC Emerging Video Coding Standard.*
Autores: Vladimir Afonso, Henrique Maich, Luciano Agostini e Denis Franco.
Qualis: B5

Evento: SIM 2012 – *27th South Symposium on Microelectronics*
Título: *A Dedicated Hardware Solution for the HEVC Interpolation Unit.*
Autores: Vladimir Afonso, Marcel Corrêa, Luciano Agostini e Denis Franco.

B) Resumos publicados em anais de congressos:

Evento: DCC 2013 – *Data Compression Conference*
Título: *Simplified HEVC FME Interpolation Unit Targeting a Low Cost and High Throughput Hardware Design.*
Autores: Vladimir Afonso, Henrique Maich, Luciano Agostini e Denis Franco.
Qualis: A2

Evento: SPC 2012 – III Seminário de Pesquisa em Computação da UFPel
Título: Desenvolvimento de Arquitetura para Interpolação das Amostras de Luminância Segundo o Padrão HEVC.
Autores: Vladimir Afonso, Luciano Agostini e Denis Franco.

Evento: SPC 2012 – III Seminário de Pesquisa em Computação da UFPel
Título: Análise do Impacto da Predição Inter e da Estimação de Movimento Fracionária na Codificação de Vídeos com o Padrão HEVC.
Autores: Henrique Maich, Vladimir Afonso, Marcelo Porto, Denis Franco e Luciano Agostini.

Evento: ENPOS 2012 – XIV Encontro de Pós-Graduação da UFPel
Título: Projeto de Hardware Com Elevada Taxa de Processamento Para Estimação de Movimento Fracionária Segundo o Padrão HEVC.
Autores: Vladimir Afonso, Luciano Agostini e Denis Franco.
Observação: Menção Honrosa na área de Ciências Exatas e da Terra, na modalidade Pôster.