Universidade Federal de Pelotas

Trabalho de Conclusão de Curso

Marcelo Schiavon Porto

INVESTIGAÇÃO DE ALGORITMOS E DESENVOLVIMENTO ARQUITETURAL PARA A ESTIMAÇÃO DE MOVIMENTO EM COMPRESSÃO DE VÍDEO DIGITAL

Pelotas, 2006

MARCELO SCHIAVON PORTO

INVESTIGAÇÃO DE ALGORITMOS E DESENVOLVIMENTO ARQUITETURAL PARA A ESTIMAÇÃO DE MOVIMENTO EM COMPRESSÃO DE VÍDEO DIGITAL

Trabalho acadêmico apresentado ao curso de Bacharelado em Ciência da Computação do Instituto de Física e Matemática, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. MSc. Luciano Volcan Agostini

Co-orientador: Prof. PhD. Sergio Bampi (UFRGS)

Pelotas, 2006

Dados de catalogação na fonte:
Ubirajara Buddin Cruz – CRB-10/901
Biblioteca de Ciência & Tecnologia - UFPel

P853i Porto, Marcelo Schiavon

Investigação de algoritmos e desenvolvimento arquitetural para a estimação de movimento em compressão de vídeo digital / Marcelo Schiavon Porto ; orientador Luciano Volcan Agostini ; co-orientador Sergio Bampi. – Pelotas, 2006. – 69f. : il. – Monografia (Conclusão de curso). Curso de Bacharelado em Ciência da Computação. Departamento de Informática. Instituto de Física e Matemática. Universidade Federal de Pelotas. Pelotas, 2006.

1. Informática. 2.Compressão de vídeo. 3.Estimação de movimento. 4.Descrição VHDL. I. Agostini, Luciano Volcan. II.Bampi, sergio. III.Título.

CDD: 006.7

Investigação de Algoritmos e Desenvolvimento Arquitetural para a Estimação de Movimento em Compressão de Vídeo Digital

Marcelo Schiavon Porto

Monografia defendida e aprovada em 06 de março de 2006, pela banca examinadora constituída pelos seguintes professores:

Prof. MSc. Luciano Volcan Agostini – Orientador
Universidade Federal de Pelotas – UFPel

Prof. Dr. José Luis Almada Güntzel
Universidade Federal de Pelotas – UFPel

Prof. Dr. Altamiro Amadeu Susin
Universidade Federal do Rio Grande do Sul – UFRGS

MSc. Vagner Santos da Rosa
Universidade Federal do Rio Grande do Sul – UFRGS

Agradecimentos

Aos meus pais, Roberto Garcia Porto e Ivone Schiavon Porto, pela educação que me deram, pelo apoio nas etapas difíceis da vida e pela confiança incondicional que sempre depositaram em mim.

Ao meu irmão, Matheus Schiavon Porto, por ter sido um amigo, e companhia durante muitas horas de trabalho em frente ao computador.

A minha tia, Maria Helena Schiavon, por estar sempre disposta a ajudar, e por me acolher como um filho durante toda a vida, sendo uma segunda mãe para mim.

Ao meu orientador, Luciano Agostini, pelo incentivo, pelo apoio constante e por confiar na minha capacidade de trabalho, até mesmo quando eu não acreditava. Por ser, além de um orientador, um grande amigo que adquiri durante estes anos de faculdade, que me fez crescer, não só como profissional, mas também como pessoa.

Ao professor, José Luís Güntzel, pelo apoio em diversos momentos, pelo esforço em manter o grupo de pesquisa e pela dedicação empregada aos membros do grupo.

A todos os meus colegas e amigos que, de uma forma ou de outra, contribuíram nesta longa jornada.

Resumo

PORTO, Marcelo S. **Algoritmos e Arquiteturas Para a Estimação de Movimento em Compressão de Vídeo Digital**. 2006. 62f. Monografia – Curso de Bacharelado em Ciência da Computação, Universidade Federal de Pelotas, Pelotas.

A compressão de vídeo é extremamente importante para a manipulação de vídeos digitais. Sem a compressão fica praticamente impossível enviar ou armazenar vídeos digitais devido a sua grande quantidade de informações. A etapa dos compressores de vídeo atuais que proporciona a maior parte dos ganhos em termos de compressão é a estimação de movimento. A estimação de movimento busca a relação de similaridade entre os quadros vizinhos de uma cena, reduzindo consideravelmente a redundância temporal existente entre as imagens. No entanto, o processo de estimação é uma tarefa computacionalmente complexa, que pode tornar inviáveis aplicações em software para vídeos de alta resolução.

Este trabalho tem como objetivo a investigação o de alguns dos algoritmos de estimação de movimento existentes e o desenvolvimento de uma arquitetura de hardware para implementar o processo estimação. Os algoritmos investigados foram implementados em linguagem C e seus resultados serviram de base para a escolha do algoritmo a ser implementado na arquitetura. A arquitetura desenvolvida trabalha com o algoritmo *Pel Decimation* 4:1 e opera sobre blocos de 16x16, em uma área de pesquisa de 64x64 amostras. A arquitetura foi desenvolvida em VHDL e direcionada para FPGAs de diferentes famílias. Os resultados de síntese obtidos para o dispositivo Virtex-II Pro XC2VP70 da Xilinx indicam que a arquitetura pode operar em tempo real para vídeos SDTV, utilizando um total de 28% dos recursos de hardware do dispositivo.

Palavras-chave: Compressão de vídeo. Estimação de movimento. Descrição VHDL.

Abstract

PORTO, Marcelo S. **Algoritmos e Arquiteturas Para a Estimação de Movimento em Compressão de Vídeo Digital**. 2006. 62f. Monografia – Curso de Bacharelado em Ciência da Computação, Universidade Federal de Pelotas, Pelotas.

The vídeo compression is extremely important for the handling of digital videos. Without the compression is practically impossible to send or to store digital videos due theirs high amount of information. The stage of the current video compressors that provides most of the profits in terms of compression is the motion estimation. The motion estimation searches the similarity relationship among the neighboring frames of a scene, reducing the temporal redundancy among the images. However, the estimation process is a computational complex task, that can lead to impracticable software applications for high resolution videos.

The goal of this work is the investigation of some of the algorithms for motion estimation and the design of a hardware architecture to implement the motion estimation process. The investigated algorithms had been implemented in C language and its results had been used to define the algorithm that was designed in hardware. The designed architecture works with the Pel Decimation 4:1 algorithm and operates with blocks of 16x16, in a search area of 64x64 samples. The architecture was described in VHDL and its synthesis was targeted to FPGAs of different families. The synthesis results for the Xilinx Virtex-II Pro XC2VP70 device indicate that the architecture can operate in real time for SDTV videos, using a total of 28% of the available device hardware resources.

Key-words: Video compression. Motion estimation. VHDL design.

Lista de Ilustrações

Lista de Tabelas

Lista de Abreviaturas e Siglas

ASIC	*Application-specific Integrated Circuit*
CIF	*Common Intermediate Format*
CODEC	codificador/decodificador
DVD	*Digital Versatile Disk*
FPGA	*Field Programmable Gate Array*
HDTV	*High Definition Digital Television*
HSI	*Hue, Saturation, Intensity*
LDSP	*Large Diamond Search Pattern*
MAE	*Minimum Absolute Error*
MC	Motion Estimation
ME	*Motion Estimation*
MSB	*Most Significant Bit*
MSE	*Minimum Square Error*
QCIF	*Quarter Common Intermediate Format*
RAM	*Random Access Memory*
RFP11	Requisição Formal de Proposta 11
RGB	*Red, Green, Blue*
RLB	Registrador de Linha de Bloco
RLP	Registrador de Linha de Pesquisa
RRM	Registrador de resultado de Memória
SAD	*Sum of Absolute Differences*

SB	Seletor de Linha de Bloco
SBTVD	Sistema Brasileiro de Televisão Digital
SDSP	*Small Diamond Search Pattern*
SDTV	*Standart Definition Television*
SP	Seletor de Linha de Pesquisa
UP	Unidade de Processamento
UFRN	Universidade Federal do Rio Grande do Norte
VGA	*Video Graphics Adapter*
VHDL	VHSIC *Hardware Description Language*
YCbCr	*Luminance, Chrominance Blue, Chrominance Red*

Sumário

1. Introdução

A compressão de vídeos digitais é um tema de extrema importância na atualidade, principalmente pelas altas taxas de compressão proporcionadas pelos padrões de compressão de vídeos atuais. Esta compressão significativa na quantidade de informações contidas nos vídeos digitais possibilitou a sua aplicação em diversos dispositivos com celulares, DVD *players*, televisão digital de alta definição (HDTV), entre outros.

Os padrões de compressão de vídeo são formados por um conjunto de algoritmos e técnicas que tentam reduzir o número de informações necessárias para representar um vídeo. Dentro dos compressores de vídeo digital atuais existem diversas etapas as quais as informações que compões o vídeo são submetidas, dentre elas as principais são: Estimação de movimento, Compensação de movimento, Transformadas, Quantização e Codificação de entropia. Dentre as etapas dos compressores atuais, a mais complexa e que resulta nos melhores resultados de taxa de compressão é a estimação de movimento (ME – *Motion Estimation*). O ME é responsável por encontrar uma co-relação entre os *frames*, descobrindo a redundância temporal entre os *frames* vizinhos de uma cena.

Este trabalho apresenta uma investigação sobre alguns dos algoritmos de estimação de movimento mais conhecidos, com o intuito de desenvolver uma arquitetura em hardware para a estimação de movimento. Na primeira etapa deste trabalho, alguns dos algoritmos estudados são implementados em software. Seus resultados são apresentados e serviram de base para a escolha do algoritmo que foi utilizado na segunda etapa. A segunda etapa deste trabalho é a proposta arquitetural em hardware para a estimação de movimento. A solução arquitetural proposta neste trabalho foi fortemente baseada em uma solução desenvolvida na UFRN. A arquitetura foi descrita em VHDL e direcionada para FPGAs. A proposta

arquitetural deste trabalho utiliza o algoritmo de busca *Pel Decimation* 4:1 e o SAD como critério de distância. A arquitetura trabalha com blocos de 16x16 em uma área de pesquisa de 64x64 amostras. O trabalho original, da UFRN, utiliza o algoritmo de busca *Full Search* sobre uma área de busca de 32x32 amostras. A solução da UFRN também trabalha com o SAD como critério de distância e também utiliza blocos com tamanho de 16x16. Os resultados de síntese da arquitetura desenvolvida são apresentados e indicam que a arquitetura possui um taxa de processamento suficiente para processar vídeos SDTV em tempo real.

O capítulo dois desta monografia apresenta alguns conceitos básicos de compressão de vídeos digitais que serão úteis para a compreensão dos demais capítulos deste trabalho. No capítulo três são apresentados alguns conceitos de estimação de movimento, bem como a explicação detalhada dos algoritmos e dos critérios de distorção estudados. O capítulo quatro apresenta os resultados das implementações em C para os algoritmos de estimação de movimento estudados. No capítulo cinco, será apresentada a arquitetura do estimador, detalhando cada um dos principais blocos que o compõem. Por fim, o capítulo 6 apresenta as conclusões e propostas de trabalhos futuros.

2. Conceitos Básicos de Compressão de Vídeo Digital

Para contextualizar a estimação de movimento na compressão de vídeo, este capítulo apresentará, superficialmente, alguns conceitos básicos da compressão de vídeo. As explicações contidas neste capítulo serão importantes para a compreensão dos demais capítulos apresentados nesta monografia.

2.1 Conceitos Básicos de Vídeo Digital

Um vídeo digital é formado por imagens, que por sua vez são formadas por pontos (*pixels*). Uma cena de vídeo é formada por uma seqüência de imagens. Pontos pertencentes à mesma imagem e, espacialmente vizinhos, são chamados de pontos vizinhos. Imagens, também chamadas de quadros (*frames*), pertencentes à mesma cena e temporalmente próximas são chamadas de imagens vizinhas. Pontos e imagens vizinhas são geralmente muito similares e esta é uma característica importante de imagens e vídeos digitais. Esta similaridade resulta em grande redundância de informações na sua representação.

Todo vídeo digital é formado por uma seqüência de quadros (*frames*), que por sua vez, são divididos em blocos. Estes blocos podem ter tamanhos variados, mas comumente são utilizados tamanhos de 4x4, 8x8 ou 16x16 *pixels*. A fig. 2.1 ilustra uma seqüência de quadros de um vídeo digital, salientando a divisão dos blocos do *frame*.

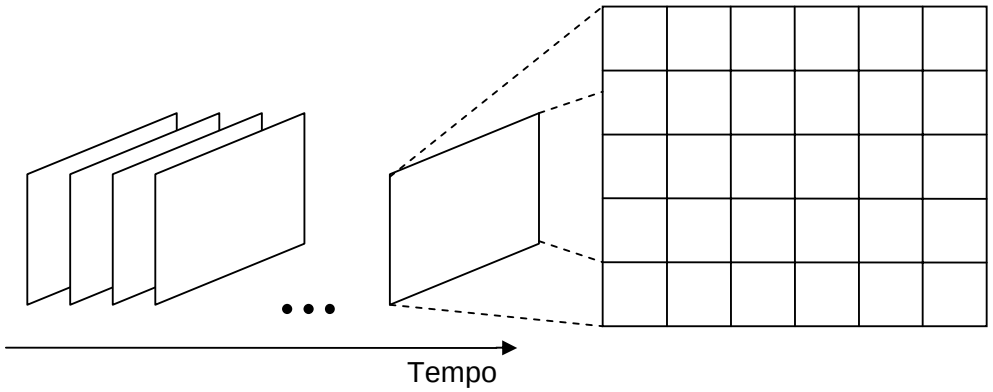


Figura 2.1 – Seqüência de *frames* em um vídeo digital.

2.2 Redundância de Informação na representação de vídeos digitais

A compressão é baseada na eliminação de dados redundantes existentes em vídeos e imagens. Um dado é considerado redundante quando seu valor não representa uma nova informação relevante para a representação da imagem. Basicamente, existem quatro tipos diferentes de redundâncias exploradas na compressão de vídeos: redundância espacial, redundância temporal, redundância psicovisual e redundância entrópica. Existem controvérsias entre os autores a respeito da definição dos tipos de redundância. A classificação apresentada neste trabalho, com quatro diferentes tipos de redundância, é baseada em (SHI, 1999) e (GONZALEZ, 2003).

A redundância espacial, também chamada de “redundância intraframe” (GHANBARI, 2003), é resultado da correlação existente entre os pixels vizinhos do mesmo quadro. *Pixels* vizinhos tendem a possuir valores semelhantes e, por conseqüência, com elevado grau de redundância de informação, gerando pouco acréscimo de informação visual a cada ponto em relação aos vizinhos.

As características do sistema visual humano fazem com que não consigamos captar alguns tipos de informações que podem estar presentes na imagem. Além disso, algumas informações da imagem, como o brilho, por exemplo, são mais importantes para o sistema visual humano do que outras, como as cores. Este tipo de redundância de informação é chamado de redundância psicovisual (GONZALEZ, 2003). Para explorar este tipo de redundância, parte da informação original da imagem é eliminada de forma irreversível pelo codificador. Existem dois tipos principais de processos utilizados para este fim. O primeiro, chamado de “sub-

amostragem” é utilizado na entrada do vídeo e elimina parte das informações de cores. O segundo, conhecido por “quantização”, normalmente é aplicado no domínio das freqüências e elimina ou atenua as freqüências de menor percepção para o sistema visual humano. Por isso, este tipo de processo de codificação é chamado de “codificação com perdas”. É importante destacar que a eliminação destas informações contribui para que o codificador atinja elevadas taxas de compressão, com pequeno impacto na qualidade visual da imagem que, eventualmente, pode mesmo ser nulo.

A redundância entrópica está relacionada com a forma de representação computacional dos símbolos codificados e não se relaciona diretamente ao conteúdo da imagem. A entropia é uma medida da quantidade média de informação transmitida por símbolo do vídeo (SHI, 1999). A quantidade de informação nova transmitida por um símbolo diminui na medida em que a probabilidade de ocorrência deste símbolo aumenta. Então, os codificadores que exploram a redundância entrópica têm por objetivo transmitir o máximo de informação possível por símbolo codificado e, deste modo, representar mais informações com um número menor de símbolos. A “codificação de entropia”, como é chamada, utiliza diferentes técnicas e algoritmos de compressão sem perdas para atingir este objetivo.

A redundância temporal, também chamada de “redundância interframe” (GHANBARI, 2003), é causada pela correlação existente entre quadros vizinhos. Na verdade, a redundância temporal poderia ser classificada como apenas mais uma dimensão da redundância espacial, como faz (GONZALEZ, 2003). Muitos blocos de *pixels* simplesmente não mudam de valor de um quadro para outro em um vídeo, como por exemplo, em um fundo que não foi alterado de um quadro para outro. Outros pixels apresentam uma pequena variação de valores causada, por exemplo, por uma variação de iluminação. Também é possível que o bloco de pixels simplesmente tenha se deslocado de um quadro para o outro, como por exemplo, em um movimento de um objeto em uma cena. Estes fatores resultam em um pequeno acréscimo de informação visual de cada imagem com relação às imagens vizinhas. Todos os padrões de codificação de vídeo atuais visam eliminar ou diminuir a redundância temporal. A exploração eficiente da redundância temporal conduz a elevadas taxas de compressão e é fundamental para o sucesso dos codificadores.

2.3 Sub-Amostragem de Cores

A representação digital de um vídeo colorido está associada à interpretação das cores pelo sistema visual humano. Existem muitas formas de se representar as cores de forma digital. Um sistema para representar cores é chamado de espaço de cores e a definição do espaço de cor a ser utilizado para representar um vídeo é essencial para a eficiência da codificação deste vídeo. São vários os espaços de cores usados para representar imagens digitais, tais como: RGB, HSI e YCbCr (SHI, 1999). O espaço de cores RGB é um dos mais comuns, tendo em vista que é este o espaço de cores utilizado nos monitores coloridos. O RGB representa, em três matrizes distintas, as três cores primárias captadas pelo sistema visual humano: vermelho, verde e azul.

No espaço de cores YCbCr, as três componentes utilizadas são luminância (Y), que define a intensidade luminosa ou o brilho; crominância azul (Cb) e crominância vermelha (Cr) (MIANO, 1999). Os componentes R, G e B possuem um elevado grau de correlação, tornando difícil o processamento de cada uma das informações de cor de forma independente. Por isso, a compressão de vídeos é aplicada para espaços de cores do tipo luminância e crominância, como o YCbCr (RICHARDSON, 2002). Outra vantagem do espaço de cor YCbCr sobre o espaço RGB é que, no espaço YCbCr, a informação de cor está completamente separada da informação de brilho. Deste modo, estas informações podem ser tratadas de forma diferenciada pelos compressores de imagens estáticas e vídeos.

O sistema visual humano é menos sensível as informações de cor contidas nas imagens (GONZALEZ, 2003). Então os padrões de compressão de imagens estáticas e vídeos podem explorar esta característica psicovisual humana para aumentar a eficiência de codificação através da redução da taxa de amostragem dos componentes de crominância em relação aos componentes de luminância (RICHARDSON, 2002). Esta operação é chamada de sub-amostragem de cores e é realizada sob o espaço de cores YCbCr nos padrões de compressão de vídeos atuais.

Existem várias formas de relacionar os componentes de crominância com o componente de luminância para realizar a sub-amostragem. Os formatos mais comuns são o 4:4:4, o 4:2:2 e o 4:2:0. No formato 4:4:4, para cada quatro amostras de luminância (Y), existem quatro amostras de crominância azul (Cb) e quatro

amostras de crominância vermelha (Cr). Assim, a sub-amostragem não está sendo aplicada. No formato 4:2:2, para cada quatro informações de Y apenas duas informações de Cb e Cr são representadas, e para o formato 4:2:0, para cada quatro informações de Y apenas uma informação de Cb e uma de Cr são representadas.

A sub-amostragem de cor aumenta significativamente a eficiência da compressão, uma vez que parte da informação da imagem é simplesmente descartada, sem causar impacto visual perceptível. Considerando o formato 4:2:0 como exemplo, uma vez que cada componente de crominância possui exatamente um quarto das amostras presentes no componente de luminância, então um vídeo YCbCr no formato 4:2:0 irá utilizar exatamente a metade das amostras necessárias para um vídeo RGB ou YCbCr no formato 4:4:4. Isso implica em uma taxa de compressão de 50%, considerando apenas a sub-amostragem.

3. Estimação de Movimento

Quando analisamos um trecho de um vídeo, podemos perceber que as imagens vizinhas são muito similares. As diferenças, quando existem, são geradas em sua maioria por movimentos da câmera ou de objetos pertencentes à cena. Esta característica dos vídeos produz uma grande redundância temporal, ou seja, se comparados dois *frames* vizinhos pode-se perceber uma enorme quantidade de informações repetidas, causadas por regiões dentro do *frame* que não são alteradas de um *frame* para outro.

As figs. 3.1 e 3.2 representam, respectivamente, o primeiro e o segundo *frame* de uma seqüência de vídeo. Podemos observar que as diferenças entre os *frames* 1 e 2 são praticamente imperceptíveis. Grande parte da imagem permanece estática e não sofre alterações de um *frame* para outro. A fig. 3.3 ilustra as diferenças entre os dois *frames*. Esta imagem foi gerada a partir da subtração *pixel* a *pixel* entre os *frames* 1 e 2. Nas regiões onde a imagem não se altera o valor dos pixels também não se altera, o que resulta em um valor nulo após a subtração. Para as regiões que sofrem algum tipo de alteração, a diferença entre os pixels dos dois *frames* resulta em um valor não nulo. Para facilitar a percepção das diferenças de movimento de um *frame* para outro, o resultado da subtração foi somado à 128. Deste modo, os pontos em zero (pontos idênticos nos dois *frames*) passaram a possuir o valor 128, que é o cinza predominante na fig. 3.3.

O objetivo principal da estimação de movimento é reduzir esta redundância temporal entre imagens vizinhas. A técnica consiste em identificar estas redundâncias e mapear as diferenças através de vetores denominados de vetores de movimento. A partir de um ou mais *frames*, denominados de *frames* de referência, o *frame* atual, que é o próximo frame após o frame de referência, é estimado usando vetores de movimento obtidos a partir do(s) *frame(s)* de referência.

Para cada bloco do *frame* atual será gerado um vetor de movimento que corresponderá a posição onde este bloco obteve a melhor relação de similaridade no *frame* de referência.

Figura 3.1 – *Frame* 1.

Figura 3.2 – *Frame* 2.

Figura 3.3 – Resultado da subtração entre os *Frames* 1 e 2.

Uma área de pesquisa será determinada no *frame* de referência para cada bloco do *frame* atual. A área de pesquisa pode conter o *frame* inteiro mas, normalmente, esta área é definida como uma fração do *frame*, para diminuir a complexidade computacional da estimação de movimento. Um algoritmo de busca determina a maneira como esse bloco se desloca dentro desta área de pesquisa para que a melhor relação de similaridade (melhor *matching*) seja encontrada. A fig. 3.4 ilustra um *frame* de referência e um *frame* atual. No *frame* de referência está destacada a área de pesquisa e o bloco que está sendo pesquisado.

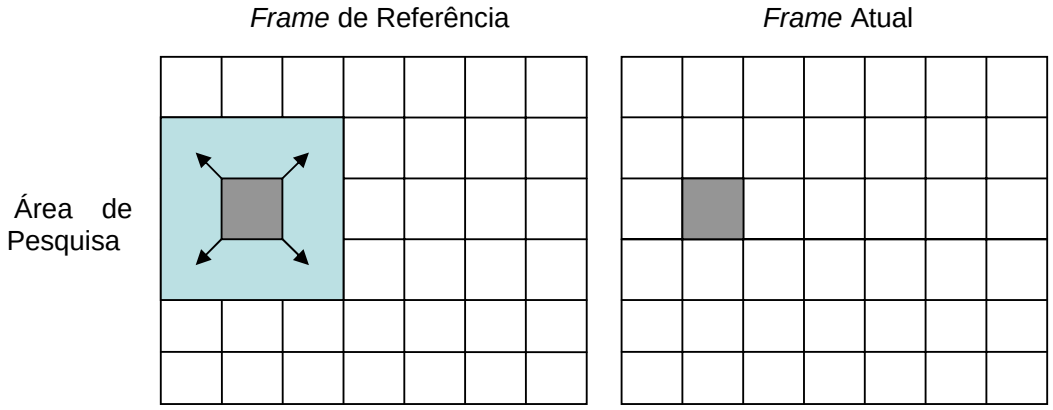


Figura 3.4 – Determinação do vetor de movimento para um bloco.

Esta informação do vetor de movimento é gerada para cada bloco de luninância do *frame* atual, já que as informações de crominância são irrelevantes para a estimação de movimento. Esta operação implica em uma grande complexidade computacional. No entanto, quando encontrado um bom casamento, proporciona uma redução significativa da quantidade de informação necessária para formar o vídeo, pois todas as informações contidas em um bloco são substituídas por um vetor de duas dimensões e mais um resíduo. O resíduo é a diferença *pixel* a *pixel* do bloco do *frame* atual e o bloco escolhido do *frame* de referência. O resíduo, além de possuir baixa amplitude, tendendo a zero, ainda é muito comprimido através de outras técnicas de compressão, como quantização e codificação de entropia. Por isso, a estimação de movimento (ME) facilita os demais processos do compressor de vídeo, pois menos informações devem ser processadas, tornando a ME uma ferramenta essencial em compressores de vídeo atuais. Dentro de um compressor existe um processo denominado compensação de movimento (MC). Este processo é

o responsável por remontar os *frames* a partir dos vetores de movimentos, gerados pela estimação de movimento, e a partir do *frame* de referência.

Alguns critérios devem ser cuidadosamente analisados para se obter o resultado esperado, tais como, o tamanho da área de pesquisa e dos blocos dos *frames*. O tamanho da área de pesquisa pode influenciar diretamente na qualidade dos vetores gerados e também na complexidade computacional do processo de estimação. Quanto maior a área de pesquisa, maior será a chance de se encontrar um bloco que possua o melhor casamento com relação ao bloco que esta sendo pesquisado. No entanto, a complexidade computacional também cresce, pois quantidade de comparações a serem feitas é bem maior. O tamanho dos blocos deve ser suficientemente pequeno para que áreas menores da imagem sejam comparadas, gerando erros menores. No entanto, deve ser grande o suficiente para que não exista um número muito grande de blocos em um *frame*, pois para cada bloco é necessário um vetor, que deve ser representado por um par de números inteiros. Estes vetores devem ser enviados junto com a informação comprimida e, se existe um número muito grande de vetores, o ganho obtido através dos erros menores é perdido em função da codificação do próprio vetor de movimento.

Pode-se concluir que diversos fatores devem ser observados para a implementação de um estimador de movimento. Fatores como a complexidade computacional e a qualidade e quantidade dos vetores são sempre contraditórios. Além disso, outro fator importante é a taxa de processamento, que para vídeos em tempo real deve estar entre 24 e 30fps (*frames* por segundo). Considerando estes fatores e as necessidades específicas que se deseja, sejam elas qualidade ou taxa de processamento, é que devemos determinar as características do estimador. A complexidade computacional, agregada ao processo de estimação, torna praticamente impossíveis aplicações de estimação de movimento em *software* para vídeos de alta definição, tornando indispensável à utilização de *hardware* dedicado.

3.1 Algoritmos de Busca para Estimação de Movimento

Os algoritmos de busca para a estimação de movimento determinam a forma como a busca dentro da área de pesquisa ocorrerá. O algoritmo tem influência direta na complexidade computacional da estimação, bem como na qualidade dos vetores gerados. Pode-se dividir os algoritmos de estimação em dois grandes grupos:

algoritmos ótimos e sub-ótimos. Algoritmos ótimos analisam todas as posições possíveis dentro da área de pesquisa para gerar o vetor que represente a menor diferença entre as regiões pesquisadas. Algoritmos sub-ótimos utilizam determinados métodos de busca que eliminam alguns cálculos, diminuindo a complexidade e gerando vetores que podem não ser os que resultam no menor erro.

Nesta seção serão apresentados alguns algoritmos de estimação como, busca completa (*Full Search*) (BHA 99) e (LIN, 2005), *Three Step Search* (TSS) (RICHARDSON, 2002), *Diamond Search* (KUHN, 1999), busca logarítmica (SHI, 1999), *One at a Time Search* (RICHARDSON, 2002) e *Pel Decimation* (KUHN, 1999).

3.1.1 Busca Completa (*Full Search*)

O algoritmo de busca completa procura a melhor relação entre as áreas de pesquisa (melhor *matching*) comparando o bloco que está sendo pesquisado com todas as posições possíveis dentro da área de pesquisa. O bloco é deslocado de um *pixel* dentro da área de pesquisa, começando do canto superior esquerdo, até que todas as posições tenham sido comparadas e a menor diferença tenha sido registrada. Ao final, será gerado um vetor de movimento referente ao deslocamento do bloco para a região de melhor *matching* (LIN, 2005). A fig. 3.5 ilustra o processo de busca completa.

A fig. 3.5 ilustra uma região de pesquisa de +/- 7 *pixels* em torno do bloco corrente (supondo que a posição original do bloco seja a posição central da área de pesquisa). Cada ponto na figura representa um vetor de movimento, sendo que o vetor que resultar no melhor casamento será o escolhido. Para cada ponto marcado na figura uma função de similaridade será aplicada para cada *pixel* do maclobloco, em comparação com o *pixel* da área de referência. Logo após, o algoritmo desloca o bloco atual um *pixel* dentro da área de pesquisa e re-calcula a diferença. Esta operação é repetida para todas as posições possíveis dentro da área de pesquisa.

Alguns critérios de parada podem ser adotados para acelerar o processo do algoritmo de busca completa como, por exemplo, determinar uma diferença mínima, quando o algoritmo encontrar um valor abaixo desse mínimo ele pára sua execução. Vale lembrar que todas as técnicas de redução do número de cálculos do algoritmo de busca completa podem implicar na geração de um vetor de movimento que pode não ser o vetor ótimo.

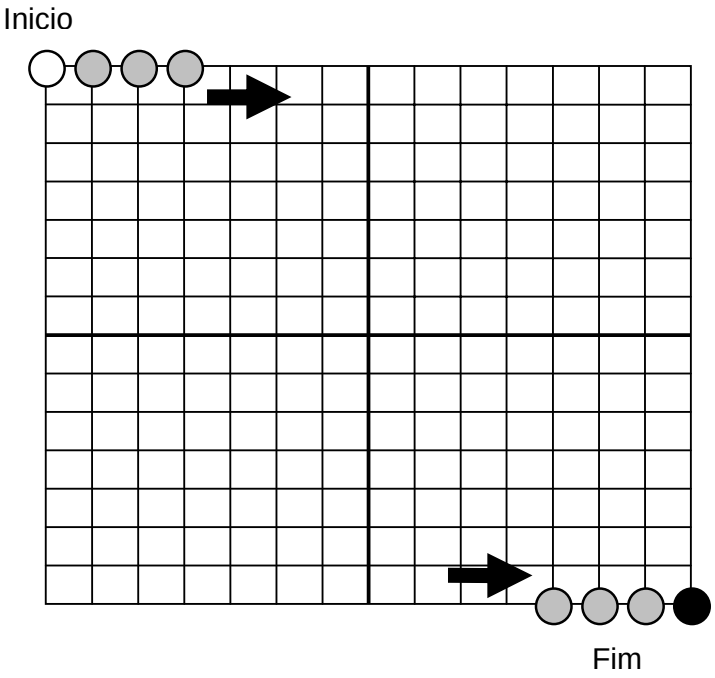


Figura 3.5 – Busca Completa.

A complexidade computacional do algoritmo de busca completa dificulta a sua aplicação em CODECs (codificadores/ decodificadores) que necessitam operar em tempo real para vídeos de alta resolução. Implementações em hardware podem minimizar a complexidade do algoritmo ao paralelizar as operações de comparação. Isto acelera o processo de estimação tornando viável a utilização deste algoritmo, que é uma ótima opção quando se deseja alta qualidade e taxa de compressão do vídeo comprimido.

Os próximos algoritmos que serão apresentados neste trabalho são considerados algoritmos rápidos (sub-ótimos), ideais para implementações de estimadores de movimento que operem em tempo real, principalmente em *software.* É importante salientar que estes algoritmos usam técnicas para diminuir o número de comparações dentro da área de pesquisa, o que pode desviar o rumo da pesquisa para uma região que não leve ao vetor ótimo. Mesmo assim, estes algoritmos podem encontrar regiões com erros muito próximos aos encontrados pelo algoritmo de busca completa, mesmo que gerando vetores de movimento bem diferentes.

3.1.2 *Three Step Search* (TSS)

Este algoritmo é mais difundido com o nome de busca em três passos. No entanto, ele pode ser estendido para *n* passos (RICHARDSON, 2002). Para uma área de pesquisa de +/- (2^n – 1) *pixel*s, o passo inicial S deve ser inicializada com S = 2^{n-1}.

O primeiro passo consiste em posicionar a busca no centro da área de pesquisa e calcular o erro para esta posição. Em seguida, calcular mais oito valores +/- S *pixel*s em torno da posição (0,0) (centro da área de pesquisa). Comparar os nove valores de erro obtidos e determinar como nova posição de origem a posição de menor erro.

No segundo passo a variável S é dividida por dois e novamente oito valores +/- S *pixel* em torno da origem são calculados. Desta vez, oito valores são comparados e, novamente, a origem será substituída pela posição com o menor erro.

No terceiro e último passo, mais uma vez a variável S é dividida por dois. Desta vez ela irá conter o valor um, o que indica o último estágio de busca. Mais uma vez, os oito valores de erros são comparados e o vetor de movimento será gerado para a posição com o menor valor de erro. A fig. 3.6 ilustra o algoritmo *Three Step Search* para uma área de pesquisa de +/- 7 *pixel*s. Os pontos de melhor *matching* escolhidos a cada passo do algoritmo estão destacados em cinza.

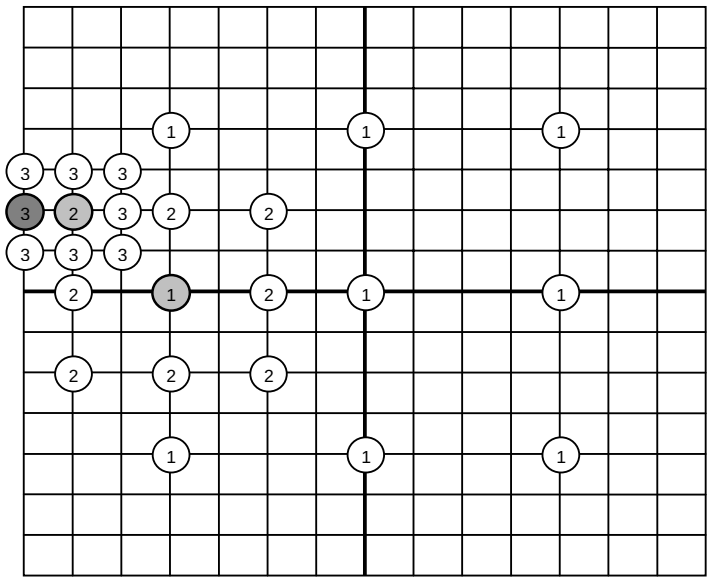


Figura 3.6 – *Three Step Search.*

Após os três passos do algoritmo, 25 posições são comparadas (9 + 8 + 8). Em geral 8n + 1 comparações são necessárias para uma busca em uma área de pesquisa de +/- (2^n-1). Se utilizássemos o algoritmo de busca completa, para a mesma área de pesquisa teríamos um total de 225 (15 x 15) comparações.

3.1.3 *One at a Time Search*

Este algoritmo é bastante simples e consiste na idéia de se encontrar primeiro um mínimo horizontal e, em seguida, um mínimo vertical (RICHARDSON, 2002). O algoritmo começa calculando o erro para a posição central da área de pesquisa. Em seguida, mais dois valores, um imediatamente à direita e outro imediatamente à esquerda, são calculados. Caso o valor do centro seja o menor, o algoritmo passa para o estágio de busca vertical. Caso contrário, redefine-se o centro com a posição do menor erro e calcula-se o valor para o vizinho, seja ele à direita ou à esquerda (dependendo do valor escolhido no passo anterior). O estágio de cálculo dos valores verticais é muito semelhante, variando a busca para cima e para baixo. A fig. 3.7 ilustra uma busca de um algoritmo *One at a Time Search* em uma área de pesquisa de +/- 7 *pixel*s.

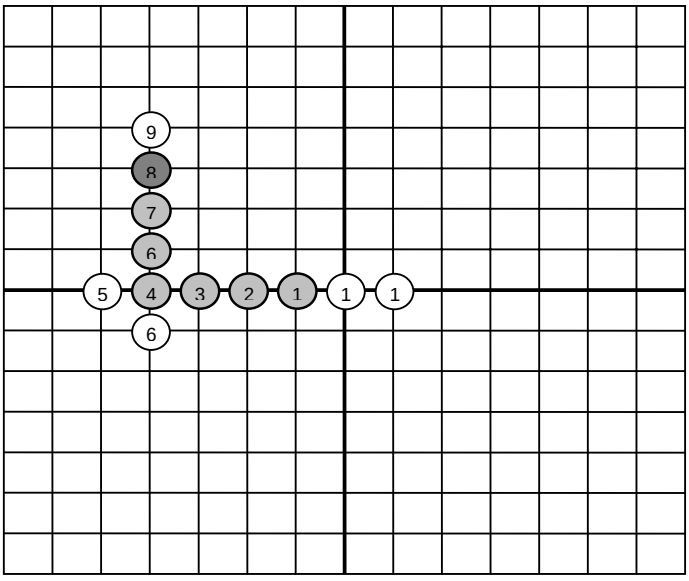


Figura 3.7 – *One at a time Search.*

No ponto número 5 o resultado do erro é maior que no ponto “4”, então o algoritmo encerra a busca horizontal e inicia a busca vertical. São calculados dois

valores, um acima e outro abaixo da posição número “4”. O menor erro é encontrado acima. O algoritmo segue deslocando um *pixel* acima até que a posição número “9” resulta em um erro maior que a posição “8”. O algoritmo encerra a busca e gera o vetor para a posição “8”.

Para este algoritmo, não é possível definir o número exato de operações, pois isso dependerá diretamente das informações contidas na área de pesquisa e no bloco. Estas informações determinarão o número de cálculos na horizontal e na vertical. No entanto, pode-se definir o número máximo de comparações que o algoritmo pode realizar. Este valor será igual a:

$$Max = 2*(P+2)$$

Onde *P* é o tamanho da área de pesquisa.

Para o exemplo apresentado, o número máximo de comparações será igual a dezoito (2*(7+2)). No entanto, no exemplo ilustrado na fig. 3.7, apenas doze comparações são realizadas.

Pode-se perceber, pelo reduzido número de operações, que este algoritmo é bastante rápido, no entanto, está muito suscetível a encontrar mínimos locais, ou seja, determinar o vetor de movimento para regiões bem próximas ao início da pesquisa, podendo desviar o rumo da pesquisa numa direção contrária a região de menor erro.

3.1.4 Busca Logarítmica

O algoritmo de busca logarítmica possui uma estrutura um pouco mais complexa, que tenta minimizar o problema dos mínimos locais (SHI, 1999). O processo começa inicializando uma variável S, que para o exemplo da fig. 3.8 é igual a “2”.

Logo após, o algoritmo calcula o erro para o centro e para mais quatro posições +/- S *pixel*s em torno do centro, formando um “+”. Os cinco valores de erro são comparados e a nova origem será a posição correspondente à posição de menor erro. Este processo será repetido até que o menor erro seja encontrado no centro. Quando isto ocorre, a variável S é dividida por “2”. Se o valor da variável S for igual a “1”, oito valores imediatamente ao redor do centro serão calculados. Por

fim, o algoritmo escolhe a posição de menor erro entre estes oito valores e mais o centro, determinando assim a região de menor erro.

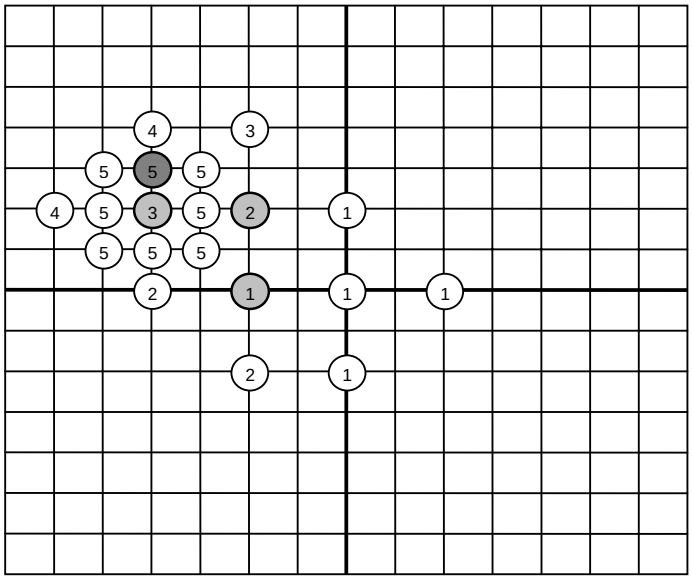


Figura 3.8 – Busca Logarítmica.

Mais uma vez, não podemos determinar quantas operações o algoritmo irá realizar a cada busca, pois isto está diretamente ligado às informações contidas nas regiões de pesquisa. Uma característica importante deste algoritmo é que ele pode, inicialmente, se desviar da região de menor erro devido a um mínimo local, porém pode mudar o rumo de sua busca ao longo do processo. Para o exemplo mostrado, vinte comparações são necessárias para encontrar o melhor *matching*.

3.1.5 – *Diamond Search*

O algoritmo *Diamond Search* possui dois padrões diamante que são usados na etapa inicial e final do algoritmo. A fig. 3.9 ilustra os padrões *Large Diamond Search Pattern* (LDSP) e o padrão *Small Diamond Search Pattern* (SDSP). O padrão LDSP consiste em 9 comparações e é utilizado na etapa inicial da pesquisa. Já o padrão SDSP consiste em 4 comparações e é utilizado na etapa final da pesquisa, com o intuito de refinar o resultado obtido na etapa anterior (KUHN, 1999).

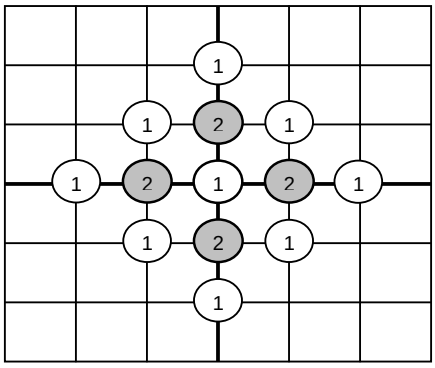


Figura 3.9 – *Large Diamond* (LDSP) (1) e *Small Diamond* (SDSP) (2).

O algoritmo começa aplicando o padrão LDSP ao centro da área de pesquisa. Caso o valor de menor erro seja encontrado no centro, o algoritmo aplica o padrão SDSP para refinar o resultado obtido. Caso contrário, um novo padrão diamante é aplicado à posição de menor erro da etapa anterior. Esta posição pode pertencer a uma aresta ou a um vértice do diamante. As figs. 3.10 e 3.11 representam, respectivamente, a busca por uma aresta e a busca por um vértice.

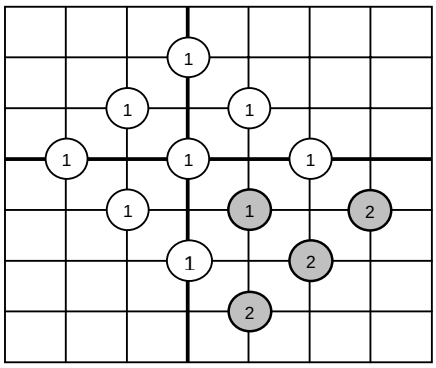


Figura 3.10 – Busca por uma aresta.

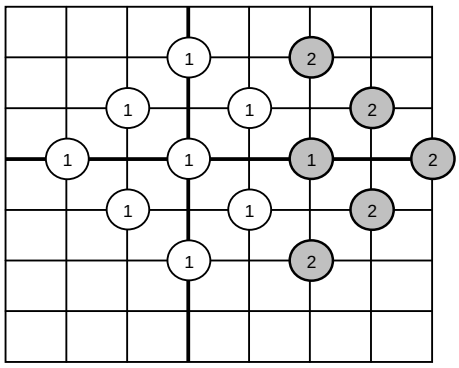


Figura 3.11 – Busca por um vértice.

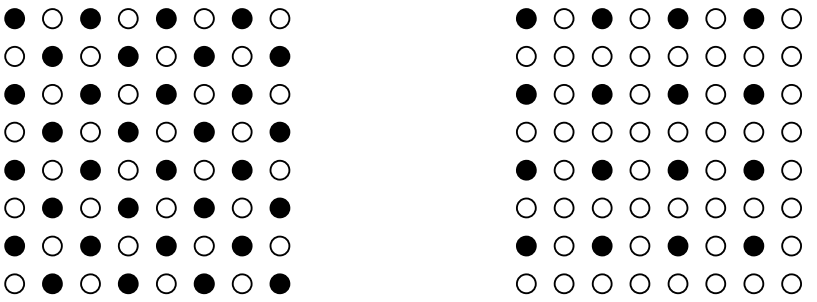
No caso da busca por uma aresta, mais 3 valores são calculados para formar um novo diamante em torno da nova origem. Quando o novo centro é um vértice do diamante, mais 4 valores são calculados para formar o novo diamante em torno do centro. Caso o menor erro não seja encontrado no centro do novo diamante, a etapa de busca será repetida, seja ela por uma aresta ou por um vértice. Quando o menor erro for encontrado para o centro do diamante o padrão SDSP é aplicado. Então mais quatro valores imediatamente ao redor do centro serão calculados e a posição com o menor erro será a escolhida.

Novamente não se pode determinar o número de operações do algoritmo. Assim como o algoritmo de busca logarítmica, o algoritmo *Diamond Search* pode começar a sua busca em uma direção e desviar a pesquisa ao longo do processo, o que pode evitar que o algoritmo caia em um mínimo local.

3.1.6 *Pel Decimation*

A técnica de *Pel Decimation* (também chamada de *Pel Subsampling*) consiste em reduzir o número de cálculos realizados pelo critério de distorção, a cada comparação (KUHN, 1999). O *Pel Decimation* é baseado no algoritmo de busca completa, no entanto, a distorção não é calculada para todos os *pixel*s do bloco. Isto diminui o tempo de processamento e a complexidade computacional do algoritmo. A técnica de *Pel Decimation* geralmente é aplicada nas proporções 2:1 e 4:1. A fig. 3.12 ilustra a técnica para as proporções 2:1(a) e 4:1(b) para uma bloco de 8x8 *pixel*s. Apenas as posições em preto são calculadas, as posições em branco são desconsideradas.

Com o algoritmo *Pel Decimation* pode-se reduzir significativamente o número de operações do algoritmo de busca completa. Neste caso, para um bloco de 8x8 *pixels*, a cada comparação, o algoritmo de busca completa deve realizar 64 operações (8x8). Para o exemplo da fig. 3.12(a) o algoritmo deve realizar 32 operações (4x8) e para o exemplo da fig. 3.12(b) o algoritmo deve realizar apenas 16 operações (4x4).



a) *Pel Decimation* 2:1 b) *Pel Decimation* 4:1

Figura 3.12 – Técnica de *Pel Decimation*.

Esta técnica reduz a complexidade computacional do algoritmo e o tempo de operação. No entanto, os vetores resultantes podem não ser os vetores ótimos. Mesmo comparando todas as posições da área de pesquisa, ao desconsiderar

alguns elementos do bloco, o algoritmo pode conduzir a escolha de uma região que não gera o resultado ótimo.

3.2 Funções de Similaridade

A função similaridade (critério de distorção) é a maneira como as diferenças entre as regiões comparadas são avaliadas. Os critérios podem considerar desde a simples diferença aritmética entre as regiões até cálculos que tentam fazer uma relação com o resultado visual da comparação.

Os critérios mais utilizados são: o menor erro quadrático MSE (*Minimum Square Error*) (RICHARDSON, 2002), o menor erro absoluto MAE (*Minimum Absolute Error*) (RICHARDSON, 2002) e a soma das diferenças absolutas SAD (*Sum of Absolute Differences*) (RICHARDSON, 2002) e (VASSILIADIS, 1998).

A função de similaridade MSE calcula a distorção entre as regiões comparadas como sendo o valor médio da diferença elevada ao quadrado, para cada ponto da região de referência e de procura. A função para o cálculo do MSE é dada por (1):

$$\text{Erro}_{x,y} = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\left(R_{i,j} - P_{i+x,j+y}\right)^2 \Big/ N^2 \quad (1)$$

Onde:

- $\text{Erro}_{x,y}$ representa o erro para a posição (x,y).
- R representa o ponto da área de referência.
- P representa o ponto da área de pesquisa.
- N é o tamanho do bloco.

O critério MAE calcula a distorção entre as regiões comparadas como sendo o valor médio do módulo da diferença, para cada ponto da região de referência e de procura. Este cálculo resulta no erro absoluto entre as duas regiões. A função para o cálculo do MAE é dada por (2):

$$\text{Erro}_{x,y} = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\left|R_{i,j} - P_{i+x,j+y}\right| \Big/ N^2 \quad (2)$$

A função de similaridade SAD calcula a distorção entre as regiões comparadas como sendo o somatório das diferenças absolutas, para cada ponto da região de referência e de procura. Este cálculo resulta em um erro proporcional ao MAE que se difere por um fator N^2. A função para o cálculo do SAD é dada por (3):

$$\text{Erro}_{x,y} = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\left|R_{i,j} - P_{i+x,j+y}\right| \quad (3)$$

O valor do erro será sempre positivo. Isto está garantido, pois para o MSE a diferença é elevada ao quadrado e, para MAE e SAD, a função considera o valor absoluto da diferença. Isto garante que erros negativos não compensem erros positivos (ZANDONAI, 2003).

Avaliando os critérios apresentados, podemos perceber que o MSE possui uma complexidade computacional maior. No entanto, o MSE tenta buscar uma relação com um sentido físico mais relevante que os critérios MAE e SAD. A grande vantagem dos critérios MAE e SAD é a significativa redução na complexidade para seu cálculo, pois consideram apenas a diferença absoluta entre dois *pixels*.

O critério SAD, por possuir um resultado equivalente ao MAE e por requisitar um número menor de operações é o critério de distorção mais utilizado para os algoritmos de estimação de movimento, principalmente para aplicações em hardware.

4. Avaliação Algorítmica

Neste capítulo serão apresentados os resultados da implementação em software de alguns dos algoritmos apresentados no capítulo anterior. Foram implementados os algoritmos: busca completa (*Full Search*), *Three Step Search*, *Diamond Search*, *One at a Time Search* e os algoritmos *Pel Decimation* 2:1 e 4:1. A linguagem escolhida foi C (KERNIGHAN, 1999) e os algoritmos foram combinados com os três critérios apresentados anteriormente, SAD, MSE e MAE.

Para todas as implementações dos algoritmos, a área de pesquisa foi delimitada em +/- 15 *pixels* em torno de um bloco com 16x16 amostras (*pixels*), ou seja, a área de pesquisa possui um tamanho total de 46x46 amostras. A seqüência de vídeo utilizada foi a "src21_ref__525_480i@30.yuv" (FEDORA, 2005) que possui uma resolução de 720x480 *pixels* no formato 4:2:0. Todos os algoritmos foram compilados no compilador DEV-C++ 4.9.9.2 (BLOODSHED, 2005) em um processador Intel Pentium 4 de 3.2GHz com 1GB de memória RAM.

Os resultados serão apresentados em duas seções distintas. Uma avaliando o tempo de execução para as implementações C dos algoritmos, para todos os critérios de distorção apresentados, e outra apresentando os resultados de erro gerados pelos algoritmos de estimação. Na terceira seção serão feitas considerações comparativas sobre os resultados obtidos.

4.1 Resultados de Tempo de Execução

Nesta seção serão apresentados os resultados de tempo de execução gerados por todas as implementações dos algoritmos em software. Os resultados estão divididos em três grupos, um para cada critério de distorção. Para todos os algoritmos, os tempos apresentados referem-se ao tempo de execução do núcleo do

algoritmo, ou seja, a parte do processamento específica de cada algoritmo. As demais etapas de cada implementação, como leitura e escrita em arquivos, não foram consideradas no cálculo do tempo de execução. Estes valores de leitura e escrita, para os algoritmos rápidos, podem ser consideravelmente altos, e macular os resultados de tempo obtidos para comparação com os algoritmos mais lentos.

A tab. 4.1 apresenta os resultados de tempo de execução de 100 *frames* da seqüência de vídeo utilizada como entrada, considerando todos os algoritmos implementados e utilizando o critério SAD. O algoritmo *Full Search* foi o que obteve o pior desempenho, em termos de tempo de execução, dentre os algoritmos implementados. Este resultado já era esperado, devido a sua característica de pesquisar todas as posições possíveis da área de pesquisa e calcular a distorção para todos os *pixels* do bloco. Os algoritmos *Pel Decimation* 2:1 e 4:1 também não obtiveram bons resultados em termos de tempo de execução. Este resultado também é devido as suas características de implementação que, apesar de não calcularem a distorção para todos os *pixels* do bloco, realizam a busca para todas as posições possíveis da área de pesquisa.

Tabela 4.1 – Resultados de tempo para o critério SAD.

Algoritmos	**Tempo para 100 *frames* (s)**	**Tempo por Bloco (ms)**
Full Search	419,9	3,11
Diamond Search	8,9	0,065
Three Step Search	7,7	0,057
One at a Time Search	2,5	0,018
Pel Decimation 2:1	310,2	2,30
Pel Decimation 4:1	108,6	0,80

Estimação gerada para 100 *frames* na resolução de 720x480 *pixels*
Processador Intel Pentium 4, 3.20 GHz, 1Gb RAM

Dentre os algoritmos mais rápidos, todos obtiveram tempos de execução muitas vezes menor que os tempos *Full Search* e *Pel Decimation*, operando em tempos, pelo menos, 12 vezes menor. O mais rápido foi o algoritmo *One at a Time Search*, que merece destaque especial, pois foi o único algoritmo, dentre os algoritmos implementados, que alcançou a taxa de processamento necessária para operar em tempo real. Este algoritmo processou 100 *frames* em 2,5 segundos, superando a taxa de processamento necessária para operar em tempo real que é de

30 *frames* por segundo. Se comparados os resultados de tempo de execução para o algoritmo mais lento, *Full Search*, e o mais rápido, *One at a Time Search*, podemos perceber que a diferença é superior a 165 vezes.

A tab. 4.2 apresenta os resultados de tempo de execução para todos os algoritmos implementados, utilizando o critério MAE. Os tempos de execução para o critério MAE seguem a mesma tendência dos tempos apresentados na tab. 4.1, para o critério SAD. Os algoritmos mais lentos, para o critério SAD, continuaram sendo os mais lentos também para o critério MAE. A única diferença importante é que, para todos os algoritmos, o tempo de execução é maior, se comparado aos apresentados na tab. 4.1. Isto se deve ao fato do critério MAE possuir uma operação de divisão para gerar o erro de cada bloco, como é apresentado na fórmula (2), o que aumenta a complexidade computacional do algoritmo.

Tabela 4.2 – Resultados de tempo para o critério MAE.

Algoritmos	**Tempo para 100 *frames* (s)**	**Tempo por Bloco (ms)**
Full Search	592,5	4,38
Diamond Search	11,0	0,081
Three Step Search	9,9	0,073
One at a Time Search	3,0	0,022
Pel Decimation 2:1	390,2	2,89
Pel Decimation 4:1	153,8	1,14

Estimação gerada para 100 *frames* na resolução de 720x480 *pixels*
Processador Intel Pentium 4, 3.20 GHz, 1Gb RAM

Mesmo com esse aumento de complexidade computacional, e por conseqüência aumento de tempo de execução, o algoritmo *One at a Time Search* alcançou uma taxa de processamento suficiente para operar em tempo real, processando os 100 *frames* da amostra em 3 segundos.

A tab. 4.3 apresenta os resultados de tempo de execução para todos os algoritmos implementados, utilizando o critério MSE. Mais uma vez os tempos de execução para o critério MSE seguem a mesma tendência dos tempos apresentados na tab. 4.1, para o critério SAD, e na tab. 4.2, para o critério MAE. Os algoritmos mais lentos, para o critério SAD e MAE, continuaram sendo os mais lentos também para o critério MSE. A diferença mais relevante é que, para todos os algoritmos, o tempo de execução é maior se comparado aos apresentados na tab. 4.1 e na tab. 4.2. Isto se deve a elevada complexidade do cálculo do critério MSE, que define a

diferença como sendo o valor médio da diferença elevada ao quadrado, como é apresentado na fórmula (3). Além da inserção da divisão, este critério também calcula o quadrado da diferença entre os *pixels* do bloco, para cada bloco, o que aumenta a complexidade do cálculo se comparados aos critérios SAD e MAE. Mesmo com esse aumento de complexidade computacional, se comparado aos critérios SAD e MAE, o algoritmo *One at a Time Search* continuou obtendo uma taxa de processamento suficiente para operar em tempo real, processando 100 *frames* da amostra em 3,1 segundos.

Tabela 4.3 – Resultados de tempo para o critério MSE.

Algoritmos	**Tempo para 100 *frames* (s)**	**Tempo por Bloco (ms)**
Full Search	604,0	4,47
Diamond Search	11,1	0,082
Three Step Search	10,2	0,075
One at a Time Search	3,1	0,022
Pel Decimation 2:1	399,4	2,96
Pel Decimation 4:1	164,1	1,21

Estimação gerada para 100 *frames* na resolução de 720x480 *pixels*
Processador Intel Pentium 4, 3.20 GHz, 1Gb RAM

Vale salientar, que o critério MSE busca explorar uma relação de similaridade com um sentido físico mais relevante que os critérios SAD e MAE, que apenas exploram a diferença simples entre os *pixels*. Esta característica do critério MSE é o principal fator responsável pela sua mais elevada complexidade computacional e maior tempo de execução.

Os resultados de tempo de execução apresentados nesta seção não devem ser avaliados isoladamente. Geralmente os algoritmos que possuem elevados tempos de execução tendem a gerar os melhores resultados em termos de erro acumulado. A próxima seção apresentará os resultados de erro para todas as implementações dos algoritmos de estimação investigados neste trabalho.

4.2 Resultados de Erro

Nesta seção serão apresentados os resultados de erro gerados por todas as implementações dos algoritmos em software. Os resultados estão divididos em três grupos, um para cada critério de distorção. Para avaliar os resultados de erro

gerados pelos algoritmos de estimação, a tab. 4.4 apresenta o somatório do erro absoluto, ou seja, o somatório da diferença, *pixel* a *pixel*, entre os *frames* vizinhos (sem estimação), gerado para os 100 primeiros *frames* da amostra utilizada.

Tabela 4.4 – Erro absoluto para os 100 primeiros *frames* da amostra.

	Somatório do erro absoluto	**Erro por Bloco**
Erro	213.158.877	1.578,95

Somatório da diferença para os 100 primeiros *frames*.

Todos os valores apresentados para erro, a partir deste ponto, estarão considerando o somatório do erro absoluto para uma comparação justa com os valores de erro apresentados na tab. 4.4.

A tab. 4.5 apresenta os resultados de erro gerados para todos os algoritmos com o critério SAD, após a estimação dos 100 primeiros *frames* da amostra utilizada. Estes valores de erro são gerados a partir da soma dos erros para cada bloco escolhido pela estimação.

Os resultados da tab. 4.5 demonstram a eficiência dos algoritmos que realizam a busca em todas as posições possíveis da área de pesquisa, como os algoritmos *Full Search* e *Pel Decimation* 2:1 e 4:1. O algoritmo *Full Search*, como já era esperado, resultou no menor erro para a estimação dos 100 primeiros *frames* da amostra, seguido de perto pelo algoritmo *Pel Decimation* 2:1. Ambos alcançaram uma diminuição do erro superior a 64%, se comparado ao erro apresentado na tab. 4.4. O algoritmo *Pel Decimation* 4:1 ficou com o terceiro melhor desempenho na avaliação do erro gerado, diminuindo o erro em aproximadamente 59% em relação ao erro absoluto. Entre os algoritmos rápidos, o que obteve o melhor desempenho foi o algoritmo *Three Step Search*, que reduziu o erro em mais de 56%. O algoritmo *Diamond Search* obteve um resultado regular, diminuindo o erro em mais de 41%. O pior desempenho, em termos de diminuição de erro, foi o do algoritmo *One at a Time Search*, que obteve uma diminuição inferior a 9%.

É interessante salientar que os resultados de erro para os algoritmos sub-ótimos dependem diretamente da amostra de vídeo utilizada. Dependendo do tipo do vídeo, se houver muito movimento ou se a cena é praticamente estática, por exemplo, os resultados podem variar consideravelmente.

Tabela 4.5 – Resultados de erro para o critério SAD.

Algoritmos	Erro para 100 *frames*	Erro por Bloco	Diminuição do erro(%)
Full Search	74.282.817	550,24	65,15
Diamond Search	124.357.591	921,17	41,77
Three Step Search	92.714.430	686,77	56,50
One at a Time Search	194.984.582	1.444,33	8,53
Pel Decimation 2:1	75.068.732	556,06	64,78
Pel Decimation 4:1	87.351.770	647,05	59,02

Estimação gerada para 100 *frames* na resolução de 720x480 *pixels*
Processador Intel Pentium 4, 3.20 GHz, 1Gb RAM

A tab. 4.6 apresenta os resultados de erro gerados para todos os algoritmos com o critério MSE, após a estimação dos 100 primeiros *frames* da amostra utilizada. Pode-se perceber valores de erro superiores aos apresentados na tab. 4.5, para o critério SAD. É interessante observar que o algoritmo *One at a Time Search* obteve, inclusive, um resultado de erro superior ao apresentado da tab. 4.4, isso é, apresentou um resultado pior em termos de erro do que quando a ME simplesmente não é aplicada.

É importante salientar que estes resultados ocorrem porque se está considerando o somatório do erro absoluto como parâmetro de comparação. O critério MSE busca uma relação com um sentido físico mais relevante (baseado na fórmula (2)). Isto pode gerar um erro absoluto maior. No entanto, o resultado visual pode ser melhor. Sendo assim, até mesmo o algoritmo *One at a Time Search*, que gerou um erro absoluto maior que o encontrado antes da estimação, pode resultar em uma imagem de boa qualidade após o processo de estimação e compensação de movimento.

Tabela 4.6 – Resultados de erro para o critério MSE.

Algoritmos	Erro para 100 *frames*	Erro por Bloco	Diminuição do erro(%)
Full Search	74.658.263	553,02	64,97
Diamond Search	186.250.702	1.379,63	12,62
Three Step Search	102.179.857	756,89	52,06
One at a Time Search	229.735.761	1.701,75	-
Pel Decimation 2:1	75.337.813	558,06	64,66
Pel Decimation 4:1	87.389.357	647,33	59,00

Estimação gerada para 100 *frames* na resolução de 720x480 *pixels*
Processador Intel Pentium 4, 3.20 GHz, 1Gb RAM

Os demais algoritmos obtiveram resultados de erro muito próximos aos apresentados na tab. 4.5. Apenas o algoritmo *Diamond Search* teve uma queda considerável na redução do erro de cerca de 30%.

A tabela com os resultados para o critério MAE será omitida, pois o resultado de erro gerado pelo critério difere por um fator de N^2, se comparado ao critério SAD, como está apresentado na fórmula (3). O critério MAE divide o resultado do erro, para cada bloco, por N^2, o que resulta no erro absoluto para este bloco. No entanto, como se está considerando o somatório do erro absoluto como parâmetro de comparação, os critérios MAE e SAD são passíveis de comparação.

4.3 Avaliação dos Resultados

Avaliar os resultados gerados pelas implementações dos algoritmos de estimação apresentados pode ser uma tarefa muito complexa. Se forem avaliados isoladamente os resultados de tempo de execução apresentados, concluir-se-á, sem sombra de dúvida, que o melhor desempenho pertence ao algoritmo *One at a Time Search*. Caso se analise apenas os resultados de erro apresentados, se chegará a conclusão que o melhor resultado pertence ao algoritmo *Full Search*. Enfim, existe sempre uma compromisso entre o tempo de execução e a qualidade do resultado gerado.

Para uma avaliação coerente dos resultados apresentados, deve-se ter em mente, primeiramente, o tipo de implementação que se deseja aplicar ao algoritmo. Implementações em software podem assimilar mais facilmente a complexidade gerada pelas simplificações dos algoritmos sub-ótimos e pelos critérios mais elaborados, como o MAE e MSE. No entanto, aplicações em software são praticamente incapazes de suportar a quantidade de cálculos necessárias para as implementações que realizam a busca em toda a área de pesquisa para aplicações que operem com vídeos de alta resolução em tempo real. Implementações em hardware são mais facilmente desenvolvidas para algoritmos regulares, como os que realizam a busca em toda a área de pesquisa, e também para critérios de distorção mais simples, como o SAD, que realiza apenas uma subtração em módulo entre os pixels. Os demais critérios introduzem operações, como divisão e potenciação, que são bem mais difíceis de implementar em hardware. Os algoritmos mais regulares (como *Full Search* e *Pel Decimation*), apesar de realizarem um número muito

elevado de comparações, podem ter diversas etapas paralelizadas em implementações em hardware, o que acelera o seu processamento.

Uma característica relevante e negativa dos algoritmos rápidos avaliados neste trabalho (*Diamond Search*, *Three Step Search* e *One at a Time Search*) é que eles possuem uma elevada dependência de dados, dificultando a exploração do paralelismo em hardware. Por isso, mesmo sem que estas soluções tenham sido implementadas em hardware, é possível estimar que os ganhos em termos de tempos de execução obtidos por estes algoritmos, quando considerada sua implementação em software (tab. 4.1, 4.2 e 4.3), não serão mantidos quando for considerada a implementação em hardware. Isto é, quando considera-se a implementação em hardware, os algoritmos *Full Search* e *Pel Decimation* 2:1 e 4:1 terão um tempo de processamento muito próximo aos tempos que seriam exibidos pelos algoritmos rápidos.

Também é importante salientar que os algoritmos rápidos não possuem um tempo fixo para encontrar o melhor *matching* do bloco atual dentro da área de pesquisa. Isso implica em dificuldades para implementar estes algoritmos em hardware, pois seria necessária uma estrutura de *buffers* na entrada e na saída para garantir o sincronismo da ME com os demais blocos de um codificador de vídeo. Estes *buffers* utilizariam um número elevado de recursos de hardware e, eventualmente, não seria possível explorar toda a eficiência dos algoritmos rápidos, pois seria possível que a ME tivesse que ser colocada em estado de espera para viabilizar a sincronização com os outros blocos. Tendo em vista o que foi exposto nos parágrafos anteriores, e como o foco principal deste trabalho é propor uma arquitetura de hardware para a estimação de movimento, a escolha do algoritmo de busca e do critério de similaridade irá desconsiderar os algoritmos rápidos. Então, serão avaliados mais detalhadamente os resultados apresentados pelos algoritmos *Full Search* e *Pel Decimation* 2:1 e 4:1, utilizando o critério SAD.

Para melhor visualizar os resultados destes algoritmos a tab. 4.7 apresenta os resultados de tempo de execução, erro e diminuição de erro, apenas para os algoritmos *Full Seach*, *Pel Decimation* 2:1 e 4:1.

Comparando os resultados de tempo de execução e erro gerados por estes algoritmos, podemos perceber que o algoritmo *Full Search* possui um tempo de execução aproximadamente 1,4 vezes maior, gerando uma diminuição do erro cerca de 1% maior, se comparado ao algoritmo *Pel Decimation* 2:1. Com relação ao

algoritmo *Pel Decimation* 4:1, podemos perceber que o algoritmo *Full Search* utilizou um tempo aproximadamente 3,8 vezes maior para processar os 100 *frames* e gerou uma diminuição do erro aproximadamente 6% maior.

Tabela 4.7 – Resultados para os algoritmos *Full Search* e *Pel Decimation* 2:1 e 4:1

Algoritmos	Tempo (s)	Erro para 100 *frames*	Diminuição do erro(%)
Full Search	419,9	74.282.817	65,15
Pel Decimation 2:1	310,2	75.068.732	64,78
Pel Decimation 4:1	108,6	87.351.770	59,02

Estimação gerada para 100 *frames* na resolução de 720x480 *pixels*
Processador Intel Pentium 4, 3.20 GHz, 1Gb RAM

Os resultados da tab. 4.7 mostram que as características do algoritmo *Pel Decimation*, apresentadas no capítulo 2, reduzem consideravelmente o tempo de execução e geram uma diminuição de erro muito próxima se comparado ao algoritmo *Full Search*. Estes resultados fizeram com que a escolha do algoritmo *Full Search*, para a implementação da arquitetura desenvolvida neste trabalho, tenha sido descartada.

O algoritmo escolhido para a implementação da arquitetura foi o *Pel Decimation* 4:1. Esta escolha ocorreu pois o algoritmo obteve uma diminuição de erro em torno de 5% menor que o algoritmo *Pel Decimation* 2:1, mas no entanto, realizou a estimação em um tempo quase 3 vezes menor. Esta diferença na diminuição do erro foi consideravelmente pequena se comparada ao ganho em termos de tempo de execução. A escolha do algoritmo *Pel Decimation* 4:1 para a implementação em hardware ainda possui outras vantagens. A sub-amostragem dos valores dos pixels, como ilustrado na fig. 3.13(b), reduz a quantidade de cálculos necessários para determinar o menor SAD, reduzindo a quantidade de recursos de hardware necessários para a implementação da arquitetura. A implementação do algoritmo *Pel Decimation* também diminui pela metade o tamanho da memória que armazena o bloco do *frame* atual. Os resultados obtidos através das implementações em software, aliadas às facilidades de implementação deste algoritmo em hardware, foram fatores decisivos para a escolha do algoritmo *Pel Decimation* 4:1. Além disso, como a arquitetura proposta neste trabalho está fortemente baseada em uma arquitetura que utilizou *Full Search* (UFRN), será

possível comparar os impactos em hardware do uso do *Pel Decimation* 4:1 em relação ao *Full Search*.

No capítulo seguinte serão apresentadas, detalhadamente, as características da arquitetura para implementar o algoritmo *Pel Decimation* 4:1 com SAD em hardware, bem como os resultados de implementação.

5. Proposta Arquitetural para a Estimação de Movimento

Neste capítulo, será detalhada a arquitetura de hardware proposta para implementar o algoritmo *Pel Decimation* 4:1, utilizando SAD como critério de similaridade. Esta arquitetura teve como base a arquitetura proposta, e ainda não publicada, pelo grupo da Universidade Federal do Rio Grande do Norte (UFRN) que faz parte do consórcio H.264Brasil, da RFP11 do SBTVD (MINISTÉRIO 2005). Este trabalho foi desenvolvido na UFRN sob a orientação do professor Ivan Saraiva da Silva (UFRN). Esta arquitetura trabalha com o algoritmo *Full Search* e o critério SAD, utilizando blocos de 16x16 amostras em uma área de pesquisa de 32x32 amostras. A arquitetura desenvolvida na UFRN é capaz de atingir uma taxa de processamento capaz de operar em tempo real para resoluções elevadas mas, para tanto, utiliza uma quantidade excessiva de recursos de hardware. Utilizando o *Pel Decimation* ao invés do *Full Search*, é possível simplificar a arquitetura mantendo o desempenho elevado. Além disso, é possível explorar características adicionais, como o aumento na área de busca.

A arquitetura proposta nesta monografia trabalha com o algoritmo *Pel Decimation* 4:1 e o critério SAD, utilizando blocos de 16x16 e uma área de pesquisa de 64x64 amostras. As vantagens e desvantagens da implementação do algoritmo *Pel Decimation* 4:1 sobre a arquitetura base e as modificações necessárias serão apresentadas no decorrer deste capítulo.

As entradas da estimação de movimento são o bloco do quadro atual e a área de pesquisa do quadro de referência. As informações destes quadros ficam armazenadas na memória externa ao codificador e, quando são necessárias, são enviadas para a memória interna do estimador de movimento (ME – *Motion*

Estimation). A saída da ME são os vetores de movimento, que devem ser codificados junto com os resíduos da imagem.

A produção dos vetores de movimento é realizada através da comparação de regiões do quadro atual com regiões do quadro de referência, buscando uma maior semelhança entre elas. Essas regiões são representadas por blocos de 16x16 amostras de luminância. O diagrama de blocos da arquitetura da estimação de movimento está apresentada na fig. 5.1. Vários sinais foram ocultados para permitir uma melhor visualização da figura. Os detalhes dos principais subsistemas que formam a arquitetura serão apresentados nas próximas seções e incluem a linha de cálculo de SAD e as suas unidades de processamento (UPs), o controle, o comparador e o sistema de memória, incluindo as memórias internas e o gerenciador de memória. Os outros blocos incluem um conjunto de registradores para uma linha da área de pesquisa (RLP), um conjunto de registradores para uma linha do bloco do quadro atual (RLB) e seletores dos dados dos registradores (SP e SB).

A memória interna está organizada em 5 memórias distintas como está apresentado na fig. 5.1. Uma memória é destinada ao armazenamento do bloco do quadro atual e as outras quatro memórias são usadas para armazenar a área de pesquisa, que está dividida em quatro partes distintas. A memória da área de pesquisa foi dividida em quatro partes devido às características da arquitetura, que opera com palavras de 32 amostras (de oito bits cada). Desta forma, a arquitetura opera sobre uma das memórias de pesquisa de cada vez, necessitando de quatro iterações, uma com cada memória, para cobrir toda a área de pesquisa.

Os dados da área de pesquisa e do bloco atual que estão armazenados na memória interna são acessados pelo gerenciador de memória (fig. 5.1). Como o bloco está armazenado em uma memória e a área de pesquisa em outras quatro, quando o ME inicia seu funcionamento o gerenciador de memória realiza a leitura de uma palavra, de uma das memórias da área de pesquisa, e outra da memória do bloco atual. Com isso, são acessadas uma linha de uma das partes da área de pesquisa e uma linha do bloco atual. Essas linhas são armazenadas nos registradores da linha de pesquisa (RLP na fig. 5.1) e nos registradores de linha de bloco (RLB na fig. 5.1). Nesse momento, o gerenciador disponibiliza estes dados na sua saída e ativa o início das atividades dos seletores.

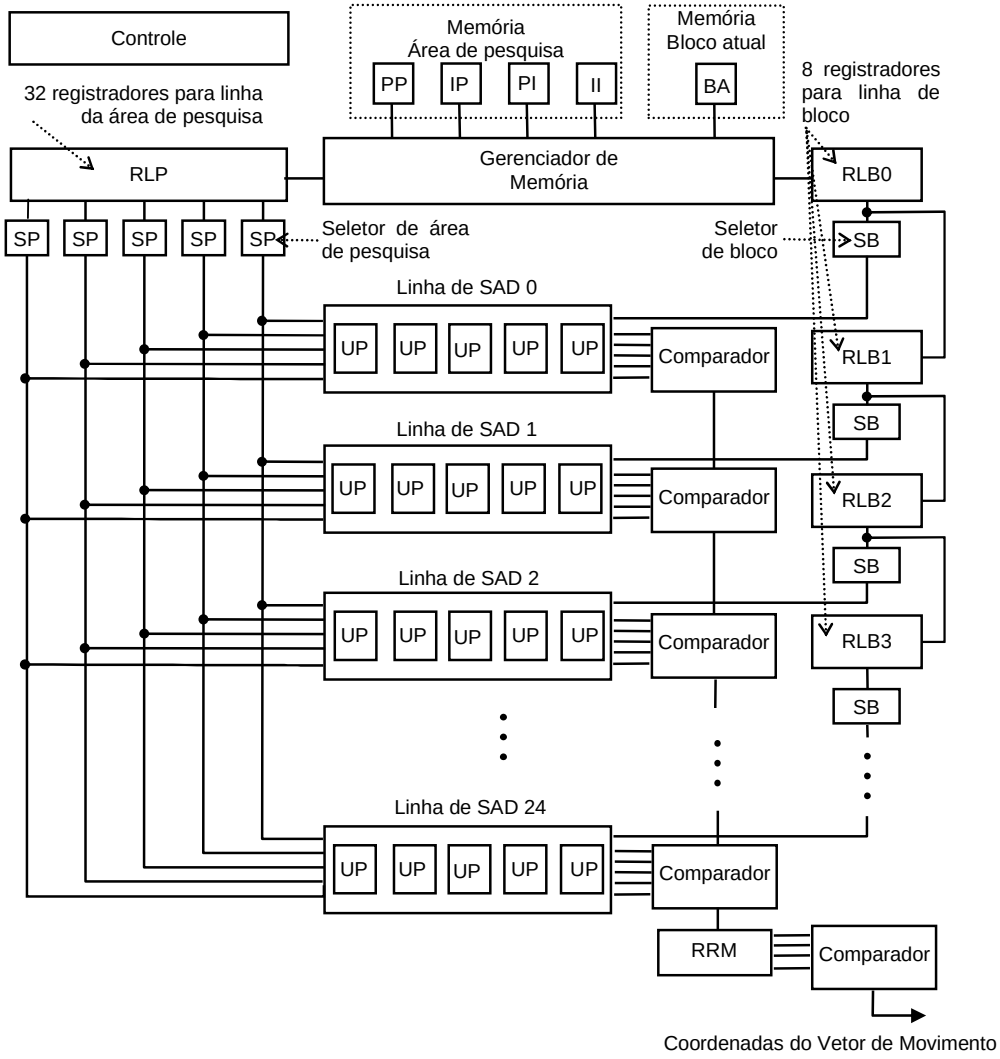


Figura 5.1 – Arquitetura do Estimador de Movimento.

Os registradores RLP armazenam uma linha completa de uma das áreas de busca (32 amostras), utilizando 32 Bytes, e os registradores RLB armazenam a linha do bloco atual, com oito Bytes. Na verdade, uma linha completa da área de pesquisa possui 64 amostras, no entanto, como a área de pesquisa foi dividida em quatro partes distintas, apenas 32 amostras são armazenadas no RLP a cada leitura de memória.

No caso do bloco atual, a linha do bloco atual deveria conter 16 amostras, visto que o tamanho do bloco é de 16x16 amostras. No entanto, devido à sub-amostragem definida pelo *Pel Decimation* 4:1, apenas oito amostras precisam ser armazenadas a cada linha.

A unidade de processamento (UP na fig. 5.1) calcula a distorção entre ¼ de palavra do bloco do quadro atual e ¼ da palavra do bloco candidato. A decisão de calcular ¼ de palavra na UP, ao invés de ½ da arquitetura base, foi tomada devido ao alto desempenho da arquitetura base. Calculando ¼ de palavra serão necessários duas vezes mais interações para o cálculo do SAD do bloco candidato. Isto diminuirá o desempenho da arquitetura, mas, proporcionará uma redução significativa na área da arquitetura. Cinco UPs são combinadas para formar uma linha de SAD. Cada UP é responsável por processar cinco blocos candidatos. Logo, a linha de SAD com cinco UPs processa os 25 blocos candidatos de uma linha. Um conjunto de 25 linhas forma uma matriz de SAD, como está apresentado na fig. 5.1, realizando a busca completa sobre uma das quatro partes da área de busca do quadro de referência.

O controle gerencia as operações e define em que estágio do pipeline cada uma das linhas da matriz de SADs está operando. Depois de finalizado o processamento de toda a região de busca, o comparador recebe os valores de distorção de todos os blocos candidatos e escolhe aquele com menor valor. O comparador também recebe os vetores de movimento correspondentes a cada bloco candidato. Estes vetores são gerados por um gerador de vetores, omitido na fig. 5.1. Ao final do processamento do comparador, o bloco candidato que possuir o menor valor de SAD tem o seu vetor de movimento enviado para a saída.

A primeira linha de SADs calcula o SAD para os 25 blocos candidatos que iniciam na primeira linha da área de pesquisa e compara o bloco atual a estes blocos candidatos. A segunda linha de SADs calcula o SAD para os 25 blocos que iniciam na segunda linha da área de referência e assim por diante, até que, na vigésima quinta linha de SADs os SADs dos últimos 25 blocos candidatos são calculados.

A arquitetura proposta na fig. 5.1 otimiza o compartilhamento das amostras da área de busca e das amostras do bloco do quadro atual sobre o qual deseja-se encontrar o melhor casamento e, deste modo, gerar o melhor vetor de movimento. Esta otimização permite diminuir o número de bits de memória necessários e permite reduzir o número de acessos à memória. O cálculo do SAD sempre acontece linha a linha, como será explicado em maiores detalhes na próxima seção. Inicialmente, a primeira linha do bloco atual é usada para calcular o SAD para os blocos candidatos da primeira linha da área de busca. Então, a segunda linha da área de busca é lida da memória da área de pesquisa e a segunda linha do bloco atual é lida da memória

do quadro atual. A primeira linha de SAD utiliza estas duas novas informações para calcular a segunda linha de SADs dos blocos que iniciam na primeira linha da área de busca. Mas a segunda linha de SADs irá receber a segunda linha da área de busca e a primeira linha do bloco atual, para iniciar os cálculos dos SADs dos blocos candidatos que começam na segunda linha da área de busca. E assim acontece sucessivamente, até que todos os SADs dos blocos candidatos da área de busca estejam calculados. É por isso que todas as linhas de SAD da fig. 5.1 recebem sempre a mesma linha da área de busca (RLP), mas recebem linhas diferentes do bloco atual (RLB).

Considerando-se as memórias internas da arquitetura (apresentadas na fig. 5.1) inicialmente preenchidas, um novo vetor é gerado a cada 653 ciclos de *clock*. Este novo vetor refere-se ao menor SAD encontrado para uma das memórias de pesquisa. Após encontrar os resultados para as quatro memórias de pesquisa, que foram armazenadas no registrador RRM, mais três ciclos são necessários para comparar estes quatro resultados e assim determinar o menor SAD e enviar o vetor correspondente para a saída. Desta forma, 2615 ciclos de *clock* são necessários para gerar o vetor de movimento para o bloco atual.

Os blocos principais da arquitetura apresentada na fig. 5.1 serão apresentados em mais detalhes nas próximas seções do texto.

5.1 Gerenciamento de Memória

As memórias internas utilizadas no estimador de movimento armazenam apenas as informações do componente luminância, pois as informações de crominância não são necessárias para os cálculos do ME, como já foi exposto na seção 3.

O ME foi projetado utilizando cinco memórias internas, como está apresentado na fig. 5.1. Uma memória armazena o bloco do quadro atual e as outras quatro memórias armazenam, cada um delas, um conjunto distinto de dados da área de pesquisa. A memória que armazena o bloco do quadro atual possui oito palavras de 64 bits cada, onde cada posição da memória armazena uma linha do bloco atual, com oito amostras de oito bits cada. Já as quatro memórias da área de pesquisa (PP, PI, IP e II na fig. 5.1) seguem a seguinte divisão:

PP – Contém os elementos pares das linhas pares
IP – Contém os elementos ímpares das linhas pares
PI – Contém os elementos pares das linhas ímpares
II – Contém os elementos ímpares das linhas ímpares

Todas as memórias da área de pesquisa possuem 32 palavras de 256 bits cada. Os dados contidos nas quatro memórias de pesquisa representam toda a área de pesquisa de 64x64 amostras. Esta divisão foi adotada devido à sub-amostragem utilizada pelo algoritmo *Pel Decimation* 4:1, que calcula o SAD para a primeira amostra e desconsidera a amostra imediatamente seguinte e também a imediatamente inferior. Desta forma, por exemplo, se o primeiro elemento, posição (0,0) na área de pesquisa, for considerado, o próximo elemento desta linha a ser calculado será o elemento (0,2), e assim sucessivamente, sempre desconsiderando os elementos ímpares desta linha, até que todas as amostras desta linha sejam calculadas. A primeira amostra da próxima linha a ser calculada será a de posição (2,0), ou seja, a linha ímpar deve ser desconsiderada, e assim sucessivamente até que todas as linhas do bloco tenham sido calculadas. Desta forma, com apenas uma leitura da memória podemos calcular os SADs de uma linha para todos os blocos candidatos que comecem em linhas e colunas pares. Assim, ao final dos cálculos para a memória PP, teremos um vetor de movimento resultante para os blocos que começam em linhas e colunas pares. O mesmo processo deve ser realizado para as outras três memórias, gerando os vetores de movimento resultantes para os blocos candidatos que começam em linhas pares e colunas ímpares (PI), linhas ímpares e colunas pares (IP) ou linhas ímpares e colunas ímpares (II).

O gerenciador de memória na fig. 5.1 é responsável por controlar as leituras de memória de acordo com as necessidades da arquitetura. As amostras lidas das memórias de área de pesquisa são copiadas para os registradores RLP, enquanto que as amostras de bloco atual são armazenadas nos registradores RLB. Os seletores de linha SP e SB dividem a palavra lida da memória, de modo que cada UP receba uma informação correta na sua entrada.

O ME acessa duas palavras das memórias, uma da memória do bloco atual e uma da memória da área de pesquisa, a cada 20 ciclos, em um total de 320 bits sendo acessados em intervalos de 20 ciclos. Cada leitura realizada sobre as duas

memórias gera uma linha completa de uma das memórias da área de pesquisa (com 32 amostras) e uma linha completa do bloco atual (com oito amostras).

5.2 Arquitetura para o Cálculo do SAD

Para encontrar a maior similaridade entre um bloco do quadro atual e os blocos da área de pesquisa do quadro de referência, é utilizado o algoritmo SAD (*Sum of Absolute Differences*) (VASSILIADIS, 1998). O SAD recebe um bloco do quadro atual e a área de pesquisa do quadro de referência e realiza a subtração em módulo das posições correspondentes aos blocos, obtendo valores denominados resíduos. Em seguida, os resíduos são acumulados em um único valor (chamado de distorção) que representa o grau de similaridade entre as imagens.

Na arquitetura do ME proposta neste trabalho, foram considerados blocos de 16x16 amostras de luminância e a área de pesquisa foi definida em 64x64 amostras, estando sempre incluída em um único quadro de referência. Deste modo, existem 49x49 blocos candidatos dentro da área de pesquisa, gerando um total de 2401 blocos candidatos ao melhor casamento. Considerando blocos 16x16 e levando em conta a sub-amostragem inserida pelo algoritmo *Pel Decimation* 4:1, 64 cálculos de SAD devem ser realizados para cada bloco candidato. Portanto 153.664 cálculos de SAD devem ser realizados para realizar a busca completa em toda a área de pesquisa para encontrar o melhor casamento de bloco.

A arquitetura para o cálculo do SAD foi construída hierarquicamente. A instância de mais elevado nível hierárquico é a matriz de SADs, que é formada por 25 linhas de SADs que, por sua vez, são formadas por cinco unidades de processamento (UPs), como está apresentado na fig. 5.1. Cada UP calcula a medida de similaridade de um bloco candidato da área de pesquisa do quadro de referência com o bloco do quadro atual.

A fig. 5.2 apresenta a arquitetura simplificada de uma UP, onde os sinais de controle estão omitidos. Cada UP recebe como entrada duas amostras do bloco atual (B0 e B1 na fig. 5.2) e duas amostras do bloco candidato (R0 e R1 na fig. 5.2). Isso significa que cada UP calcula paralelamente a similaridade de ¼ linha do bloco candidato em relação ao bloco atual, uma vez que um bloco 16x16 após a subamostragem definida pelo *Pel Decimation* 4:1, passará a conter apenas 8x8 amostras utilizadas (oito amostras por linha).

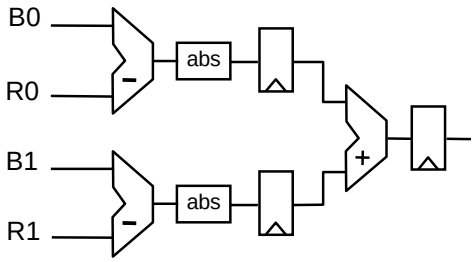


Figura 5.2 – Arquitetura da unidade de processamento de SAD.

As UPs foram projetadas para operar em um pipeline de três estágios, como está apresentado na fig. 5.2. Inicialmente, é realizada uma subtração entre as duas amostras do bloco atual (B0 e B1) e as duas amostras do bloco candidato (R0 e R1). O resultado do subtrator passa por um processo que extrai o módulo. Então, os módulos são somados para gerar um valor único. O resultado representa o módulo do erro entre os dois blocos para as duas primeiras amostras, ou seja, para ¼ da primeira linha dos blocos.

O SAD parcial do bloco candidato (¼ de linha) gerado pela UP deve ser armazenado e adicionado aos SADs das demais partes do bloco candidato para gerar o SAD total deste bloco, que é formado por oito linhas com oito amostras em cada linha (32 SADs devem ser acumulados). A linha de SADs, apresentadas na fig. 5.3, agrupa cinco UPs e realiza a acumulação para gerar o valor final do SAD dos blocos candidatos. Ao todo são 25 acumuladores, um para cada bloco candidato existente na linha.

Cada UP gera valores parciais para cinco acumuladores diferentes. Cada conjunto de cinco acumuladores possui um somador para realimentação e um conjunto multiplexador/demultiplexador, para selecionar a entrada e a saída corretas, de acordo com os sinais de controle.

Como já foi mencionado, cada UP processa ¼ da linha dos blocos (2 amostras) em cada operação e demora três ciclos para concluir este cálculo. A tab. 5.1 mostra que a arquitetura das UPs calcula os SADs de duas amostras da linha de cinco blocos (indicados pela letra **a** na tab. 5.1) para só então retornar ao primeiro bloco e realizar a segunda etapa do cálculo sobre a linha, processando mais duas amostras dos blocos (indicados pela letra **b** na tab. 5.1), e assim sucessivamente para a outra metade da linha (indicados pelas letras **c** e **d** na tab. 5.2).

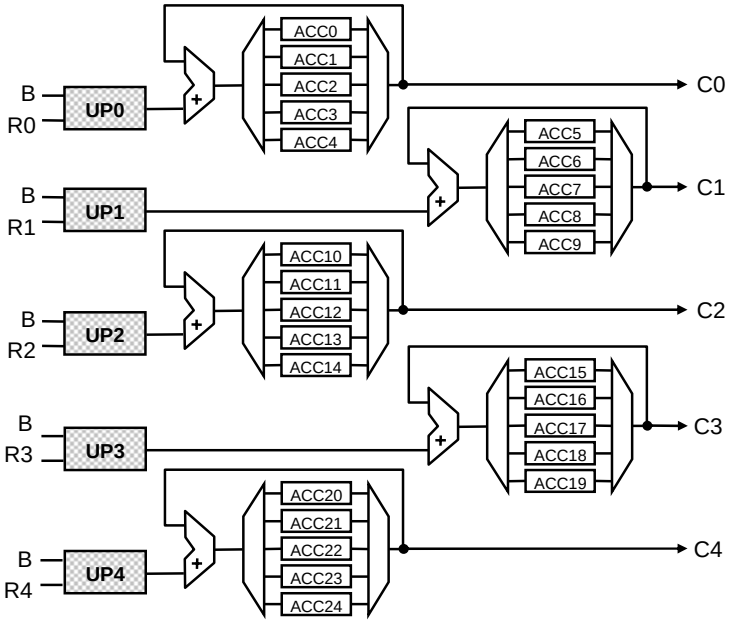


Figura 5.3 – Diagrama em blocos de uma linha de SAD.

Quando o pipeline da UP estiver cheio, um novo resultado, referente a ¼ de linha, será gerado a cada ciclo, desta forma, após a geração do valor da primeira etapa, mais três ciclos serão necessários para a conclusão do cálculo do SAD de uma linha do bloco, totalizando 6 ciclos.

O resultado final é gerado depois que as oito linhas são processadas para cada bloco candidato. Como cada UP processa uma linha do bloco em quatro etapas, então cada UP gera 32 resultados parciais de SAD para cada bloco processado. Estes 32 resultados parciais devem ser somados para gerar o SAD final do bloco. Como estes resultados, para um mesmo bloco não são gerados em paralelo, uma simples estrutura de somador e registrador de acumulação é suficiente para fazer esta totalização. Como as UPs calculam em paralelo o SAD de mais de um bloco, então um registrador é utilizado para armazenar o SAD de cada bloco (ACC0 a ACC24 na fig. 5.3) e um par demultiplexador e multiplexador é necessário para controlar o acesso correto aos registradores, como está apresentado na fig. 5.3. Quando uma linha de SADs concluir seus cálculos, então os registradores ACC0 a ACC24 da fig. 5.3 irão conter os SADs finais dos 25 blocos candidatos.

Tabela 5.1: Exemplo do escalonamento de operações nas UPs de uma linha de SADs.

Tempo	Estágio	UP0	UP1	UP2	UP3	UP4
0	1	Bloco 0 a	Bloco 5 a	Bloco 10 a	Bloco 15 a	Bloco 20 a
1	2	Bloco 1 a	Bloco 6 a	Bloco 11 a	Bloco 16 a	Bloco 21 a
2	3	Bloco 2 a	Bloco 7 a	Bloco 12 a	Bloco 17 b	Bloco 22 a
3	4	Bloco 3 a	Bloco 8 a	Bloco 13 a	Bloco 18 b	Bloco 23 a
4	5	Bloco 4 a	Bloco 9 a	Bloco 14 a	Bloco 19 b	Bloco 24 a
5	1	Bloco 0 b	Bloco 5 b	Bloco 10 b	Bloco 15 b	Bloco 20 b
6	2	Bloco 1 b	Bloco 6 b	Bloco 11 b	Bloco 16 b	Bloco 21 b
7	3	Bloco 2 b	Bloco 7 b	Bloco 12 b	Bloco 17 b	Bloco 22 b
8	4	Bloco 3 b	Bloco 8 b	Bloco 13 b	Bloco 18 b	Bloco 23 b
9	5	Bloco 4 b	Bloco 9 b	Bloco 14 b	Bloco 19 b	Bloco 24 b
10	1	Bloco 0 c	Bloco 5 c	Bloco 10 c	Bloco 15 c	Bloco 20 c
11	2	Bloco 1 c	Bloco 6 c	Bloco 11 c	Bloco 16 c	Bloco 21 c
12	3	Bloco 2 c	Bloco 7 c	Bloco 12 c	Bloco 17 c	Bloco 22 c
13	4	Bloco 3 c	Bloco 8 c	Bloco 13 c	Bloco 18 c	Bloco 23 c
14	5	Bloco 4 c	Bloco 9 c	Bloco 14 c	Bloco 19 c	Bloco 24 c
15	1	Bloco 0 d	Bloco 5 d	Bloco 10 d	Bloco 15 d	Bloco 20 d
16	2	Bloco 1 d	Bloco 6 d	Bloco 11 d	Bloco 16 d	Bloco 21 d
17	3	Bloco 2 d	Bloco 7 d	Bloco 12 d	Bloco 17 d	Bloco 22 d
18	4	Bloco 3 d	Bloco 8 d	Bloco 13 d	Bloco 18 d	Bloco 23 d
19	5	Bloco 4 d	Bloco 9 d	Bloco 14 d	Bloco 19 d	Bloco 24 d

É importante notar que, para uma UP calcular cinco SADs diferentes (em forma de pipeline), é necessário que, a cada ciclo, duas novas amostras estejam disponíveis para a UP. A seleção dos corretos registradores da linha de bloco e dos registradores da linha da área de pesquisa para cada UP é realizada pelos seletores (SB e SP na fig. 5.1).

5.3 Arquitetura do Comparador

O comparador foi desenvolvido para realizar cinco comparações de SADs em paralelo, sendo projetado em um pipeline de cinco estágios. As cinco saídas da linha de SADs (C0 a C4 na fig. 5.1) entregam para o comparador quatro valores de SADs de blocos candidatos em cada ciclo de *clock* e, em cinco ciclos, todos os 25 valores de SAD de uma linha de SAD já foram entregues ao comparador. A arquitetura do comparador deve ser capaz de identificar qual é o menor SAD dentre os 25 SADs da linha de SAD. Quando este menor valor é encontrado, então o comparador compara este valor de SAD com o menor valor dentre os SADs da linha

de SAD anterior. A ligação entre os diversos comparadores pode ser observada na fig. 5.1. Então, o menor SAD dentre todos os SADs calculados até então e o vetor de movimento do bloco candidato que gerou este menor SAD, são disponibilizados na saída para o comparador da próxima linha de SADs ou diretamente para o RRM, quando a comparação é realizada na última linha de SAD.

A fig. 5.4 apresenta a arquitetura proposta para o comparador. Nesta proposta, o comparador é formado, basicamente, por subtratores que subtraem o valor de dois diferentes SADs. Se o resultado é negativo (MSB=1) então o SAD da entrada positiva do subtrator possui uma magnitude menor que o SAD da entrada negativa e, deste modo, o valor da entrada positiva do subtrator deve ser entregue na saída do estágio de comparação.

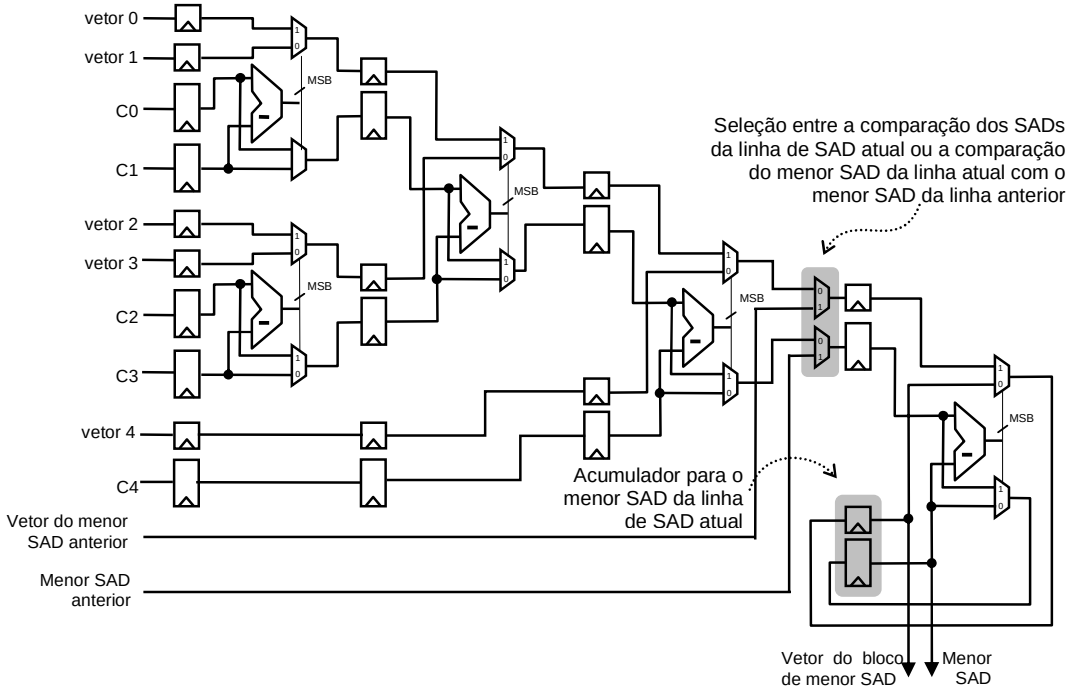


Figura 5.4 – Diagrama em blocos do comparador.

Caso o resultado da subtração seja positivo (MSB=0) então o SAD da entrada negativa do subtrator deve ser entregue na saída. Além dos valores de SAD, o vetor de movimento relativo ao bloco que gerou o menor SAD também deve ser enviado para a saída, para identificar qual bloco gerou o SAD vencedor. Multiplexadores controlados pelo bit de sinal do resultado da subtração selecionam os valores corretos do menor SAD e do respectivo vetor relativo ao bloco de menor

SAD. Esta estrutura de comparações com subtratores funciona porque somente valores positivos são permitidos nas entradas. As entradas são SADs de blocos que, no melhor caso, possuem valor igual a zero.

Os multiplexadores destacados em cinza, na fig. 5.4, são utilizados quando é realizada a comparação do menor SAD da linha de SAD atual, com o menor SAD da linha de SAD anterior. Esta comparação é sempre a última comparação realizada no comparador e, para ser realizada, o SAD de menor magnitude da linha já deve ter sido encontrado.

Os registradores destacados em cinza na fig. 5.4 armazenam o menor SAD da linha de SAD, armazenando o menor SAD entre o último menor SAD e o SAD atual. Estes registradores funcionam como um acumulador do menor SAD da linha. Após testar os 25 blocos candidatos de uma linha de SAD e encontrar o bloco de menor SAD, os registradores destacados em cinza também armazenam o menor SAD entre o resultado da linha de SADs atual e as linhas de SADs anteriores. Deste modo, o valor destes registradores é enviado para o comparador da próxima linha de SADs e, se a linha de SADs for a de número 24 (última linha da matriz) então o resultado do vetor de movimento do bloco de menor SAD dentre todos os blocos candidatos é enviado para o registrador de resultado de memória (RRM na fig. 5.1).

Em cinco ciclos, a arquitetura calcula o menor SAD dentre os cinco primeiros valores de entrada (C0 a C4 na fig. 5.4). Mais quatro ciclos são necessários para que a linha de SAD entregue os 20 valores restantes, completando os 25 valores de SAD candidatos. Só então a arquitetura compara o menor SAD encontrado na linha com o valor do SAD anterior. Assim, no décimo ciclo, um sinal indicando que o resultado da saída está correto é ativado. O valor do SAD enviado a saída, após a ativação do sinal, corresponde ao menor SAD entre os valores de SAD candidatos da linha e o valor de SAD anterior.

5.4 Controle do Estimador de Movimento

O Controle do ME é umas das etapas de maior complexidade no desenvolvimento do estimador, devido à grande quantidade de hardware que necessita de controle. Devido a isto, o controle foi dividido para facilitar a sua implementação. Um bloco de controle local foi agregado à arquitetura da linha de SAD, para controlar a escrita nos registradores internos e também disparar o início

das operações do comparador, ao indicar dado válido em suas saídas. Outro bloco de controle gerencia globalmente a arquitetura, gerando os sinais de controle para o gerenciador de memória e seletores de linha, bem como controlando a reutilização da matriz de linhas de SAD, que é utilizada quatro vezes para cobrir toda a área de pesquisa.

O bloco de controle local, além de gerar os sinais para os registradores internos da linha de SAD (ACC0 a ACC24 na fig. 5.3) e controlar as realimentações para a acumulação do SAD total para o bloco candidato, também é responsável por disparar as operações na linha de SAD seguinte. Ao término dos cálculos de uma linha do bloco atual, o controle gera um sinal para disparar as operações na linha seguinte. Assim, as próprias linhas de SAD são as responsáveis pelo gerenciamento do pipeline entre as linhas de SAD. Ao final das operações da linha, ou seja, após o cálculo do SAD para as oito linhas do bloco atual, o controle local gera um sinal que dispara as operações do comparador.

O controle global gera os sinais que controlam o gerenciador de memória, os seletores de linha de bloco e de pesquisa e também a reutilização da matriz de linhas de SAD. É tarefa do controle global determinar quando o gerenciador de memória dever ser ativado e quando deve estar inativo, bem como determinar a carga nos registradores RLP e RLB ilustrados na fig. 5.1. Ele também gera os sinais de *reset* para as linhas de SAD, necessários para zerar os acumuladores das linhas, após o término das operações em uma das memórias de pesquisa. Outra tarefa do controle global é controlar a escrita no registrador RRM, que deverá armazenar os resultados para as quatro memórias de pesquisa, bem como disparar o comparador para determinar o menor SAD dentre os quatro valores armazenados no RRM.

Esta divisão do controle facilitou a sua implementação pois, desta forma, o controle global não precisa gerar sinais de controle para as etapas internas das linhas de SAD, reduzindo a sua complexidade e, conseqüentemente, o seu tamanho.

5.5 Resultados de Síntese

Nesta seção serão apresentados os resultados de síntese para as arquiteturas apresentadas neste capítulo. Todas as arquiteturas foram descritas em VHDL (ASHENDEN, 1998) e sintetizadas em FPGAs. Inicialmente, as arquiteturas foram direcionadas para o FPGA StratixII EP2S30F484C3 da Altera, utilizando a ferramenta de síntese QuartusII 5.0, também da Altera (ALTERA, 2005).

A tab. 5.2 apresenta os resultados de síntese obtidos com a ferramenta QuartusII 5.0, para os principais blocos do estimador.

Pode-se observar que o bloco que utiliza mais recursos de hardware é a linha de SAD e também é o bloco que atingiu a menor freqüência de operação. Todos os demais blocos atingiram freqüências superiores, sendo que alguns blocos, como a UP, por exemplo, chegaram a atingir 500MHz, que é a freqüência máxima do dispositivo. A tab. 5.2 mostra os resultados de síntese para os principais blocos do estimador, no entanto, a ferramenta Quartus II, em função de algum bug não identificado, não conseguiu sintetizar o código para os maiores blocos, como a matriz de SAD que reúne as 25 linhas de SAD, bem como o estimador, que reúne a matriz de SAD e mais os blocos do gerenciador de memórias e seletor de linha de pesquisa, formando a arquitetura completa do estimador de movimento.

Tabela 5.2 – Resultados de síntese para ferramenta QuartusII.

Bloco	ALUTs	Registradores	Freqüência (MHz)	Bits de Memória
Controle Global	141	56	311,43	-
Controle Local	31	15	302,21	-
UP	66	45	500,00	-
Linha de SAD	918	590	149,68	-
Comparador	499	457	213,63	-
Gerenciador de Memória	420	22	170,13	32286
Seletor de Linha de Bloco	48	5	500,00	-
Seletor de Linha de Pesquisa	691	5	500,00	-
Gerador de vetores	26	3	500,00	
Matriz de SAD	-	-	-	-
Estimador	-	-	-	-

Dispositivo StratixII EP2S30F484C3

Devido a este problema de síntese da ferramenta, as arquiteturas foram direcionadas para outro dispositivo, o FPGA Virtex-II Pro XC2VP70 (XILINX, 2005) da Xilinx, utilizando a ferramenta ISE, também da Xilinx (XILINX, 2006). A tab. 5.3 ilustra os resultados obtidos para a síntese na ferramenta ISE.

Com a ferramenta ISE foi possível gerar os resultados de síntese também para a matriz de SAD e o estimador completo. Um resultado um tanto curioso é que a matriz de SAD utiliza mais elementos lógicos (LUTs para o Virtex-II Pro) do que o estimador completo. Isto se deve ao fato de que a matriz de SAD possui vários pinos de entrada a mais que o estimador, como as palavras da área de pesquisa e de busca, que são lidas pelo gerenciador de memória e entregues à matriz de SAD. O estimador não possui estes pinos de entrada, pois o gerenciador de memória é um dos seus blocos. Isto causa uma utilização adicional de LUTs, que são usadas como conexões entre as entradas e os componentes internos da arquitetura. Em geral, não é possível prever com exatidão os resultados de síntese das arquiteturas, a partir dos blocos que as compõe, tendo em vista que as ferramentas de síntese possuem mecanismos de otimização que podem reduzir significativamente a utilização de recursos. As ferramentas de síntese também podem se utilizar de elementos pré-definidos dentro do FPGA para implementar otimizações, dependendo das características da arquitetura.

Tabela 5.3 – Resultados de síntese para ferramenta ISE.

Bloco	LUTs	Registradores	Freqüência (MHz)	Bits de Memória
Controle Global	157	97	267,2	-
Controle Local	23	25	290,6	-
UP	45	45	381,9	-
Linha de SAD	350	350	357,0	-
Comparador	235	462	224,6	-
Gerenciador de Memória	605	41	284,0	32.286
Seletor de Linha de Bloco	63	36	1,025,6	-
Seletor de Linha de Pesquisa	589	106	1,008,9	-
Gerador de vetores	16	38	909,8	
Matriz de SAD	27.265	28.212	173,9	-
Estimador	19.695	19.856	228,0	32.286

Dispositivo Virtex-II Pro XC2VP70

Um dos resultados mais importantes mostrados na tab. 5.3 é a freqüência de operação da arquitetura do estimador. Com a freqüência de 228MHz, e gerando um novo vetor a cada 2.615 ciclos, a arquitetura do estimador pode gerar mais de 87 mil vetores de movimento por segundo. Como cada vetor de movimento se refere a um bloco, que para esta arquitetura é de 16x16 amostras, mais de 22 milhões de amostras podem ser processadas a cada segundo.

Para melhor visualizar estas informações, a tab. 5.4 mostra a taxa de processamento do estimador para alguns dos padrões de vídeo existentes. Os resultados mostram que a arquitetura desenvolvida neste trabalho pode atingir tempo real para a maioria dos padrões de vídeo existentes, exceto para HDTV (1920 x 1080), onde obteve uma taxa de processamento inferior a 30 quadros por segundo.

Tabela 5.4 – Taxa de processamento da arquitetura.

Padrão	Quadros por Segundo
QCIF (176 x 144)	880,3
CIF (352 x 268)	220,0
VGA (640 x 480)	72,6
SDTV (720 x 480)	64,7
HDTV (1920 x 1080)	10,7

Outro resultado importante foi que a arquitetura completa do estimador utilizou 19,7 mil elementos lógicos, isto implica em cerca de 28% dos elementos lógicos do dispositivo alvo. Este é um bom resultado, tendo em vista que a arquitetura opera sobre uma área de pesquisa de 64x64 amostras.

5.6 Resultados Comparativos

Esta seção tem como objetivo comparar os resultados de síntese obtidos entre a arquitetura desenvolvida e a arquitetura base, desenvolvida na UFRN. A arquitetura base utiliza o algoritmo de busca *Full Search* com o critério de similaridade SAD e opera sobre blocos de 16x16 em uma área de pesquisa de 32x32 amostras. Nesta arquitetura, são necessários 333 ciclos de *clock* para que um novo vetor de movimento seja gerado. A arquitetura desenvolvida neste trabalho utiliza o algoritmo de busca *Pel Decimation* 4:1 com o SAD como critério de

similaridade e também opera sobre blocos de 16x16. No entanto, a área de pesquisa foi quadruplicada, tratando 64x64 amostras.

Outra diferença importante entre as duas soluções está na arquitetura da UP, que para a arquitetura base calcula ½ da linha do bloco candidato e que na arquitetura desenvolvida processa ¼ da linha do bloco. Esta opção reduz significativamente a quantidade de hardware utilizado na UP, mas gera um número duas vezes maior de realimentações no cálculo de cada bloco candidato.

A tab. 5.5 mostra os resultados de síntese da ferramenta ISE, direcionados ao FPGA Virtex-II Pro XC2VP70 da Xilinx, para as duas arquiteturas.

Tabela 5.5 – Resultados comparativos.

Bloco	LUTs	Freqüência (MHz)	Throughput (Msamples/s)
Arquitetura Base	37.561	172,1	132
Arquitetura Desenvolvida	19.695	228,0	22

Dispositivo Virtex-II Pro XC2VP70

A arquitetura base pode operar a uma freqüência de 172,1MHz, podendo processar mais de 132 milhões de amostras por segundo. Este resultado possibilita a utilização da arquitetura para aplicações HDTV em tempo real. No entanto, esta arquitetura opera sobre uma área de pesquisa de 32x32 amostras e, mesmo assim, utiliza mais de 37,5 mil elementos lógicos, o que resulta em mais de 50% dos recursos de hardware do FPGA alvo, que é um dos maiores dispositivos da atualidade.

Vale ressaltar que a arquitetura desenvolvida neste trabalho opera sobre uma área de 64x64 amostras, o que resulta em 2401 blocos candidatos, ao invés dos 289 blocos candidatos da área de pesquisa de 32x32 amostras da arquitetura base. Isto implica em um aumento significativo no tamanho da arquitetura, no entanto, a adoção do algoritmo *Pel Decimation* 4:1, juntamente com as simplificações da UP, possibilitaram uma redução significativa da utilização de recursos de hardware do FPGA, que ficou em torno de 28% dos recursos do dispositivo. No entanto, as simplificações que resultaram em uma redução do hardware utilizado pela arquitetura causou uma queda de desempenho que impossibilita a utilização da arquitetura para HDTV em tempo real, para o dospositivo utilizado neste trabalho. Utilizando outras possibilidades de implementação, como

ASIC por exemplo, ou até mesmo uma nova geração de FPGAs, esta mesma arquitetura pode alcançar a taxa de processamento necessária para processar vídeos HDTV em tempo real.

Considerando que a área de pesquisa é quatro vezes maior do que a solução base e considerando que os dados da área de pesquisa são processados através de quatro realimentações sobre a matriz de SADs (memórias PP, IP, II e IP) enquanto que a solução base não utiliza nenhuma realimentação, então é possível estimar que a arquitetura desenvolvida neste trabalho teria um desempenho quatro vezes mais lento que a arquitetura base, considerando a mesma freqüência de operação.

Por outro lado, considerando que a arquitetura proposta processa ¼ da linha do bloco em cada passada pela UP, enquanto que a arquitetura base processa ½ linha do bloco em cada passada, foi possível estimar que o processamento de um bloco será realizado com o dobro de ciclos de clock, fazendo com que a performance da arquitetura proposta neste trabalho seja duas vezes menor do que a arquitetura base.

Por fim, considerando que o cálculo das UPs está inserido no cálculo das linhas de SAD e, por conseqüência, nos cálculos da matriz de SAD, foi possível estimar que a arquitetura proposta neste trabalho seria até oito vezes mais lenta do que a arquitetura base.

Os resultados de síntese, fundamentalmente em função da freqüência de operação mais elevada atingida pela arquitetura proposta neste trabalho em relação ao trabalho base, indicaram que o desempenho da arquitetura proposta neste trabalho possui um *throughput* apenas seis vezes inferior.

É importante salientar que mesmo com este *throughput* inferior, a arquitetura desenvolvida neste trabalho também atinge um desempenho elevado, sendo suficiente para codificar vídeos SDTV (720 x 480 pixels) em tempo real. Além disso, a área de busca foi quadruplicada e mesmo assim os recursos de hardware utilizados foram reduzidos quase pela metade em relação à solução base.

6. Conclusões

Esta monografia apresentou um estudo sobre diversos algoritmos de estimação de movimento direcionados à compressão de vídeos digitais. Também foram apresentados os resultados de implementações em software de alguns destes algoritmos. Por fim, este trabalho apresentou o desenvolvimento de uma arquitetura para a estimação de movimento utilizando o algoritmo *Pel Decimation* 4:1, bem como os seus resultados de síntese para FPGAs.

Foram implementados seis algoritmos em linguagem C, para que seus resultados formassem a base para a definição do algoritmo que seria utilizado na implementação da arquitetura. Os resultados das implementações em software mostraram que o algoritmo *Full Search* gera os vetores de movimento que resultam no menor erro, no entanto é o algoritmo mais lento dentre todos os algoritmos estudados. As variações do algoritmo *Pel Decimation*, 4:1 e 2:1, obtiveram os melhores resultados na relação entre o custo computacional e o desempenho. Devido a isto e em função das facilidades de implementação em hardware, o algoritmo *Pel Decimation* 4:1 foi o escolhido para a implementação da arquitetura.

A arquitetura desenvolvida neste trabalho foi baseada em uma arquitetura que utiliza o algoritmo *Full Search* e foi desenvolvida na UFRN. A arquitetura foi descrita em VHDL e sintetizada para FPGAs. A arquitetura desenvolvida trabalha com blocos de 16x16 amostras e opera sobre uma área de pesquisa de 64x64 amostras. Os resultados de síntese indicam que esta arquitetura é capaz de gerar mais de 87 mil vetores de movimento por segundo, o que implica no processamento de mais de 22 milhões de amostras a cada segundo. Com estes resultados de desempenho a arquitetura desenvolvida pode codificar com folga vídeos SDTV em tempo real. Outro resultado importante foi a diminuição na utilização de recursos, se comparado a arquitetura base, que chegou a mais de 47%.

Como trabalhos futuros pretende-se concluir a validação da arquitetura, que não foi possível por que as ferramentas necessárias para a validação não estavam disponíveis no laboratório utilizado para o desenvolvimento deste trabalho.

Também pretende-se aumentar o paralelismo da UP, para que cada UP possa processar ½ linha do bloco, ao invés de processar ¼ de linha. Esta modificação permitirá dobrar o desempenho da arquitetura e, com mais alguns pequenos ajustes arquiteturais, é provável que esta solução atinja o desempenho exigido para HDTV (1920 x 1080 pixels) em tempo real. Neste caso, a quantidade de hardware utilizada, segundo estimativas, será próxima a 26 mil LUTs, com um aumento superior a 6 mil LUTs causado por esta modificação. Ainda assim, a arquitetura modificada utilizaria cerca de 30 % menos LUTs do que a arquitetura base.

Outra possibilidade é diminuir a área de pesquisa para fazer com que a arquitetura atinja os requisitos de HDTV em tempo real. Esta diminuição da área de pesquisa, além de melhorar o desempenho, pode reduzir ainda mais o tamanho da arquitetura.

Outra proposta de trabalho futuro é fazer com que a arquitetura opere sobre múltiplos tamanhos de bloco. Neste caso o estimador não operaria apenas sobre blocos de 16x16 e sim sobre blocos de tamanhos variados, como 16x8, 8x8, 8x4, 4x4 entre outros. A escolha de qual tamanho de bloco deve ser utilizado é feita pelo estimador e pode variar de acordo com a quantidade de movimento da região da imagem que está sendo processada.

Referências

ALTERA Corporation, “Altera: The Programmable Solutions Company”. Disponível em: <http://www.altera.com>, Acesso em: dez. 2005.

ASHENDEN, Peter J. **The Student’s Guide to VHDL**. San Francisco: Morgan Kaufmann, 1998. 312p.

BHASKARAN, Vasudev; KONSTANTINIDES, Konstantinides**. Image and video compression standard: algorithms and architectures**. 2. ed. Massachusetts: Kluwer Academic Publisher, 1999. 454p.

BLOODSHED Software – DEV-C++ , “Providing Free Software to the Internet Community”. Disponível em: <http://www.bloodshed.net/devcpp.html>, Acesso em out. 2005.

FEDORA Core Test Page. Disponível em: <http://ginfo05.dee.ime.eb.br/>, Acesso em out 2005.

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

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

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

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

MIANO, J. **Compressed Image File Formats: JPEG, PNG, GIF, XBM, BMP**. New York: Assison-Wesley, 1999.

MINISTÉRIO DAS COMUNICAÇÕES. Sistema de TV Digital. Brasília: MC, 2005. Disponível em: <http://sbtvd.cpqd.com.br/>. Acesso em: 30 nov. 2005.

RICHARDSON, I. **H.264 and MPEG-4 Video Compression – Video Coding for Next-Generation Multimedia**. Chichester: John Wiley and Sons, 2003. 281p.

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

KERNIGHAN, Brian W.; RITCHIE, Dennis M. **C: a Linguagem de Programação Padrão Ansi**. Rio de Janeiro: Campus, 1999.

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

VASSILIADIS, S.; et all. The Sum-Absolute-Difference Motion Estimation Accelerator. In: 24TH EUROMICRO CONFERENCE. **Proceedings...** [S.l.]: IEEE, 1998, v. 2, p. 559-566.

XILINX INC. **Xilinx University Program – Virtex-II Pro Development System – Hardware Reference Manual**. [S.l.], 2005. Disponível em: <www.digilentinc.com>. Acesso em: dez. 2005.

XILINX INC. Xilinx: “The Programmable Logic Company”. Disponível em: <www.xilinx.com>. Acesso em: jan. 2006.

ZANDONAI, Diogo. **Uma Arquitetura de Hardware para Estimação de Movimento aplicada à Compressão de Vídeo Digital**. Porto Alegre, 2003. 104f. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal do Rio Grande do Sul.

ANEXO A – Publicações Durante o Curso de Graduação

Este anexo apresenta as publicações referentes aos trabalhos de pesquisa que foram realizados durante o curso de gradução. Os diversos trabalhos desenvolvidos junto ao grupo de arquiteturas e circuitos intergrados (GACI), foram publicados em diveros eventos e revistas da área. Grande parte dos trabalhos publicados não tem relação direta com o tema abordado nesta monografia, no entanto, estes trabalhos deram o embasamento necessário para o desenvolvimento deste trabalho de conclusão. Os trabalhos publicados serão apresentados em ordem de-crescente da data de publicação.

Artigos Publicados revistas:

1- Título: Arquiteturas de Cálculo da DCT 2-D para Codecs HDTV Seguindo o Padrão H.264/AVC.

Autores: PORTO, Marcelo Schiavon; ROSA, Leandro Zanetti Paiva da; SILVA, Thaísa Leal Sa; PORTO, Roger Endrigo Carvalho; GÜNTZEL, José Luis Almada; BAMPI, Sergio; SILVA, Ivan Saraiva da; AGOSTINI, Luciano Volcan.

Evento: Revista Hífen.

Ano: 2005.

2- Título: Comparison between OCP, PVCI and BVCI Hardware Reuse Interfaces Designed in VHDL and Mapped to FPGAs.

Autores: SILVA, Thaísa Leal da; PORTO, Marcelo Schiavon; ROSA, Leandro Zanetti Paiva da; PORTO, Roger Endrigo Carvalho; GÜNTZEL, José Luis Almada; SILVA, Ivan Saraiva da; BAMPI, Sergio; AGOSINI, Luciano Volcan.

Evento: Revista Hífen.

Ano: 2005.

Artigos Publicados em eventos internacionais:

1- Título: Ivestigação de Algorítmos e Proposta Arquitetural para a Estimação de Movimento Direcionada à Vídeos de Alta Resolução.

Autores: PORTO, Marcelo Schiavon; GÜNTZEL, José; SILVA, Ivan; BAMPI, Sergio; AGOSTINI, Luciano Volcan.

Evento: XII WORKSHOP IBERCHIP, San José, Costa Rica.

Ano: 2006.

2- Título: Quantização Direta e Inversa de Alta Performance para a Compressão de Vídeo H.264/AVC direcionada para HDTV.

Autores: PORTO, Marcelo Schiavon; PORTO, Roger; GÜNTZEL, José; SILVA, Ivan; BAMPI, Sergio; AGOSTINI, Luciano Volcan.

Evento: XII WORKSHOP IBERCHIP, San José, Costa Rica.

Ano: 2006.

3- Título: Exploração no Espaço de Projeto da Hadamard 4x4 Direta do Padrão de Compressão de Vídeo H.264/AVC.

Autores: SILVA, André Marcelo Coelho da; SILVA, Thaísa Leal da; PORTO, Marcelo Schiavon; PORTO, Roger Endrigo Carvalho; GÜNTZEL, José Luís Almada; SILVA, Ivan Saraiva; BAMPI, Sergio; AGOSTINI, Luciano Volcan.

Evento: XII WORKSHOP IBERCHIP, San José, Costa Rica.

Ano: 2006.

4- Título: Forward and Inverse 2-D DCT Architectures for H.264/AVC Video Compression Directed to HDTV.

Autores: PORTO, Roger Endrigo Carvalho; PORTO, Marcelo Schiavon; SILVA, Thaísa Leal da; ROSA, Leandro Zanetti Paiva da; GÜNTZEL, José Luís Almada; BAMPI, Sergio; SILVA, Ivan Saraiva; AGOSTINI, Luciano Volcan.

Evento: 2nd SOUTHERN CONFERENCE ON PROGRAMMABLE LOGIC, Mar Del Plata, Argentina.

Ano: 2006.

Prêmio: ALTERA Best Paper Award SPL 2006, Universidad CAECE Mar del Plata.

5- Título: Design Space Exploration on the H.264 4x4 Hadamard Transform.

Autores: PORTO, Marcelo Schiavon; SILVA, Thaísa Leal da; PORTO, Roger Endrigo Carvalho; AGOSTINI, Luciano Volcan.

Evento: XXIII NORCHIP CONFERENCEOulu. XXIII, Oulu, Finlândia.

Ano: 2005.

6- Título: Projeto, Síntese e Simulaçao das Interfaces de Reuso do Padrão OCP (Open Core Protocol).

Autores: PORTO, Marcelo Schiavon; PORTO, Roger Endrigo Carvalho; GÜNTZEL, José Luís Almada; AGOSTINI, Luciano Volcan.

Evento: XI WORKSHOP IBERCHIP, Salvador, Brasil.

Ano: 2005.

7- Título: Impactos so Uso de Diferentes Arquiteturas de Somadores em FPGAs Altera.

Autores: PORTO, Marcelo Schiavon; SILVA, André Marcelo Coelho da; PORTO, Roger Endrigo Carvalho; GÜNTZEL, José Luís Almada; AGOSTINI, Luciano Volcan.

Evento: XI WORKSHOP IBERCHIP, Salvador, Brasil.

Ano: 2005.

Artigos Publicados em eventos nacionais:

1- Título: A Hardware Architecture for Motion Estimation Using Pel Decimation 4:1 and SAD.

Autores: PORTO, Marcelo; GÜNTZEL, José; BAMPI, Sergio; SILVA, Ivan; AGOSTINI, Luciano.

Evento: XXI SIMPÓSIO SUL DE MICROELETRÔNICA, Porto Alegre, Brasil.

Ano: 2006.

2- Título: Design Space Exploration on theH.264 4x4 Hadamard Transform.

Autores: SILVA, André; PORTO, Roger; PORTO, Marcelo; SILVA, Thaísa; GÜNTZEL, José; BAMPI, Sergio; SILVA, Ivan; AGOSTINI, Luciano.

Evento: XXI SIMPÓSIO SUL DE MICROELETRÔNICA, Porto Alegre, Brasil.

Ano: 2006.

3- Título: FPGA Prototype of a H.264/AVC Inverse Transform and Quantization Architecture for HDTV.

Autores: PORTO, Roger; PORTO, Marcelo; AGOSTINI, Luciano GÜNTZEL, José; SILVA, Ivan; BAMPI, Sergio;.

Evento: XXI SIMPÓSIO SUL DE MICROELETRÔNICA, Porto Alegre, Brasil.

Ano: 2006.

4- Título: Design, Synthesis and Simulation of OCP (Open Core Protocol) Hardware Reuse Interface.

Autores: PORTO, Marcelo Schiavon; ROSA, Leandro Zanetti Paiva da; PORTO, Roger Endrigo Carvalho; GÜNTZEL, José Luís Almada; AGOSTINI, Luciano Volcan.

Evento: XX SIMPÓSIO SUL DE MICROELETRÔNICA, Santa Cruz do Sul, Brasil.

Ano: 2005.

5- Título: Impacts of Different Adder Architectures in Designs Directed to FPGAs.

Autores: PORTO, Marcelo Schiavon; SILVA, André Marcelo Coelho da; GÜNTZEL, José Luís Almada; AGOSTINI, Luciano Volcan.

Evento: XX SIMPÓSIO SUL DE MICROELETRÔNICA, Santa Cruz do Sul, Brasil.

Ano: 2005.

6- Título: An Integer 2-D DCT Architecture for the H.264/AVC Video Coding Standard.

Autores: PORTO, Roger Endrigo Carvalho; PORTO, Marcelo Schiavon; SILVA, Thaísa Leal da; ROSA, Leandro Zanetti Paiva da; GÜNTZEL, José Luís Almada; AGOSTINI, Luciano Volcan.

Evento: XX SIMPÓSIO SUL DE MICROELETRÔNICA, Santa Cruz do Sul, Brasil.

Ano: 2005.

7- Título: Experiments With OCP (Open Core Protocol) Hardware Reuse Interface.

Autores: PORTO, Marcelo Schiavon; AGOSTINI, Luciano Volcan.

Evento: XIX SIMPÓSIO SUL DE MICROELETRÔNICA, São Miguel das Missões, Brasil.

Ano: 2004.

Resumos publicados em eventos:

1- Título: Design and Comparison between PVCI, BVCI and OCP Hardware Reuse Interfaces Mapped to FPGA.

Autores: PORTO, Marcelo Schiavon; SILVA, Thaísa L da; PORTO, Roger Endrigo Carvalho; GÜNTZEL, José Luiz Almada; AGOSTINI, Luciano Volcan.

Evento: V STUDENT FORUM ON MICROELECTRONICS, Florianópolis, Brasil.

Ano: 2005.

2- Título: Algoritmos e Arquiteturas para a Estimação de Movimento na Compressão de Vídeos Digitais.

Autores: PORTO, Marcelo Schiavon; AGOSTINI, Luciano Volcan.

Evento: XIV CONGRESSO DE INICIAÇÃO CIENTÍFICA DA UFPEL, Pelotas, Brasil.

Ano: 2005.

3- Título: Desenvolvimento de Hardware Dedicado para a Quantização Direta e Inversa segundo o Padrão de Compressão de Vídeo H.264.

Autores: PORTO, Marcelo Schiavon; PORTO, Roger Endrigo Carvalho; AGOSTINI, Lucinao Volcan.

Evento: XIV CONGRESSO DE INICIAÇÃO CIENTÍFICA DA UFPEL, Pelotas, Brasil.

Ano: 2005.

4- Título: Comparação entre as Interfaces OCP, PVCI e BVCI para o Reuso de Blocos de Hardware Implementadas em VHDL e Mapeadas para FPGAs.

Autores: SILVA, Thaísa Leal da; PORTO, Marcelo Schiavon; AGOSTINI, Luciano Volcan.

Evento: XIV CONGRESSO DE INICIAÇÃO CIENTÍFICA DA UFPEL, Pelotas, Brasil.

Ano: 2005.

5- Título: Exploração do Espaço de Projeto de Hardware para as Transformadas Discretas do Padrão de Compressão de Vídeo H.264.

Autores: ROSA, Leandro Zanetti Paiva da; PORTO, Roger Endrigo Carvalho; PORTO, Marcelo Schiavon; SILVA, Thaísa Leal da; SILVA, André Marcelo Coelho da; AGOSTINI, Luciano Volcan.

Evento: XIV CONGRESSO DE INICIAÇÃO CIENTÍFICA DA UFPEL, Pelotas, Brasil.

Ano: 2005.

6- Título: Descrição e Validação das Interfaces de Reuso do Padrão OCP (Open Core Protocol).

Autores: PORTO, Marcelo Schiavon; ROSA, Leandro Zanetti Paiva da; AGOSTINI, Luciano Volcan.

Evento: XIII CONGRESSO DE INICIAÇÃO CIENTÍFICA DA UFPEL, Pelotas, Brasil.

Ano: 2004.

Prêmio: Prêmio Jovem Pesquisador - Primero lugar na área de engenharias para o trabalho apresentado no XIII CIC (pôster).

7- Título: Experimentos com o Padrão de Reuso de Hardware OCP (Open Core Protocol).

Autores: PORTO, Marcelo Schiavon; AGOSTINI, Luciano Volcan.

Evento: X WORKSHOP IBERCHIP, Cartagena de Índias, Colômbia.

Ano: 2004.

8- Título: Estudo Sobre o Padrão de reuso de Hardware OCP (Open Core Protocol).

Autores: PORTO, Marcelo Schiavon; AGOSTINI, Luciano Volcan.

Evento: XII CONGRESSO DE INICIAÇÃO CIENTÍFICA DA UFPEL, Pelotas, Brasil.

Ano: 2003.