UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ROGER ENDRIGO CARVALHO PORTO

Desenvolvimento Arquitetural para Estimação de Movimento de Blocos de Tamanhos Variáveis Segundo o Padrão H.264/AVC de Compressão de Vídeo Digital

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Sergio Bampi
Orientador

Porto Alegre, junho de 2008.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Porto, Roger Endrigo Carvalho

Desenvolvimento Arquitetural para Estimação de Movimento de Blocos de Tamanhos Variáveis Segundo o Padrão H.264/AVC de Compressão de Vídeo Digital / Roger Endrigo Carvalho Porto – Porto Alegre: Programa de Pós-Graduação em Computação, 2008.

96 p.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação. Porto Alegre, BR – RS, 2008. Orientador: Sergio Bampi.

1. Compressão de Vídeo. 2. H.264/AVC. 3. Estimação de Movimento. I. Bampi, Sergio. II. Título.

AGRADECIMENTOS

Aos meus pais, José Sotero e Norma: pelo exemplo de perseverança e humildade; por não se dobrarem nunca diante das dificuldades que a vida impõe; e por serem, assim, os grandes motivadores de tudo o que faço.

À minha irmã, Aline, uma grande amiga que tenho: pelo companheirismo; pelas horas de mate; e pelas conversas sobre "nuestra tierra sureña", "folklore" e assuntos vermelhos.

À Marina Mota, minha noiva: pelo apoio incondicional; pelo carinho; por compartilhar meus sonhos; por não deixar que as desilusões me vençam; por ser essa mulher incrível. Tudo teria sido mais difícil sem ela.

Ao professor Sergio Bampi, meu orientador, pessoa que muito admiro e respeito: por ter me aceitado como aluno de mestrado; pelo apoio e incentivo; e por ter sido tolerante em relação as minhas ausências. Sinto-me feliz em participar de um grupo de trabalho liderado por uma pessoa experiente como ele.

Ao professor Luciano Agostini: por ser praticamente o co-orientador deste trabalho; por ter sido um amigo essencial nesta caminhada estando presente desde o início dela, sempre disposto a ajudar, das mais diversas formas.

Aos colegas da UFRGS, Marcelo Porto, Bruno Zatt, Cláudio Diniz, Dieison Deprá, Thaísa da Silva, Guilherme de Freitas, Vagner da Rosa: pelo convívio; pela amizade e pelas conversas da "hora do café", às vezes técnicas, às vezes cômicas, mas sempre proveitosas.

Aos colegas da UFPel; Leandro e Fabiane; pela ajuda nas avaliações em software, avaliações estas que foram essências no meu trabalho.

Aos professores da UFRGS, Altamiro Susin, Ricardo Reis, Flávio Wagner, Luigi Carro, Fernanda Kastensmidt, Marcelo Lubaszewski, e Érika Cota: por terem colaborado na minha formação neste período de mestrado.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

2-D	*Two Dimensional*
ASIC	*Application Specific Integrated Circuit*
AVC	*Advanced Video Coding*
B	*Bi-predictive*
BS	*Block Subsampling*
BRAM	*Block RAM*
CIF	*Common Intermediate Format*
CMOS	*Complementary Metal Oxide Semicondutor*
FPS	*Frame per second*
FS	*Full Search*
FPGA	*Field Programmable Gate Array*
HD	*High Definition*
HDTV	*High Definition Digital Television*
HHR	*Half Horizontal Resolution*
HJTC	*HeJian Technology Corporation*
I	*Inter*
IEEE	*Institute of Electric and Electronics Engineers*
IOB	*Input / Output Block*
ISO	*International Organization for Standardization*
ITU-T	*International Telecommunication Union - Telecommunication*
JVT	*Joint Video Team*
LUT	*Logic Unit*
MAE	*Mean Absolute Error*
MC	*Motion Compensation*
ME	*Motion Estimation*
MPEG	*Moving Picture Experts Group*
MSE	*Mean Square Error*

NAL	*Network Adaptation Layer*
P	*Predictive*
PS	*Pel Subsampling*
Q	*Quantization*
Q^{-1}	*Inverse Quantization*
QCIF	*Quarter CIF*
QVGA	*Quarter VGA*
RAM	*Random Access Memory*
RCB	Registrador de Linha de Bloco Atual
RSA	Registrador de Linha de Área de Busca
SAE	*Sum of Absolute Errors*
SAD	*Sum of Absolute Differences*
SD	*Standard Definition*
SDTV	*Standard Definition Television*
SI	*Switching I*
SIF	*Source Input Format*
SP	*Switching P*
SQCIF	*Sub Quarter CIF*
SVGA	*Super VGA*
SXGA	*Super XGA*
T	*Transform*
T^{-1}	*Inverse Transform*
TSMC	*Taiwan Semiconductor Manufacturing Company*
UMC	*United Microelectronics Corporation*
UP	Unidade de Processamento
VBSME	*Variable Block Size Motion Estimation*
VCEG	*Video Coding Experts Group*
VGA	*Video Graphics Array*
VQEG	*Video Quality Experts Group*
VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuit*
XGA	*Extended Graphics Array*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Apesar de as capacidades de transmissão e de armazenamento dos dispositivos continuarem crescendo, a compressão ainda é essencial em aplicações que trabalham com vídeo. Com a compressão reduz-se significativamente a quantidade de bits necessários para se representar uma seqüência de vídeo.

Dentre os padrões de compressão de vídeo digital, o mais novo é o H.264/AVC. Este padrão alcança as mais elevadas taxas de compressão se comparado com os padrões anteriores mas, por outro lado, possui uma elevada complexidade computacional. A complexidade computacional elevada dificulta o desenvolvimento em software de aplicações voltadas a definições elevadas de imagem, considerando a tecnologia atual. Assim, tornam-se indispensáveis implementações em hardware.

Neste escopo, este trabalho aborda o desenvolvimento de uma arquitetura para estimação de movimento de blocos de tamanhos variáveis segundo o padrão H.264/AVC de compressão de vídeo digital. Esta arquitetura utiliza o algoritmo *full search* e SAD como critério de similaridade. Além disso, a arquitetura é capaz de gerar os 41 diferentes vetores de movimento referentes a um macrobloco e definidos pelo padrão.

A solução arquitetural proposta neste trabalho foi descrita em VHDL e mapeada para FPGAs da Xilinx. Também foi desenvolvida uma versão *standard cell* da arquitetura. Considerando-se as versões da arquitetura com síntese direcionada para FPGA, os resultados mostraram que a arquitetura pode ser utilizada em aplicações voltadas para alta definição como SDTV ou HDTV. Para a versão *standard cells* da arquitetura os resultados indicam que ela pode ser utilizada para aplicações SDTV.

Palavras-Chave: Compressão de Vídeo, H.264/AVC, Estimação de Movimento.

Architectural Design for Variable Block-Size Motion Estimation of the H.264/AVC Digital Video Compression Standard

ABSTRACT

The transmission and storage capabilities of the digital communications and processing continue to grow. However, compression is still necessary in video applications. With compression, the amount of bits necessary to represent a video sequence is dramatically reduced.

Amongst the video compression standards, the latest one is the H.264/AVC. This standard reaches the highest compression rates when compared to the previous standards. On the other hand, it has a high computational complexity. This high computational complexity makes it difficult the development of applications targeting high definitions when a software implementation running in a current technology is considered. Thus, hardware implementations become essential.

Addressing the hardware architectures, this work presents the architectural design for the variable block-size motion estimation defined in the H.264/AVC standard. This architecture is based on full search motion estimation algorithm and SAD calculation. This architecture is able to produce the 41 motion vectors within a macroblock that are specified in the standard.

The architecture designed in this work was described in VHDL and it was mapped to Xilinx FPGAs. Extensive simulations of the hardware architecture and comparisons to the software implementation of the same variable-size algorithm were used to validate the architecture. It was also synthesized to standard cells. Considering the synthesis results, the architecture reaches real time for high resolution videos, as HDTV when mapped to FPGAs. The standard cells version of this architecture is able to reach real time for SDTV resolution, considering a physical synthesis to 0.18μm CMOS.

Keywords: Video Compression, H.264/AVC, Motion Estimation.

1 INTRODUÇÃO

Um dos assuntos mais explorados atualmente, tanto pela academia quanto pela indústria, e que possui um grande número de aplicações é a compressão de vídeos digitais. Com ela é possível reduzir drasticamente a quantidade de bits utilizada para representar as seqüências de vídeo facilitando o armazenamento ou a transmissão das mesmas. Entre as aplicações onde a compressão de vídeos digitais é empregada estão, por exemplo, televisores digitais de alta resolução, filmadoras digitais portáteis, DVD *players*, computadores pessoais e portáteis, telefones celulares, e vários outros.

A compressão de vídeos está baseada em três tipos de redundância: entrópica, espacial e temporal (AGOSTINI, 2007). A redundância entrópica diz respeito a como os símbolos são codificados, explorando a probabilidade de ocorrências dos símbolos, entre outras características. A redundância espacial ou (intraquadro) diz respeito à redundância existente dentro de um mesmo quadro, ou seja, à tendência de que *pixels* vizinhos possuam valores semelhantes. Já a redundância temporal (interquadros) trata de explorar a semelhança entre blocos de *pixels* que variam pouco de um quadro para outro. Para tratar eficientemente da redução da redundância temporal, os padrões de compressão de vídeos digitais empregam técnicas de estimação e compensação de movimento.

Dentre os padrões de compressão de vídeo, o mais novo é o H.264/AVC (ITU-T, 2003). O H.264/AVC foi desenvolvido na tentativa de criar-se um padrão com uma taxa de compressão superior aos padrões anteriores. Este objetivo foi alcançado, mas, como resultado, houve um grande aumento na complexidade computacional de suas operações. Este aumento na complexidade computacional dificulta o desenvolvimento, em software, de aplicações voltadas para tempo real quando definições elevadas de imagem são processadas e encoraja implementações em hardware.

Neste escopo, este trabalho aborda o desenvolvimento de uma arquitetura para estimação de movimento de blocos de tamanhos variáveis segundo o padrão H.264/AVC de compressão de vídeo digital. Esta arquitetura é capaz de gerar os 41 diferentes vetores de movimento referentes a um macrobloco definidos pelo padrão. É importante destacar que o modo de decisão definido pelo padrão (ITU-T, 2003) não será tema deste trabalho. O foco do trabalho é a arquitetura para a estimação de movimento que é capaz de gerar os 41 vetores previstos pelo padrão para cada macrobloco. A definição de qual ou quais destes vetores de movimento gerados serão efetivamente utilizados é função justamente do modo de decisão. Em função da complexidade relacionada ao modo de decisão e como o modo de decisão deve considerar também os resultados gerados pela predição intra (ITU-T, 2003), o desenvolvimento arquitetural

para este módulo é, por si só, um desafio de grande porte, que não será abordado neste trabalho.

A solução arquitetural proposta neste trabalho foi desenvolvida em VHDL e mapeada para FPGAs da Xilinx (XILINX, 2007). Após, a arquitetura foi validada com dados referentes a seqüências de vídeo reais. Também foi desenvolvida uma versão *standard cell* da arquitetura. Os resultados obtidos foram satisfatórios tanto para a síntese quanto para a validação. Considerando-se as versões da arquitetura com síntese direcionada para FPGA, os resultados mostraram que a arquitetura pode ser utilizada em aplicações, tanto SDTV como HDTV. Para a versão *standard cells* da arquitetura os resultados indicam que ela pode ser utilizada, no máximo, para aplicações SDTV.

A estrutura do texto desta dissertação está organizada da seguinte maneira. No capítulo dois é apresentado um resumo sobre o padrão H.264/AVC, com breve histórico e características. Após, o capítulo três relaciona assuntos sobre estimação de movimento baseada em blocos, apresentando uma introdução, algoritmos de busca, estratégias de sub-amostragem, e as principais inovações para a estimação de movimento que o padrão H.264/AVC especifica. O capítulo quatro apresenta as avaliações do uso de sub-amostragem e do tamanho da área de busca. Ambas as avaliações feitas a partir da implementação em *software* dos algoritmos. A arquitetura para estimação de movimento de blocos de tamanho 4x4 *pixels* e os módulos que a compõem são apresentados no capítulo cinco. No capítulo seis, a arquitetura para estimação de tamanhos de bloco variáveis é apresentada em detalhes. Os resultados de síntese e de validação da arquitetura são mostrados no capítulo sete. O capítulo oito apresenta comparações da arquitetura com trabalhos relacionados encontrados na literatura assim como uma breve descrição dos mesmos. Finalmente, as conclusões desta dissertação são apresentadas no capítulo nove.

2 O PADRÃO H.264/AVC DE COMPRESSÃO DE VÍDEO DIGITAL

Este capítulo do trabalho apresenta um resumo sobre o padrão de compressão de vídeo H.264/AVC. Este resumo é composto por uma rápida introdução com histórico, os objetivos que levaram a criação do padrão, a terminologia utilizada, os perfis e níveis empregados e, por fim, uma breve abordagem dos módulos que compõem o codec H.264/AVC.

2.1 Introdução e Breve Histórico

O padrão H.264/AVC, também conhecido como MPEG-4 parte 10, é o mais novo padrão de compressão de vídeo (WIEGAND, 2003). Foi idealizado pelo *Joint Video Team* (JVT), um consórcio entre o *Video Coding Experts Group* (VCEG) da ITU e o *Moving Pictures Experts Group* (MPEG) da ISO, na tentativa de criar um padrão com desempenho superior aos padrões anteriores. Sua aprovação pela ITU-T e pela ISO ocorreu em 2003 (ITU-T, 2003).

O objetivo de obter-se um padrão superior em taxa de compressão em relação aos padrões anteriores foi alcançado. O custo desta superioridade foi um aumento significante na complexidade computacional, através da introdução de novas técnicas de compressão e do aperfeiçoamento de técnicas anteriores. Essa grande complexidade computacional das operações do padrão H.264/AVC em relação aos padrões anteriores dificulta o desenvolvimento de implementações em software de aplicações voltadas a definições elevadas e que visem tempo real. Assim tornam-se imprescindíveis soluções em hardware que, eficientemente, implementem o que está definido pelo padrão. Neste escopo, a estimação de movimento é justamente a operação mais complexa dentro do codificador (PURI, 2004), demandando uma solução em hardware o mais eficiente possível.

2.2 Terminologia

A formação de uma imagem codificada, para o padrão H.264/AVC, pode acontecer a partir de um campo ou quadro, quando o vídeo é entrelaçado, ou a partir de um quadro, quando o vídeo é progressivo (RICHARDSON, 2003). Dessa forma, para o vídeo entrelaçado metade das linhas de um quadro (linhas ímpares ou linhas pares) é reproduzida de cada vez. Alternadamente, cada parte do quadro (campo) é reproduzida para formar a imagem. Para o vídeo progressivo a reprodução é realizada linha a linha. Assim, a imagem é formada por todas as linhas do quadro seqüencialmente.

Em relação ao espaço de cores utilizado para representar o vídeo de entrada, o padrão H.264/AVC adota o espaço do tipo YCbCr (luminância e crominância). A relação entre os elementos Y, Cb e Cr é dependente do perfil do padrão que está sendo considerado.

Um quadro codificado é formado por um número determinado de macroblocos, cada um contendo 16x16 amostras de luminância, associadas às amostras de crominância. Além das amostras da imagem presentes nos macroblocos, o quadro codificado também contém informações de controle, que indicam o tipo de codificação adotado, o início de macrobloco, o tipo de macrobloco, etc. Cada elemento codificado, seja com informações de controle ou com amostras da imagem, é chamado de elemento sintático (RICHARDSON, 2003).

Os quadros codificados anteriormente podem ser utilizados como quadros de referência para a predição de outros quadros. Os quadros de referência são organizados em duas listas, chamadas de lista 0 e lista 1, de acordo com um número de quadro. Este número de quadro é sinalizado no *bitstream* e não está, necessariamente, relacionado à ordem de decodificação deste quadro.

Os macroblocos são organizados em *slices* dentro de cada quadro. Um *slice* é um grupo de macroblocos em uma ordem de varredura tipo *raster*, como mostrado na figura 2.1. Um *slice* do tipo I pode conter somente macroblocos do tipo I. Um *slice* do tipo P pode conter macroblocos do tipo P e I e um *slice* do tipo B pode conter macroblocos do tipo B e I. O padrão H.264/AVC também permite a existência de outros dois tipos de *slices*: SI e SP (ITU-T, 2005).Os *slices* SI e SP são *slices* especiais que possibilitam a eficiente mudança entre *streams* de vídeo e eficiente acesso aleatório para codificadores de vídeo. Os *slices* SI são compostos por um tipo especial de macrobloco intra, enquanto que os *slices* SP são compostos por *slices* P e/ou *slices* I (RICHARDSON, 2003).

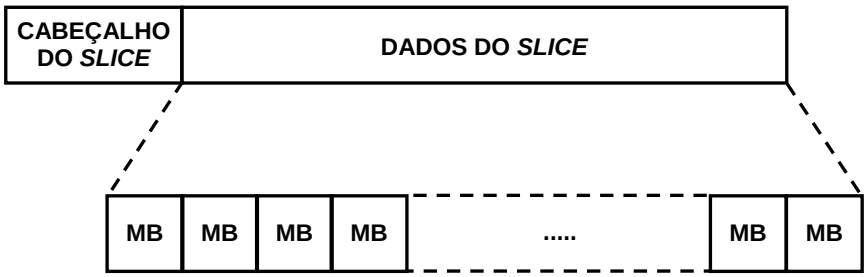


Figura 2.1: Composição de um *slice*.

Os macroblocos do tipo I são codificados usando a codificação intraquadro, a partir das amostras do *slice* atual. A codificação pode acontecer sobre macroblocos completos ou para cada bloco. Os macroblocos do tipo P são codificados usando a codificação interquadros, a partir de quadros de referência previamente codificados. Um macrobloco codificado no modo interquadros pode ser dividido em partições de macroblocos, isto é, em blocos de 16x16, 16x8, 8x16 ou 8x8. Se o tamanho de partição escolhido for o 8x8, então cada sub-macrobloco 8x8 pode ser dividido, novamente, em partições de sub-macrobloco de tamanho 8x8, 8x4, 4x8 ou 4x4. Cada partição de macrobloco é codificada utilizando como referência um quadro da lista 0. Se existir uma partição de

sub-macrobloco, então esta partição é codificada utilizando o mesmo quadro da lista 0 utilizado para codificar a partição de macrobloco. Os macroblocos do tipo B também são codificados usando a codificação interquadros. Cada partição de macrobloco pode ser codificada utilizando um ou dois quadros de referência, um na lista 0 e outro na lista 1. Se existir uma partição em sub-macrobloco, cada sub-macrobloco é codificado a partir dos mesmos um ou dois quadros de referência utilizados na codificação da partição de macrobloco.

2.3 Perfis e Níveis

A primeira versão do padrão H.264/AVC, de maio de 2003 (ITU-T, 2003), define um grupo de três diferentes perfis: *Baseline*, *Main* e *Extended*. Cada perfil suporta um grupo particular de funções de codificação.

Segundo Agostini (AGOSTINI, 2007), o perfil *Baseline* é direcionado a aplicações como vídeotelefonia, videoconferência e vídeo sem fio. O perfil *Baseline* suporta codificação intraquadro e interquadros (usando somente *slices* I e P) e uma codificação de entropia com códigos de comprimento de palavra variável adaptativos ao contexto (CAVLC).

O perfil *Main* é focado na transmissão de televisão e no armazenamento de vídeo. O perfil *Main* inclui o suporte para vídeo entrelaçado, o suporte à codificação interquadros utilizando *slices* do tipo B e utilizando predição ponderada e o suporte à codificação de entropia utilizando codificação aritmética adaptativa ao contexto (CABAC). A predição ponderada utiliza pesos diferentes sobre as amostras dos macroblocos dos slices P ou B. Então, a predição é construída a partir de uma combinação linear destas predições (AGOSTINI, 2007).

O perfil *Extended* é mais voltado para aplicações em *streaming* de vídeo e não suporta vídeo entrelaçado ou codificação aritmética adaptativa ao contexto, mas agrega modos para habilitar uma troca eficiente entre *bitstreams* codificados (através de *slices* do tipo SP e SI).

Ainda de acordo com o exposto por Agostini, os perfis *Baseline*, *Main* e *Extended* também possuem outra característica em comum: nos três perfis são usados 8 bits por amostra. Estes três perfis inicialmente propostos pelo padrão H.264/AVC não incluíram suporte para vídeos com qualidade mais elevada, como as necessárias em ambientes profissionais. Para responder às exigências deste tipo de aplicação, uma continuação do projeto JVT foi realizada para adicionar novas extensões para as capacidades do padrão original. Estas extensões foram chamadas de extensões para alcance de fidelidade (*fidelity range extensions* - FRExt). O FRExt produziu um grupo de quatro novos perfis chamados coletivamente de perfis *High* (SULLIVAN, 2004).

2.4 O Codec H.264/AVC

As figuras a 2.2 e 2.3 apresentam o codec H.264/AVC e os módulos que o compõem. O diagrama em módulos do codificador H.264/AVC é apresentado na figura 2.2. A figura 2.3 mostra o diagrama em módulos do decodificador H.264/AVC. O codificador H.264/AVC é composto pelos seguintes módulos: predição interquadros (constituída de estimação e compensação de movimento, ME e MC), predição intraquadro, transformadas diretas (T), transformadas inversas (T^{-1}), quantização (Q), quantização inversa (Q^{-1}), codificação de entropia e filtro redutor de efeito de bloco.

A figura 2.3 apresenta o decodificador H.264/AVC. Pode-se notar que diversos dos módulos que compõem o codificador estão presentes também no decodificador, com algumas alterações. Assim, os módulos que formam o decodificador H.264/AVC são: predição interquadros (apenas com compensação de movimento), predição intraquadro, transformadas inversas (T^{-1}), quantização inversa (Q^{-1}), decodificação de entropia e filtro redutor de efeito de bloco.

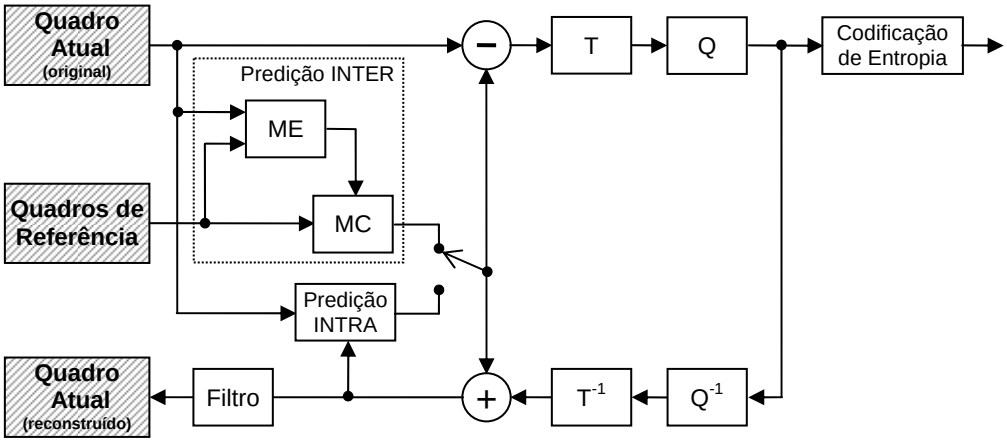


Figura 2.2: Diagrama em módulos do codificador H.264/AVC (AGOSTINI, 2007).

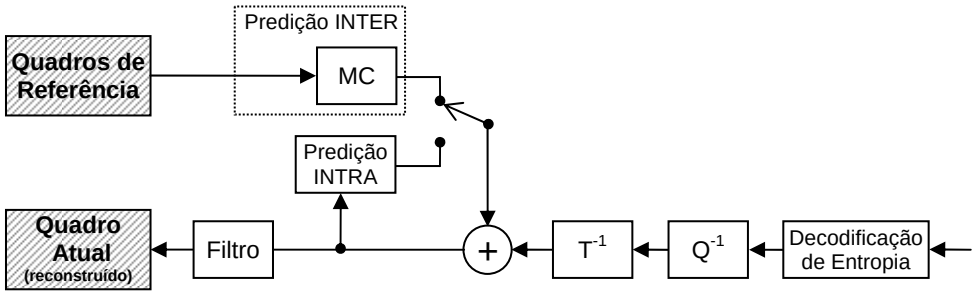


Figura 2.3: Diagrama em módulos do decodificador H.264/AVC (AGOSTINI, 2007).

O foco deste trabalho é o módulo da estimação de movimento do codificador mostrado na figura 2.2. Assim, serão apresentados, nos próximos capítulos, um estudo sobre algoritmos, estratégias de sub-amostragem, critérios de similaridade, além de uma avaliação aprofundada dos algoritmos para posterior proposição da arquitetura.

Este capítulo apresentou um resumo sobre o padrão de compressão de vídeo H.264/AVC com histórico, terminologia, perfis e níveis, e, por fim, uma abordagem dos módulos que compõem o codec H.264/AVC. O próximo capítulo abordará a estimação de movimento e aspectos como algoritmos, critérios de similaridade e sub-amostragem. As novas técnicas utilizadas pelo padrão H.264/AVC para estimação de movimento também serão apresentadas.

3 ESTIMAÇÃO DE MOVIMENTO BASEADA EM BLOCOS

Este capítulo aborda a estimação de movimento e aspectos relevantes como algoritmos, critérios de similaridade e sub-amostragem. Um resumo sobre as novas técnicas utilizadas pela estimação de movimento do padrão H.264/AVC também é apresentado. A estimação de movimento pode ser baseada em objetos ou blocos (RICHARDSON, 2003). Neste trabalho é abordada a estimação de movimento baseada em blocos.

3.1 Introdução

A estimação de movimento de um bloco (*motion estimation*, ME) envolve encontrar, no quadro de referência, uma amostra que mais aproximadamente case com o bloco atual. O objetivo é diminuir a redundância entre os quadros transmitidos. A figura 3.1 ilustra este processo.

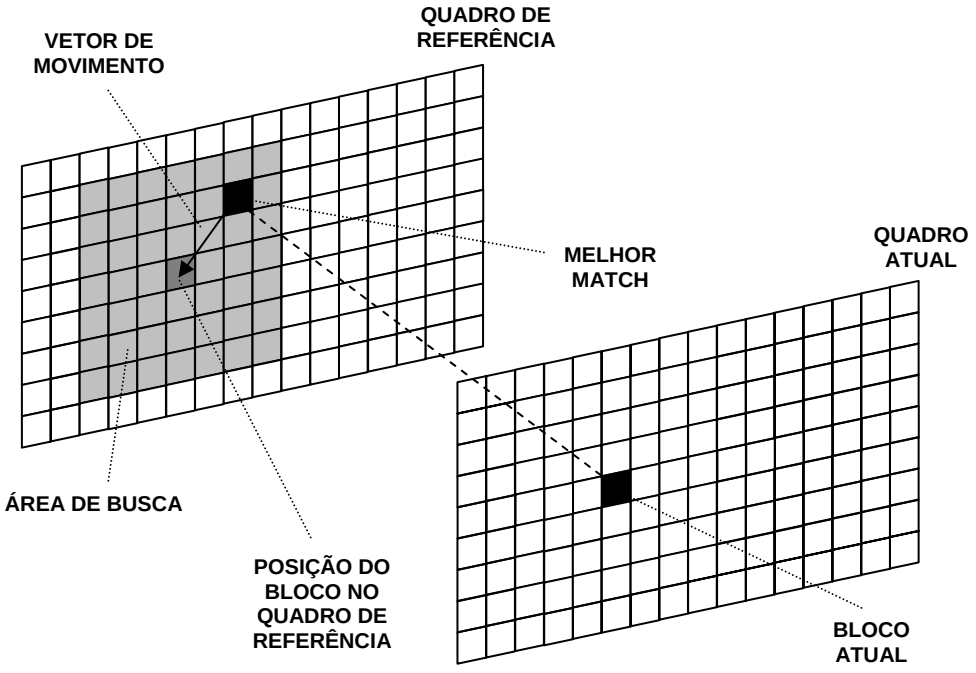


Figura 3.1: Exemplo do processo de estimação de movimento.

Para cada bloco do quadro atual será gerado um vetor de movimento que corresponderá à posição onde o bloco obteve a melhor relação de similaridade no quadro de referência. Para isso, é definida, no quadro de referência, uma área de busca em torno da posição original do macrobloco. A partir daí, utiliza-se um algoritmo de busca associado a um critério de similaridade para fazer a escolha da melhor similaridade. O vetor de movimento é gerado assim que for encontrado, no quadro atual, o bloco mais similar.

O intuito de se realizar estimação de movimento é minimizar a energia do quadro residual obtido. Com isso o desempenho da compressão é maximizado. Por outro lado, buscar o melhor casamento para cada bloco do quadro torna-se uma tarefa computacionalmente intensiva (RICHARDSON, 2003).

As figuras 3.2, 3.3, 3.4 e 3.5 ilustram bem o que foi discutido. Considerando dois quadros de uma seqüência de vídeo como exemplo (figuras 3.2 e 3.3), a figura 3.4 mostra um quadro residual sem estimação de movimento, obtido apenas com a subtração simples entre estes dois quadros da seqüência de vídeo. Já na figura 3.5 observa-se o quadro obtido com a realização de estimação de movimento sobre os mesmos dois quadros. Nota-se na figura 3.4 uma área acinzentada, onde a diferença é zero; áreas escuras, onde a diferença é negativa e áreas claras, onde a diferença é positiva. Quanto maiores as áreas escuras ou claras, maior a quantidade de energia ainda existente no resíduo. Resíduo é a subtração entre os valores do bloco original e os resultados da codificação. Para a figura 3.5, as áreas escuras ou claras são menores mostrando que há menos energia residual e que a estimação de movimento cumpriu sua função.

Figura 3.2: Quadro 1 (RICHARDSON, 2003).

Figura 3.3: Quadro 2 (RICHARDSON, 2003).

Figura 3.4: Resíduo sem estimação de movimento (RICHARDSON, 2003).

Figura 3.5: Resíduo com estimação de movimento (RICHARDSON, 2003).

3.2 Algoritmos de Busca para Estimação de Movimento

Nesta seção são apresentadas algumas alternativas de algoritmos de busca para estimação de movimento.

3.2.1 Algoritmo de Busca Completa

O algoritmo de busca completa é o mais simples procedimento de busca e o que encontra o melhor resultado, mas é computacionalmente intensivo, devido ao grande número de comparações que necessita efetuar (RICHARDSON, 2002; KUHN, 1999).

A busca completa realiza uma comparação exaustiva do bloco a ser buscado com todos os blocos candidatos possíveis dentro da janela de busca, encontrando, assim, o resultado ótimo para cada bloco. Todas as posições são comparadas até que se tenha encontrado a menor diferença, diferença esta que é determinada a partir de um critério de similaridade. Os critérios de similaridade mais comumente usados serão apresentados no item 3.4 deste capítulo. Finalmente, gera-se um vetor de movimento referente ao deslocamento do bloco para o melhor casamento. A figura 3.6 ilustra o processo de busca completa que pode ser linha a linha (*raster*) ou em espiral.

O alto custo computacional requerido por este método dificulta aplicações em tempo real para vídeos de alta resolução. Dessa forma, pode-se usar algoritmos que realizem mais rapidamente a estimação de movimento através da redução no número de operações. Para reduzir-se o número de operações pode-se optar tanto por diminuir o número de blocos candidatos como por reduzir os cálculos necessários para cada um deles. Estas técnicas de sub-amostragem serão apresentadas no item 3.3 deste capítulo.

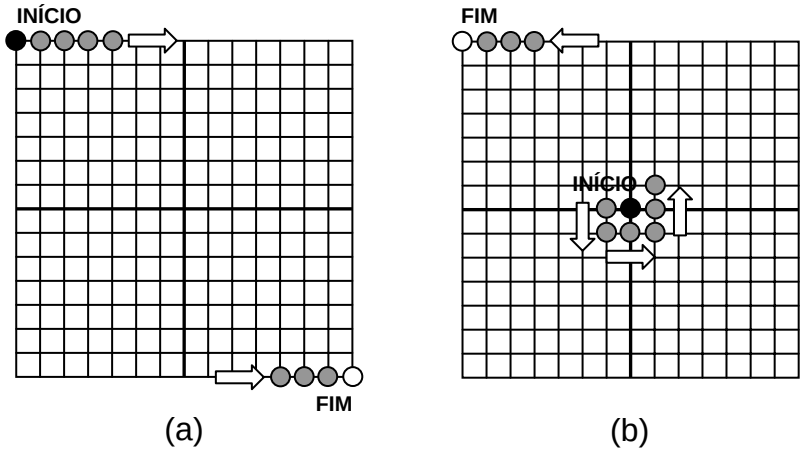


Figura 3.6: Busca completa (a) linha a linha e (b) em espiral.

A arquitetura desenvolvida neste trabalho irá utilizar o algoritmo de busca completa para realizar a estimação de movimento, como será apresentado no capítulo 5.

Além da busca completa, pode-se efetuar uma busca rápida. Diversos algoritmos foram desenvolvidos para este tipo de busca, alguns deles são apresentados no próximo item.

3.2.2 Algoritmos de Busca Rápida

Efetuar busca rápida é outra forma de diminuir o número de operações necessárias para a estimação de movimento. Neste tipo de busca poucos blocos candidatos são comparados, sendo que o número de comparações cai de forma significativa em relação à busca completa.

Diversos algoritmos foram desenvolvidos para este tipo de busca, cada um com sua peculiaridade (KUHN, 1999). Entre eles estão: *One-at-a-Time Search*, *2-D Log Search*, *Cross Search*, *Orthogonal Search*, *Four Step Search*, *Three Step Search*, *Binary Search,* entre outros. Um inconveniente é que alguns algoritmos de busca rápida acabam recaindo em um mínimo local e não são capazes de encontrar o mínimo global.

É importante ressaltar que a diferença encontrada entre blocos pela busca rápida contém mais energia do que a diferença encontrada pela busca completa (RICHARDSON, 2002).

3.3 Estratégias de Sub-Amostragem

3.3.1 Sub-Amostragem de *Pixels*

Uma alternativa usada para reduzir o número de cálculos realizados pelo critério de similaridade é a técnica de sub-amostragem de *pixels*. Esta técnica também é conhecida como *pel decimation* ou *pel subsampling* (KUHN, 1999; LEE, 2004a). O *pel decimation* tem por base o 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 figura 3.7 ilustra a técnica para as proporções 2:1(a) e 4:1(b) para um bloco de 4x4 *pixel*s. Apenas as posições em cinza são calculadas desconsiderando-se as posições em branco.

a) *pel decimation* 2:1 b) *pel decimation* 4:1

Figura 3.7: Técnica de *pel decimation*.

Com o algoritmo *pel decimation* pode-se reduzir significativamente o número de operações do algoritmo de busca completa. Neste caso, para um bloco de 4x4 *pixels*, a cada comparação, o algoritmo de busca completa deve realizar 16 operações (4x4), para o exemplo da figura 3.7(a) o algoritmo deve realizar 8 operações (2x4) e para o exemplo da figura 3.7(b) o algoritmo deve realizar apenas 4 operações (2x2).

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 busca, ao desconsiderar alguns elementos do bloco, o algoritmo pode conduzir a escolha de um bloco que não é o resultado ótimo.

3.3.2 Sub-Amostragem de Blocos

Assim como a sub-amostragem de *pixel* também é possível realizar a sub-amostragem de blocos (*block subsampling*). Esta é outra técnica utilizada para reduzir o número de operações necessárias para gerar o frame estimado (KORAH, 2005) (DE VOS, 2005).

Na sub-amostragem de blocos desconsideram-se blocos candidatos em uma determinada razão. Normalmente se usam as razões 2:1 ou 4:1. Dessa forma, em 2:1, de cada 2 blocos que seriam utilizados na comparação, um é descartado. Similarmente, para 4:1, de 4 blocos descartam-se 3. Com isso, é possível reduzir o número de comparações pela metade com a sub-amostragem 2:1 e para um quarto com a sub-amostragem 4:1, acelerando-se o processo de busca.

As figuras 3.8 e 3.9 ilustram a aplicação da técnica de *block subsampling* para uma área de busca de 17x17 *pixels* e um bloco a ser buscado de 8x8 *pixels*. Na figura 3.8 pode ser observado o número de blocos candidatos sem o uso de sub-amostragem de blocos. A técnica de *block subsampling* é apresentada na figura 3.9. Nas figuras, os blocos da área de busca utilizados estão marcados com linha cheia e numerados. Para as dimensões apresentadas, aplicando-se a técnica de *block subsampling* na proporção 4:1, obtém-se uma redução no número de comparações de 100 para 25.

3.4 Critérios de Similaridade

Os critérios de similaridade (ou de distorção) servem para encontrar a maior semelhança entre um bloco do quadro atual e os blocos da área de busca do quadro de referência. O processo está fundamentado em calcular os resíduos para cada *pixel* e, em seguida, acumulá-los em um único valor chamado de distorção. A distorção representa o grau de similaridade entre os blocos.

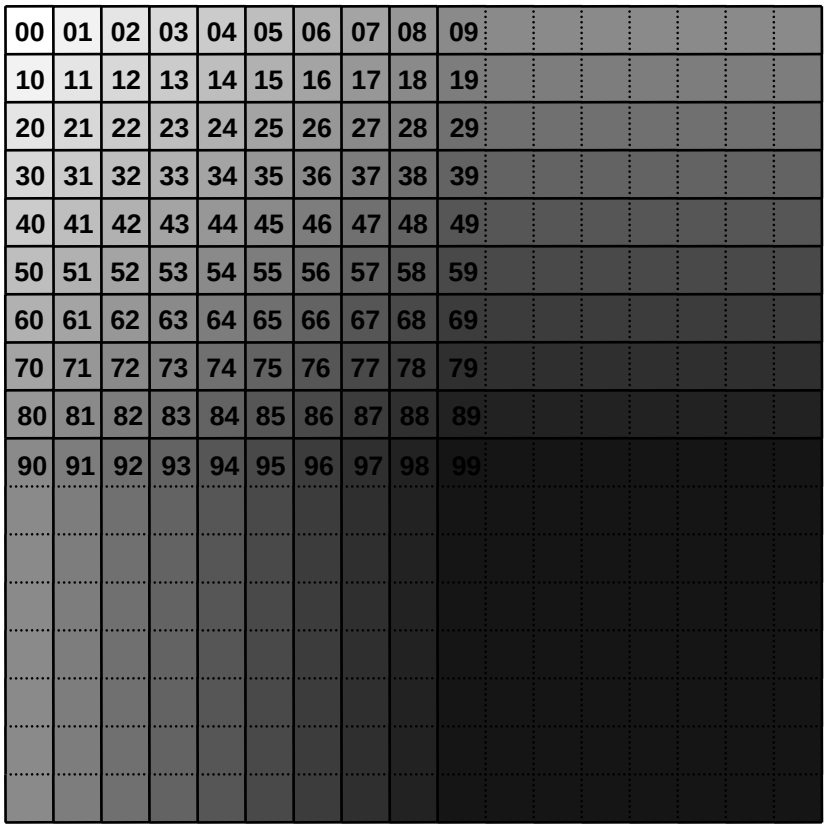


Figura 3.8: Busca sem sub-amostragem de blocos.

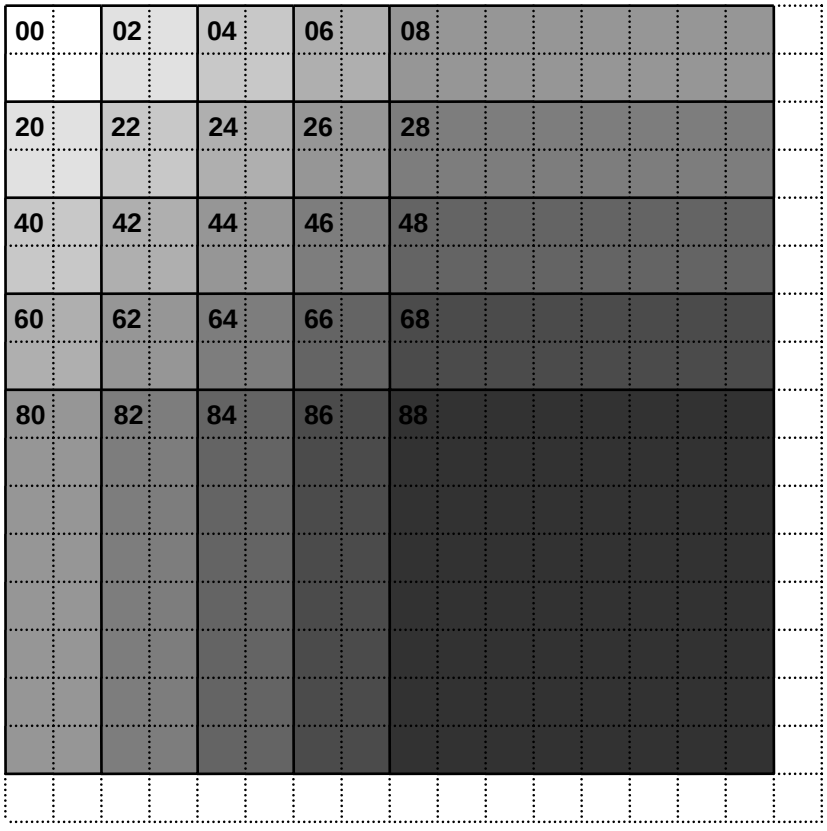


Figura 3.9: Busca com sub-amostragem de blocos.

Vários critérios de similaridade foram propostos e analisados na literatura (KUHN, 1999). Neste capítulo serão apresentados apenas os mais comumente usados: Erro Quadrático Médio, Erro Absoluto Médio e Soma dos Erros Absolutos.

3.4.1 Erro Quadrático Médio

O erro quadrático médio (*Mean Square Error*, MSE) provê uma medida da energia restante no bloco de diferença (RICHARDSON, 2002). O MSE para blocos de N x N amostras está definido em (1).

$$MSE = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (C_{ij} - R_{ij})^2 \quad (1)$$

onde C_{ij} é uma amostra do bloco atual e R_{ij}, uma amostra da área de referência.

3.4.2 Erro Absoluto Médio

Além de ser mais fácil de calcular do que o MSE, por não apresentar a potenciação necessária ao MSE, o erro absoluto médio (*Mean Absolute Error*, MAE) obtém uma aproximação razoavelmente boa da energia residual (RICHARDSON, 2002). O MAE está definido em (2).

$$MAE = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \left| C_{ij} - R_{ij} \right| \quad (2)$$

3.4.3 Soma dos Erros Absolutos

Pode-se simplificar a comparação através da omissão do termo $1/N^2$ na equação (2) e simplesmente calcular a soma dos erros absolutos (*Sum of Absolute Errors*, SAE) ou soma das diferenças absolutas (*Sum of Absolute Differences*, SAD), como também é conhecida (RICHARDSON, 2002). A equação do SAE está apresentada em (3).

$$SAE = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \left| C_{ij} - R_{ij} \right| \quad (3)$$

Por sua simplicidade e por prover uma aproximação razoável da energia do bloco, o SAD é comumente usado como critério de similaridade para estimação de movimento baseada em blocos. Em função destas características, será este o critério utilizado neste trabalho.

3.5 Inovações do Padrão H.264/AVC para a Estimação de Movimento

O padrão H.264/AVC introduz várias inovações, a maior parte delas nas etapas de estimação e compensação de movimento. Dentre elas estão o uso de blocos de tamanho variável, precisão de um quarto de *pixel* e utilização de múltiplos quadros de referência (RICHARDSON, 2002; WIEGAND, 2003).

3.5.1 Múltiplos Tamanhos de Blocos

Uma das primeiras tarefas realizadas na compressão de um quadro de vídeo é dividi-lo em macroblocos de tamanho 16x16 *pixels*. A partir daí a estimação de movimento é realizada sobre blocos do tamanho do macrobloco. No H.264/AVC há ainda a

possibilidade de segmentar cada macrobloco em sub-blocos de diferentes tamanhos e realizar uma estimação de movimento mais precisa (SULLIVAN, 2005; LEE, 2004).

O menor tamanho de bloco possível no padrão H.264/AVC é de 4x4 *pixels*. Assim, são possíveis 41 vetores de movimento para 7 combinações possíveis de sub-blocos, um vetor para cada partição do macrobloco. Os tamanhos possíveis de sub-partições em um macrobloco são 4x4, 4x8, 8x4, 8x8, 8x16, 16x8 e 16x16 *pixels* (OU, 2005), como pode ser visto na figura 3.10. Partições grandes são apropriadas para áreas mais homogêneas do quadro, enquanto que partições menores costumam ser mais apropriadas para áreas com muitos detalhes (AGOSTINI, 2007).

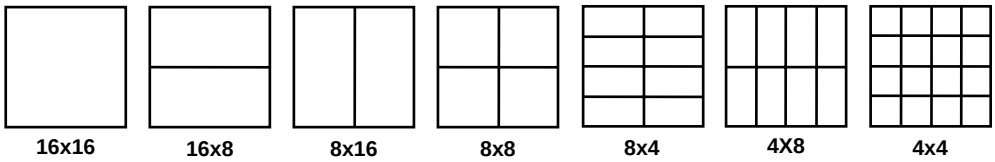


Figura 3.10: Segmentação do macrobloco para estimação de movimento no padrão H.264/AVC.

Com o uso de múltiplos tamanhos de bloco, mais vetores de movimento deverão ser gerados (um para cada partição). Para que essa quantidade maior de vetores seja gerada, são necessários mais cálculos. Por outro lado, nota-se o aumento da qualidade da estimação através da redução da energia do quadro residual exemplificada nas figuras 3.11, 3.12 e 3.13.

Figura 3.11: Resíduo com estimação de movimento de blocos de 16x16 *pixels* (RICHARDSON, 2003).

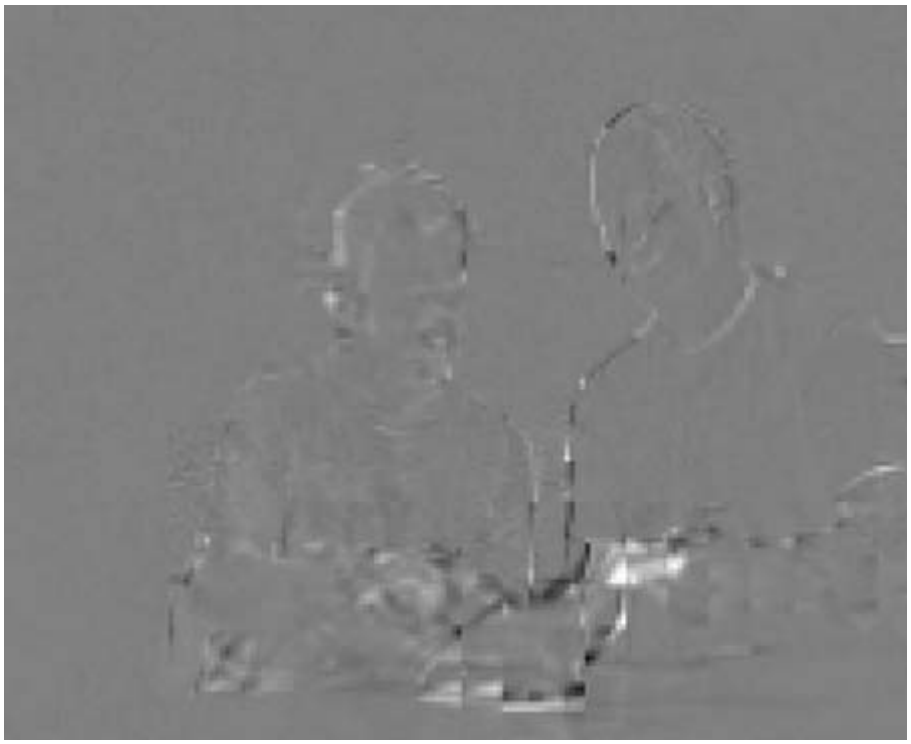


Figura 3.12: Resíduo com estimação de movimento de blocos de 8x8 *pixels* (RICHARDSON, 2003).

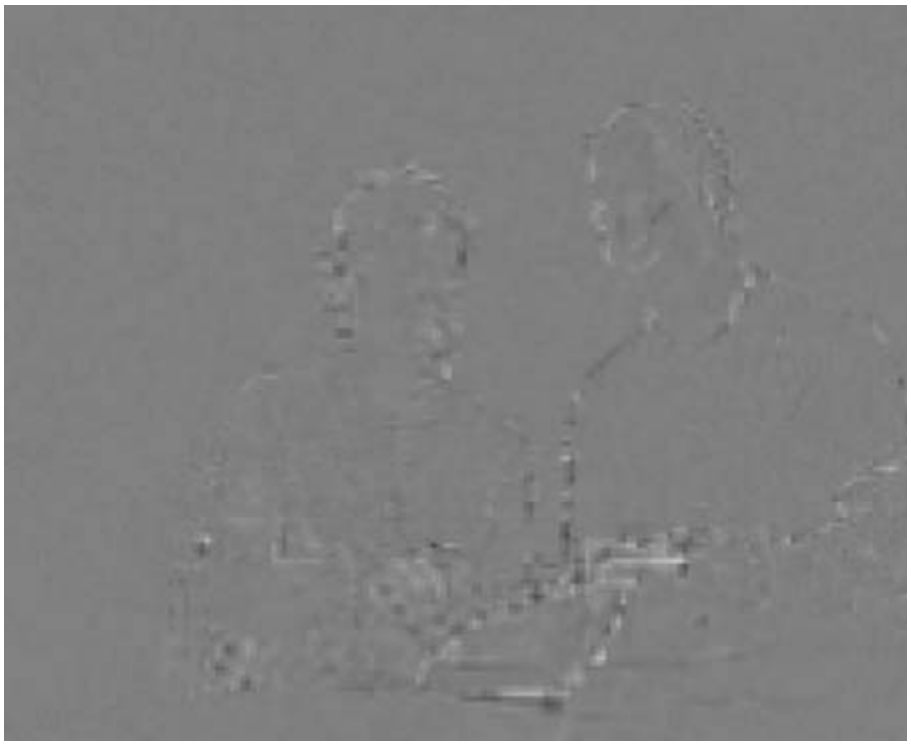


Figura 3.13: Resíduo com estimação de movimento de blocos de 4x4 *pixels* (RICHARDSON, 2003).

A figura 3.11 mostra o resíduo resultante da estimação de movimento de blocos de tamanho 16x16 *pixels*. Já na figura 3.12 o resíduo mostrado é de uma estimação de movimento para blocos de tamanho 8x8 pixels. Nota-se, na figura 3.12, que o resíduo é menor, pois há uma redução das áreas claras e escuras em relação ao resultado mostrado na figura 3.11. A energia residual pode ser menor ainda se forem usados blocos com tamanho de 4x4 *pixels*, resultado este que pode ser observado na figura 3.13. Assim sendo, aumenta-se a qualidade da estimação, mas aumenta-se a complexidade, já que mais comparações deverão ser realizadas e mais vetores terão de ser transmitidos. Assim, é possível perceber que a escolha do tamanho da partição possui um impacto significativo no desempenho da compressão (RICHARDSON, 2003). Adaptar o tamanho de bloco de acordo com as características de movimento dos quadros é a forma correta de se projetar uma arquitetura eficiente. O foco deste trabalho é exatamente o desenvolvimento arquitetural para a estimação de movimento capaz de operar sobre todos os tamanhos de blocos definidos pelo padrão H.264/AVC.

3.5.2 Precisão de Um Quarto de *Pixel*

Outra inovação apresentada pelo padrão H.264/AVC é a precisão de um quarto de *pixel* (RICHARDSON, 2003; SULLIVAN 2005). Em muitos casos o melhor casamento de um bloco não está em uma posição inteira. Assim, podem-se interpolar os valores inteiros da área de busca e encontrar o melhor casamento em posições fracionárias em torno do melhor casamento inteiro. O processo de buscar o melhor casamento para um bloco utilizando precisão de um quarto de *pixel* tem por base refinar a busca. A figura 3.14 ilustra este processo. Primeiramente, busca-se o melhor casamento para as posições inteiras (círculos). Após, buscam-se as posições de meio *pixel* em torno deste melhor casamento (quadrados) para buscar por um resultado melhor do que o anterior. Se necessário, uma nova busca é feita em torno da posição de meio *pixel* que apresentou melhor casamento em busca de um casamento para um quarto de pixel (triângulos).

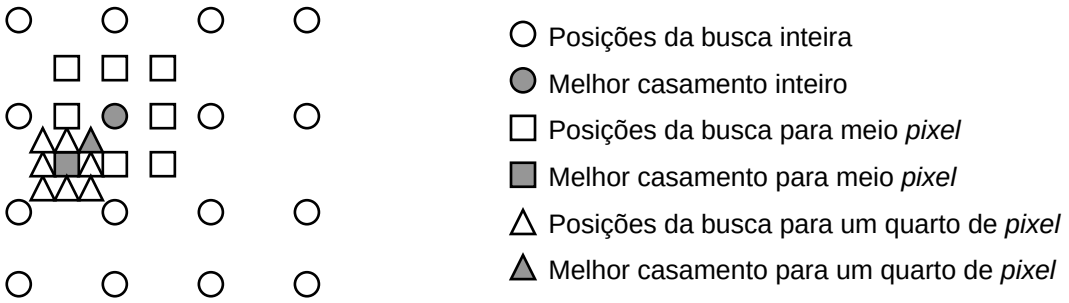


Figura 3.14: Estimação de movimento para as precisões inteira e fracionária.

Em geral, fazer uma busca em posições fracionárias fornece um resíduo menor, mas aumenta a complexidade do processo, pois mais cálculos devem ser realizados. O ganho em desempenho tende a diminuir de acordo com o incremento no nível de interpolação (RICHARDSON, 2002). De qualquer forma, utilizar estimação de movimento em posições fracionárias leva a melhores resultados. Isso pode ser notado na figura 3.15, onde é apresentado o resíduo com estimação de movimento de blocos de 4x4 *pixels* com precisão de meio-*pixel*. Esse resultado é melhor do que o apresentado na figura 3.13.

Aumentando-se a precisão para um quarto de *pixel* melhora-se ainda mais o resultado, mesmo não sendo em uma proporção tão significativa. O resíduo com estimação de movimento de blocos de 4x4 *pixels* e precisão de um quarto de *pixel* é mostrado na figura 3.16.

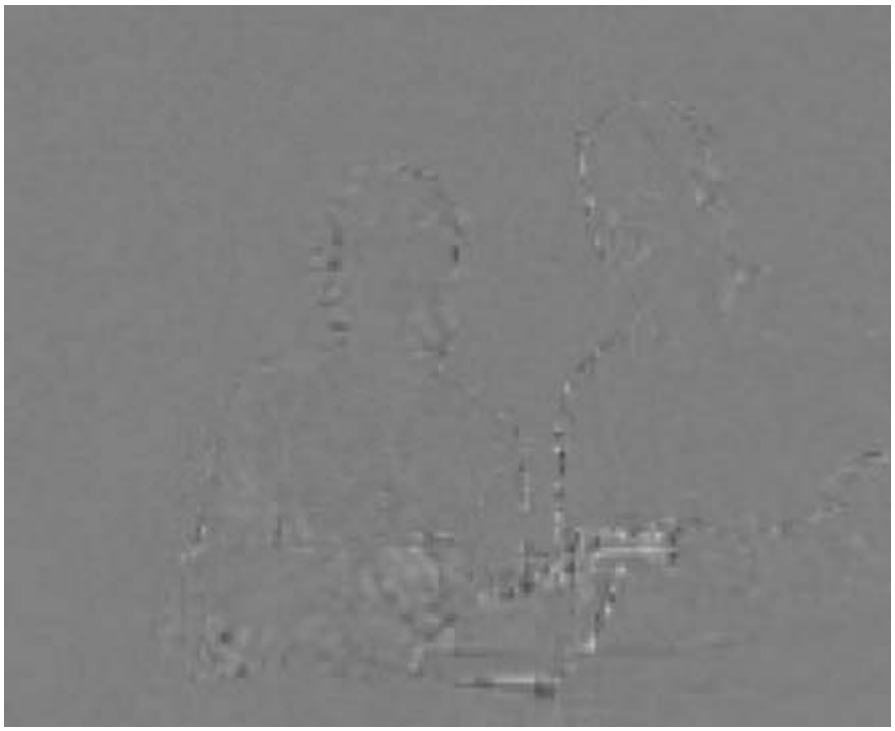


Figura 3.15: Resíduo com estimação de movimento de blocos de 4x4 *pixels* com precisão de meio-*pixel* (RICHARDSON, 2003).

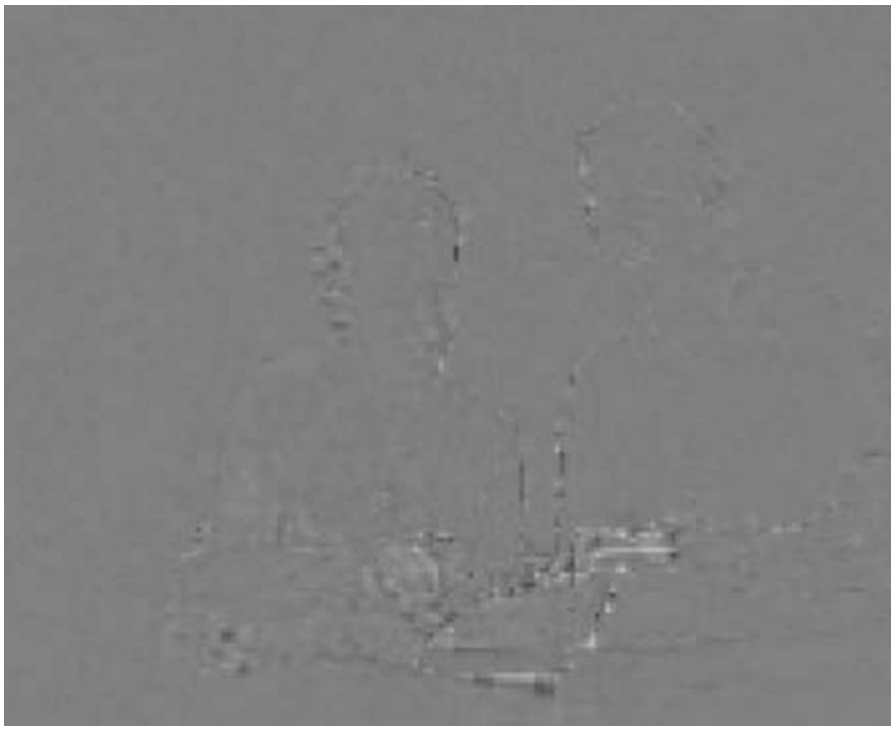


Figura 3.16: Resíduo com estimação de movimento de blocos de 4x4 *pixels* e precisão de um quarto de *pixel* (RICHARDSON, 2003).

3.5.3 Múltiplos Quadros de Referência

O uso de múltiplos quadros de referência é outra inovação do padrão H.264/AVC. Assim, é possível utilizar como referência múltiplos quadros, tanto para, frente quanto para trás do quadro atual (RICHARDSON, 2003). A figura 3.17 exemplifica este processo.

Ainda em relação aos quadros de referência, há a necessidade de se definir duas listas chamadas de lista 0 e lista 1. A lista 0 contém o quadro passado mais próximo, outros quadros passados, e quadros futuros. A lista 1 contém o quadro futuro mais próximo, outros quadros futuros, e quadros passados.

É importante destacar que o padrão H.264/AVC permite que o codificador escolha uma ordem dos quadros para codificação completamente diferente da ordem dos quadros para apresentação do vídeo. Por isso, é possível armazenar nas listas 0 e 1, quadros futuros que, neste caso, são quadros futuros para a apresentação do vídeo, mas são quadros que já foram codificados (AGOSTINI, 2007).

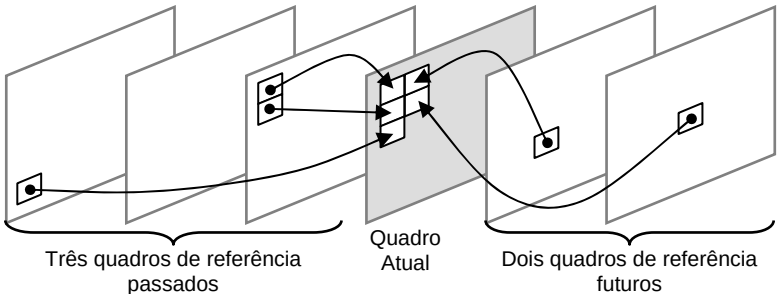


Figura 3.17: Uso de múltiplos quadros de referência (AGOSTINI, 2007)

Este capítulo abordou a estimação de movimento e aspectos como algoritmos, critérios de similaridade e sub-amostragem. Apresentou também as novas técnicas utilizadas pelo padrão H.264/AVC para estimação de movimento. No capítulo seguinte será apresentada a avaliação realizada para definir as arquiteturas que foram implementadas.

4 AVALIAÇÃO ATRAVÉS DE IMPLEMENTAÇÕES EM SOFTWARE DOS ALGORITMOS

Este capítulo apresenta dois estudos que foram extremamente importantes para a definição das arquiteturas que serão apresentadas. Um deles é uma avaliação do uso de sub-amostragem na estimação de movimento. Este primeiro estudo foi dividido em duas avaliações de implementações: uma para blocos de tamanho 16x16 *pixels* e outra para blocos de 4x4 *pixels*. O segundo estudo trata de uma avaliação do tamanho da área de busca a ser usada pelas arquiteturas.

No grupo de pesquisa do Programa de Pós-Graduação em Computação da Universidade Federal do Rio Grande do Sul foi realizado outro estudo sobre os algoritmos de estimação de movimento para a codificação pelo padrão H.264/AVC (PORTO, 2008). Este estudo apresentou uma investigação sobre algoritmos de estimação de movimento visando implementações em hardware. Foram investigados os algoritmos *Full Search*, *Three Step Search*, *Diamond Search*, *One at a Time Search*, *Hexagon Based Search*, e *Dual Cross Search.* Para cada algoritmo também foram geradas versões com sub-amostragem de *pixel* e de bloco. Todos os algoritmos foram desenvolvidos primeiramente em linguagem C e submetidos a diversos testes para avaliação de desempenho e custo computacional. Os algoritmos foram aplicados a diversas amostras de vídeo. Ao final deste estudo, o algoritmo *Diamond Search* foi o que obteve o maior destaque dentre todos os algoritmos avaliados. As vantagens decorrentes da sub-amostragem de *pixel* e de blocos também foram ressaltadas naquele estudo. É importante salientar que na presente dissertação a arquitetura foi desenvolvida para o algoritmo *full search* devido a sua linearidade e por encontrar sempre o resultado ótimo. Através dessas duas características foi possível implementar uma arquitetura que explora o máximo de qualidade de estimação.

Para a primeira parte do estudo foram implementados os algoritmos busca completa ou *full search* (FS), *full search* com *pel subsampling* 4:1(PS 4:1) e *full search* com *pel subsampling* 4:1 com *block subsampling* 4:1(PS 4:1 BS 4:1). A linguagem de programação escolhida para desenvolver os algoritmos foi C (KERNIGHAN, 1999). O critério de similaridade utilizado pelos algoritmos foi o SAD. A área de busca foi delimitada em 46x46 amostras. As seqüências de vídeo utilizadas na avaliação foram obtidas na página do *Video Quality Experts Group* (VQEG, 2007). A figura 4.1 apresenta *screenshots* dos primeiros quadros e nomes destas seqüências. Estas seqüências de vídeo possuem uma resolução de 720x480 *pixels* no formato 4:2:0 (figura 4.2). O formato 4:2:0 significa que para cada quatro elementos Y de luminância estão

associados, um elemento Cb e um Cr de crominância. Todos os algoritmos foram compilados no compilador DEV-C++ 4.9.9.2 (BLOODSHED, 2007) em um processador Intel Pentium IV de 3.2GHz com 512MB de memória RAM.

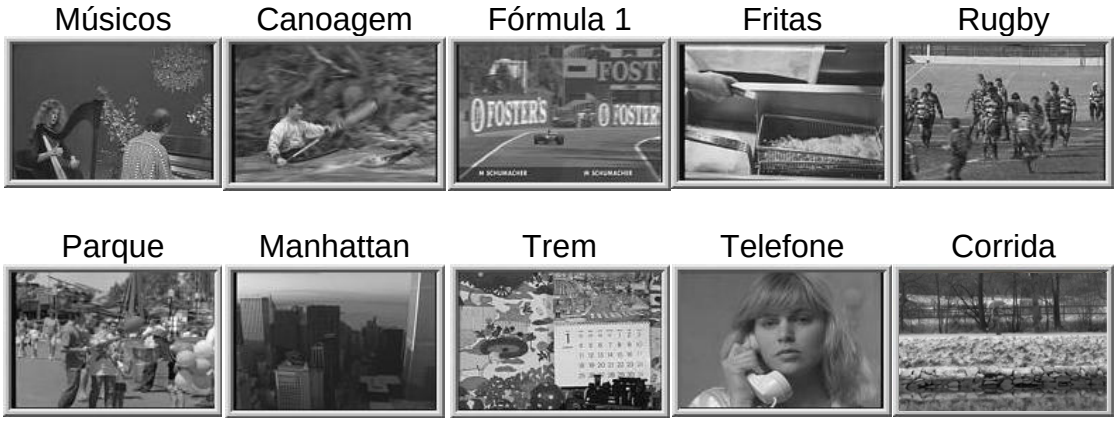


Figura 4.1: Seqüências de vídeo utilizadas na avaliação algorítmica.

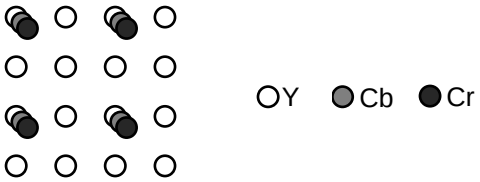


Figura 4.2: Formato 4:2:0 de sub-amostragem de crominância.

4.1 Avaliação do Uso de Sub-Amostragem para Blocos de Tamanho 16x16 *Pixels*

Os resultados deste item são apresentados em duas seções distintas: uma avaliando o tempo de execução para as implementações C dos algoritmos (seção 4.1.1) e outra apresentando os resultados de erro gerados pelos algoritmos de estimação (seção 4.1.2). Os resultados apresentados nestas seções referem-se às implementações para blocos de 16x16 *pixels*. Na seção 4.1.3 deste capítulo são feitas considerações comparativas sobre os resultados obtidos.

4.1.1 Resultados de Tempo de Execução

A partir de implementações em software dos algoritmos apresentados foram obtidos os tempos de execução. Estes tempos referem-se somente ao tempo de execução do núcleo do algoritmo, ou seja, da parte de 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.

A tabela 4.1 apresenta os resultados de tempo de execução de 100 quadros da seqüência de vídeo utilizada como entrada, considerando os algoritmos implementados e utilizando o critério SAD. O algoritmo FS foi o que obteve o pior desempenho dentre os algoritmos implementados. Este resultado já era esperado devido ao FS analisar todas as posições possíveis da área de busca e calcular a distorção para todos os *pixels* do

bloco. O algoritmo mais rápido foi o PS 4:1 BS 4:1 operando num tempo de execução 16 vezes menor do que o tempo de execução do FS justamente por fazer 16 vezes menos comparações.

Tabela 4.1: Resultados de tempo.

Seqüência de Vídeo	Algoritmo		
	FS	FS com PS 4:1	FS com PS 4:1 e BS 4:1
Músicos	487s	120s	32s
Canoagem	376s	89s	23s
Fórmula 1	417s	98s	26s
Fritas	400s	96s	25s
Rugby	413s	100s	27s
Parque	408s	96s	25s
Manhattan	401s	100s	25s
Trem	465s	108s	29s
Telefone	451s	112s	29s
Corrida	466s	113s	29s
Média	**428s**	**103s**	**27s**

Estimação gerada para 100 quadros na resolução de 720x480 *pixels*
Processador Intel Pentium 4, 3.2GHz, 512MB RAM.

4.1.2 Resultados de Redução do Erro Total

Os resultados de erro gerados são avaliados considerando o somatório do erro absoluto como base de comparação. O somatório do erro absoluto refere-se ao somatório da diferença, *pixel* a *pixel*, entre os quadros vizinhos, sem estimação. A tabela 4.2 apresenta os resultados de erro gerados para todos os algoritmos após a estimação dos 100 primeiros quadros de cada amostra utilizada. Estes valores de erro são gerados a partir da soma dos erros para cada bloco escolhido pela estimação.

Tabela 4.2: Porcentagem de redução do erro total.

Seqüência de Vídeo	Algoritmo		
	FS	FS com PS 4:1	FS com PS 4:1 e BS 4:1
Músicos	44,75	42,89	3,58
Canoagem	52,06	48,06	42,05
Fórmula 1	59,94	56,61	38,64
Fritas	58,83	57,46	49,08
Rugby	51,62	48,60	40,35
Parque	40,94	34,21	27,01
Manhattan	55,62	49,41	13,34
Trem	50,49	43,30	6,53
Telefone	39,92	29,36	-7,59
Corrida	64,63	64,11	52,04
Média	**51,88**	**47,40**	**26,50**

Estimação gerada para 100 quadros na resolução de 720x480 *pixels*
Processador Intel Pentium 4, 3.2GHz, 512MB RAM.

O algoritmo FS, como já era esperado, apresentou o menor erro para a estimação dos 100 primeiros quadros da amostra, seguido de perto pelo algoritmo FS com *pel decimation* 4:1. Ambos alcançaram uma redução do erro, em média, superior a 45%. O algoritmo FS com PS 4:1 e BS 4:1 ficou com o terceiro melhor desempenho na avaliação do erro gerado, diminuindo o erro em aproximadamente 26% em relação ao erro absoluto.

É interessante salientar que os resultados de erro para estes algoritmos dependem diretamente da amostra de vídeo utilizada. Dependendo do tipo do vídeo, se houver muito movimento ou se a cena é praticamente estática, os resultados podem variar consideravelmente. Isso pode ser notado pela diferença de comportamento dos algoritmos para os vídeos "Corrida" (com muito movimento) e "Telefone" (com pouco movimento), por exemplo. A seqüência de vídeo denominada "Telefone" apresentou uma porcentagem de -7,59% de redução do erro total. Este valor indica que, para este caso, a estimação levou a resultados piores do que os obtidos pela subtração simples entre quadros Esse resultado é considerado atípico e se justiçada pelo pouco movimento apresentado por esta seqüência e por ter sido utilizada uma taxa de amostragem onde apenas $1/16$ das amostras são comparadas.

Outro aspecto importante é o número de SADs calculados. Aplicando-se *pel subsampling* e *block subsampling* diminui-se significativamente o número de comparações para o cálculo de SAD. Com PS 4:1, o número de cálculos cai para um quarto do valor obtido para FS. Combinando-se BS 4:1 com PS 4:1 este valor cai mais um quarto chegando a um dezesseis avos do número de SADs calculados para o algoritmo FS. A tabela 4.3 apresenta o número de SADs calculados para cada seqüência de vídeo, levando-se em conta os três algoritmos implementados. O número de SADs calculados é o mesmo para cada uma das seqüências de vídeo.

Tabela 4.3: Número de SADs calculados (x 10^9).

Algoritmo		
FS	FS com PS 4:1	FS com PS 4:1 e BS 4:1
33,21 x 10^9	8,30 x 10^9	2,21 x 10^9

4.1.3 Avaliação dos Resultados

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. No entanto, aplicações em software, atualmente, são incapazes de suportar a quantidade de cálculos necessários para as implementações que realizam a busca em toda a área de busca 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 busca, e, também, para critérios de distorção mais simples, como o SAD, que realiza apenas uma subtração em módulo entre as amostras de luminância dos *pixels*. Os algoritmos apresentados neste trabalho, apesar de realizarem um número

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

Para melhor visualizar os resultados dos algoritmos a tabela 4.4 apresenta os resultados médios para o tempo de execução, o número de SADs calculados e a porcentagem de redução de erro.

Comparando os resultados apresentados na tabela 4.4 percebe-se que o algoritmo FS atinge o maior percentual de redução de erro, mas possui um tempo de execução aproximadamente quatro vezes maior do que o tempo do algoritmo FS com PS 4:1. Se comparado com FS com PS 4:1 e BS 4:1 esse tempo é 16 vezes maior. A mesma proporção ocorre para o número de SADs calculados.

Os resultados da tabela 4.4 mostram que o uso do algoritmo FS com PS 4:1 reduz consideravelmente o tempo de execução e gera uma redução de erro muito próxima da redução atingida pelo FS. Neste caso, o problema é que ele perde para o FS com PS 4:1 e BS 4:1 em tempo de execução e número de SADs calculados.

Tabela 4.4: Resultados comparativos entre os algoritmos.

Algoritmos	Tempo (s)	SADs calculados (x 10^9)	Redução do erro (%)
FS	428	33,21	51,88
FS com PS 4:1	103	8,30	47,40
FS com PS 4:1 e BS 4:1	27	2,21	26,50

Estimação gerada para 100 quadros na resolução de 720x480 *pixels*
Processador Intel Pentium 4, 3.2GHz, 512MB RAM.

4.2 Avaliação do Uso de Sub-Amostragem para Blocos de Tamanho 4x4 *Pixels*

Para notar o real impacto de estimar movimento para blocos de tamanho menor partiu-se para a segunda parte das implementações em software: a implementação dos algoritmos apresentados anteriormente mas agora para blocos de tamanho 4x4 *pixels*. Foram utilizadas as mesmas seqüências de vídeo usadas anteriormente.

A tabela 4.5 apresenta a porcentagem média de redução do erro total partindo do mesmo processo apresentado na tabela 4.4. Nesta tabela é possível notar um aumento na redução do erro total se estes resultados forem comparados aos obtidos na implementação para blocos de 16x16 *pixels*. Isso é a demonstração de que a estimação de movimento para blocos de tamanhos menores tende a gerar menos resíduo.

Tabela 4.5: Porcentagem média de redução do erro total.

Algoritmo		
FS	FS com PS 4:1	FS com PS 4:1 e BS 4:1
74,29%	49,14%	48,01%

Estimação gerada para 100 quadros na resolução de 720x480 *pixels*
Processador Intel Pentium 4, 3.2GHz, 512MB RAM.

Um resultado extremamente importante apresentado na tabela 4.5 é a redução do erro total de 74,29% apresentada pelo algoritmo FS nesta implementação. Comparado aos 51,88% do resultado da tabela 4.4, há uma melhoria de 22,41%. Outro resultado interessante é que a implementação do algoritmo FS com PS 4:1 aproxima-se, em redução de erro, da implementação FS para blocos de 16x16 *pixels*.

Com base nos resultados obtidos nos estudos apresentados decidiu-se implementar uma arquitetura para estimar movimento de blocos de 4x4 *pixels* utilizando-se o algoritmo FS. O motivo da escolha do algoritmo FS deve-se a grande diferença que o mesmo apresentou em relação ao FS com PS 4:1 em relação à redução do erro total, sendo 25% mais eficiente.

Em compensação o algoritmo FS necessita um número maior de comparações, como já foi discutido. A próxima parte do estudo, que será apresentada no item a seguir, busca equilibrar a redução do erro total com um número de operações aceitável.

4.3 Avaliação do Tamanho da Área de Busca

Após a escolha do algoritmo FS para a implementação da arquitetura da estimação de movimento, partiu-se para a avaliação do melhor tamanho de área de busca, levando-se em conta a eficiência versus o número de operações. O tamanho de bloco utilizado foi 4x4 *pixels*.

Para que pudessem ser gerados os dados comparativos, foram obtidos a partir da implementação em software, os resultados de erro total (sem estimação de movimento e usando apenas codificação diferencial), apresentados na tabela 4.6; e de erro obtido com a estimação de movimento, apresentados na tabela 4.7.

Tabela 4.6: Erro total ($x10^8$).

Vídeo	Erro Total
Músicos	3,55
Canoagem	6,43
Fórmula 1	4,09
Fritas	5,34
Rugby	6,28
Parque	5,57
Manhattan	1,36
Trem	6,50
Telefone	1,23
Corrida	8,49
Média	**4,88**

Na tabela 4.7 foram obtidos os erros para 100 quadros das dez seqüências de vídeo para seis diferentes áreas de busca: 16x16, 20x20, 26x26, 32x32, 38x 38 e 44x44 *pixels*. Estes resultados, combinados com os da tabela 4.6, geraram os resultados comparativos apresentados na tabela 4.8.

Os dados adicionais apresentados na tabela 4.8 servem de subsídio para o cálculo das operações necessárias. Partindo-se da equação (4), pode-se calcular o número de blocos candidatos.

$$BC = (T_{AB} - T_B)^2 \quad (4)$$

Na equação (4), T_{AB} refere-se ao tamanho da área de busca e T_B ao tamanho do bloco, ambos em *pixels*. Por exemplo, em uma área de 16x16 amostras, são169 blocos candidatos ((16-4+1)x(16-4+1)). Multiplicando-se o resultado obtido (13x13=169) pelo número de operações por bloco (4x4=16) chega-se ao valor de 2.704 operações por bloco candidato. Tendo-se 21.600 blocos por frame ((720/4)x(480/4)) então são $0{,}5 \cdot 10^{10}$ operações para os 100 frames de cada seqüência de vídeo utilizada. Os dados sobre redução do erro total foram calculados a partir dos dados obtidos na tabela 4.6.

Tabela 4.7: Erro obtido com estimação (x10^8).

Seqüência de vídeo	Área (*pixels*)					
	16x16	20x20	26x26	32x32	38x38	44x44
Músicos	1,57	1,54	1,50	1,47	1,45	1,43
Canoa	2,41	2,09	1,76	1,57	1,41	1,28
Fórmula 1	1,43	1,29	1,10	1,01	0,93	0,88
Fritas	2,46	2,22	1,76	1,44	1,22	1,06
Rugby	2,27	2,06	1,82	1,69	1,59	1,52
Parque	2,42	2,27	2,07	1,94	1,84	1,77
Manhattan	0,47	0,46	0,44	0,43	0,42	0,41
Trem	2,47	2,44	2,40	2,36	2,33	2,31
Telefone	0,58	0,57	0,56	0,56	0,55	0,55
Corrida	2,54	2,52	2,49	2,47	2,45	2,43
Média	**1,86**	**1,75**	**1,59**	**1,49**	**1,42**	**1,36**

Tabela 4.8: Resultados comparativos para as diferentes áreas de busca.

Área de busca	16x16	20x20	26x26	32x32	38x38	44x44
Blocos candidatos	169	289	529	841	1.225	1.681
Operações por bloco candidato	2.704	4.624	8.464	13.456	19.600	26.896
Blocos por frame	21.600	21.600	21.600	21.600	21.600	21.600
Operações para 100 frames	$0{,}5 \cdot 10^{10}$	$0{,}9 \cdot 10^{10}$	$1{,}8 \cdot 10^{10}$	$2{,}9 \cdot 10^{10}$	$4{,}2 \cdot 10^{10}$	$5{,}8 \cdot 10^{10}$
Redução do erro total	61,80%	64,18%	67,39%	69,37%	70,89%	72,02%

Para uma melhor visualização dos dados apresentados na tabela 4.8, foi organizada uma nova tabela (tabela 4.9) e um gráfico (figura 4.3). Assim, pode-se ter uma idéia mais precisa do custo de se aumentar a área de busca, ou seja, é possível prever a quantidade de operações a mais que serão necessárias para o cálculo de SAD sobre a nova área. Os dados percentuais da tabela 4.9 foram obtidos comparando-se os resultados das outras áreas de busca em relação à área de busca de 16x16 *pixels*.

Tabela 4.9: Número de operações x erro total.

Área de busca (*pixels*)	16x16	20x20	26x26	32x32	38x38	44x44
Aumento no número de operações	-	71%	213%	397%	624%	894%
Redução do erro total	-	2,38%	5,59%	7,57%	9,09%	10,22%

Considerando os dados da Tabela 4.9, aumentando-se a área de busca de 16x16 para 20x20, por exemplo, tem-se um aumento no número de operações da ordem de 71% mas apenas o pequeno aumento de 2,38% na redução do erro total, o que não se demonstra um bom investimento. É preciso ter sempre em mente que a quantidade de operações necessárias reflete-se diretamente na área ocupada pela arquitetura, para uma mesma taxa de processamento. Dessa forma, os altos índices de aumento no número de operações refletem-se em grande aumento na quantidade de hardware necessário para implementar a arquitetura. A situação piora quanto maior for a área de busca. Para uma área de 44x44 *pixels*, por exemplo, são necessárias 894% mais operações obtendo-se um ganho de apenas 10%.

No gráfico apresentado na figura 4.3 é possível visualizar as proporções dos números apresentados na tabela 4.9. Neste gráfico, a redução do erro é apresentada em preto e o aumento no número de operações, em cinza. Todas as cinco áreas apresentadas têm seus percentuais comparativos calculados sobre os resultados obtidos para a área de busca de 16x16.

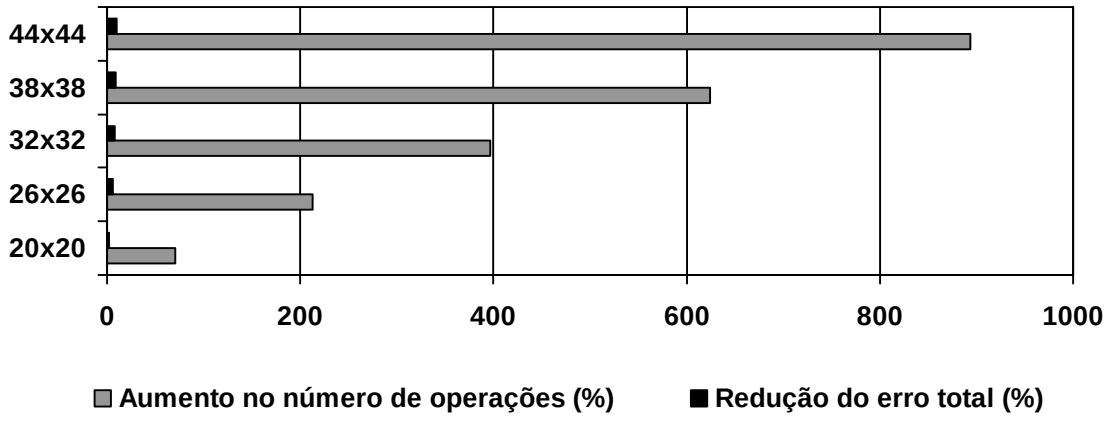


Figura 4.3: Relação entre a redução do erro total e o número de operações.

Com base nos resultados apresentados, a arquitetura definida para implementar a estimação de movimento de blocos de 4x4 *pixels* utiliza o algoritmo *full search* associado ao critério SAD sobre uma área de busca de 16x16 *pixels*, pois é a que apresentou a melhor relação custo-benefício.

Este capítulo apresentou os estudos realizados para a definição das arquiteturas que foram implementadas. Foram apresentadas as avaliações do uso de sub-amostragem na estimação de movimento e do tamanho da área de busca a ser usado pelas arquiteturas.

O próximo capítulo apresenta detalhes da implementação da arquitetura proposta e desenvolvida neste trabalho para realizar estimação de movimento sobre blocos de 4x4 *pixels*.

5 ARQUITETURA PARA ESTIMAÇÃO DE MOVIMENTO DE BLOCOS DE TAMANHO 4X4 *PIXELS*

A arquitetura em hardware proposta para implementar a estimação de movimento sobre blocos de 4x4 *pixels* é apresentada neste capítulo. Essa arquitetura usa o algoritmo *full search* como método de busca e SAD como critério de similaridade. A área de busca utilizada é de 16x16 pixels. A base para a implementação dessa arquitetura é apresentada na Tese de Doutorado de Luciano Volcan Agostini (AGOSTINI, 2007). É importante salientar que essa estimação é feita sobre blocos de 4x4 amostras de luminância não estando as informações de crominância incluídas nesse processo

5.1 Arquitetura para Estimação de Movimento de Blocos de Tamanho 4x4 *Pixels*

Na arquitetura desenvolvida neste trabalho, 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. Inicialmente, cada macrobloco de 16x16 *pixels* é dividido em 16 sub-blocos de 4x4 *pixels*. Esses sub-blocos, juntamente com suas respectivas áreas de busca, são as entradas da estimação de movimento. O bloco de 4x4 *pixels* é proveniente do quadro atual enquanto que a área de busca é procedente do quadro de referência. As informações destes quadros ficam armazenadas em uma memória externa à estimação de movimento e, quando são necessárias, são enviadas para duas memórias internas da ME. A saída da ME são os vetores de movimento calculados para os blocos de 4x4 *pixels*. Estes devem ser codificados junto com os resíduos dos blocos.

A figura 5.1 apresenta o diagrama de blocos da arquitetura da estimação de movimento. Para permitir uma melhor visualização da figura, vários sinais foram ocultados, principalmente os sinais de controle. Os detalhes dos principais módulos que compõem a arquitetura serão apresentados nas próximas seções. Incluídos nestes módulos estão as linhas de cálculo de SAD e as suas unidades de processamento (UPs), o controle, os comparadores e a parte da memória, composta pelas memórias de área de busca e de bloco atual e pelo gerenciador de memória. Outros módulos menos complexos mas que também fazem parte da arquitetura são dois conjuntos de registradores: um para a linha de área de busca (RSA) e outro para a linha do bloco do quadro atual (RCB).

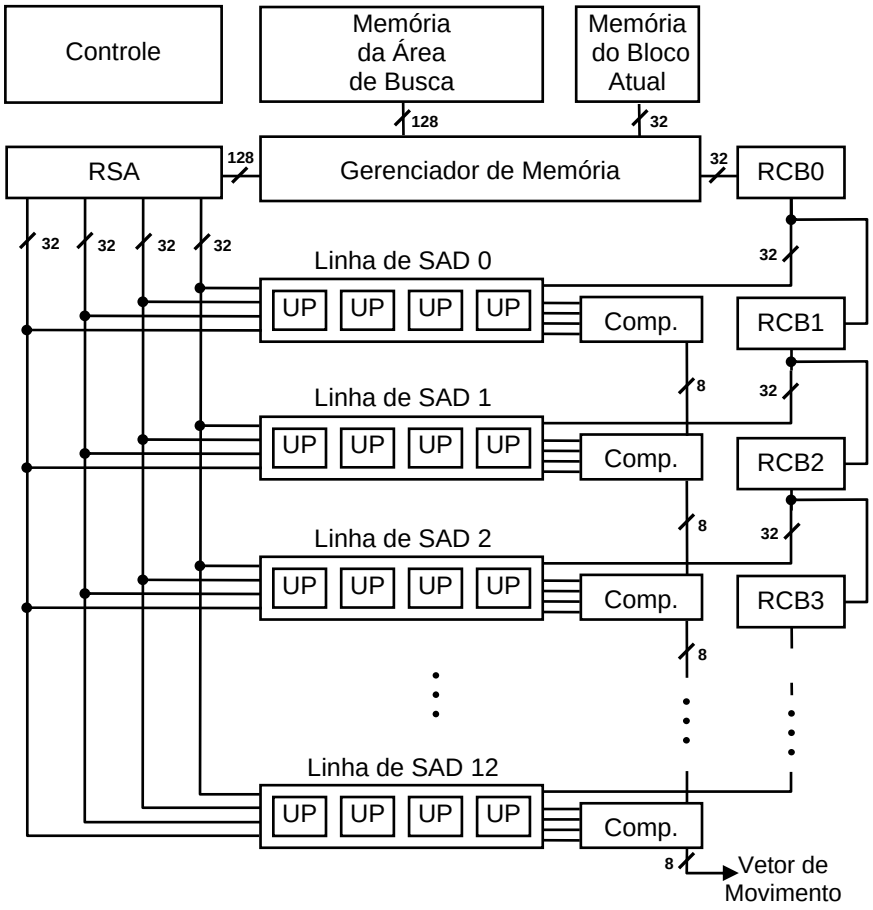


Figura 5.1: Arquitetura do estimador de movimento para blocos com 4x4 *pixels* e área de busca de 16x16 *pixels*.

A memória interna está organizada em duas memórias distintas como está apresentado na figura 5.1. Uma memória é destinada ao armazenamento do bloco do quadro atual e a outra é usada para armazenar a área de busca. A organização das memórias é feita de forma que cada palavra da memória contenha uma linha do bloco atual ou da área de busca respectivamente. A memória para o bloco atual está organizada em 4 palavras de 32 bits. Já a memória para a área de busca possui 16 palavras de 128 bits, 16 vezes maior do que a memória para o armazenamento do bloco atual. Cada amostra é composta por 8 bits. Assim sendo, cada palavra contém uma linha.

Os dados da área de busca e do bloco atual que estão armazenados na memória interna são acessados pelo gerenciador de memória (figura 5.1). Quando a ME inicia seu funcionamento o gerenciador de memória realiza a leitura de uma palavra da memória da área de busca e outra da memória do bloco atual. Dessa forma, são acessadas uma linha da área de busca e uma linha do bloco atual. Essas linhas são armazenadas nos registradores da linha de busca (RSA na figura 5.1) e nos registradores de linha de bloco (RCB na figura 5.1). Os registradores RSA armazenam uma linha completa da área de busca utilizando 128 bits (16 amostras); os registradores RCB guardam a linha do bloco atual com 32 bits (4 amostras). Devido ao fato de que toda linha do bloco atual deve ser comparada com cada linha da área de busca, as linhas de bloco atual vão sendo passadas adiante nos RCBs durante o cálculo do SAD.

A unidade de processamento (UP na figura 5.1) calcula a distorção entre uma linha do bloco do quadro atual e uma linha do bloco candidato. Quatro UPs são combinadas para formar uma linha de SAD. As três primeiras UPs da linha de SAD são responsáveis por processar quatro blocos candidatos; a última processa apenas um. Logo, a linha de SAD com suas quatro UPs processa os 13 blocos candidatos de uma linha. Um conjunto de 13 linhas de SAD forma uma matriz de SAD realizando a busca completa sobre a área de busca do quadro de referência. O controle das linhas de SAD é interno. Essa foi uma decisão tomada com o intuito de reduzir a complexidade do controle dos módulos hierarquicamente superiores.

O controle define a operação dos módulos da ME. Gerenciar as operações através da sincronização das linhas de SAD, dos comparadores e dos registradores é tarefa do controle. Ele é responsável por gerar os sinais que informam quando a linha de SAD disponibiliza os valores de distorção e os vetores de movimento correspondentes a cada bloco candidato. Com esses dados disponíveis, o controle ativa o comparador. Isso acontece linha a linha até que, na última linha, o vetor de movimento do bloco candidato que apresentar menor distorção, será enviado para a saída.

A primeira linha de SADs calcula o SAD para os 13 blocos candidatos que iniciam na primeira linha da área de busca, comparando o bloco atual a estes blocos candidatos. A segunda linha de SADs faz o mesmo processo para os 13 blocos candidatos que iniciam na segunda linha da área de referência. Este processo é realizado até que, na décima terceira linha de SADs, os SADs dos últimos 13 blocos candidatos são calculados, finalizando a comparação para os 169 blocos candidatos.

A arquitetura proposta na figura 5.1 otimiza o compartilhamento das amostras da área de busca e das amostras do bloco do quadro atual sobre o qual se deseja encontrar o melhor casamento e, deste modo, gerar o melhor vetor de movimento. Esta otimização 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. Todas as linhas do bloco atual são comparadas com todas as linhas da área de busca. Para isso, todas as linhas de SAD recebem a mesma linha da área de busca (através dos registradores RSA) e linhas diferentes do bloco atual (disponibilizadas pelos registradores (RCB).

Depois de preenchidas as memórias internas da arquitetura (apresentadas na figura 5.1), um novo vetor de movimento é gerado a cada 64 ciclos de relógio. Este novo vetor de movimento é referente ao menor SAD encontrado para um bloco de 4x4 *pixels*.

5.2 Gerenciamento de Memória

A ME utiliza duas memórias internas, conforme apresentado na figura 5.1. Uma memória armazena o bloco do quadro atual e a outra, dados da área de busca. A memória que armazena o bloco do quadro atual possui 4 palavras de 32 bits cada. Cada palavra é composta por 4 amostras de 8 bits formando uma linha. Já a memória da área de busca está organizada em 16 palavras de 128 bits. Dezesseis amostras de 8 bits compõem cada uma das palavras desta memória. A memória que armazena o bloco atual tem 128 bits de tamanho. Já a área de busca está armazenada em uma memória de 2048 bits ou 2 KB.

O gerenciador de memória na figura 5.1 é responsável por controlar as leituras de acordo com as necessidades da arquitetura. As amostras lidas da memória de área de

busca são copiadas para os registradores RSA, enquanto que as amostras de bloco atual são armazenadas nos registradores RCB. Esses registradores, através do gerenciamento realizado pelo controle, disponibilizam os dados corretos nas entradas de cada UP.

O acesso à memória é feito a cada 64 ciclos. Dentro deste intervalo de 64 ciclos, os 16 primeiros ciclos são utilizados para a leitura das 16 palavras referentes à área de busca e os 4 primeiros ciclos são utilizados para a leitura das palavras relativas ao bloco atual. Assim, para a posterior leitura de uma palavra de qualquer uma das memórias internas é necessário apenas um ciclo.

5.3 Arquitetura para o Cálculo do SAD

Para encontrar a maior similaridade entre um bloco do quadro atual e os blocos da área de busca do quadro de referência, é utilizado o algoritmo SAD (*Sum of Absolute Differences*) (RICHARDSON, 2003). O SAD é calculado através da subtração, em módulo, das posições correspondentes das amostras do quadro atual e da área de busca, 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 (RICHARDSON, 2003).

Na arquitetura para estimação de movimento proposta neste trabalho foram considerados blocos de 4x4 amostras de luminância e área de busca de 16x16 amostras. Deste modo, existem 13x13 blocos candidatos dentro da área de busca, gerando um total de 169 blocos candidatos ao melhor casamento. Considerando essas medidas, 16 cálculos de SAD devem ser realizados para cada bloco candidato, totalizando 2700 cálculos de SAD necessários para realizar a busca completa em toda a área de busca e para encontrar o melhor casamento para um bloco.

A arquitetura para o cálculo do SAD foi construída hierarquicamente. A instância de mais elevado nível hierárquico é a matriz de SADs. A matriz de SADs é formada por 13 linhas de SADs que, por sua vez, são formadas por 4 unidades de processamento (UPs).

A figura 5.2 apresenta a arquitetura de uma UP. Os sinais de controle foram omitidos. Cada UP recebe como entrada quatro amostras do bloco atual (A0 a A3 na figura 5.2) e quatro amostras do bloco candidato (R0 a R3 na figura 5.2). Isso significa que cada UP calcula, paralelamente, a similaridade de uma linha do bloco candidato em relação ao bloco atual. As UPs foram projetadas para operar em um *pipeline* de três estágios, como está apresentado na figura 5.2. Inicialmente, é realizada uma subtração entre as quatro amostras do bloco atual (A0 a A3) e as quatro amostras do bloco candidato (R0 a R3). O resultado do subtrator passa por um processo que extrai o módulo. Os módulos são somados para gerar um valor único. O resultado representa o módulo do erro entre os dois blocos para as quatro primeiras amostras, ou seja, para uma linha dos blocos.

O SAD parcial do bloco candidato (SAD da linha) gerado pela UP deve ser armazenado e adicionado aos SADs das demais linhas do bloco candidato para gerar o SAD total. Este, por sua vez, é formado por quatro linhas com quatro amostras cada (16 SADs devem ser acumulados). A linha de SADs, apresentada na figura 5.3, agrupa quatro UPs e realiza a acumulação para gerar o valor final do SAD dos blocos candidatos. Ao todo são 13 acumuladores, um para cada bloco candidato existente na linha.

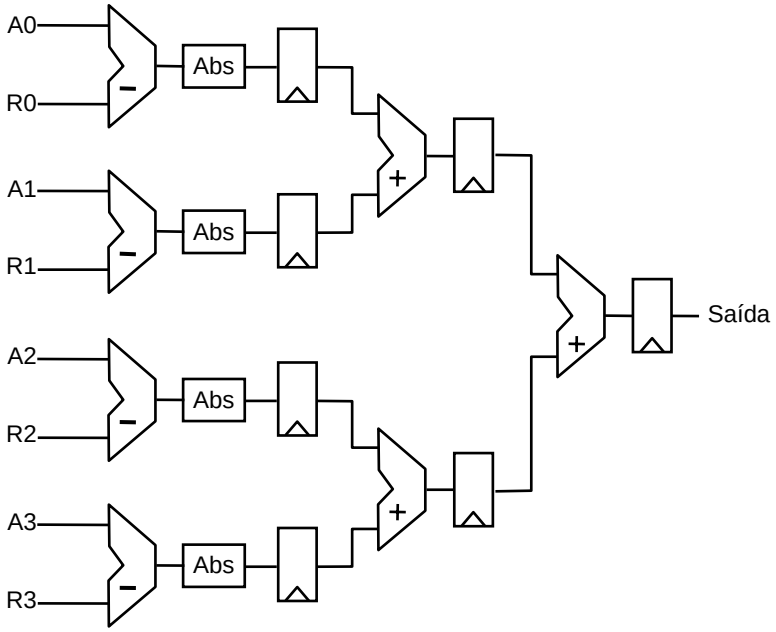


Figura 5.2: Arquitetura da unidade de processamento de SAD.

Cada UP gera valores parciais para quatro acumuladores diferentes (menos a última UP). Cada conjunto de 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.

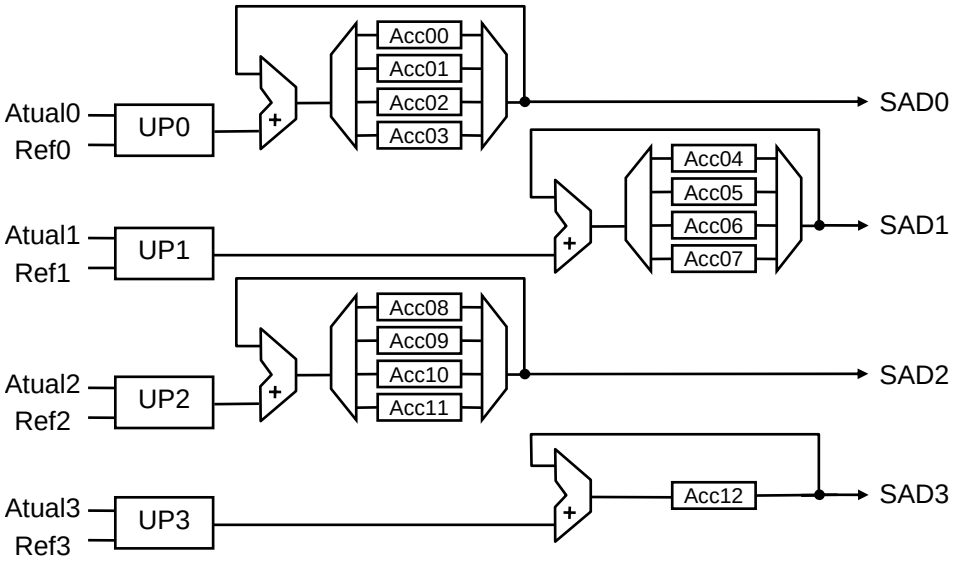


Figura 5.3: Diagrama em módulos de uma linha de SAD.

Cada UP é responsável pelo cálculo, em tempos distintos, da similaridade de diferentes blocos candidatos. Com esta solução, as UPs processam os SADs relativos a quatro diferentes blocos candidatos, exceto a última UP que processa SADs para apenas um bloco candidato. Portanto, uma linha de SADs calcula paralelamente o SAD de 13

diferentes blocos candidatos. Para o cálculo de todos os SADs das linhas dos blocos candidatos é usada sempre a mesma linha do bloco atual. A tabela 5.1 apresenta um exemplo das linhas dos blocos candidatos que são processados em cada estágio de *pipeline* de cada UP de uma linha de SADs.

Como já foi mencionado, cada UP processa uma linha dos blocos (4 amostras) em cada operação e demora três ciclos para concluir este cálculo. A tabela 5.1 mostra o escalonamento das operações das UPs em uma linha de SAD. A arquitetura das UPs inicialmente calcula os SADs das primeiras linhas dos treze blocos (indicadas pela letra **a**). Após os três ciclos necessários para estes cálculos as UPs processam as segundas linhas dos blocos (marcadas com **b**). O processo repete-se até que as quatro linhas dos 13 blocos candidatos tenham sido calculadas.

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

Tempo	Estágio	UP0	UP1	UP2	UP3
0	1	Bloco 00 a	Bloco 04 a	Bloco 08 a	Bloco 12 a
1	2	Bloco 01 a	Bloco 05 a	Bloco 09 a	-
2	3	Bloco 02 a	Bloco 06 a	Bloco 10 a	-
3	4	Bloco 03 a	Bloco 07 a	Bloco 11 a	-
4	1	Bloco 00 b	Bloco 04 b	Bloco 08 b	Bloco 12 b
5	2	Bloco 01 b	Bloco 05 b	Bloco 09 b	-
6	3	Bloco 02 b	Bloco 06 b	Bloco 10 b	-
7	4	Bloco 03 b	Bloco 07 b	Bloco 11 b	-
8	1	Bloco 00 c	Bloco 04 c	Bloco 08 c	Bloco 12 c
9	2	Bloco 01 c	Bloco 05 c	Bloco 09 c	-
10	3	Bloco 02 c	Bloco 06 c	Bloco 10 c	-
11	4	Bloco 03 c	Bloco 07 c	Bloco 11 c	-
12	1	Bloco 00 d	Bloco 04 d	Bloco 08 d	Bloco 12 d
13	2	Bloco 01 d	Bloco 05 d	Bloco 09 d	-
14	3	Bloco 02 d	Bloco 06 d	Bloco 10 d	-
15	4	Bloco 03 d	Bloco 07 d	Bloco 11 d	-

Assim sendo, o resultado final é gerado depois que as quatro linhas são processadas para cada bloco candidato. Como cada UP processa uma linha do bloco a cada passo, então cada UP gera 4 resultados parciais de SAD para cada bloco processado. Estes 4 resultados parciais devem ser somados para gerar o SAD final do bloco. Estes resultados, para um mesmo bloco, não são gerados em paralelo. Torna-se necessária uma estrutura simples de somador e registrador de acumulação para fazer esta totalização. Como as UPs calculam em paralelo o SAD de mais de um bloco, um registrador é utilizado para armazenar o SAD de cada bloco (ACC0 a ACC12 na figura 5.3). Também é necessário um par demultiplexador/multiplexador para controlar o acesso correto aos registradores. Quando uma linha de SADs concluir seus cálculos, os registradores ACC0 a ACC12 irão conter os SADs finais dos 13 blocos candidatos.

É importante ressaltar que os valores dos SADs finais para os 13 blocos candidatos de cada linha são enviados para a saída da estimação de movimento de blocos de 4x4 *pixels*. Estes valores serão utilizados para estimar movimento de blocos de outros

tamanhos, tarefa esta a ser realizada pela extensão que trata da estimação de tamanhos de bloco variáveis.

Outro aspecto importante a ser observado é a geração dos vetores de movimento. Na arquitetura que foi implementada neste trabalho uma parte do vetor é gerada dentro da linha de SAD e a outra fora. Isso se deve ao fato de todas as linhas terem o mesmo índice conforme apresentado na figura 5.4. Em destaque na figura um bloco candidato de 4x4 *pixels* (em cinza) e seu respectivo vetor (em negrito). No total são 169 blocos candidatos para cada bloco de 4x4 *pixels* a ser predito.

(-6,-6)	(-6,-5)	(-6,-4)	(-6,-3)	(-6,-2)	(-6,-1)	(-6,0)	(-6,1)	(-6,2)	(-6,3)	(-6,4)	(-6,5)	(-6,6)			
(-5,-6)	(-5,-5)	(-5,-4)	(-5,-3)	(-5,-2)	(-5,-1)	(-5,0)	(-5,1)	(-5,2)	(-5,3)	(-5,4)	(-5,5)	(-5,6)			
(-4,-6)	(-4,-5)	(-4,-4)	(-4,-3)	(-4,-2)	(-4,-1)	(-4,0)	(-4,1)	(-4,2)	(-4,3)	(-4,4)	(-4,5)	(-4,6)			
(-3,-6)	(-3,-5)	(-3,-4)	(-3,-3)	(-3,-2)	(-3,-1)	(-3,0)	(-3,1)	(-3,2)	(-3,3)	(-3,4)	(-3,5)	(-3,6)			
(-2,-6)	(-2,-5)	(-2,-4)	(-2,-3)	(-2,-2)	(-2,-1)	(-2,0)	(-2,1)	(-2,2)	(-2,3)	(-2,4)	(-2,5)	(-2,6)			
(-1,-6)	(-1,-5)	(-1,-4)	(-1,-3)	(-1,-2)	(-1,-1)	(-1,0)	(-1,1)	(-1,2)	(-1,3)	(-1,4)	(-1,5)	(-1,6)			
(0,-6)	(0,-5)	(0,-4)	(0,-3)	(0,-2)	(0,-1)	**(0,0)**	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)			
(1,-6)	(1,-5)	(1,-4)	(1,-3)	(1,-2)	(1,-1)	(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)			
(2,-6)	(2,-5)	(2,-4)	(2,-3)	(2,-2)	(2,-1)	(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)			
(3,-6)	(3,-5)	(3,-4)	(3,-3)	(3,-2)	(3,-1)	(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)			
(4,-6)	(4,-5)	(4,-4)	(4,-3)	(4,-2)	(4,-1)	(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)			
(5,-6)	(5,-5)	(5,-4)	(5,-3)	(5,-2)	(5,-1)	(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)			
(6,-6)	(6,-5)	(6,-4)	(6,-3)	(6,-2)	(6,-1)	(6,0)	(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)			

Figura 5.4: Vetores de movimento para os blocos candidatos.

5.4 Arquitetura do Comparador

O comparador foi projetado em um *pipeline* de quatro estágios podendo realizar a comparação de quatro SADs em paralelo. As quatro saídas da linha de SAD (SAD0 a SAD3 na figura 5.3) entregam para o comparador quatro valores de SADs de blocos candidatos a cada ciclo de relógio. Em quatro ciclos todos os 13 valores de uma linha de SAD já foram entregues ao comparador. A arquitetura do comparador deve ser capaz de identificar qual é o menor dentre os 13 valores de uma linha de SAD. Quando este

menor valor é encontrado, o comparador necessita ainda compará-lo com o menor valor dentre os SADs da linha anterior. A ligação entre os diversos comparadores pode ser observada na figura 5.1. O menor SAD e o vetor de movimento do bloco candidato que gerou este menor SAD são disponibilizados na saída para o comparador da próxima linha de SAD.

A arquitetura do comparador é apresentada na figura 5.5. Essa arquitetura é formada por operadores que realizam a subtração entre dois valores de SAD. A partir daí, o MSB do resultado é testado para que se saiba qual dos dois valores é o menor. O menor valor de SAD e seu respectivo vetor de movimento são passados ao próximo estágio para nova comparação. Este processo termina quando todos os valores de uma linha de SAD tiverem sido comparados. Como somente valores positivos são permitidos nas entradas, esta estrutura de comparações com subtratores funciona corretamente. As entradas são SADs de blocos. No melhor caso, esses SADs possuem valor igual à zero.

Há, em determinado momento, a necessidade de se comparar o menor SAD da linha com o menor SAD da linha anterior. Esta é sempre a última comparação a ser realizada.

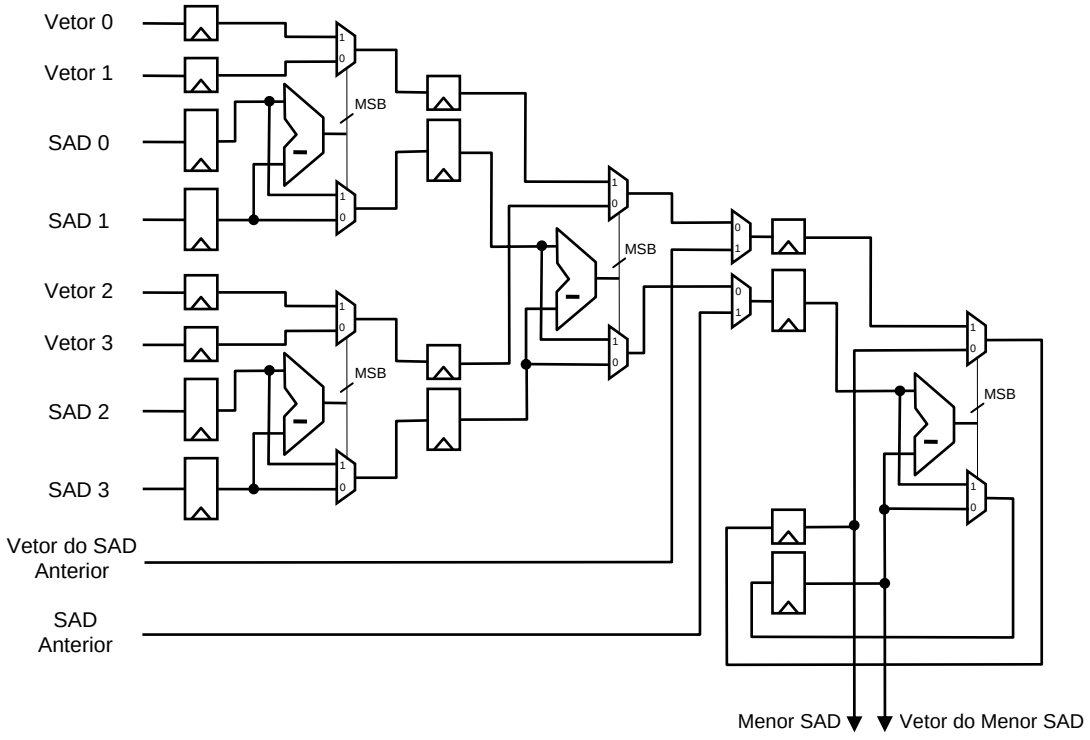


Figura 5.5: Diagrama em módulos do comparador.

No quarto estágio do *pipeline* (figura 5.5) estão o registrador do menor SAD da linha e o registrador para o vetor de movimento desse menor SAD. O valor destes registradores é enviado para o comparador da próxima linha de SADs. Se a linha de SADs for a última, então o vetor de movimento do bloco de menor SAD dentre todos os blocos candidatos é enviado para a saída da estimação de movimento.

5.5 Controle da Arquitetura para Estimação de Movimento de Blocos de Tamanho 4x4 *pixels*

O controle da ME é umas das etapas de maior complexidade no desenvolvimento do estimador, devido à grande quantidade de hardware que deve ser controlada. Em virtude disso, o controle foi dividido com o objetivo de facilitar a sua implementação.

A primeira parte do controle, a mais baixa hierarquicamente, controla a linha de SAD. Assim, cada linha de SAD tem seu próprio controle interno reduzindo em muito a complexidade do controle externo. Esta etapa do controle é responsável por sincronizar unidades de processamento, acumuladores, somadores, multiplexadores e demultiplexadores. Também nesta etapa são gerados os quatro bits mais significativos do vetor de movimento. Os outros quatro bits que indicam a posição que o vetor de movimento aponta são gerados fora desta etapa. Oito estados foram utilizados na máquina de estados do controle da linha de SAD.

O segundo nível da hierarquia de controle é aplicado para disparar as atividades das linhas de SAD e dos comparadores. Para isso são gerados 13 sinais que habilitam a operação de cada uma das linhas de SAD e mais 13 sinais que habilitam o funcionamento dos comparadores. Além disso, esta parte do controle sincroniza os registradores de linhas de bloco atual e de área de busca com os outros módulos da arquitetura. Esta etapa do controle foi implementada com uma máquina de 64 estados.

O terceiro e último nível da hierarquia controla o uso das memórias pelo restante da arquitetura. Nesta etapa do controle são gerados, em tempos fixos, os endereços de leitura e de escrita das memórias de bloco atual e de área de busca. Também são gerados os sinais de habilitação de escrita ou leitura nas memórias e demais sinais de controle que sincronizam os módulos que compõem o estimador de movimento. Novamente foi utilizada uma máquina de estados com 64 estados para implementar esta etapa do controle.

Neste capítulo foram apresentados os detalhes da implementação da arquitetura proposta e desenvolvida neste trabalho para realizar estimação de movimento sobre blocos de 4x4 *pixels*. No capítulo seguinte serão apresentados detalhes sobre a arquitetura para estimação de movimento para tamanhos de blocos variáveis.

6 ARQUITETURA PARA ESTIMAÇÃO DE MOVIMENTO DE BLOCOS DE TAMANHOS DE VARIÁVEIS

A estratégia utilizada neste trabalho para realizar estimação de movimento de blocos de tamanhos variáveis foi reusar os SADs calculados para os blocos menores para calcular os SADs dos blocos maiores.

Ao se reusar os SADs calculados anteriormente é necessário ter cuidado. Ao se reusar cálculos, nesse caso, não se pode simplesmente acumular os melhores resultados encontrados para o tamanho de bloco menor e formar o resultado para o bloco maior. As figuras 6.1, 6.2 e 6.3 exemplificam o que foi exposto através da busca de dois blocos de 4x4 *pixels*, 0 e 1, primeiro separadamente e, após, juntos, compondo um bloco 8x4 *pixels*, 0-1. Na figura 6.1 é apresentada a busca de um bloco de 4x4 *pixels*, o bloco numerado com 0. A mesma busca, mas agora para o bloco 1 é mostrada na figura 6.2. Tanto na figura 6.1 quanto na figura 6.2, a área da esquerda representa o quadro atual e a área da direita, o quadro de referência. Dentro do quadro de referência, a área listrada representa a área de busca em torno da posição original do bloco 4x4, marcada por um quadrado pontilhado. Os melhores casamentos, resultantes da estimação de movimento, são representados pelas novas posições de 0 e 1. Agora, se a busca fosse pelo melhor casamento para o bloco de 8x4 *pixels* 0-1 formado pelos blocos 0 e 1, não podemos simplesmente acumular os SADs de 0 e 1 para obtermos o resultado. Além disso, os resultados para 0 e 1 não são contíguos no quadro de referência e, portanto, não formam um bloco 8x4. Assim, acumular os menores SADs de blocos 4x4 nem sempre leva ao menor SAD do bloco de 8x4 *pixels*. Quando agrupados, os SADs dos dois blocos menores podem levar a outro resultado que pode não ser o melhor casamento para este tamanho de bloco. O resultado para a busca do bloco 0-1 é apresentada na figura 6.3. Nota-se facilmente que o resultado encontrado na figura 6.3 não tem relação alguma com os resultados independentes encontrados nas figuras 6.1 e 6.2. A busca de um bloco maior a partir do agrupamento de dois blocos menores caracteriza uma busca totalmente independente. Dessa forma, para reusar cálculos de SAD, deve-se guardar os SADs de todos os blocos candidatos tanto do bloco 0, quanto do bloco 1. Assim, na busca do melhor casamento para o bloco 0-1, o resultado deverá ser o menor SAD de todas as possíveis somas dos SADs de 0 e 1 em posições contíguas e não a soma do menor SAD para o bloco 0 com o menor SAD para o bloco 1.

A forma de reuso de SADs apresentada no exemplo serve, também, para os outros tamanhos de bloco possíveis e segue a estrutura apresentada na figura 6.4. Blocos

candidatos de 8x8 *pixels* serão formados por blocos candidatos de 4x8 *pixels*; os de 16x8, formados pelos de 8x8; e assim por diante.

A arquitetura para estimação de movimento de tamanhos de bloco variáveis (VBSME) é apresentada na figura 6.5. Essa arquitetura leva em conta o reuso de SADs já calculados, conforme o que foi explicado anteriormente. Os módulos que compõem esta arquitetura serão apresentados a seguir.

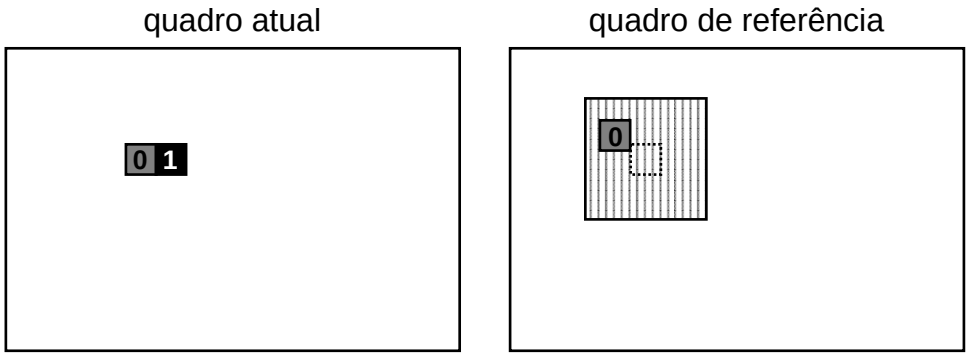


Figura 6.1: Estimação de movimento do bloco 0 (4x4 *pixels*).

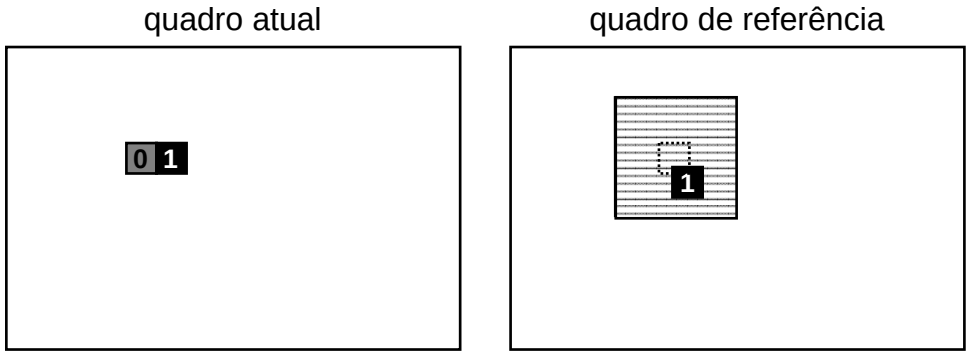


Figura 6.2: Estimação de movimento do bloco 1 (4x4 *pixels*).

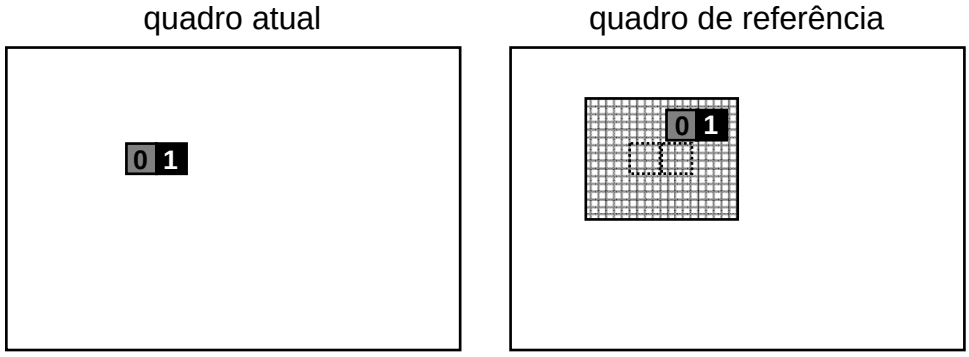


Figura 6.3: Estimação de movimento do bloco 0-1 (8x4 *pixels*).

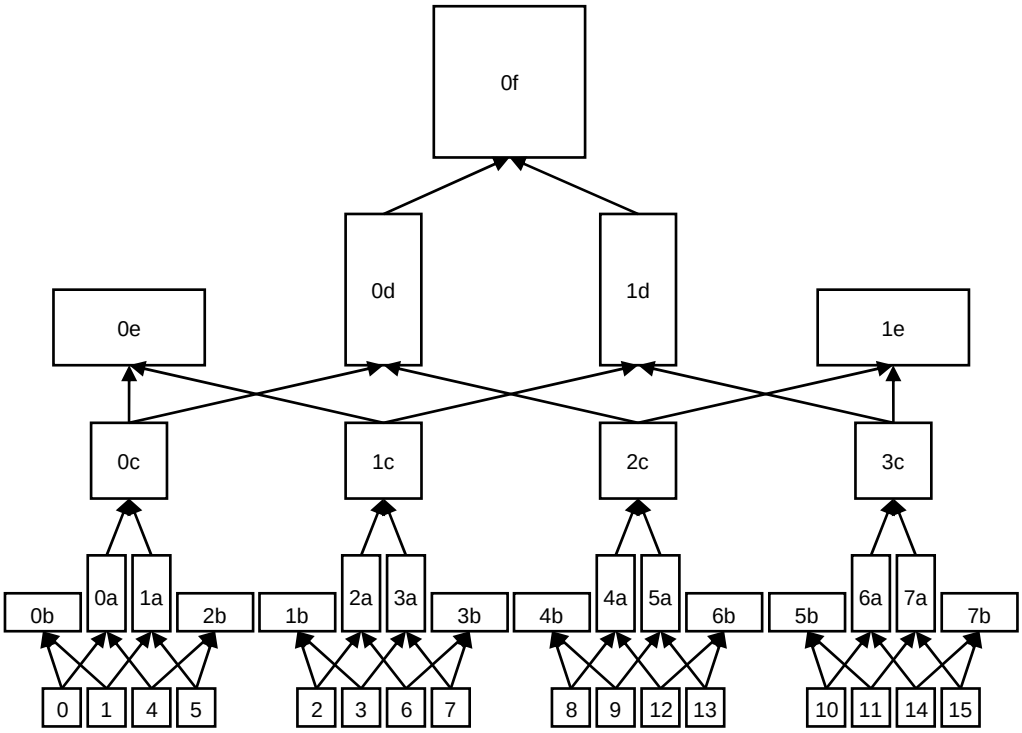


Figura 6.4: Agrupamento de SADs no padrão H.264/AVC.

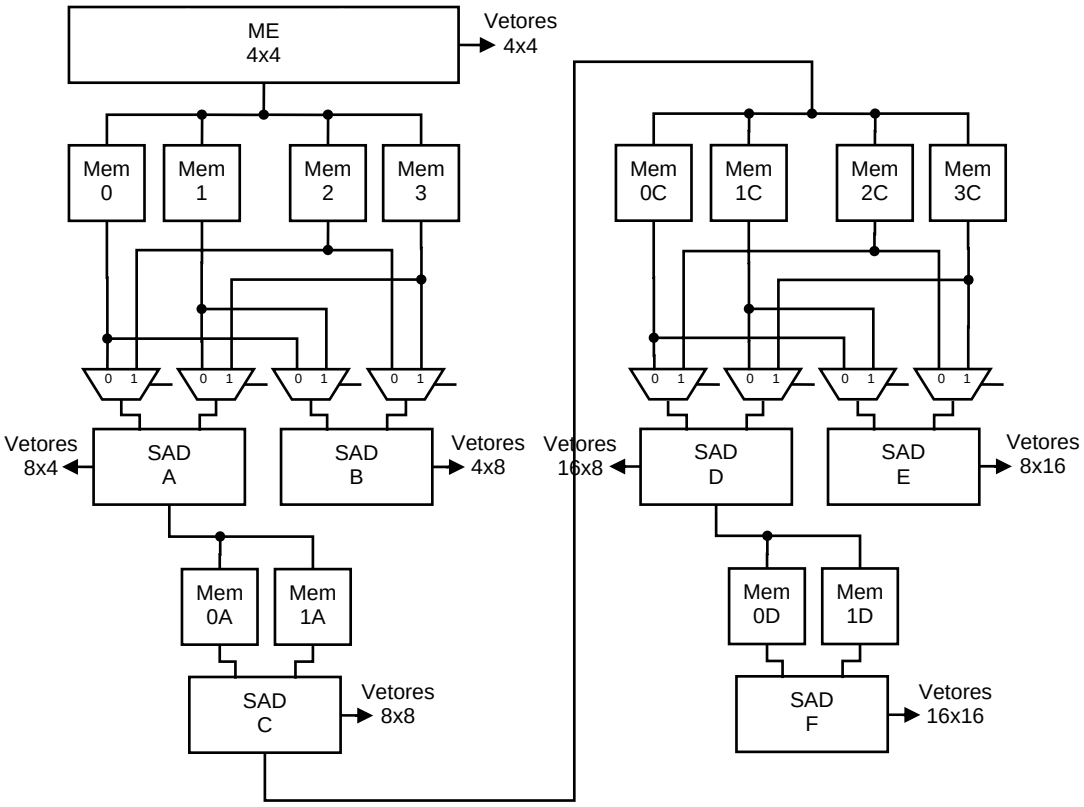


Figura 6.5: Diagrama em módulos da arquitetura para tamanhos de bloco variáveis.

6.1 Arquitetura para Estimação de Movimento de Blocos de 4x4 *Pixels*

O módulo ME 4x4 apresentado na figura 6.5 trata-se da arquitetura para estimação de movimento para blocos de 4x4 *pixels* apresentada no capítulo anterior. Este módulo foi adaptado para que a idéia de reaproveitar SADs anteriormente calculados pudesse funcionar. Dessa forma, este módulo apresenta duas saídas: uma na qual são disponibilizados os 16 vetores gerados na estimação de movimento para os 16 blocos de 4x4 *pixels* que compõem um bloco de 16x16 *pixels* e outra que envia, para quatro memórias, todos os SADs calculados para os blocos candidatos dos 16 blocos de 4x4 *pixels*.

Os resultados dos cálculos de SAD que são armazenados nas memórias são retirados antes da etapa de comparação da ME 4x4. Posteriormente esses resultados intermediários poderão ser agrupados para escolha de um melhor casamento entre blocos de outros tamanhos.

6.2 Memórias

As memórias apresentadas na figura 6.5 são responsáveis por armazenar os SADs dos blocos candidatos para que estes possam, posteriormente, ser agrupados para formar SADs de blocos maiores. A memória total utilizada fica em torno de 51 KB, incluindo as memórias utilizadas pela ME 4x4.

Foram implementados quatro tamanhos diferentes de memória para armazenar os resultados do cálculo de SADs para os blocos 4x4, 8x4, 8x8 e 16x8 *pixels*. Outros tamanhos, como 4x8 e 8x16 *pixels*, não foram necessários, pois já havia a opção de calcular os SADs 8x8 a partir dos SADs 8x4, no caso dos SADs 4x8, e os 16x16 a partir dos 16x8, no caso dos 8x16.

6.2.1 Memórias para SADs de Blocos Candidatos de Tamanho 4x4 *Pixels*

As memórias Mem 0, Mem 1, Mem 2 e Mem 3, mostradas na figura 6.5, servem para armazenar, cada uma, 169 SADs referentes aos blocos candidatos de cada bloco de 4x4 *pixels*. É como se as quatro memórias, juntas, armazenassem SADs referentes a um bloco inteiro de 8x8 *pixels*. Isso pode ser visto através da figura 6.6. De 0 a 15 estão numerados os 16 blocos de 4x4 *pixels* que formam um macrobloco 16x16 *pixels*. Na figura 6.6 também é apresentada a ordenação utilizada pela arquitetura, o "duplo Z". Com ordenação em "duplo Z" torna-se mais fácil agrupar os SADs do que em ordem normal além de necessitar uma menor quantidade de memória. Em ordem normal seria necessário o dobro de memória. Isso se deve ao fato de que, com ordenação em "duplo Z", as dependências de dados ficam restritas a blocos de 8x8 *pixels* necessitando, assim, apenas quatro memórias para armazenamento dos SADs. Se a ordenação escolhida fosse a ordenação normal, seriam necessárias oito memórias já que a dependência ficaria dentro de blocos 16x8 pixels. Em ordenação normal, por exemplo, seria necessário armazenar os SADs referentes ao bloco 3 até o momento em que os SADs do bloco 7 estivessem prontos. Isso porque são necessários os dados do bloco 3 e do bloco 7 para calcular os SADs dos blocos candidatos do bloco 3-7.

Com as memórias preenchidas, os SADs 4x4 são usados para calcular os SADs 8x4 e 4x8. Os SADs 8x4 são obtidos através do arranjo entre os dados lidos das memórias Mem0 e Mem1 (SADs dos blocos candidatos do primeiro bloco de 8x4 *pixels*) e Mem2 e Mem3 (SADs dos blocos candidatos do segundo bloco de 8x4 *pixels*). Os SADs 4x8

são obtidos agrupando-se os valores lidos das memórias Mem0 e Mem2 (blocos candidatos do primeiro bloco de 4x8 *pixels*) e Mem1 e Mem3 (blocos candidatos do segundo bloco de 4x8 *pixels*). Após o cálculo dos SADs dos blocos candidatos do segundo bloco de 4x8 *pixels*, as memórias podem ler novos valores que serão usados nos cálculos dos próximos blocos candidatos.

A seqüência de funcionamento é a seguinte: preenchem-se as quatro memórias com os SADs dos blocos candidatos 0, 1, 4 e 5; calculam-se os SADs para os blocos candidatos 0-1, 0-4, 4-5 e 1-5; preenchem-se as memórias com os valores referentes a 2, 3, 6 e 7; calculam-se os SADs para os blocos candidatos 2-3, 2-6, 6-7 e 3-7; e assim por diante até que se tenha calculado os SADs para 11-15, completando um macrobloco.

Cada memória para blocos candidatos de 4x4 *pixels* possui 3840 bits com 64 palavras de 60 bits. As quatro memórias totalizam 15 KB.

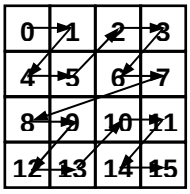


Figura 6.6: Blocos de 4x4 *pixels* que compõem um macrobloco e ordenação em “duplo Z”.

6.2.2 Memórias para SADs de Blocos Candidatos de Tamanho 8x4 *Pixels*

As memórias utilizadas para armazenar os SADs de blocos candidatos com tamanho de 8x4 pixels trabalham da mesma forma que as memórias usadas no tamanho 4x4, mas, por sua vez, armazenam SADs de blocos candidatos de 8x4 *pixels*. Da mesma forma que para blocos 4x4, armazenam 169 valores cada uma.

Os resultados de SAD gerados por SAD A (figura 6.5), referentes a blocos de 8x4 *pixels* são armazenados nestas memórias para que possam, posteriormente, ser utilizados no cálculo dos SADs de blocos candidatos de tamanho 8x8 *pixels*. Na figura 6.5 estas memórias estão representadas por Mem 0A e Mem 1A. Como o funcionamento desta etapa é semelhante ao funcionamento da etapa anterior, o uso de “duplo Z” como ordenação levou à utilização de apenas duas memórias ao contrário das quatro memórias que seriam necessárias em ordem normal.

As memórias para blocos candidatos de 8x4 *pixels* são compostas, cada uma, por 64 palavras de 64 bits, resultando em 4096 bits. Ao todo, 8 KB são utilizados para as duas memórias.

É importante salientar que, como os SADs para blocos de 8x8 *pixels* são calculados a partir de SADs de blocos de 8x4, não há a necessidade de armazenar os resultados de SAD dos blocos candidatos de 4x8 *pixels*.

6.2.3 Memórias para SADs de Blocos Candidatos de Tamanho 8x8 *Pixels*

A parte da arquitetura posterior ao cálculo dos SADs dos blocos candidatos de 8x8 *pixels* é muito semelhante à anterior. Isso se deve ao particionamento de um macrobloco 16x16 *pixels* ser igual ao particionamento de um bloco 8x8 *pixels* como apresentado no

capítulo 3 (figura 3.10). Dessa forma, neste estágio são utilizadas quatro memórias para armazenar SADs referentes a blocos candidatos de tamanho 8x8 *pixels.* Essas memórias são mostradas na figura 6.5 como Mem 0C, Mem 1C, Mem 2C e Mem 3C.

Nesta etapa da arquitetura as memórias são compostas por 64 palavras de 68 bits totalizando 4352 bits em cada memória, em um total de 17 KB para as quatro memórias.

6.2.4 Memórias para SADs de Blocos Candidatos de Tamanho 16x8 *Pixels*

As memórias Mem 0D e Mem 1D na figura 6.5 servem para armazenar, cada uma, 169 SADs de blocos candidatos de 16x8 *pixels* provenientes do módulo SAD D. Esses valores serão utilizados posteriormente para calcular os SADs dos blocos 16x16 *pixels*. Cada memória é composta por 64 palavras de 72 bits chegando ao total de 4608 bits. As duas memórias somam 9 KB.

Novamente, não há a necessidade de armazenar os SADs provenientes do módulo SAD E (8x16 *pixels*) já que o cálculo para 16x16 *pixels* é feito através de resultados oriundos do módulo SAD D (8x16 *pixels*).

6.3 Arquitetura para Estimação de Movimento de Blocos de Tamanho 4x8, 8x4, 8x8, 8x16, 16x8 e 16x16 *Pixels*

As arquiteturas para o cálculo de SADs para blocos de tamanhos 4x8, 8x4, 8x8, 8x16, 16x8 e 16x16 *pixels* são muito semelhantes à arquitetura do comparador mostrada no item 5.4. A diferença está nos quatro somadores necessários para agrupar os SADs antes de compará-los na busca do melhor casamento. Na figura 6.5 essas arquiteturas estão representadas com os nomes SAD A, SAD B, SAD C, SAD D, SAD E, e SAD F.

Estes módulos foram projetados em um *pipeline* de cinco estágios. No primeiro estágio, são agrupados oito SADs, quatro referentes a cada bloco, em quatro novos SADs. Nos últimos quatro estágios os quatro SADs gerados são comparados em busca do menor SAD. O último estágio guarda o menor SAD encontrado e seu respectivo vetor de movimento.

Para uma melhor ilustração do funcionamento desses módulos será mostrado como trabalha o módulo SAD A. Os outros módulos (B, C, D, E, e F) funcionam de forma semelhante, o que muda é a origem dos dados de suas entradas.

A cada ciclo, o módulo SAD A recebe quatro SADs provenientes de Mem0 e quatro de Mem1 ou então quatro de Mem2 e quatro de Mem3, dependendo do bloco 4x8 que estiver sendo buscado. Em quatro ciclos, cada memória repassa ao módulo 13 SADs referentes aos blocos candidatos de uma linha, totalizando 26 valores de SAD. As entradas In0 e In1 são utilizadas para este fim. Os respectivos vetores também são passados para o módulo. A partir daí, esses SADs, que são referentes a blocos candidatos de 4x4 *pixels*, são agrupados para formar SADs de tamanho 8x4 *pixels* referentes aos novos blocos candidatos. Recorrendo às figuras 6.1, 6.2 e 6.3 pode-se dizer que para obter o SAD do primeiro bloco candidato de 8x4 (de “0-1”), é necessário somar o SAD do primeiro bloco candidato de “0” com o primeiro bloco candidato de “1”.

Após essa etapa de soma de SADs, estes valores percorrem os próximos estágios para comparação e busca do menor valor. A parte responsável pela comparação é

semelhante ao comparador mostrado no item 5.4, utilizando-se da mesma organização anterior, conforme pode ser observado na figura 6.7.

Resumindo o processo, no total 169 valores de SADs de cada memória são agrupados dois a dois para formar 169 novos valores de SAD referentes aos blocos candidatos de um bloco 4x8 *pixels*. Ao final, o menor destes 169 novos SADs é escolhido e seu respectivo vetor de movimento é enviado para a saída. Os 169 novos valores de SAD são enviados para a saída para serem armazenados em outra memória. Essa saída está ligada aos registradores do segundo estágio de *pipeline* e não é mostrada na figura 6.7 para simplicidade de visualização. Finalizando, para um macrobloco 16x16 *pixels,* são gerados oito vetores de movimento referentes aos oito blocos 4x8 possíveis.

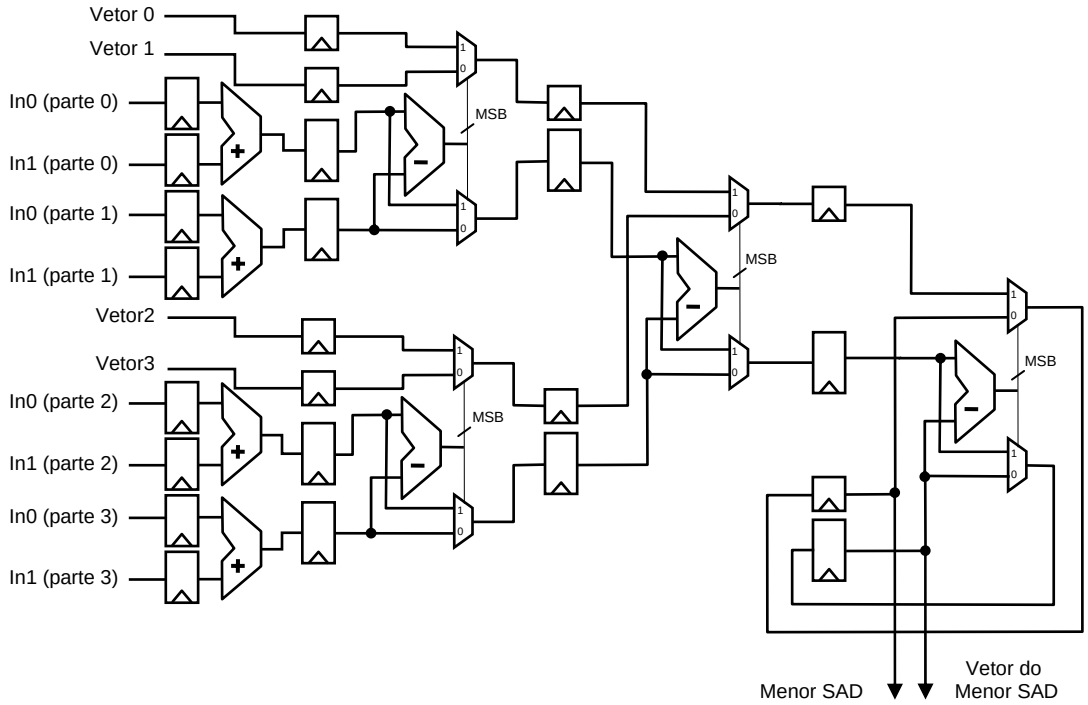


Figura 6.7: Arquitetura para o acúmulo do SAD.

O módulo SAD B trabalha de forma semelhante ao módulo SAD A, apenas mudando a orientação, ou seja, calcula os SADs para blocos de 4x8 *pixels*. Dessa forma, enquanto o SAD A agrupa valores de Mem0 e Mem1, o SAD B agrupa valores de Mem0 e Mem2. Quando o SAD A agrupa valores de Mem2 e Mem3, o SAD B agrupa valores de Mem1 e Mem3. Da mesma forma que para o SAD A, para um macrobloco 16x16 *pixels* são gerados oito vetores de movimento referentes aos oito blocos 8x4 possíveis.

Com os SADs dos blocos de 8x4 *pixels* tendo sido calculados, pode-se agrupá-los para formarem os SADs dos blocos de 8x8 *pixels*. De forma semelhante ao SAD A, o SAD C utiliza valores de SAD armazenados nas memórias Mem0A e Mem1A para calcular os SADs dos novos blocos candidatos. A forma de agrupamento de valores e de geração de vetores é a mesma utilizada pelo SAD A e os demais blocos. Quatro vetores de movimento são gerados e referem-se aos quatro blocos de 8x8 *pixels* possíveis em

um macrobloco 16x16 *pixels*. Os SADs calculados para os blocos candidatos de tamanho 8x8 *pixels* são armazenados nas memórias Mem0C, Mem1C, Mem2C, e Mem3C. Posteriormente os módulos SAD D e SAD E utilizarão estes valores para fazer a estimação de movimento dos blocos de 16x8 e 8x16 *pixels*. Serão gerados, nesta etapa, dois vetores para os blocos 16x8 e dois para os blocos 8x16. Finalizando o processo de estimação de movimento para tamanhos de bloco variáveis, o vetor para o macrobloco 16x16 *pixels* é gerado. Este vetor é gerado pelo módulo SAD F a partir dos valores armazenados nas memórias Mem0D e Mem1D.

Com isso, os seis módulos adicionais mostrados e a ME 4x4 geram os 41 vetores de movimento especificados pelo padrão H.264/AVC. Estes vetores são encontrados sem que nenhuma busca adicional seja necessária, como foi mostrado.

6.4 Controle da Arquitetura para Estimação de Movimento de Blocos de Tamanhos Variáveis

O controle da arquitetura para tamanhos de blocos variáveis abrange o controle das arquiteturas para o cálculo de SADs para blocos de tamanhos 4x8, 8x4, 8x8, 8x16, 16x8 e 16x16 *pixels* e o controle das memórias com os SADs de blocos candidatos. O controle da estimação de movimento para blocos de 4x4 *pixels* é interno, mas dependente de sinais gerados pelo controle global da arquitetura que trata múltiplos tamanhos de blocos. Os sinais gerados pelo controle da arquitetura para tamanhos de bloco variáveis garantem que os módulos internos da arquitetura funcionem de forma sincronizada e que nenhum dado seja perdido durante o processamento.

Em relação às arquiteturas de cálculo de SADs para blocos de tamanhos 4x8, 8x4, 8x8, 8x16, 16x8 e 16x16 *pixels* o controle gera sinais que sincronizam o funcionamento destes módulos com o funcionamento das memórias. Há sinais de controle para os muxes disponibilizarem, de acordo com a etapa do processamento, as entradas corretas para esses módulos. Há ainda sinais de *enable* para iniciar ou finalizar o funcionamento das arquiteturas de cálculo de SADs, bem como sinais indicando quando há um vetor de movimento válido na saída. Esses sinais são disponibilizados independentemente para cada módulo de cálculo de SAD sendo ativados ou desativados em tempos diferentes.

Em se tratando das memórias com os SADs de blocos candidatos, o controle gera os sinais para habilitação de escrita ou leitura, de acordo com a disponibilidade de dados válidos das arquiteturas de cálculo de SADs. Além disso, o controle gera os endereços de escrita e de leitura para as memórias.

O controle da estimação de movimento para blocos de 4x4 *pixels* é feito com alguns poucos sinais que habilitam ou desabilitam seu funcionamento. Como dito anteriormente, o controle da estimação de movimento para blocos de 4x4 *pixels* é interno e descentralizado, mas dependente dos sinais gerados pelo controle global. Essa forma descentralizada de controle diminuiu, em muito, a complexidade necessária no controle de toda a arquitetura para tamanhos de bloco variáveis.

Os detalhes sobre a arquitetura para estimação de movimento de blocos de tamanhos variáveis foram apresentados neste capítulo. No capítulo a seguir serão apresentados os resultados de síntese e de validação da arquitetura.

7 RESULTADOS DE SÍNTESE E DE VALIDAÇÃO DA ARQUITETURA

Este capítulo aborda os resultados de síntese e de validação da arquitetura para estimação de movimento para blocos de tamanhos variados. A arquitetura e todos os seus sub-módulos foram descritos em VHDL (ASHENDEN, 1998). A descrição foi realizada de forma hierárquica, totalizando aproximadamente 6800 linhas de código em 24 arquivos. A síntese da arquitetura foi direcionada para dois FPGAs e para *standard cells*.

Os próximos itens do texto apresentam detalhes dos resultados de síntese e de validação, bem como uma discussão sobre as resoluções de vídeo suportadas pela arquitetura. Esses resultados servem de suporte ao próximo capítulo deste trabalho, onde é realizada uma comparação com trabalhos relacionados encontrados na literatura.

7.1 Resultados de Síntese

As tabelas 7.1, 7.2 e 7.3 apresentam os resultados de síntese da VBSME desenvolvida nesta dissertação, sendo que as tabelas 7.1 e 7.2 apresentam resultados para FPGAs e a tabela 7.3 para uma versão *standard cell* da arquitetura. A síntese para FPGAs foi direcionada para os dispositivos VP 30, da família Virtex 2 Pro, e VLX 25, da família Virtex 4, ambos da Xilinx (XILINX, 2007). A ferramenta utilizada na descrição e síntese da arquitetura foi o ISE, também da Xilinx. No caso da síntese da arquitetura para *standard cells*, foi utilizada a tecnologia TSMC 0.18um e, como ferramenta de síntese, o Leonardo Spectrum da Mentor Graphics (MENTOR GRAPHICS, 2007).

Utilizando-se o FPGA Virtex 2 Pro VP 30 foi possível atingir 242,41 MHz como freqüência máxima. Com este desempenho a arquitetura é capaz de processar, por exemplo, 171 quadros SDTV (720 x 480 *pixels*) por segundo. Para HDTV (1920x1080 *pixels*) a arquitetura pode processar 28 quadros por segundo. Estes resultados mostram que a arquitetura pode operar em tempo real para as duas resoluções considerando-se, como tempo real, taxa de amostragem entre 25 e 30 quadros por segundo.

Em relação à utilização dos recursos do FPGA, a tabela 7.1 apresenta os resultados de síntese da arquitetura para um dispositivo da família Virtex 2 Pro da Xilinx. Na síntese direcionada para este dispositivo, a arquitetura utilizou 8.906 *slices* (65% do total), 11.710 *slice flip flops* (42% do total), 15.080 LUTs (55% do total) e 29 BRAMs (21% do total). Estes percentuais mostram que a área ocupada pela arquitetura é

bastante grande. Se um codificador completo tivesse de ser sintetizado para FPGA seria necessário mapeá-lo para um dispositivo maior para que coubessem, além da estimação de movimento, os outros módulos que o compõe.

Tabela 7.1: Resultados de síntese para um dispositivo da família Virtex 2 Pro.

	Slices	Slice Flip Flops	4 Input LUTs	BRAMs
Utilizado	8.906	11.710	15.080	29
Disponível	13.696	27.392	27.392	136
Porcentagem	65%	42%	55%	21%

Dispositivo Virtex2P 2VP30FG676-7

Na síntese direcionada para o dispositivo Virtex 4 VLX 25 a freqüência máxima alcançada foi de 266,28 MHz. Assim, podem ser processados 188 quadros SDTV por segundo e, no caso de HDTV, 31 quadros por segundo. Estes resultados são melhores que os anteriores, o que já era esperado já que os FPGAs da família Virtex 4 possuem uma tecnologia mais atual do que os da família Virtex 2 Pro. Novamente, a arquitetura pode operar em tempo real para as duas resoluções utilizadas como exemplo. No próximo item deste capítulo serão apresentadas taxas de processamento para outras resoluções de vídeo.

A tabela 7.2 apresenta os resultados de síntese da arquitetura para um dispositivo da família Virtex 4 da Xilinx. Na tabela 7.2 os resultados de síntese mostram que, para esse FPGA, a arquitetura ocupou 8.858 *slices* (82% do total), 11.727 *slice flip* flops (54% do total), 14.526 LUTs (67% do total) e 29 BRAMs (40% do total). Os resultados de área são semelhantes para os dois dispositivos, diferenciando-se apenas nos resultados percentuais, devido à diferença de capacidade (em LUTs, *flip flops*, e *Block* RAMs) entre os FPGAs. Novamente vale o que foi discutido para a versão anterior da arquitetura. Um FPGA maior seria necessário se a síntese fosse não apenas da estimação de movimento mas para o codificador inteiro.

Tabela 7.2: Resultados de síntese para um dispositivo da família Virtex 4.

	Slices	Slice Flip Flops	4 Input LUTs	BRAMs
Utilizado	8.858	11.727	14.526	29
Disponível	10.752	21.504	21.504	72
Porcentagem	82%	54%	67%	40%

Dispositivo Virtex4 4VLX25FF676-12

No caso da síntese da arquitetura para *standard cells* os resultados de síntese são apresentados na tabela 7.3. A arquitetura, nesta versão, alcançou 80 MHz de freqüência máxima, em síntese não otimizada, podendo processar 56 quadros SDTV (720 x 480 *pixels)* por segundo ou, no caso de HDTV (1920x1080 pixels), 9 quadros por segundo. Considerando-se uma taxa de amostragem entre 25 e 30 quadros por segundo como tempo real, estes resultados apontam que a arquitetura pode operar em tempo real para SDTV. Ainda sobre a síntese da arquitetura para *standard cells*, foram utilizadas em torno de 120 mil portas lógicas e 36 KB de memória.

O próximo capítulo desta dissertação fará uma comparação dos resultados obtidos neste capítulo com resultados apresentados em outros trabalhos que constam na literatura, relacionando vários parâmetros como: número de unidades de processamento, tamanho da área de busca, tecnologia utilizada, freqüência máxima alcançada, utilização de área e memória, e taxa de processamento.

Tabela 7.3: Resultados de síntese para *standard cells*.

Portas lógicas	Memória	Freqüência
120 K	36 KB	80 MHz
		TSMC 0.18um

7.2 Discussão Sobre Taxas de Processamento

Uma seqüência de vídeo é formada por amostras tomadas em intervalos periódicos. Essas amostras referem-se a quadros que, após serem capturados, podem dar idéia de movimento se reproduzidos seqüencialmente. Esse tipo de amostragem é chamado de amostragem temporal. As taxas de amostragem e seu resultado visual são apresentados na tabela 7.4. Há que se considerar, é claro, um vídeo com movimentação satisfatória, como "Fórmula 1", por exemplo. Vídeos com pouco movimento são sempre exemplos ruins.

Segundo Richardson (RICHARDSON, 2002), em uma taxa de amostragem temporal entre 10 e 20 quadros por segundo o vídeo é reproduzido normalmente, com exceção de partes com movimentos mais rápidos, onde a seqüência de vídeo aparentará estar truncada. Uma amostragem entre 25 e 30 quadros por segundo apresenta melhores resultados sendo, inclusive, usada como padrão para imagens de televisão. Nesta faixa de amostragem já é possível se ter a noção de tempo real quando a seqüência de vídeo é reproduzida. Se for possível utilizar uma taxa de bits mais elevada, 50 a 60 quadros por segundo, por exemplo, então será possível uma visualização de movimento mais suave, mas normalmente, tais taxas são usadas apenas em ambientes profissionais em função do elevado custo computacional para seu processamento.

Tabela 7.4: Taxas de amostragem temporal (RICHARDSON, 2002).

Taxa de Amostragem	Aparência do Vídeo
abaixo de 10 quadros por segundo	movimento não natural, irregular
entre 10 e 20 quadros por segundo	movimento lento é mostrado normalmente; movimento rápido fica claramente irregular
entre 20 e 30 quadros por segundo	movimento razoavelmente suave
entre 50 e 60 quadros por segundo	movimento suave

A partir do que foi exposto nos parágrafos anteriores, pode-se observar, com a ajuda da tabela 7.5, quais os casos em que a arquitetura desenvolvida neste trabalho alcança tempo real para as seqüências de vídeo. As taxas de processamento apresentadas na tabela 7.5 são dependentes da tecnologia em que a arquitetura foi implementada e das resoluções de vídeo possíveis. De acordo com a resolução da seqüência de vídeo também são apresentados, na tabela 7.5, o número de amostras e o número de macroblocos a serem processados.

Tabela 7.5: Taxa de processamento da arquitetura de estimação de movimento.

Resolução		Número de Blocos 16x16	Número de Amostras	Taxa de Processamento (QPS)		
Nome	Tamanho			Virtex2P	Virtex4	0.18um
SQCIF	128x96	48	12.288	4.818,90	5.293,48	1590,33
QCIF	176x144	99	25.344	2.336,44	2.566,53	771,07
QVGA	320x240	300	76.800	771,02	846,96	254,45
525 SIF	352x240	330	84.480	700,93	769,96	231,32
CIF	352x288	396	101.376	584,11	641,63	192,77
525 HHR	352x480	660	168.960	350,47	384,98	115,66
625 HHR	352x576	792	202.752	292,05	320,82	96,38
VGA	640x480	1.200	307.200	192,76	211,74	63,61
525 4SIF	704x480	1.320	337.920	175,23	192,49	57,83
525 SD	720x480	1.350	345.600	171,34	188,21	56,55
4CIF	704x576	1.584	405.504	146,03	160,41	48,19
625 SD	720x576	1.620	414.720	142,78	156,84	47,12
SVGA	800x600	1.875	480.000	123,36	135,51	40,71
XGA	1024x768	3.072	786.432	75,30	82,71	24,85
720p HD	1280x720	3.600	921.600	64,25	70,58	21,20
4VGA	1280x960	4.800	1.228.800	48,19	52,93	15,90
SXGA	1280x1024	5.120	1.310.720	45,18	49,63	14,91
525 16SIF	1408x960	5.280	1.351.680	43,81	48,12	14,46
16CIF	1408x1152	6.336	1.622.016	36,51	40,10	12,05
4SVGA	1600x1200	7.500	1.920.000	30,84	33,88	10,18
1080 HD	1920x1080	8.160	2.088.960	28,35	31,14	9,35
2K x 1K	2048x1024	8.192	2.097.152	28,24	31,02	9,32

A tabela 7.5 detalha a quantidade de macroblocos e o número de amostras a serem processadas para cada quadro em cada formato de vídeo. Assim, pode-se ter uma idéia precisa da quantidade de dados que necessitam ser processados para cada resolução de vídeo. Na tabela 7.5, as informações relativas à taxa de processamento foram obtidas através da freqüência alcançada por cada versão da arquitetura. Assim, é possível definir quantos quadros por segundo cada versão é capaz de processar e para quais resoluções a arquitetura opera em tempo real.

Considerando-se as versões da arquitetura com síntese direcionada para FPGA, é possível notar que ambas alcançam taxa de processamento acima de 25 quadros por segundo para todos os formatos apresentados. Estes resultados mostram que a arquitetura pode operar em tempo real para todas as resoluções apresentadas

considerando-se, como tempo real, uma taxa de amostragem entre 25 e 30 quadros por segundo. Portanto, estas versões podem ser utilizadas em aplicações como TV digital de resolução padrão (SDTV) ou de alta resolução (HDTV).

Para a versão *standard cells* da arquitetura os resultados são um pouco inferiores, haja vista que a síntese não foi otimizada para ASIC. Como a freqüência alcançada ficou em torno de 80MHz, a arquitetura alcança taxas de processamento para tempo real até a resolução XGA, 1024x768 *pixels*. Dessa forma, a arquitetura pode ser utilizada para aplicações que processam vídeo nos formatos compatíveis com SDTV.

7.3 Resultados de Validação

A validação da arquitetura para estimação de movimento apresentada neste trabalho tomou por base a comparação entre os resultados fornecidos pela arquitetura mapeada em FPGA e os resultados obtidos através de *software* específico. O fluxograma da validação é apresentado na figura 7.1. Nesta etapa do trabalho foram descritos programas em C com a finalidade de gerar arquivos com os blocos a serem estimados e suas respectivas áreas de busca, os SADs resultantes e os vetores de movimento. Além destes, foi descrito em C um programa para realizar a comparação entre vetores de movimento gerados pela arquitetura e os vetores de movimento obtidos através de *software*.

A seqüência de vídeo utilizada na validação foi a denominada “Formula 1”, obtida na página do *Video Quality Experts Group* (VQEG, 2007) e apresentada anteriormente no capítulo 4. Essa seqüência de vídeo possui uma resolução de 720x480 *pixels* (525SD). Foram utilizados os 10 primeiros quadros dessa seqüência nesta etapa do trabalho. Assim, um total de 3.456.000 amostras foram processadas na validação.

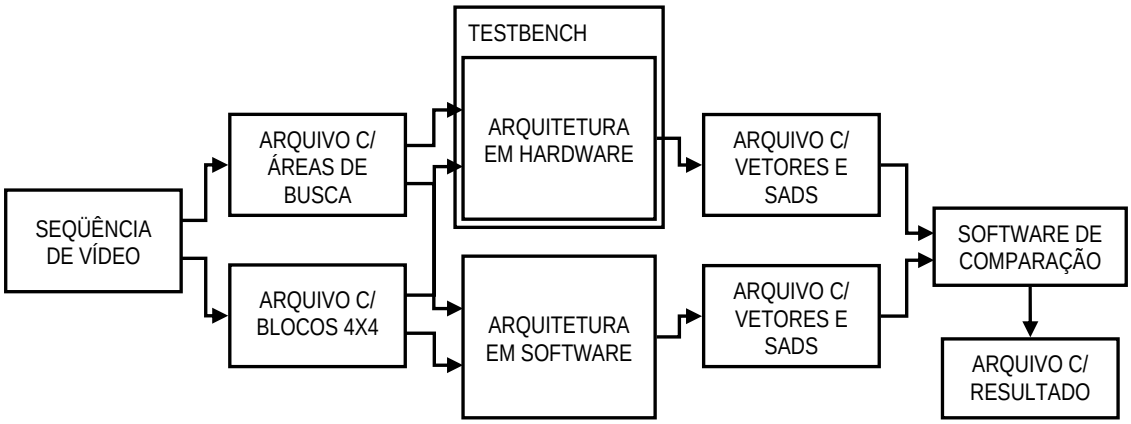


Figura 7.1: Fluxograma da validação da arquitetura.

Com a seqüência de vídeo disponível, foram gerados os arquivos de estímulo para a arquitetura e os arquivos utilizados na fase de comparação. Como o número de blocos 4x4 para 10 frames da seqüência de vídeo é 216.000 ((720/4 x 480/4) x 10 = 216.000), cada arquivo foi gerado com 216.000 elementos. Assim, foi gerado um arquivo com 216.000 blocos a serem buscados, um arquivo com 216.000 áreas de buscas referentes aos blocos, um arquivo com 216.000 vetores de movimento resultantes da estimação e outro arquivo com 216.000 SADs referentes aos vetores resultantes. Todos os arquivos

foram salvos em modo texto, ocupando os seguintes espaços: 439 MB, para o arquivo com áreas de busca; 29 MB, para o arquivo contendo os blocos; 2,1 MB, para o arquivo com os vetores de movimento; e 3,5 MB, para o arquivo incluindo os SADs calculados.

Utilizando-se a ferramenta ModelSim, da Mentor Graphics (MENTOR GRAPHICS, 2007), criou-se um *testbench* para, através dos arquivos com os blocos e com as áreas de busca, estimular a arquitetura e obter, na saída, os resultados a serem comparados posteriormente para verificar o correto funcionamento da arquitetura. Desta etapa obtiveram-se mais dois arquivos, um com os vetores de movimento gerados pela arquitetura e outro com os SADs calculados referentes a estes vetores.

O próximo passo foi a comparação entre vetores de movimento gerados pela arquitetura e os vetores de movimento obtidos através de *software*. Um programa descrito em C comparou os arquivos gerados pelas duas implementações e gerou um relatório com os resultados da comparação. Estes resultados são resumidos na tabela 7.6.

Tabela 7.6: Resultados de validação.

	Vetores de Movimento	**SADs Calculados**
Diferenças entre *hardware* e *software*	37.265	105
Total	216.000	216.000
Porcentagem	17,25%	0,05%

Valores referentes a 10 frames.

Do total de 216.000 vetores de movimento, 37.265 apontaram para posições diferentes nas áreas de busca entre as duas implementações. A porcentagem de valores discrepantes chegou a 17,25%, o que indicou a necessidade de uma avaliação cuidadosa dos dados apresentados.

Esses resultados conflitantes se devem à forma diferenciada com que os arquivos contendo os vetores de movimento foram obtidos. Assim, partiu-se para a avaliação dos SADs calculados referentes a estes vetores de movimento. De 216.000 SADs calculados apenas 105 apresentaram resultados discrepantes, cerca de 0,05%. Isso mostra que tanto a arquitetura como o programa descrito em C encontraram, em diversas situações, blocos de melhor similaridade, com SADs idênticos, mas em posições diferentes. Nota-se que, nestes casos, havia mais de um bloco dentro da área de busca que conduzia ao mesmo SAD relativo ao melhor casamento. Para todos estes casos, a diferença no vetor gerado não era, na verdade, um erro, pois não conduzia a resultados piores em termos de qualidade da estimação. Após esta constatação, verificou-se, ainda, que a maioria dos 105 valores discrepantes de SAD apresentava apenas pequenas diferenças entre as duas implementações, e que estas diferenças foram causadas em função do arredondamento dos valores calculados nas duas implementações. Outro aspecto importante a ser destacado é que a porcentagem de valores corretos aumenta na medida em que mais blocos são estimados.

Com estes resultados de validação mostrando uma porcentagem de 99,95% de vetores de movimento corretos sendo gerados foi possível concluir que a arquitetura opera corretamente e, portanto, implementa adequadamente em hardware o mesmo algoritmo para o padrão H.264/AVC.

A figura 7.2 apresenta as formas de onda obtidas na validação da arquitetura. O trecho apresentado refere-se a um macrobloco. Os sinais “sa” e “cb” referem-se às áreas de busca e aos blocos a serem buscados, respectivamente. Os sinais cujos nomes iniciam por “vetor” contêm os valores dos vetores de movimento. De forma semelhante, os sinais iniciados por “sad” contêm os valores dos SADs dos blocos candidatos. Os sinais “ok” indicam que há um vetor de movimento válido na saída da arquitetura. Na figura 7.2, os vetores de movimento referentes a um macrobloco totalizam os 41 valores discutidos anteriormente.

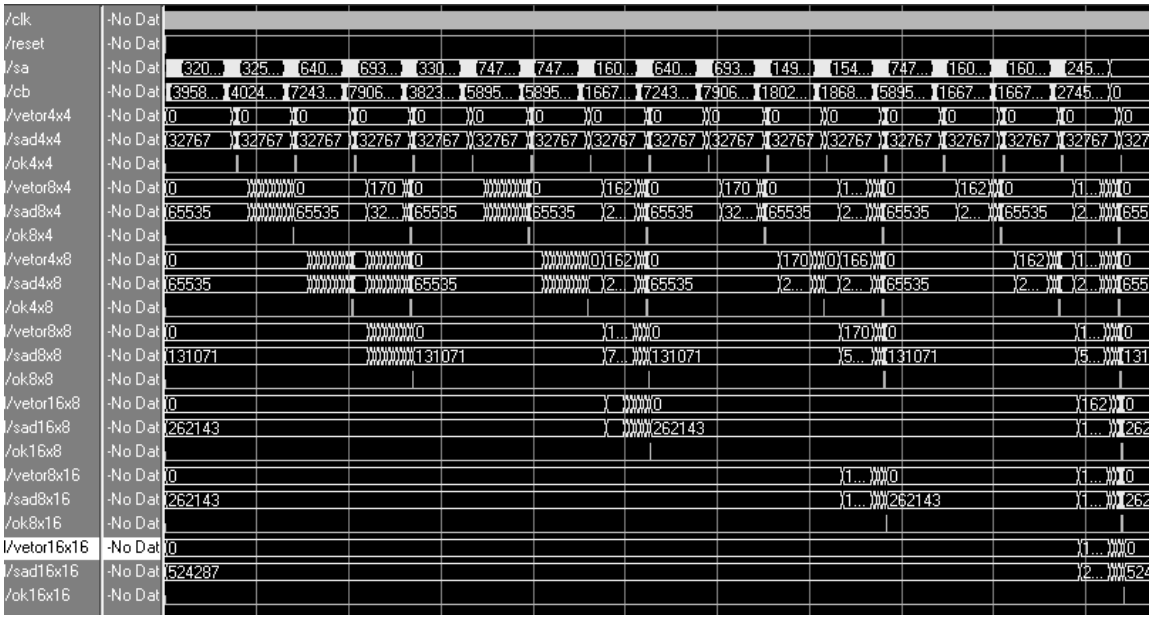


Figura 7.2: Formas de onda obtidas na validação da arquitetura.

Os resultados de síntese e de validação da arquitetura para estimação de movimento de blocos de tamanhos variáveis foram apresentados neste item do trabalho. O próximo capítulo apresentará comparações com vários trabalhos relacionados encontrados na literatura.

8 COMPARAÇÕES COM TRABALHOS RELACIONADOS

Este capítulo apresenta comparações com vários trabalhos relacionados encontrados na literatura. Os primeiros itens servem para prover uma breve descrição de cada trabalho, com suas características principais.

Todos são trabalhos atuais e utilizam FS como algoritmo de busca, SAD como critério de similaridade, H.264/AVC como foco, mas utilizam diferentes abordagens para realizar a estimação de movimento de blocos de tamanhos variáveis.

O último item deste capítulo discute os resultados comparativos relacionando vários parâmetros como: número de unidades de processamento, tamanho da área de busca, tecnologia utilizada, freqüência máxima alcançada, utilização de recursos e taxa de processamento. Os dados relativos às resoluções horizontal e vertical de vídeo que serão apresentadas neste capítulo podem ser encontrados na tabela 7.5 do capítulo anterior.

8.1 Trabalho de Bojnordi *et al.*

A arquitetura de Bojnordi (BOJNORDI, 2006) é uma arquitetura paralela multi-nível na qual a estimação de movimento é realizada através de três módulos principais: um módulo que calcula os SADs elementares (4x4 *pixels*), uma árvore para *merging* dos SADs elementares e um seletor de vetor de movimento.

Neste trabalho são utilizadas 128 unidades de processamento sobre uma área de busca de 23x16 *pixels*. A arquitetura proposta por Bojnordi foi implementada com a tecnologia CMOS Artisan UMC 0.18µm, ocupando uma área de 374 mil portas lógicas. A taxa de processamento alcançada foi de 56, 5 milhões de amostras por segundo, com a arquitetura operando em uma freqüência de 219 MHz. Com essa taxa de processamento, a arquitetura de Bojnordi é capaz de processar 27 quadros por segundo de uma seqüência de vídeos na resolução 1080HD. Dessa forma, o trabalho de Bojnordi apresenta o quinto melhor desempenho entre as arquiteturas apresentadas.

Bojnordi, no momento em que realiza comparações de sua arquitetura com trabalhos publicados anteriormente, referencia os autores Huang (HUANG, 2003), Ou (OU, 2005) e Yap (YAP, 2004).

8.2 Trabalho de Chen *et al.*

A arquitetura apresentada por Chen (CHEN, 2006) tem por base o deslocamento das linhas da área de busca para maximizar o reuso dos dados entre os blocos candidatos sucessivos. Uma árvore de SADs é aplicada posteriormente a esta etapa.

A arquitetura de Chen também foi implementada com a tecnologia CMOS Artisan UMC 0.18µm. A área ocupada foi de 330 mil portas lógicas, a freqüência alcançada foi de 108 MHz. Com esta freqüência a arquitetura é capaz de processar 30 quadros 720pHD por segundo, ou seja, 27,64 milhões de amostras por segundo. A arquitetura de Chen opera sobre uma área de busca de 128x64 *pixels*, utilizando 256 unidades de processamento.

Chen é citado em três dos trabalhos listados neste capítulo sendo eles Kim e Park (KIM, 2007), Li (LI, 2007) e Zhaoqing (ZHAOOQING, 2006). No trabalho de Chen, Huang (HUANG, 2003) este trabalho é referenciado.

8.3 Trabalho de Deng *et al.*

Deng (DENG, 2005) implementa uma arquitetura para VBSME através de uma matriz bidimensional de unidades de processamento seguidas de um esquema de *merging* e um módulo comparador.

Neste trabalho, o autor propõe o uso de 256 unidades de processamento e de uma área de busca de 65x65 *pixels*. A arquitetura de Deng é capaz de processar 12,44 milhões de amostras por segundo, ou seja, 30 quadros 625SD por segundo. TSMC 0.18µm foi a tecnologia CMOS utilizada na implementação da arquitetura. Os resultados de síntese obtiveram uma freqüência de 260 MHz e a utilização de 210 mil portas lógicas.

A arquitetura desenvolvida por Deng é referenciada no trabalho de Li (LI, 2007) e cita, nas comparações, dois trabalhos de Wei (WEI, 2004) (WEI, 2005) e os trabalhos de Huang (HUANG, 2003), Kim (KIM, 2005) e Yap (YAP, 2004).

8.4 Trabalho de Huang *et al.*

Huang (HUANG, 2003) possui o trabalho mais antigo dentre os trabalhos listados neste capítulo sendo, portanto, referenciado por vários deles.

Neste trabalho, uma matriz de elementos de processamento gera os 16 SADs de blocos de 4x4 *pixels* referentes a um macrobloco 16x16 *pixels*. Após, uma árvore com somadores em paralelo calcula os SADs restantes referentes aos tamanhos de bloco maiores.

A arquitetura de Huang é capaz de processar, por segundo, 30 quadros 525SD, ou 10,36 milhões de amostras por segundo. Para isso, utiliza 256 unidades de processamento e uma área de busca de 48x32 *pixels*. A implementação foi direcionada para a tecnologia CMOS TSMC 0.35µm. A arquitetura foi implementada com 106 mil portas lógicas e alcançou uma freqüência de 67 MHz.

8.5 Trabalho de Kim *et al.*

No trabalho de Kim (KIM, 2005), novamente uma matriz de unidades de processamento é aplicada. Com os resultados desta matriz, uma árvore de somadores gera os SADs dos blocos maiores e os entrega aos comparadores que gerarão os vetores de movimento.

Kim propõe uma arquitetura com 256 unidades de processamento e que utiliza uma área de busca de 64x64 *pixels*. Implementada com a tecnologia CMOS TSMC 0.18µm, a arquitetura alcançou a freqüência de 100 MHz, sendo capaz de processar 20 quadros no formato VGA por segundo atingindo cerca de 6,08 milhões de amostras por segundo. Os resultados mostram ainda que a arquitetura proposta por Kim ocupa uma área referente a 154 mil portas lógicas.

O trabalho de Kim é citado por Deng (DENG, 2005) e Li (LI, 2007) em seus trabalhos e referencia as arquiteturas desenvolvidas por Huang (HUANG, 2003), Wei (WEI, 2003) e Yap (YAP, 2004).

8.6 Trabalho de Kim e Park

Em seu trabalho, Kim e Park (KIM, 2007) utilizam uma matriz bidimensional com 16 unidades de processamento que recebem os dados em uma ordem diferente da convencional para maximizar o reuso de SADs e diminuir a complexidade dos cálculos.

Kim e Park apresentam, em seu trabalho, uma arquitetura capaz de processar seqüências de vídeo a uma taxa de 28 quadros do formato 720pHD por segundo ou 25,95 milhões de amostras por segundo. A arquitetura foi implementada com 39 mil portas lógicas em tecnologia CMOS DonbuAnam 0. 18µm e alcançou a freqüência de 416 MHz, a maior entre os trabalhos listados neste capítulo. Neste trabalho, foi utilizada uma área de busca de 16x16 *pixels* e 16 unidades de processamento.

Em seus resultados comparativos, Kim e Park mencionam as arquiteturas de Chen (CHEN, 2006), Wei (WEI, 2004) (WEI, 2005) e Yap (YAP, 2004).

8.7 Trabalho de Li *et al.*

A arquitetura apresentada por Li (Li, 2007) utiliza uma abordagem um pouco diferente mesmo utilizando matriz bidimensional de unidades de processamento e árvore de cálculo de SADs. Entre a matriz e a árvore de somadores há um mecanismo de rotação dos dados calculados para diminuir o número de acessos à memória. Após a árvore, há uma unidade que seleciona os vetores de movimento e os armazena juntamente com seus respectivos SADs.

A arquitetura proposta por Li ocupou 168 mil portas lógicas e foi implementada com a tecnologia CMOS Faraday 0.18µm. Os resultados de síntese mostram, ainda, que essa arquitetura pode processar 12,44 milhões de amostras por segundo o que equivale a 30 quadros 625SD por segundo. Para alcançar esta taxa, Li utilizou 256 unidades de processamento associadas a uma área de busca de tamanho 65x33 *pixels*. A freqüência alcançada foi de 216 MHz.

Por ser um dos trabalhos mais atuais, o trabalho de Li referencia, em seus resultados comparativos, os trabalhos de Huang (HUANG, 2003), Yap (YAP, 2004), Kim (KIM, 2005), Ou (OU, 2005), Chen (CHEN, 2006), Deng (DENG, 2005), e Wei (WEI, 2005).

8.8 Trabalho de Liu *et al.*

Liu (LIU, 2007) desenvolveu em seu trabalho uma arquitetura para estimação de movimento com 192 unidades de processamento, operando sobre uma área de busca de 192x128. A arquitetura proposta por Liu apresenta o terceiro melhor desempenho entre as arquiteturas apresentadas, alcançando uma taxa de processamento de 62,66 milhões de amostras por segundo. Esse desempenho é suficiente para processar vídeos 1080HD em uma taxa de 30 quadros por segundo. A arquitetura ocupou 486 mil portas lógicas e foi implementada com a tecnologia CMOS TSMC 0.18µm. A freqüência obtida neste trabalho foi de 200 MHz.

Na comparação com trabalhos relacionados Liu referencia Huang (HUANG, 2003) e Yap (YAP, 2004).

8.9 Trabalho de Ou *et al.*

O autor Ou (OU, 2005) implementa uma arquitetura com módulos que calculam todos os SADs de blocos 4x4 *pixels* em paralelo. Em seguida, outro módulo gera, concorrentemente, os 41 vetores de movimento relativos às partições do macrobloco com 16x16 *pixels*.

A arquitetura desenvolvida por Ou é a que apresenta a maior taxa de processamento dentre todas as listadas neste trabalho, alcançando 125,33 milhões de amostras por segundo. Assim sendo, a arquitetura de Ou pode processar 60 quadros 1080HD por segundo, indo além da taxa sugerida para a reprodução de vídeo HD em tempo real. A tecnologia CMOS UMC 0.18 µm foi utilizada na implementação, gerando um ASIC com 597 mil portas lógicas, a maior área entre todas as arquiteturas apresentadas. Os resultados de síntese apresentam ainda a freqüência obtida, que alcançou 123 MHz. Este trabalho utiliza 256 unidades de processamento e atua sobre uma área de busca de 16x16 *pixels*.

Com relação às comparações, o trabalho de Ou é referenciado em três trabalhos (BOJNORDI, 2006) (LI, 2007) (ZHAOQING, 2006) e referencia Yap (YAP, 2004).

8.10 Trabalho de Sayed *et al.*

A arquitetura proposta por Sayed (SAYED, 2006) utiliza uma matriz unidimensional de elementos de processamento alimentada por duas memórias, uma para a área de busca e outra contendo o bloco atual. Após esta etapa, outra matriz é utilizada, desta vez, com unidades de comparação para a geração dos vetores de movimento.

Em seu trabalho, Sayed apresenta uma arquitetura que utiliza 31 unidades de processamento e uma área de busca de 31x31 *pixels*. Os resultados de síntese, para a tecnologia CMOS TSMC 0.18 µm, indicam o uso de 68 mil portas lógicas e uma freqüência de 122 MHz. A taxa de processamento alcançada foi de 3,14 milhões de amostras por segundo. Com esta taxa de processamento, a arquitetura é capaz de processar 31 quadros CIF por segundo.

Neste trabalho, assim como em muitos outros, os trabalhos de Huang (HUANG, 2003) e de Yap (YAP, 2004) são mencionados.

8.11 Trabalho de Song *et al.*

Song (SONG, 2006) apresenta em seu trabalho uma arquitetura composta por elementos de processamento ligados ao que chamou de barramento de SADs. Como resultado do uso de um barramento, foram utilizados menos comparadores na etapa final da estimação.

A arquitetura proposta e apresentada por Song é capaz de processar 16,63 milhões de amostras por segundo utilizando 16 unidades de processamento e uma área de busca de 16x16 *pixels*. Nesta taxa, podem ser processados 40 quadros 625SD por segundo. Os resultados de síntese para a tecnologia CMOS TSMC 0.18 µm mostram o uso de 68 mil portas lógicas e uma freqüência em torno de 266 MHz.

Novamente as arquiteturas desenvolvidas apresentadas nos resultados comparativos são Huang (HUANG, 2003) e (YAP, 2004).

8.12 Trabalhos de Wei *et al.*

Wei apresenta três trabalhos sobre o tema, todos propondo arquiteturas para estimação de movimento e considerando blocos de tamanhos variáveis (WEI, 2003) (WEI, 2004) (WEI, 2005). A seguir as três arquiteturas serão apresentadas separadamente.

8.12.1 Trabalho de 2003

A arquitetura apresentada em (WEI, 2003) é composta por três partes, sendo elas: uma unidade que controla e seleciona as entradas, uma unidade para armazenamento, facilitando o cálculo de SADs para variados tamanhos de bloco e outra unidade para realizar o cálculo dos SADs.

Neste trabalho, Wei propõe uma arquitetura com 16 unidades de processamento operando sobre uma área de busca de 16x16 *pixels*. A síntese da arquitetura foi direcionada para um dispositivo da família Apex 20K da Altera (ALTERA, 2007). A freqüência obtida foi de 120 MHz. Para esse trabalho, a taxa de processamento foi de 24 quadros VGA por segundo ou o equivalente a aproximadamente 7,49 milhões de amostras por segundo.

Kim (KIM, 2005) faz referência a este trabalho de Wei em seu texto.

8.12.2 Trabalho de 2004

Neste trabalho, Wei (WEI, 2004) apresenta uma arquitetura composta por 16 unidades de processamento, 25 somadores e 25 comparadores. Dessa forma, a arquitetura de Wei calcula os SADs e gera os vetores de movimento de forma local, sem a clássica estrutura de níveis utilizada até agora.

A arquitetura desenvolvida por Wei neste trabalho foi implementada com a tecnologia CMOS TSMC 0.25 µm. A síntese resultou em 71 mil portas lógicas e uma freqüência de 150 MHz. A área de busca foi quadruplicada em relação ao trabalho anterior, ou seja, 32x32 *pixels*, para as mesmas 16 unidades de processamento. O desempenho apresentou-se pior do que o do trabalho anterior chegando a 30 quadros CIF por segundo, ou seja, 3,04 milhões de amostras por segundo.

Na comparação com trabalhos relacionados, novamente figuram, nas referências, as arquiteturas desenvolvidas por Huang (HUANG, 2003) e Yap (YAP, 2004). Dois trabalhos referenciam esse: Deng (DENG, 2005) e Kim e Park (KIM, 2007).

8.12.3 Trabalho de 2005

Em (WEI, 2005) são utilizadas uma matriz bidimensional e uma árvore de somadores para o cálculo dos SADs.

Novamente Wei propõe uma arquitetura implementada com *standard cells*. A freqüência alcançada foi de 52 MHz e 105 mil portas lógicas foram utilizadas. A tecnologia para a qual a síntese foi direcionada é a CMOS TSMC 0.25 µm. Neste terceiro trabalho, Wei apresenta uma arquitetura com o mesmo tamanho de área de busca do trabalho anterior (32x32 *pixels*), mas utiliza 256 unidades de processamento. Com uma taxa de processamento de 12,44 milhões de amostras por segundo a arquitetura proposta é capaz de processar 30 quadros 625SD por segundo.

Outra vez as referências a trabalhos relacionados dizem respeito a Huang (HUANG, 2003) e Yap (YAP, 2004). Por ser um trabalho mais interessante que seus anteriores, este é referenciado por Deng (DENG, 2005), Kim e Park (KIM, 2007), Li (LI, 2007) e Zhaoqing (ZHAOQING, 2006).

8.13 Trabalho de Yalcin *et al.*

Yalcin (YALCIN, 2005) utiliza em sua arquitetura 4 grupos de 9 elementos de processamento para o cálculo dos SADs elementares. Os resultados dessa etapa chegam a 5 grupos de somadores que geram os SADs para outros tamanhos de blocos. Após isso, comparadores são empregados para a geração dos vetores de movimento.

Em seu trabalho, Yalcin desenvolveu uma arquitetura com síntese direcionada para um FPGA da família Virtex 2 da Xilinx. A freqüência obtida foi de 68 MHz. Nesta freqüência, a arquitetura proposta é capaz de processar 27 quadros VGA por segundo, ou seja, em torno de 8,29 milhões de amostras por segundo. A arquitetura de Yalcin utiliza uma área de busca de 9x9 *pixels* e 36 unidades de processamento.

Quanto às citações, no trabalho de Yalcin novamente Huang (HUANG, 2003) é referenciado.

8.14 Trabalho de Yap e McCanny

A arquitetura desenvolvida por Yap e McCanny (YAP, 2004) é amplamente citada na literatura, sendo o trabalho mais freqüentemente referenciado nos trabalhos listados neste capítulo.

Yap e McCanny propõem uma arquitetura composta por uma matriz unidimensional de elementos de processamento que é ligada em comparadores através de vários barramentos.

A síntese da arquitetura desenvolvida por Yap e McCanny foi direcionada para a tecnologia CMOS TSMC 0.13 µm. Os resultados dessa síntese apontam uma freqüência de 294 MHz e o uso de 61 mil portas lógicas. A arquitetura de Yap e McCanny pode processar 19 quadros 720pHD por segundo. Isso é o mesmo que dizer que a taxa de processamento da arquitetura é de 18,24 milhões e amostras por segundo. Neste

trabalho, a área de busca é de 16x16 *pixels* de tamanho e são utilizadas 16 unidades de processamento.

8.15 Trabalho de Zhaoqing *et al.*

O trabalho de Zhaoqing (ZHAOQING, 2006) calcula os 16 SADs de blocos de 4x4 *pixels* referentes a um macrobloco através de uma matriz bidimensional de unidades de processamento e interconexões chamadas redes em cruz.

Zhaoqing apresenta em seu trabalho uma arquitetura que emprega 256 unidades de processamento operando sobre uma área de busca de 16x16 *pixels*. Com isso, ela é capaz de processar 55,29 milhões de amostras por segundo ou, em termos de resolução, 60 quadros 720pHD por segundo. A tecnologia CMOS com a qual a arquitetura foi implementada foi HJTC 0.18 μm resultando na utilização de 176 mil portas lógicas e em uma freqüência de 55 MHz.

O trabalho de Zhaoqing referencia Yap (YAP, 2004), Chen (CHEN, 2006), Ou (OU, 2005) e Wei (WEI, 2005) quando compara a sua arquitetura com outras arquiteturas para estimação de movimento.

8.16 Resultados Comparativos

Este item da dissertação destina-se a discutir resultados comparativos entre as arquiteturas listadas anteriormente e a arquitetura desenvolvida neste trabalho, em suas três diferentes tecnologias alvo.

Para isso, foi elaborada uma tabela geral relacionando todas as arquiteturas através de vários parâmetros tais como: número de unidades de processamento, tamanho da área de busca, tecnologia utilizada na implementação, freqüência máxima alcançada, utilização de área ou de portas lógicas, e taxa de processamento. Por resultar em uma tabela grande e complexa, optou-se por não colocá-la neste capítulo e sim apresentá-la no Apêndice A desta dissertação. Nesta seção do texto são apresentadas várias tabelas derivadas desta tabela geral. As tabelas apresentadas neste item são mais específicas e estabelecem resultados comparativos, destacando uso de hardware, taxa de processamento, e relações entre número de UPs e taxa de processamento e entre uso de hardware e taxa de processamento.

A primeira tabela apresentada é a tabela 8.1. Nesta tabela são apresentados os resultados comparativos destacando o uso de hardware. O número de unidades de processamento e o uso de área são apresentados separadamente. As arquiteturas desenvolvidas neste trabalho estão destacadas em cinza na tabela.

Nas tabelas 8.3 e 8.4, a seguir, esses dados serão relacionados com a taxa de processamento de cada arquitetura apresentada. Outros detalhes como o tamanho da área de busca e a tecnologia utilizada na implementação também são apresentados. Os dados sobre área que não são informados na tabela 8.1 referem-se a arquiteturas mapeadas para FPGAs.

É possível observar, na tabela 8.1, que a arquitetura desenvolvida neste trabalho é uma das que menos usa UPs. Outro aspecto importante a ser destacado é que a área de pesquisa utilizada nesta arquitetura é igual à área de busca da maioria das outras soluções. Os resultados de área omitidos na tabela 8.1 referem-se a soluções arquiteturais mapeadas para FPGAs.

Tabela 8.1: Resultados comparativos destacando o uso de hardware.

Arquitetura	Tamanho da Área de Busca	Tecnologia (FPGA ou ASIC)	Número de UPs	Área (KGates)
(SONG, 2006)	16x16	TSMC 0.18 µm	16	68
(WEI, 2004)	32x32	TSMC 0.25 µm	16	71
(WEI, 2003)	16x16	Altera Apex20K	16	-
(YAP, 2004)	16x16	TSMC 0.13 µm	16	61
(KIM, 2007)	16x16	DonbuAnam 0.18 µm	16	39
(SAYED, 2006)	31x31	TSMC 0.18 µm	31	68
(YALCIN, 2005)	9x9	Xilinx Virtex2	36	-
Porto em TSMC 0.18µm	16x16	TSMC 0.18 µm	52	120
Porto em Virtex2P	16x16	Xilinx Virtex2P	52	-
Porto em Virtex4	16x16	Xilinx Virtex4	52	-
(BOJNORDI, 2006)	23x16	Artisan UMC 0.18 µm	128	374
(LIU, 2007)	192x128	TSMC 0.18 µm	192	486
(KIM, 2005)	64x64	TSMC 0.18 µm	256	154
(ZHAOQING, 2006)	16x16	HJTC 0.18 µm	256	176
(HUANG, 2003)	48x32	TSMC 0.35 µm	256	106
(DENG, 2005)	65x65	TSMC 0.18 µm	256	210
(LI, 2007)	65x33	Faraday 0.18 µm	256	168
(WEI, 2005)	32x32	TSMC 0.25 µm	256	105
(CHEN, 2006)	128x64	Artisan UMC 0.18 µm	256	330
(OU, 2005)	16x16	UMC 0.18 µm	256	597

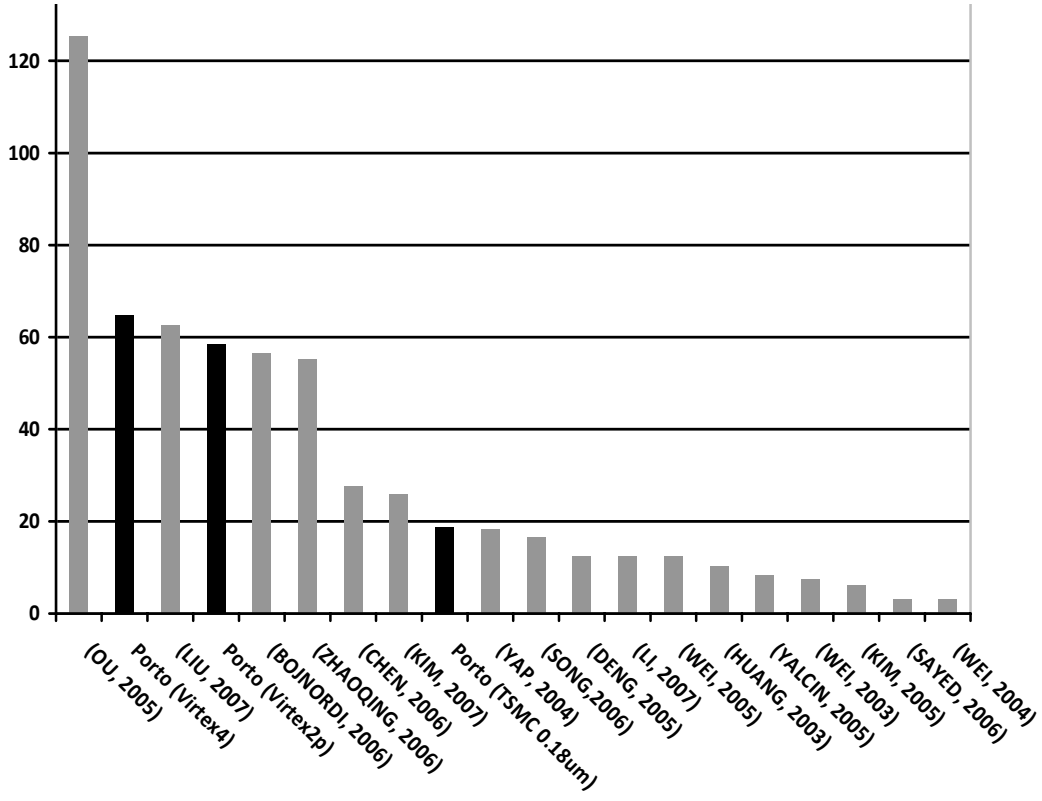
A próxima tabela a ser apresentada é a tabela 8.2. Nesta tabela, os resultados comparativos relacionados às taxas de processamento são apresentados. Novamente os resultados das arquiteturas desenvolvidas neste trabalho estão destacados em cinza na tabela. Da tabela 8.2 é possível observar que a solução desenvolvida neste trabalho apresenta desempenhos suficientes para processar HDTV 1080 quando mapeada para FPGAs de tecnologias recentes, como o Virtex 4 da Xilinx. Outro aspecto a ser observado é que apenas outras quatro soluções da literatura atingem uma taxa de processamento suficiente para suportar HDTV 1080 em taxas de quadro de 25 ou mais quadros por segundo.

Outra conclusão sobre os resultados da tabela 8.2 é de que a versão implementada com o FPGA da família Virtex4 apresenta a segunda maior taxa de processamento dentre todas as soluções investigadas. Já a versão implementada com o dispositivo da família Virtex2P ocupa o quarto lugar em número de quadros por segundo, outro bom resultado. A versão *standard cell* da arquitetura ocupa o nono lugar na tabela. A síntese deste ASIC não teve sua rede de *clock* otimizada. Sem uma rede de *clock* otimizada dificilmente esta versão da arquitetura obterá resultados melhores. Mesmo assim, esta versão da arquitetura é capaz de processar 21 quadros HDTV 720p por segundo. O destaque da tabela 8.2 é a elevada taxa de processamento alcançada pela arquitetura desenvolvida por Ou (OU, 2005). Para alcançar essa taxa, a arquitetura de Ou utiliza quase cinco vezes mais unidades de processamento do que a arquitetura desenvolvida neste trabalho. Ou seja, Ou (OU, 2005) utiliza o paralelismo de hardware de forma mais agressiva. Para que se tenha uma visão mais clara das taxas de processamento apresentadas na tabela 8.2, a figura 8.1 reapresenta estes dados na forma de um gráfico.

Tabela 8.2: Resultados comparativos destacando taxa de processamento.

Arquitetura	Tamanho da Área de Busca	Tecnologia (FPGA ou ASIC)	Taxa de Process. (Mamostras/s)	Número de Quadros por Segundo
(OU, 2005)	16x16	UMC 0.18 µm	125,33	60 (1080 HD)
Porto em Virtex4	16x16	Xilinx Virtex4	64,75	31 (1080 HD)
(LIU, 2007)	192x128	TSMC 0.18 µm	62,66	30 (1080 HD)
Porto em Virtex2P	16x16	Xilinx Virtex2P	58,49	28 (1080 HD)
(BOJNORDI, 2006)	23x16	Artisan UMC 0.18 µm	56,50	27 (1080 HD)
(ZHAOQING, 2006)	16x16	HJTC 0.18 µm	55,29	26 (1080 HD)
(CHEN, 2006)	128x64	Artisan UMC 0.18 µm	27,64	30 (720p HD)
(KIM, 2007)	16x16	DonbuAnam 0.18 µm	25,95	28 (720p HD)
Porto em TSMC 0.18µm	16x16	TSMC 0.18 µm	18,80	21 (720p HD)
(YAP, 2004)	16x16	TSMC 0.13 µm	18,24	19 (720p HD)
(SONG, 2006)	16x16	TSMC 0.18 µm	16,63	40 (625SD)
(DENG, 2005)	65x65	TSMC 0.18 µm	12,44	30 (625SD)
(LI, 2007)	65x33	Faraday 0.18 µm	12,44	30 (625SD)
(WEI, 2005)	32x32	TSMC 0.25 µm	12,44	30 (625SD)
(HUANG, 2003)	48x32	TSMC 0.35 µm	10,36	30 (525SD)
(YALCIN, 2005)	9x9	Xilinx Virtex2	8,29	27 (VGA)
(WEI, 2003)	16x16	Altera Apex20K	7,49	24 (VGA)
(KIM, 2005)	64x64	TSMC 0.18 µm	6,08	20 (VGA)
(SAYED, 2006)	31x31	TSMC 0.18 µm	3,14	31 (CIF)
(WEI, 2004)	32x32	TSMC 0.25 µm	3,04	30 (CIF)

Figura 8.1: Taxas de processamento das diversas arquiteturas (em Mamostras/s).



Mais resultados comparativos são apresentados na tabela 8.3, desta vez, relacionando número de unidades de processamento com taxa de processamento. Mais uma vez os resultados das arquiteturas desenvolvidas neste trabalho estão destacados em cinza na tabela. Para melhor avaliar essa relação foi criado um fator que é a simples divisão do número de unidades de processamento pela taxa de processamento. Quanto menor o resultado, mais eficiente é a arquitetura. Dessa forma, é possível fazer uma avaliação mais justa sobre o desempenho dos trabalhos.

As versões em FPGA da arquitetura desenvolvida neste trabalho estão entre as quatro melhores na relação entre número de unidades e taxa de processamento, sendo a versão utilizando Virtex4 a segunda melhor e a versão Virtex2P, a quarta.

A solução desenvolvida por Kim (KIM, 2007) é a melhor em uso de unidades de processamento por Mamostras/s gerada. Essa solução, mesmo tendo a melhor relação UPs/Mamostras/s, tem uma taxa de processamento inferior à obtida pelo trabalho atual com a arquitetura mapeada para Virtex4. A diferença da taxa de processamento da arquitetura atual mapeada em Virtex4 para a taxa de processamento da arquitetura de Kim possivelmente seja menor. Como a família de dispositivos Virtex4 foi desenvolvida em tecnologia de 60 nm a comparação com uma arquitetura desenvolvida em tecnologia 0.18 μm não é muito justa. A intenção de se utilizar um dispositivo Virtex4 na etapa de síntese foi a de obter resultados para um FPGA atual.

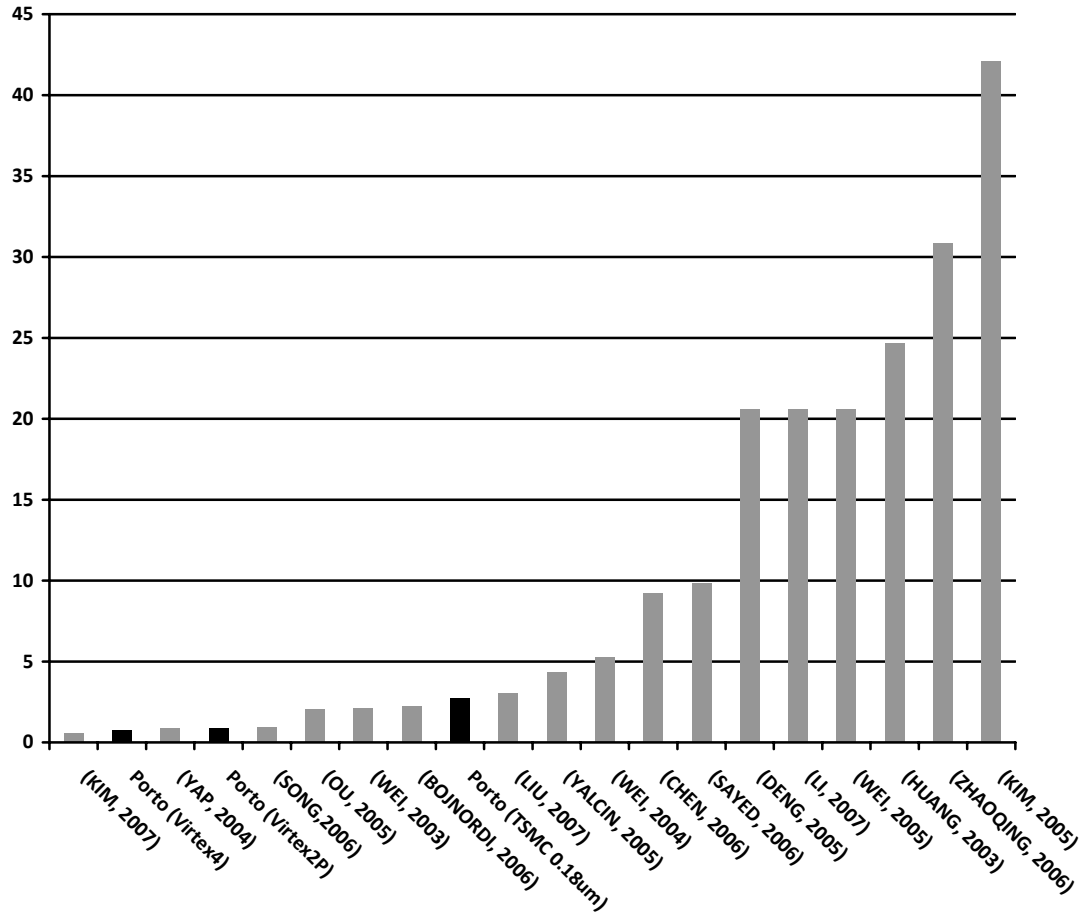
Da tabela 8.3 também é possível notar que as arquiteturas de melhor desempenho utilizam uma área de busca de 16x16 *pixels* de tamanho.

Tabela 8.3: Resultados comparativos relacionando número de UPs / taxa de processamento.

Arquitetura	Número de UPs	Tamanho da Área de Busca	Área (KGates)	Taxa de Process. (Mamostras/s)	Número de UPs / Taxa de Process.
(KIM, 2007)	16	16x16	39	25,95	0,61
Porto em Virtex4	52	16x16	-	64,75	0,80
(YAP, 2004)	16	16x16	61	18,24	0,87
Porto em Virtex2P	52	16x16	-	58,49	0,88
(SONG, 2006)	16	16x16	103	16,63	0,96
(OU, 2005)	256	16x16	597	125,33	2,04
(WEI, 2003)	16	16x16	-	7,49	2,13
(BOJNORDI, 2006)	128	23x16	374	56,5	2,26
Porto em TSMC 0.18μm	52	16x16	120	18,80	2,76
(LIU, 2007)	192	192x128	486	62,66	3,06
(YALCIN, 2005)	36	9x9	-	8,29	4,34
(WEI, 2004)	16	32x32	71	3,04	5,26
(CHEN, 2006)	256	128x64	330	27,64	9,26
(SAYED, 2006)	31	31x31	-	3,14	9,87
(DENG, 2005)	256	65x65	210	12,44	20,57
(LI, 2007)	256	65x33	168	12,44	20,57
(WEI, 2005)	256	32x32	105	12,44	20,57
(HUANG, 2003)	256	48x32	106	10,36	24,71
(ZHAOQING, 2006)	256	16x16	176	8,29	30,88
(KIM, 2005)	256	64x64	154	6,08	42,10

A figura 8.2 resume graficamente os dados da tabela 8.3, apresentando os fatores UPs/taxa de processamento alcançados pelas arquiteturas apresentadas. As três versões da arquitetura desenvolvida neste trabalho estão destacadas em preto na figura 8.2. O trabalho de Ou (OU, 2005) não alcança uma relação tão boa e, neste gráfico, fica na sexta posição.

Figura 8.2: Fator Número de UPs / Taxa de Processamento apresentado pelas diversas arquiteturas.



A tabela 8.4 apresenta resultados comparativos relacionando uso de hardware com a taxa de processamento para as diversas arquiteturas apresentadas.

Novamente o fator utilizado como comparação é uma divisão simples. Dessa vez, o fator é calculado através da divisão entre a área utilizada e a taxa de processamento. Cinco trabalhos, incluindo duas versões do trabalho atual, ficam de fora desta comparação por terem sido implementados em FPGA.

Novamente a arquitetura de Kim (KIM, 2007) aparece no topo da tabela, apresentando a melhor relação entre consumo de portas lógicas e taxa de processamento.

A versão *standard cell* da arquitetura desenvolvida neste trabalho ocupa a quinta posição. Entre a arquitetura de Kim e a versão ASIC do trabalho atual estão Yap (YAP, 2004), Ou (OU, 2005) e Song (SONG, 2006), nesta ordem.

Tabela 8.4: Resultados comparativos relacionando área e taxa de processamento.

Arquitetura	Número de UPs	Tamanho da Área de Busca	Área (KGates)	Taxa de Process. (Mamostras/s)	Área / Taxa de Process.
(KIM, 2007)	16	16x16	39	25,95	1,50
(YAP, 2004)	16	16x16	61	18,24	3,34
(OU, 2005)	256	16x16	597	125,33	4,76
(SONG, 2006)	16	16x16	103	16,63	6,19
Porto em TSMC 0.18µm	52	16x16	120	18,80	6,38
(BOJNORDI, 2006)	128	23x16	374	56,5	6,62
(LIU, 2007)	192	192x128	486	62,66	7,75
(WEI, 2005)	256	32x32	105	12,44	8,44
(HUANG, 2003)	256	48x32	106	10,36	10,23
(CHEN, 2006)	256	128x64	330	27,64	11,93
(LI, 2007)	256	65x33	168	12,44	13,50
(DENG, 2005)	256	65x65	210	12,44	16,88
(ZHAOQING, 2006)	256	16x16	176	8,29	21,23
(WEI, 2004)	16	32x32	71	3,04	23,35
(KIM, 2005)	256	64x64	154	6,08	25,32
Porto em Virtex4	52	16x16	-	64,75	-
Porto em Virtex2P	52	16x16	-	58,49	-
(WEI, 2003)	16	16x16	-	7,49	-
(YALCIN, 2005)	36	9x9	-	8,29	-
(SAYED, 2006)	31	31x31	-	3,14	-

Finalizando as comparações pode-se dizer que a solução desenvolvida neste trabalho, quando mapeada para FPGAs, apresenta resultados muito bons, atingindo a segunda maior taxa de processamento e a segunda melhor relação entre uso de UPs e taxa de processamento. As versões para FPGAs alcançam desempenho suficiente para serem utilizadas em aplicações HDTV.

Quanto à versão *standard cells*, esta não apresentou bom desempenho por não ter sido suficientemente otimizada. Esta versão foi gerada com a motivação principal de viabilizar uma comparação com outros trabalhos da literatura que foram desenvolvidos em *standard cells*. Melhorias nesta versão estão planejadas como trabalhos futuros. Mesmo assim, a versão *standard cells* da arquitetura pode ser utilizada em aplicações SDTV.

Outra conclusão interessante é que as arquiteturas de melhor desempenho utilizam uma área de busca de 16x16 *pixels* de tamanho, que é a mesma área utilizada neste trabalho.

9 CONCLUSÕES

Este trabalho apresentou o desenvolvimento de uma arquitetura para estimação de movimento de blocos de tamanhos variáveis segundo o padrão H.264/AVC de compressão de vídeo digital. Esta arquitetura é capaz de gerar os 41 diferentes vetores de movimento referentes a um macrobloco definidos pelo padrão.

Os primeiros capítulos desta dissertação apresentaram conceitos relativos ao padrão H.264/AVC de compressão de vídeo, assim como a estimação de movimento baseada em blocos. As avaliações do uso de sub-amostragem e do tamanho da área de busca, que direcionaram a etapa de desenvolvimento arquitetural, também foram apresentadas.

Após, foram apresentadas a arquitetura desenvolvida neste trabalho e seus módulos principais. A solução arquitetural proposta neste trabalho foi desenvolvida em VHDL e mapeada para FPGAs da Xilinx. Após, a arquitetura foi validada com dados referentes a seqüências de vídeo reais. Também foi desenvolvida uma versão *standard cell* da arquitetura. Os resultados obtidos foram satisfatórios tanto para a síntese quanto para a validação.

Considerando-se as versões da arquitetura com síntese direcionada para FPGA, os resultados mostraram que a arquitetura pode ser utilizada em aplicações com resoluções variadas, suportando inclusive SDTV e HDTV. Para a versão utilizando um dispositivo da família Virtex4, por exemplo, a taxa de processamento alcançou 64,75 milhões de amostras por segundo, ou seja, algo em torno de 31 quadros 1080HD por segundo. Essa taxa de processamento é a segunda maior dentre todas as soluções investigadas na literatura, perdendo apenas para uma arquitetura que utiliza cinco vezes mais unidades de processamento. Além desta versão da arquitetura, apenas outras quatro soluções apresentadas na literatura atingem uma taxa de processamento suficiente para suportar HDTV 1080 em tempo real. As versões em FPGA da arquitetura desenvolvida neste trabalho estão entre as quatro melhores na relação entre número de unidades e taxa de processamento, sendo a que a versão utilizando Virtex4 foi a segunda melhor e a versão usando Virtex2P foi a quarta melhor. Nesta comparação, a melhor solução em uso de unidades de processamento por Mamostras/s gerada apresentou uma taxa de processamento inferior à obtida pelo trabalho atual com a arquitetura mapeada para Virtex4.

Para a versão *standard cells* da arquitetura os resultados mostraram que a arquitetura pode ser utilizada para aplicações SDTV chegando a 18,8 milhões de amostras por segundo, o que é equivalente a 56 quadros 525 SD por segundo. Na relação entre consumo de portas lógicas e taxa de processamento a versão *standard cells* da

arquitetura desenvolvida neste trabalho ocupa a quinta posição. Esta versão da arquitetura foi a última etapa de desenvolvimento deste trabalho e foi gerada com a motivação principal de viabilizar uma comparação com outros trabalhos da literatura que foram desenvolvidos em *standard cells*. Mas ainda é necessário refinar os resultados gerados, pois a freqüência de operação atingida foi menor do que a esperada. Esta tarefa adicional está planejada como trabalho futuro.

Outra conclusão interessante é que as arquiteturas de melhor desempenho utilizam uma área de busca de 16x16 *pixels* de tamanho, que é a mesma área utilizada neste trabalho. Assim, a avaliação realizada neste trabalho alcançou seu objetivo.

Quanto aos resultados de validação, estes mostraram que a arquitetura alcançou uma porcentagem de 99,95% de vetores de movimento iguais aos gerados pela implementação em software também desenvolvida no trabalho. Os outros vetores foram diferentes em função da forma como o arredondamento foi realizado em software em hardware. Este índice e a verificação da discrepância mínima indicam que a arquitetura está correta.

Como trabalhos futuros pretende-se realizar a integração da arquitetura desenvolvida neste trabalho com os demais módulos do codificador H.264/AVC que foram desenvolvidos em trabalhos paralelos no grupo de pesquisa em TV Digital da UFRGS.

Outro trabalho importante a ser desenvolvido é o projeto de uma arquitetura que realize estimação de movimento com precisão de um quarto de *pixel* (*quarter pixel*). Essa arquitetura seria integrada à arquitetura desenvolvida neste trabalho agregando mais uma característica especificada pelo padrão H.264/AVC para a estimação de movimento.

A avaliação detalhada das diferenças geradas na validação é outro trabalho futuro a ser realizado. Dessa forma será possível quantificar estas diferenças e saber qual o seu efeito nos resultados.

Ainda como trabalho futuro pretende-se realizar o refinamento da versão *standard cells* em busca de melhores resultados. Como já foi dito antes, esta versão da arquitetura foi gerada com a motivação principal de viabilizar uma comparação com outros trabalhos da literatura que foram desenvolvidos em *standard cells*, e a otimização da síntese pode melhorar o desempenho do circuito.

REFERÊNCIAS

AGOSTINI, L. **Desenvolvimento de Arquiteturas de Alta Performance Dedicadas à Compressão de Vídeo Segundo o Padrão H.264.** 2007. 173 f. Tese (Doutorado em Ciência da Computação) - Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

ALTERA. **APEX 20K Devices:** System-on-a-Programmable-Chip Solutions. Disponível em: <http://www.altera.com/products/devices/apex/apx-index.html>. Acesso em: out. 2007.

ASHENDEN, P. **The Student's Guide to VHDL**. San Francisco: Morgan Kaufmann, 1998.

ATES, H.; ALTUNBASAK, Y. SAD Reuse in Hierarchical Motion Estimation for the H.264 Encoder. In: IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, ICASSP, 2005. **Proceedings…** Piscataway, NJ: IEEE, 2005. v. 2, p. 905-908.

BLOODSHED SOFTWARE. **DEV C++**: Providing Free Software to the Internet Community. Disponível em: <http://www.bloodshed.net/devcpp.html>. Acesso em: abr. 2007.

BOJNORDI, M. et al. Efficient Hardware Implementation for H.264/AVC Motion Estimation. In: IEEE ASIA PACIFIC CONFERENCE ON CIRCUITS AND SYSTEMS, APCCAS, 2006. **Proceedings…** [S.l.]: IEEE, 2006. p. 1749-1752.

CHEN, C.-Y. et al. Analysis and Architecture Design of Variable Block-Size Motion Estimation for H.264/AVC. **IEEE Transactions on Circuits and Systems**, [S.l.], v. 53, p. 578-593, Feb. 2006.

CHO, C.-Y. et al. An Embedded Merging Scheme for VLSI Implementation of H.264/AVC Motion Estimation Modules. In: IEEE INTERNATIONAL CONFERENCE ON IMAGE PROCESSING, ICIP, 2005. **Proceedings…** [S.l.]: IEEE, 2005. v.3, p. 1016-1019.

DE VOS, L.; SCHÖBINGER, M. VLSI Architecture for a Flexible Block Matching Processor. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 5, p. 417-428, Oct. 1995.

DENG, L. et al. An Efficient Hardware Implementation for Motion Estimation of AVC Standard. **IEEE Transactions on Consumer Electronics**, [S.l.], v. 51, p. 1360-1366, Nov. 2005.

HUANG, Y. et al. Hardware Architecture Design for Variable Block Size Motion Estimation in MPEG-4 AVC/JVT/ITU-T H.264. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2003. **Proceedings…** [S.l.]: IEEE, 2003. v. 2, p. 796-799.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE Xplore:** Dynamic Home Page. Disponível em: < http://ieeexplore.ieee.org>. Acesso em: mar. 2008.

INTERNATIONAL TELECOMMUNICATION UNION. **ITU-T Recommendation H.264 (05/03):** advanced video coding for generic audiovisual services. [S.l.], 2003.

INTERNATIONAL TELECOMMUNICATION UNION. **ITU-T Recommendation H.264/AVC (03/05):** advanced video coding for generic audiovisual services. [S.l.], 2005.

JUNG, H. et al. A VLSI Architecture for the Alternative Subsampling-Based Block Matching Algorithm. **IEEE Transactions on Consumer Electronics**, [S.l.], v. 41, p. 239-247, May 1995.

KERNIGHAN, B.; RITCHIE, D. **C:** a Linguagem de Programação Padrão ANSI. Rio de Janeiro: Campus, 1999.

KIM, M. et al. A Fast VLSI Architecture for Full-Search Variable Block Size Motion Estimation in MPEG-4 AVC/H.264. In: ACM ASIA AND SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE, ASP-DAC, 2005. **Proceedings…** [S.l.]: ACM, 2005. v. 1, p. 631-634.

KIM, J.; PARK, T. A Novel VLSI Architecture for Full-Search Variable Block-Size Motion Estimation. In: IEEE REGION 10 CONFERENCE, TENCON, 2007. **Proceedings…** [S.l.]: IEEE, 2007. p. 1-4.

KORAH, R. et al. Motion Estimation with Candidate Block and Pixel Subsampling Algorithm. In: IEEE INTERNATIONAL WORKSHOP ON IMAGING SYSTEMS AND TECHNIQUES, IST, 2005. **Proceedings…** [S.l.]: IEEE, 2005. p. 130-133.

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

LEE, K. et al. QME: An Efficient Subsampling-Based Block Matching Algorithm for Motion Estimation. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2004. **Proceedings…** [S.l.]: IEEE, 2004a. v. 2, p. 305-308.

LI, D-X. et al. Architecture Design for H.264/AVC Integer Motion Estimation with Minimum Memory Bandwidth. **IEEE Transactions on Consumer Electronics**, [S.l.], v. 53, p. 1053-1060, Aug. 2007.

LIU, Z. et al. 32-Parallel SAD Tree Hardwired Engine for Variable Block Size Motion Estimation in HDTV1080P Real-Time Encoding Application. In: IEEE WORKSHOP ON SIGNAL PROCESSING SYSTEMS, SiPS, 2007. **Proceedings…** [S.l.]: IEEE, 2007. p. 675-680.

MENTOR GRAPHICS. **The EDA Technology Leader**. Disponível em: <http://www.mentor.com>. Acesso em: set. 2007.

OU, C. et al. An Efficient VLSI Architecture for H.264 Variable Block Size Motion Estimation. **IEEE Transactions on Consumer Electronics**, [S.l.], v. 51, p. 1291-1299, Nov. 2005.

PORTO, M. **Arquiteturas de Alto Desempenho e Baixo Custo em Hardware para a Estimação de Movimento em Vídeos Digitais**. 2008. 100 f. Dissertação (Mestrado em Ciência da Computação) - Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

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

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

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

SAYED, M. et al. Towards an H.264/AVC Full Encoder on Chip: An Efficient Real-Time VBSME ASIC Chip. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2006. **Proceedings…** Los Alamitos, CA: IEEE, 2006. v. 1, p. 2613-2616.

SONG, Y. et al. VLSI Architecture for Variable Block Size Motion Estimation in H.264/AVC with Low Cost Memory Organization. In: IEEE INTERNATIONAL SYMPOSIUM ON VLSI DESIGN, AUTOMATION AND TEST, VLSI-DAT, 2006. **Proceedings…** [S.l.]: IEEE, 2006. p. 89-92.

SULLIVAN, G. et al. The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions. In: CONFERENCE ON APPLICATIONS OF DIGITAL IMAGE PROCESSING, SPIE, 2004. **Proceedings…** Denver: SPIE, 2004.

SULLIVAN, G.; WIEGAND, T. Video Compression – From Concepts to the H.264/AVC Standard. **Proceedings of the IEEE**, [S.l.], v. 93, n. 1, p. 18-31, Jan. 2005.

VQEG: The Video Quality Experts Group Web Site. Disponível em: <www.its.bldrdoc.gov/vqeg/>. Acesso em: nov. 2007.

WEI, C.; GANG, M. A Novel SAD Computing Hardware Architecture for Variable-Size Block Motion Estimation and Its Implementation with FPGA. In: IEEE INTERNATIONAL CONFERENCE ON ASIC, ASICON, 2003. **Proceedings…** [S.l.]: IEEE, 2003. v. 2, p. 950-953.

WEI, C. et al. VLSI Architecture Design for Variable-Size Block Motion Estimation in MPEG-4 AVC/H.264. In: IEEE ASIA-PACIFIC CONFERENCE ON CIRCUITS AND SYSTEMS, APCCS, 2004. **Proceedings…** [S.l.]: IEEE, 2004. v. 1, p. 617-620.

WEI, C.; GANG, M. A Novel VLSI Architecture for VBSME in MPEG-4 AVC/H.264. In: IEEE INTERNATIONAL SYMPOSIUM ON CIRCUITS AND SYSTEMS, ISCAS, 2005. **Proceedings…** [S.l.]: IEEE, 2005. v. 2, p.1794-1797.

WIEGAND, T. et al. Overview of the H.264/AVC Video Coding Standard. **IEEE Transactions on Circuits and Systems for Video Technology**, [S.l.], v. 13, n. 7, p. 560-576, July 2003.

XILINX. **Xilinx:** The Programmable Logic Company. Disponível em: <www.xilinx.com>. Acesso em: out. 2007.

YALCIN, S. et al. A High Performance Hardware Architecture for an SAD Reuse Based Hierarchical Motion Estimation Algorithm for H.264 Video Coding. In: IEEE INTERNATIONAL CONFERENCE ON FIELD PROGRAMMABLE LOGIC AND APPLICATIONS, FPL, 2005. **Proceedings…** [S.l.]: IEEE, 2005. p. 509-514.

YAP, S.; McCanny, J. A VLSI Architecture for Variable Block Size Video Motion Estimation. **IEEE Transactions on Circuits and Systems – II: Express Briefs**, [S.l.], v. 51, n. 7, p. 384-389, July 2004.

ZHAOQING, Z. et al. High Data Reuse VLSI Architecture for H.264 Motion Estimation. In: IEEE INTERNATIONAL CONFERENCE ON COMMUNICATION TECHNOLOGY, ICCT, 2006. **Proceedings…** [S.l.]: IEEE, 2006. p. 1-4.

APÊNDICE A

TABELA COMPLETA COM RESULTADOS COMPARATIVOS ENTRE AS ARQUITETURAS

Este apêndice destina-se a apresentar a tabela comparativa mencionada no capítulo 8. Os resultados da tabela A.1 estão ordenados pelos parâmetros: número de unidades de processamento; tamanho da área de busca; tecnologia utilizada na implementação; freqüência obtida; uso de hardware, medido em Kgates; e taxa de processamento, em número de amostras e número de quadros por segundo.

Tabela A.1: Resultados comparativos entre as arquiteturas para VBSME.

Arquitetura	Número de UPs	Tamanho da Área de Busca	Tecnologia (FPGA ou ASIC)	Freqüência (MHz)	Área (KGates)	Taxa de Processamento (Mamostras/s)	Número de Quadros por Segundo
(BOJNORDI, 2006)	128	23x16	Artisan UMC 0.18	219	374	56,50	1080 HD (1920x1080)@27fps
(CHEN, 2006)	256	128x64	Artisan UMC 0.18	108	330	27,64	720p HD (1280x720) @30fps
(DENG, 2005)	256	65x65	TSMC 0.18	260	210	12,44	625 SD (720x576) @30fps
(HUANG, 2003)	256	48x32	TSMC 0.35	67	106	10,36	525 SD (720x480) @30fps
(KIM, 2005)	256	64x64	TSMC 0.18	100	154	6,08	VGA (640x480) @20fps
(KIM, 2007)	16	16x16	DonbuAnam 0.18	416	39	25,95	720p HD (1280x720) @28fps
(LI, 2007)	256	65x33	Faraday 0.18	216	168	12,44	625 SD (720x576) @30fps
(LIU, 2007)	192	192x128	TSMC 0.18	200	486	62,66	1080 HD (1920x1080)@30fps
(OU, 2005)	256	16x16	TSMC 0.18	123	597	125,33	1080 HD (1920x1080) @60fps
(SAYED, 2006)	31	31x31	TSMC 0.18	122	68	3,14	CIF (352x288) @31fps
(SONG, 2006)	16	16x16	TSMC 0.18	266	68	16,63	625 SD (720x576) @ 40fps
(WEI, 2003)	16	16x16	Altera Apex20K	120	-	7,49	VGA (640x480)@24fps
(WEI, 2004)	16	32x32	TSMC 0.25	150	71	3,04	CIF (352x288) @30fps
(WEI, 2005)	256	32x32	TSMC 0.25	52	105	12,44	625 SD (720x576) @30fps
(YALCIN, 2005)	36	9x9	Xilinx Virtex2	68	-	8,29	VGA (640x480) @27fps
(YAP, 2004)	16	16x16	TSMC 0.13	294	61	18,24	720p HD (1280x720) @19fps
(ZHAOQING, 2006)	256	16x16	HJTC 0.18	55	176	55,29	720p HD (1280x720) @26fps
Porto em TSMC 0.18µ	52	16x16	TSMC 0.18	80	120	18,80	720p HD (1280x720) @21fps
Porto em Virte2P	52	16x16	Xilinx Virtex2P	242	-	58,49	1080 HD (1920x1080)@28fps
Porto em Virtex4	52	16x16	Xilinx Virtex4	266	-	64,75	1080 HD (1920x1080)@31fps