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

Uma Arquitetura de *Hardware* para Estimação de Movimento aplicada à Compressão de Vídeo Digital

por
DIOGO ZANDONAI

Dissertação submetida à avaliação
como requisito parcial para obtenção do grau de
Mestre em Ciência da Computação.

Prof. Dr. Sergio Bampi
Orientador

Porto Alegre, março de 2003.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Zandonai, Diogo

Uma Arquitetura de *Hardware* para Estimação de Movimento aplicada à Compressão de Vídeo Digital / por Diogo Zandonai. - Porto Alegre: PPGC da UFRGS, 2003.

104f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Ciência da Computação, Porto Alegre, BR-RS, 2003. Orientador: Bampi, Sergio.

1. Compressão de Vídeo. 2. Estimação de Movimento. 3. Arquitetura de *Hardware* para Estimação de Movimento. I. Bampi, Sergio. II. Título.

Agradecimentos

Ao colega Fábio Benevenutti que colaborou na captura de vídeo utilizada na validação da arquitetura auxiliada por *software*.

Ao líder de projetos do Genius Instituto de Tecnologia Marcel Bergerman, pela supervisão das atividades e revisão do texto.

Ao Professor Sergio Bampi, por ter apontado o rumo deste trabalho através de sua orientação.

Ao amigo e colega Luciano Volcan Agostini, que gerou o embrião deste trabalho.

À amiga e colaboradora Lisane Brisolara de Brisolara, que me auxiliou interagindo em meu nome junto ao Instituto de Informática enquanto estive ausente.

Aos colegas do Genius Instituto de Tecnologia por sua disposição em discutir muitos aspectos deste trabalho, incluindo André Trindade, Daniel Pelacani Petrini, Dieter Schwanke, José Eduardo Schwan Vianna, José Reginaldo Hughes Carvalho, Josias Oliveira, Marcello Modesto, Rodrigo Ribeiro e Silvio Mano Maeta.

Ao Professor Luigi Carro, pelas prontas respostas às minhas questões e suas recomendações.

Ao Professor Ivan Saraiva Silva, por suas sugestões com relação à validação da arquitetura.

A todos os colegas e professores do PPGC da UFRGS que de alguma maneira influenciaram este trabalho.

À UFRGS, representada pelo Instituto de Informática, por acolher-me como mestrando.

À CAPES pelo fomento à pesquisa manifestado através de uma bolsa de estudos durante o primeiro ano de curso.

Sumário

Lista de Abreviaturas

CIF	Common Interchange Format
CMY	Cyan, Magenta, Yellow
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
DVD	Digital Versatile/Video Disc
E/S	Entrada e Saída
EEPROM	Electrically Erasable Programmable Read Only Memory
EP	Elemento de Processamento
FPGA	Field Programmable Gate Array
GOP	Group Of Pictures
GOps	Giga Operations per second
HSB	Hue, Saturation, Brightness
I/O	Input and Output
ISO	International Organization for Standardization
JPEG	Joint Photographic Experts Group
Kbps	Kilo bits per second
KLT	Karhunen-Loeve Transform
MAE	Mean Absolute Error
Mbps	Mega bits per second
MFC	Microsoft Foundation Classes
MJPEG	*Motion* JPEG
MOp	Milhões de Operações
MOps	Milhões de Operações por segundo
MPEG	Moving Picture Experts Group
MSE	Mean Square Error
PC	Personal Computer
qps	quadros por segundo
RGB	Red, Green, Blue
RLE	Run Length Encoding
SAD	Sum of Absolute Differences
SVGA	Super Video Graphics Array
SXVGA	Super eXtended Video Graphics Array
USB	Universal Serial Bus
VGA	Video Graphics Array
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLC	Variable Length Coding
VLSI	Very Large Scale Integrated Circuits
XOR	eXclusive OR
YCbCr	luminance, Chrominance Blue, Chrominance Red
YIQ	luminance, Inphase, Qadrature

Lista de Figuras

Lista de Tabelas

Resumo

A tarefa de estimação de movimento, utilizada na compressão de vídeo digital, é normalmente realizada em *hardware* por processador dedicado, uma vez que demanda expressiva capacidade computacional. Este trabalho propõe e desenvolve uma arquitetura de *hardware* para realizar o cálculo dos vetores de movimento no contexto de compressão de vídeo digital.

Essa arquitetura para estimação de movimento é composta pelos blocos: interface de entrada e saída (E/S), matriz de processamento com 64 elementos de processamento, unidade de comparação e unidade de controle. A arquitetura foi descrita em linguagem VHDL de maneira que o número de *bits* utilizados para representação da luminância dos pontos é configurável. A partir desta descrição, foi gerado um protótipo para dados representados em 4 *bits* utilizando um *kit* de desenvolvimento baseado no dispositivo FPGA XC2S150 da Xilinx. Para validação do algoritmo e da arquitetura implementada, além da simulação, foi desenvolvido um *software* para plataforma PC capaz de exercitar as funcionalidades do protótipo. O PC é utilizado como dispositivo controlador de E/S para esta validação, na qual uma implementação do algoritmo em *software* e outra em linguagem de descrição de *hardware* são comparadas.

A máxima freqüência de trabalho do protótipo, estimada por simulação da arquitetura mapeada no FPGA XC2S150, é de 33 MHz. A esta freqüência o núcleo da arquitetura paralela de 64 elementos de processamento realiza cerca de 2,1 GOps (bilhões de operações inteiras por segundo). Esta arquitetura de *hardware* calcula os vetores de movimento para vídeo no formato 640x480 pontos à taxa de 107,32 quadros por segundo, ou um quadro a cada 9,3 ms. A arquitetura implementada para luminânica em 4 bits ocupa 16 pinos de E/S, 71,1% dos blocos lógicos do FPGA e 83,3% dos blocos de memória disponíveis no dispositivo XC2S150.

Palavras-chave: Compressão de Vídeo, Estimação de Movimento, Arquitetura de *Hardware* para Estimação de Movimento.

TITLE: “A HARDWARE ARCHITECTURE FOR MOTION ESTIMATION APPLIED TO DIGITAL VIDEO COMPRESSION”

Abstract

The motion estimation task, used in digital video compression, is usually performed through hardware using a dedicated processor, since it is computationally intensive. This work proposes and develops a hardware architecture to compute the motion vectors for digital video compression.

The proposed motion estimation architecture is composed of the following blocks: I/O interface, processing matrix with 64 processing elements, and comparison and control units. The architecture was described in the VHDL language in such way that the number of bits used to represent luminance data is configurable. A prototype was built using 4 bit data representation and a development kit based on the FPGA device XC2S150 from Xilinx. For algorithmic and architectural validation, besides simulation, software for the PC platform was developed to exercise the prototype functionalities. The PC was used as an I/O driver for this validation, in which the motion estimation implementation in software and in the hardware language description are compared.

The prototype maximum clock frequency, estimated by simulation of the architecture mapped for the FPGA device XC2S150, is 33 MHz. At this frequency, the kernel of the parallel architecture of 64 processing elements performs about 2.1 GOps (billons of integer operations per second). The hardware architecture computes the motion vectors for video in 640x480 pixel format at a rate of 107.32 frames per second, or one frame every 9.3 ms. The architecture implemented in 4 bits uses 16 I/O pins, 71.1% of the logic blocks and 83.3% of the memory blocks available in the XC2S150 device.

Keywords: Hardware Architecture for Motion Estimation, Motion Estimation, Video Compression.

1 Introdução

Este trabalho propõe e desenvolve uma arquitetura de *hardware* para realizar o cálculo dos vetores de movimento no contexto de compressão de vídeo digital. Este capítulo apresenta a motivação do trabalho, seus objetivos e metodologia e, por fim, a organização do texto.

1.1 Motivação

As aplicações de vídeo digital têm crescente aceitação e importância tanto no mercado de eletrônica embarcada quanto no mercado de informática. Da mesma forma, a comunicação audiovisual tem ampliado sua participação no mercado de comunicações.

A viabilidade técnico-econômica do desenvolvimento de aplicações de vídeo digital é resultado do progresso contínuo em três áreas: processos, padrões e redes [ACK 97]. O desenvolvimento de processos VLSI permite a integração de circuitos complexos de grande capacidade computacional para tratar dados de vídeo em tempo real. O desenvolvimento de padrões de compressão de vídeo com resolução e qualidade para competir com o vídeo analógico permite a interoperabilidade de dispositivos, aumentando as fronteiras do mercado de consumo. O desenvolvimento de redes com crescente velocidade de transferência de dados permite a utilização do vídeo como meio de comunicação. Estas tecnologias têm difundido largamente a utilização de vídeo digital, ampliando os limites de aplicação e substituindo cada vez mais as aplicações de vídeo analógico.

Aplicações de vídeo digital necessariamente envolvem a comunicação de grandes quantidades de dados, necessitando uma largura de banda da ordem de 3 a 10 milhões de *bits* por segundo (Mbps) para transferência de informações [ACK 97]. Considerando a capacidade de transferência de dados dos meios de comunicação atualmente existentes e a quantidade de dados presente na informação de vídeo, na prática é impossível a transmissão de vídeo em tempo real em seu formato original.

A informação de vídeo digital, quando representada através de uma seqüência de imagens sem compressão, gera uma taxa de dados alta. A Tabela 1.1 apresenta a taxa de dados necessária para representação de diversos formatos de vídeo digital sem compressão. Considerando um formato de imagem com qualidade comparável à televisão analógica, como o CIF, a uma taxa de 30 quadros por segundo e uma rede de velocidade 56,6 Kbps, percebe-se que a razão entre a velocidade da rede e a taxa de dados do vídeo original é de 1:1.290. Isto mostra que para viabilizar a transmissão em tempo real deste vídeo através desta rede seria necessária uma taxa compressão de pelo menos 1290 vezes [SHI 2000]; para o formato SVGA esta taxa seria de 10.004. O papel da compressão de vídeo é o de reduzir a taxa de dados do sinal de vídeo, viabilizando, em algumas situações, sua transmissão em tempo real. Atualmente, a transmissão em tempo real de vídeo em redes de 56,6 Kbps é realizada apenas para vídeos de baixa resolução e ainda assim com significativas perdas na qualidade do vídeo reproduzido.

TABELA 1.1 – Taxa de dados de diversos formatos de vídeo sem compressão.

Formato de Vídeo	Resolução a 30 qps	Taxa de Dados (Mbps)	Taxa de Compressão Necessária para Transmissão a 56,6 Kbps
CIF	352x288	72,99	1289,59
VGA	640x480	221,18	3907,84
SVGA	800x600	345,60	6106,01
SVGA	1024x768	566,23	10004,08
SXVGA	2048x1536	2264,92	40016,33

A compressão é uma operação importante, uma vez que o processamento de dados é mais barato que sua transmissão ou armazenamento. Para exemplificar a relação existente entre processamento e transmissão basta lembrar que, para um vídeo digital no formato CIF sem compressão, seriam necessários 1290 segundos para transmissão de um segundo de vídeo por uma rede na velocidade de 56,6 Kbps. Considerando a taxa de processamento atingida pela arquitetura desenvolvida neste trabalho, operando a uma freqüência de 33 MHz, este vídeo pode ser codificado em 0,27 segundos. Isto mostra que o tempo de processamento para compressão é desprezível diante do tempo de transmissão. Se a taxa de compressão fosse 1:10, a transmissão poderia ser realizada em cerca de 129 segundos; neste caso, o tempo de codificação representaria 0,02% deste tempo.

A maior parte dos conceitos aqui apresentados pode ser aplicada a qualquer padrão de compressão de vídeo, uma vez que os padrões atuais são baseados nas mesmas metodologias de compressão. Assim, embora os padrões de compressão possam ser comparados em termos de qualidade de imagem ou taxa de compressão; o mais importante é sua aceitação e, por isto, a denominação “padrão”. Embora existam diferentes padrões para compressão de vídeo, o MPEG é o padrão mais aceito e difundido, e tornou-se sinônimo de compressão de vídeo, tendo sido utilizado em aplicações como DVD e televisão digital.

A compressão de vídeo utiliza técnicas de processamento de vídeo e imagens conhecidas e consolidadas mesmo antes da existência de padrões. A Figura 1.1 apresenta um diagrama simplificado dos principais blocos utilizados na compressão de vídeo. O sinal de entrada passa inicialmente por um pré-processamento no qual os dados são preparados para a aplicação dos algoritmos de compressão. Nos dois blocos seguintes os algoritmos de compressão de imagens em movimento e imagens estáticas são processados. No último bloco o *bitstream* de saída é composto de acordo com algum padrão de compressão, por exemplo, o MPEG.

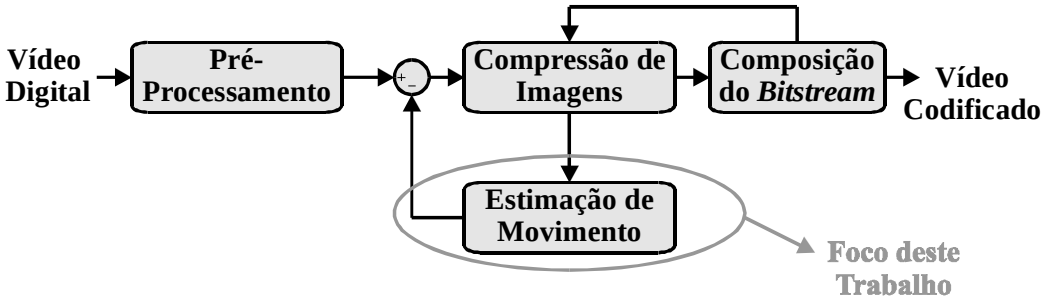

FIGURA 1.1 – Diagrama dos principais blocos para compressão de vídeo.

Dentre os algoritmos que implementam os blocos da Figura 1.1, o de estimação de movimento é o que exige maior capacidade computacional. A Tabela 1.2 apresenta o número operações, expressas em MOp (milhões de operações), necessárias para estimação de movimento para um quadro. Os valores apresentados nesta tabela foram calculados com base na arquitetura para estimação de movimento apresentada neste trabalho e nas arquiteturas para compressão de imagens apresentadas em [AGO 2002]. Analisando a Tabela 1.2 percebe-se que o número de operações é diretamente proporcional à área da imagem. Assim, para aplicações de vídeo de boa qualidade de imagem torna-se necessária a utilização de recursos de alta capacidade computacional.

TABELA 1.2 – Capacidade computacional necessária para as tarefas de compressão de vídeo, em Milhões de Operações.

Formato de Vídeo	Resolução	Estimação de Movimento	Compressão de Imagens
CIF	352x288	6,49	2,13
VGA	640x480	19,66	6,45
SVGA	800x600	30,72	10,08
SVGA	1024x768	50,33	16,52
SXVGA	2048x1536	201,33	66,06

Realizar a tarefa de estimação de movimento em um processador de uso geral, responsável pelo gerenciamento de um sistema, consome grande parte de seus recursos e torna o sistema lento. Por outro lado, a implementação dos demais algoritmos utilizados na compressão MPEG é perfeitamente viável neste tipo de processador. Assim, o desenvolvimento de uma arquitetura dedicada à estimação de movimento tem uma grande vantagem frente a processadores de uso geral, pois nesta arquitetura é explorado o fato de que as operações para estimação de movimento em geral envolvem o cálculo do módulo da diferença entre dois números, como será visto na Seção 3.2. Uma arquitetura dedicada também é capaz de garantir um bom desempenho para implementação do algoritmo para o qual foi projetada quando comparada com uma arquitetura de uso geral.

Este cenário estimulou o desenvolvimento deste trabalho, que consiste na concepção, projeto estrutural, descrição em linguagem VHDL e validação de uma arquitetura VLSI que implementa o cálculo dos vetores de movimentos necessários para

a estimação de movimento e em um *software*, executando em plataforma PC, que alimenta a arquitetura de dados e lê o valor dos vetores de movimento.

1.2 Objetivos e Metodologia

O objetivo principal deste trabalho é o projeto e validação de uma arquitetura VLSI dedicada ao cálculo dos vetores de movimento aplicados à compressão de vídeo digital. Outro objetivo do trabalho é a implementação de um *software* capaz de alimentar esta arquitetura de dados e ler os vetores de movimento resultantes. Esta arquitetura foi descrita em linguagem VHDL, e para validação funcional sua prototipagem foi realizada no dispositivo FPGA XC2S150 da Xilinx [XIL 2002].

O *software* implementado pode operar de modo dependente ou independente. No modo dependente sua função é auxiliar no teste da arquitetura desenvolvida, sendo responsável pela alimentação de dados e leitura dos vetores de movimento. No modo independente o *software* serve como parâmetro de comparação com a arquitetura para estimação de movimento, pois ambos implementam o mesmo algoritmo.

Todo projeto de *hardware* tem como objetivo encontrar o ponto ótimo para atender alguns requisitos conflitantes, tais como: aumentar a capacidade computacional e a freqüência de relógio e também reduzir a quantidade de pinos de E/S, a utilização de área e a potência.

Para encontrar este ponto ótimo é necessário definir algumas condições de contorno em função da aplicação alvo. Neste trabalho as condições de contorno representam os seguintes objetivos de projeto: aumentar a capacidade de processamento para calcular um vetor de movimento a cada ciclo de processamento e reduzir a quantidade de pinos de entrada e saída para realizar um único acesso a cada dado. A prototipagem em dispositivo FPGA implica em restrições na freqüência de relógio e na potência, porém, a principal limitação neste dispositivo é a disponibilidade de células lógicas, ou seja, área. Com relação a este aspecto o objetivo mais relevante buscado no projeto foi tornar o circuito mais compacto de maneira a acomodá-lo no dispositivo disponível.

A metodologia de trabalho é a seguinte:

- Estudo dos critérios para cálculo da distorção entre regiões e dos algoritmos para estimação de movimento e definição do algoritmo a ser implementado.
- Realização do projeto teórico da arquitetura que é inspirado nas arquiteturas apresentada em [BHA 99] e [ZAN 2001].
- Descrição das entidades fundamentais em linguagem VHDL utilizando como ambiente de desenvolvimento a ferramenta *WebPack* da Xilinx.
- Desenvolvimento do *software* para validação.
- Implementação da comunicação entre o *software* e a arquitetura para estimação de movimento.
- Testes funcionais e validação dos resultados utilizando a ferramenta de testes ModelSim da Mentor.

1.3 Organização do Texto

Este trabalho está estruturado da seguinte forma:

- No Capítulo 2 são apresentados os conceitos de compressão de vídeo, os tipos de redundância existentes na informação de vídeo, outras teorias de processamento de imagens aplicadas à compressão de vídeo e a técnica de compressão de vídeo considerando as características dinâmicas da imagem.
- No Capítulo 3 é discutida a estimação de movimento. São analisados criticamente diferentes algoritmos e implementações de *hardware* para estimação de movimento previamente apresentados na literatura.
- No Capítulo 4 é apresentada a arquitetura de *hardware* desenvolvida para estimação de movimento.
- No Capítulo 5 é apresentada a metodologia de validação da arquitetura desenvolvida e os respectivos testes e resultados.
- No Capítulo 6 são apresentadas as conclusões do trabalho e sugestões de trabalhos futuros.

Ao longo do texto, todas as palavras que representam nome de sinais, unidades operacionais ou de controle ou que fazem parte do código VHDL foram escritas utilizando a fonte `Courier New`. Para distinguir dos números decimais, os números binários de um *bit* são representados entre aspas simples (‘0’) enquanto os números binários de múltiplos *bits* são representados entre aspas duplas (“010101”).

2 Compressão de Vídeo Digital

Para contextualizar a aplicação da estimação de movimento são apresentados neste capítulo os fundamentos de compressão de vídeo. Inicialmente são apresentados os conceitos básicos de vídeo digital, desde a representação de um ponto até as características de um vídeo. Partindo da premissa que um vídeo é formado por uma seqüência de imagens, são apresentados os princípios de compressão de imagens estáticas e compressão de imagens em movimento. Maiores detalhes podem ser encontrados em [ELY 95], [GON 92], [MIT 2001] e [SHI 2000].

Para esclarecer os conceitos tratados neste trabalho são apresentadas neste capítulo as seguintes definições: ponto, imagem, vídeo, imagem natural, imagem sintética, cena de vídeo, pontos vizinhos, imagens vizinhas, espaços de representação de cores, varredura progressiva, varredura entrelaçada, redundância espacial, redundância temporal, redundância psicovisual, redundância entrópica, algoritmos de compressão interquadro, algoritmos de compressão intraquadro, compressão com perdas e compressão sem perdas.

2.1 Conceitos Básicos de Vídeo Digital

Um vídeo digital é formado por imagens, que por sua vez são formadas por pontos. Nesta dissertação as referências ao tamanho de imagem ou região da imagem são sempre expressas em pontos. A altura de uma imagem é definida pela quantidade de pontos na vertical enquanto a largura é a quantidade de pontos na horizontal. Uma cena de vídeo é formada por uma seqüência de imagens gravadas de maneira contínua. Pontos pertencentes à mesma imagem e espacialmente próximos são considerados pontos vizinhos. Analogamente, imagens pertencentes à mesma cena de vídeo e temporalmente próximas são consideradas imagens vizinhas. Pontos vizinhos e imagens vizinhas são geralmente muito similares, reflexo da grande redundância na representação digital de imagens e vídeo [GON 92].

Nesta dissertação todas as referências a vídeo, imagens ou pontos tratam de vídeo digital, imagens digitais ou pontos digitalmente representados. Vídeos naturais são gerados a partir de um sensor que capta imagens do ambiente. Um vídeo também pode ser gerado computacionalmente, o que caracteriza um vídeo sintético ou sintetizado. Em alguns casos, o vídeo sintético pode ser entendido como uma simulação computacional do vídeo natural.

Um vídeo é reproduzido pela exibição de imagens projetadas seqüencialmente na tela durante um processo denominado varredura. Em um vídeo a varredura da tela pode ser feita de duas formas diferentes: progressiva ou entrelaçada. Na varredura progressiva o canhão de elétrons percorre a tela linha após linha seqüencialmente, enquanto na varredura entrelaçada o canhão percorre as linhas pares e as linhas ímpares da tela de forma alternada. Cada varredura completa apresentará na tela uma imagem denominada quadro. Na varredura entrelaçada, cada conjunto de linhas pares ou ímpares pertencentes ao mesmo quadro constitui um campo, o campo par e o campo ímpar do quadro, respectivamente.

O potencial para compressão de imagens está no fato de que pontos vizinhos da mesma imagem são similares. O potencial para compressão de vídeo, por sua vez, está no fato de que imagens vizinhas da mesma cena são similares. Técnicas de compressão que tiram vantagem da similaridade existente entre pontos da mesma imagem são

chamadas de compressão espacial. Técnicas de compressão que tiram vantagem da similaridade existente entre imagens da mesma cena são chamadas de compressão temporal.

2.2 Redundância na Informação de Vídeo Digital

A compressão é fundamentada na redução de redundâncias existentes em vídeos e imagens. Esta seção apresenta as classes de redundância e as classes de algoritmos de compressão que exploram estas redundâncias.

Existem diferentes propostas de classificação das redundâncias na informação de vídeo, tais como as apresentadas em [SHI 2000], [MIT 2001] e [ELY 95]. A classificação apresentada por este trabalho é uma fusão destas propostas, com as redundâncias classificadas de maneira coerente com o escopo do trabalho, e quatro tipos básicos: espacial, temporal, entrópica e psicovisual.

A redundância espacial resulta da correlação existente entre um ponto e os pontos vizinhos da mesma imagem. Em uma imagem geralmente existe uma grande similaridade entre o valor que representa um ponto e o valor dos pontos vizinhos. Assim, o acréscimo de informação visual de cada ponto em relação aos pontos vizinhos é pequeno [MIT 2001].

A redundância temporal resulta da correlação existente entre imagens vizinhas em um vídeo. Em trechos de vídeo nos quais não há mudança de cena existe uma pequena variação no valor que representa um certo ponto quando comparado com o do ponto de mesma posição na imagem seguinte ou anterior, reflexo da grande similaridade entre estas imagens. Assim, o acréscimo de informação visual de cada imagem com relação às imagens vizinhas é pequeno [ELY 95].

A redundância entrópica está relacionada à forma de representação computacional da imagem e não ao seu conteúdo, como é o caso das redundâncias espacial e temporal. A entropia, isto é, a desordem dos símbolos que representam o vídeo, é uma medida da quantidade média de informação transmitida por símbolo que representa o vídeo [SHI 2000]. A quantidade de informação transmitida por um certo símbolo diminui com o aumento da probabilidade de ocorrência deste símbolo. Os padrões de compressão de vídeo prevêem a utilização de algoritmos para aumentar a quantidade de informação representada por símbolo de forma a representar mais informações utilizando menor quantidade de símbolos.

A redundância psicovisual está relacionada à forma como o sistema humano de visão interpreta as imagens. Existem aspectos visuais aos quais o sistema humano de visão não é muito sensível, como por exemplo à variação rápida da imagem e detalhes da textura da imagem, entre outros. A sensibilidade visual também é dependente da região da imagem; o cérebro interpreta bem apenas uma região central da imagem chamada “foco”, guardando informações vagas sobre as demais regiões. Assim, a redundância psicovisual resulta de informações representadas no vídeo mas que o sistema humano de visão não é capaz de interpretar ou às quais é pouco sensível [SHI 2000].

Os algoritmos empregados para compressão de vídeo fundamentam-se na eliminação de um ou mais tipos de redundância. Estes algoritmos podem ser classificados quanto à sua utilização em intraquadro ou interquadro. Os algoritmos intraquadro são aqueles aplicados à compressão de imagens estáticas que foram herdados pela compressão de vídeo, uma vez que um vídeo é uma seqüência de

imagens. Os algoritmos interquadro consideram a similaridade entre imagens vizinhas e têm por objetivo representar uma imagem em função de suas imagens vizinhas [MIT 2001].

Os algoritmos para eliminação de redundâncias também podem ser classificados, quanto à degradação que causam à imagem, em algoritmos de compressão com perdas e compressão sem perdas. Os algoritmos de compressão sem perdas permitem a exata reconstrução das informações codificadas, enquanto nos algoritmos com perdas isto não é possível. Os algoritmos com perdas exploram amplamente as redundâncias psicovisuais das cenas, enquanto os algoritmos sem perdas exploram as demais redundâncias. O incremento em termos de compressão que se obtém quando se admite uma pequena perda na qualidade do vídeo reproduzido é da ordem de dez vezes [MIT 2001].

A Tabela 2.1 apresenta a classificação dos algoritmos para eliminação de redundâncias. Em geral os algoritmos de compressão de vídeo eliminam mais de um tipo de redundância. A Tabela 2.1 apresenta, para cada classe de algoritmo, o principal tipo de redundância eliminada. Isto significa que pode existir algum algoritmo que elimine um tipo de redundância não apresentado na Tabela 2.1, porém pelo menos o tipo de redundância apresentado na Tabela 2.1 é eliminado.

TABELA 2.1 – Classificação dos algoritmos para eliminação de redundâncias.

Classe	Principal Redundância Eliminada
interquadro	temporal
intraquadro	espacial
com perdas	psicovisual
sem perdas	entrópica

2.3 Pré-Processamento

O bloco de pré-processamento da Figura 1.1 é composto, quando necessário, pelos blocos de conversão do espaço de cores e de sub-amostragem de cores conforme mostra a Figura 2.1. As subseções que seguem apresentam estes blocos.

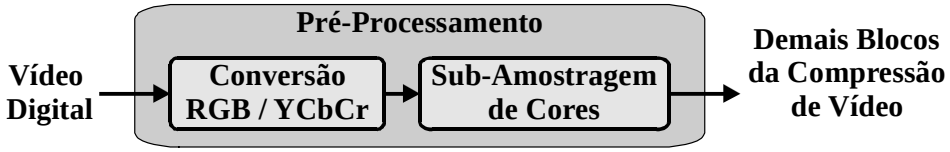

FIGURA 2.1 – Diagrama em blocos do pré-processamento.

2.3.1 Conversão do Espaço de Cores

A representação computacional de um ponto colorido está associada à interpretação humana das cores. O sistema humano de visão possui elementos sensíveis à luz chamados bastonetes e cones. Os bastonetes são sensíveis à intensidade luminosa,

enquanto os cones são sensíveis às cores primárias. Existe um tipo de cone sensível ao vermelho, um ao verde e outro ao azul [BIO 2002]. Uma vez que o sistema humano de visão é sensível às três cores primárias, a representação computacional de um ponto de uma imagem colorida é feita a partir de três componentes.

O tipo de componente que representa um ponto define o espaço de representação de cores deste ponto. Existem diversos espaços de representação de cores, tais como RGB, CMY, YCbCr, YIQ e HSB [GON 92]. Os espaços de representação de cores relevantes para este trabalho são o RGB, por ser o formato nativo dos sensores de imagem e cinescópios, e o YCbCr, por ser o formato utilizado para transmissão de vídeo. No formato RGB as três componentes utilizadas são vermelho, verde e azul. No formato YCbCr as três componentes utilizadas são luminância, crominância azul e crominância vermelha. Existe uma relação direta entre os formatos de cor RGB e YCbCr dada pela Equação 2.1. A definição do formato RGB em função do YCbCr é obtida invertendo-se a matriz quadrada desta equação.

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ -0{,}169 & -0{,}331 & 0{,}500 \\ 0{,}500 & -0{,}419 & 0{,}081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{2.1}$$

2.3.2 Sub-Amostragem de Cores

O sistema humano de visão possui cerca de 240 milhões de bastonetes e 13 milhões de cones. Isto fundamenta a observação, feita através de testes psicológicos, de sua sensibilidade maior à informação de luminância em comparação com a informação de crominância [BIO 2002]. Este é um aspecto psicovisual explorado pelos padrões de compressão de vídeo, que admitem uma redução da taxa de amostragem das cores com relação à da luminância, ou seja, a sub-amostragem de cores.

Para possibilitar a sub-amostragem de cores é necessário que esta informação esteja desvinculada da informação de luminância; para isto o vídeo deve ser representado no formato YCbCr. Esta representação é conveniente pois é o formato utilizado em televisão, porém, em se tratando de vídeo digital, é necessário realizar uma conversão do espaço de cores que é feita conforme definido na Seção 2.3.1.

O padrão MPEG, por exemplo, admite três formatos de cores. Suas denominações não têm relação com a taxa de amostragem mas sim com motivos históricos, e são: o 4:2:0, o 4:2:2 e o 4:4:4. A Figura 2.2 apresenta a relação entre a taxa de amostragem destes três formatos de cor. No formato 4:4:4 não é realizada sub-amostragem, no formato 4:2:2 a taxa de amostragem vertical das cores é de metade da taxa da luminância, e no formato 4:2:0 a taxa de amostragem vertical e horizontal é de metade da taxa da luminância.

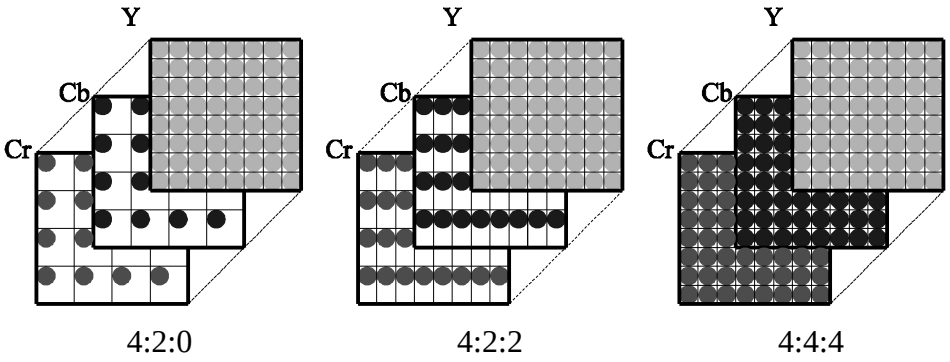

FIGURA 2.2 – Esquemas da sub-amostragem de cores.

2.4 Compressão de Imagens Estáticas

Uma vez que um vídeo é formado por uma seqüência de imagens, esta seção apresenta os princípios de compressão de imagens estáticas que são importantes para a contextualização deste trabalho. Os princípios de compressão de imagens tratados nesta seção são: transformada discreta do cosseno (DCT), quantização e codificação de entropia.

A Figura 2.3 apresenta em maior nível de detalhes o bloco de compressão de imagens da Figura 1.1. A entrada deste diagrama é uma imagem em representação de cores YCbCr. Após a aplicação da DCT, os dados são quantizados. Por fim, os coeficientes da DCT quantizados passam pelo bloco de codificação de entropia para compor a imagem codificada.

FIGURA 2.3 – Diagrama em blocos da compressão de imagens.

2.4.1 DCT – Transformada Discreta do Cosseno

Uma transformação na maneira de representar uma imagem tem a finalidade de aumentar a correlação entre os dados que representam a imagem, explicitando assim as redundâncias e possibilitando a compressão nos blocos após a transformação. Para exemplificar o que uma transformação faz com os dados que representam uma imagem, considere uma transformação simples aplicada à uma região da imagem na Figura 2.4. Esta transformação consiste em tomar os pontos da imagem em pares ordenados (x,y) e redefinir o sistema de coordenadas de maneira a explicitar a correlação entre os pontos, concentrando a informação em uma das componentes do par ordenado.

FIGURA 2.4 – Exemplo de aplicação de transformação a uma região da imagem.

À esquerda na Figura 2.5 é apresentado um gráfico, construído utilizando a ferramenta Matlab [MAT 2002], no qual os pontos x do par (x,y) são as abscissas e os pontos y do par (x,y) são as ordenadas. Percebe-se que a maior parte dos pontos concentra-se na reta em que x é igual a y, pois devido à redundância espacial da imagem, pontos espacialmente próximos têm valores muitos similares. À direita na Figura 2.5 é apresenta uma transformação no sistema de coordenadas que consiste em uma rotação de 45^{o} nos eixos de representação seguida de uma translação. Assim, após a transformação proposta, os pontos são representados pelo para ordenado (x',y') no qual o valor de x' é sempre muito próximo de zero.

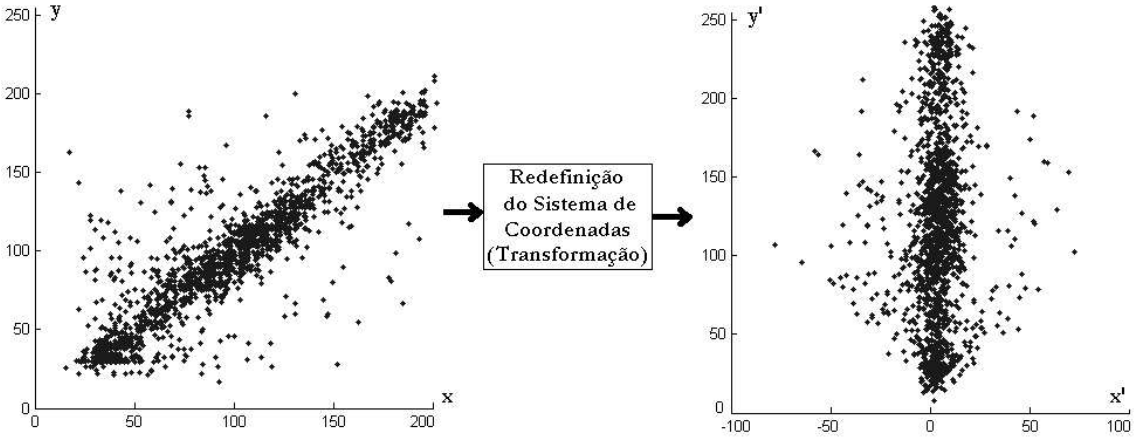

FIGURA 2.5 – Aplicação de uma transformação aos pontos da Figura 2.4.

A transformação apresentada como exemplo mostra que é possível encontrar um sistema de coordenadas no qual os pontos da imagem sejam representados por valores mais próximos de zero quando comparados com a representação original. Nesta representação alternativa (transformada) existe a possibilidade de reduzir o número de *bits* necessários para representar os pontos da imagem. A DCT representa uma rotação no eixo de representação dos pontos, porém seu espaço de representação não é no espaço bi-dimensional (x,y), mas sim um espaço de oito componentes.

A DCT é uma transformada ortogonal e reversível na qual a função original é representada através de uma composição de funções cosseno. A cada ponto da matriz de entrada corresponde um coeficiente de saída da DCT. Existem oito definições

matemáticas da DCT [OPP 99] decorrentes de diferentes hipóteses feitas na definição da mesma. Este trabalho apresenta apenas a definição indicada adotada pelo padrão MPEG [INT 94].

A entrada do bloco que implementa a DCT é uma matriz que representa uma região de 8x8 pontos da imagem denominada macrobloco. Este tamanho de macrobloco é proposto pelo padrão MPEG e foi assim escolhido por oferecer algumas facilidades computacionais. Um dos requisitos é que o tamanho do macrobloco seja múltiplo de dois para viabilizar a implementação de algoritmos baseados em divisões sucessivas. A região não pode ser muito grande pois a taxa de operações para o cálculo da DCT cresce exponencialmente com o seu tamanho, nem muito pequeno a ponto de não conter informação. Assim, o tamanho 8x8 foi adotado pelo MPEG por ser a melhor relação entre a taxa de operações e a taxa de compressão.

As Equações (2.2) e (2.3) apresentam as definições da DCT direta e inversa, respectivamente [OPP 99].

$$F(u,v) = \frac{2}{N} C(u)C(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2y+1)v\pi}{2N} \tag{2.2}$$

$$f(x,y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v)F(u,v) \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2y+1)v\pi}{2N} \tag{2.3}$$

onde:

- x, y são as coordenadas do ponto no domínio espacial das amostras.
- u, v são as coordenadas do ponto no domínio das freqüências.
- u, v, x, y $\in$ [0, 1, 2, ... N-1].
- $C(u), C(v) = \begin{cases} \frac{1}{\sqrt{2}} & para \quad u,v = 0 \\ 1 & nos\ demais\ casos \end{cases}$

Variações bruscas de cor e luminosidade em uma imagem são representadas por componentes de alta freqüência na sua transformada. Por outro lado, áreas de preenchimento com cor e luminosidade praticamente constantes na imagem são representadas por componentes de baixas freqüências na sua transformada. Em geral, imagens naturais são basicamente compostas por baixas freqüências pois os objetos têm bordas pouco definidas e seu preenchimento representa a maior parte da área da imagem. A DCT explicita a concentração da informação nas baixas freqüências maximizando a quantidade de coeficientes nulos após a quantização e aumentando a taxa de compressão na codificação de entropia, como será mostrado nas seções seguintes.

Muitas outras transformadas são utilizadas em processamento de imagens como é o caso da transformada discreta de Fourier (a DFT), da transformada de Haar, da transformada de Hadamard, da transformada de Hartley e da transformada de Karhunen-Loeve (KLT) [HOF 2001]. Merece destaque a KLT por ser a transformada que define o espaço de representação que possibilita a máxima taxa de compressão, porém, para regiões relativamente pequenas, a DCT representa uma boa aproximação da KLT. A DCT foi adotada pelos padrões de compressão de vídeo e imagens pois sua implementação computacional é mais simples que as demais transformadas, uma vez

que tem como saída apenas números reais. Isso não ocorre com a DFT, por exemplo, que utiliza uma representação através de números complexos. Além disso, existem inúmeros algoritmos rápidos para o cálculo da DCT.

2.4.2 Quantização

Conforme apresentado na Figura 2.5 a DCT representa a informação da imagem, que é composta principalmente por baixas freqüências, em torno do valor nulo. É desejado que valores muito pequenos sejam aproximados pelo valor nulo, o que maximizaria a taxa de compressão quando aplicada a codificação de entropia [MIT 2001].

A quantização é uma operação irreversível, na qual a precisão dos coeficientes da DCT é reduzida. Na quantização os coeficientes da DCT são divididos por um valor inteiro positivo e o resultado é arredondado para o inteiro mais próximo. Nesta operação, os coeficientes muito pequenos são aproximados pelo valor nulo e a quantidade de *bits* necessária para representar os demais coeficientes é reduzida.

As matrizes de quantização utilizadas para compressão são compostas por coeficientes relativamente pequenos para as componentes de alta freqüência, que carregam pouca informação, e por coeficientes maiores para as componentes de baixa freqüência, que carregam a maior parte da informação [AGO 2002]. A matriz de quantização é um grau de liberdade do codificador; este é um assunto bem estudado e existem propostas de matrizes de quantização adequadas para diferentes tipos de cena.

2.4.3 Codificação de Entropia

Após a transformação dos dados pela aplicação da DCT e sua quantização, o último bloco na compressão de imagens é a codificação de entropia. A transformação e quantização da imagem não implicam na redução da quantidade de dados; seu papel é preparar os dados explicitando as redundâncias existentes de forma a diminuir a diferença de amplitude entre os dados, aumentar o número de dados de igual valor, e aumentar a quantidade de dados com valor nulo. A seguir, na codificação de entropia, são utilizados diferentes tipos de codificação visando reduzir ou eliminar as redundâncias explicitadas pelos blocos anteriores.

A função do bloco de codificação de entropia é escolher os melhores símbolos, ou códigos, para a representação dos dados da imagem. Este bloco realiza os seguintes tipos de codificação:

- Codificação diferencial: codifica apenas a diferença entre os dados.
- VLC: a codificação de comprimento variável (VLC do inglês *variable length coding*) descarta os *bits* que não são significativos no dado a ser codificado.
- RLE: a codificação por número de ocorrências (RLE do inglês *run length coding*) considera o fato de muitos dados consecutivos são de igual valor e agrupa-os indicando apenas o número de ocorrências.
- Codificação de Huffman: utiliza dados estatísticos para escolher símbolos compactos para representar os dados que ocorrem com maior freqüência.

2.5 Estimação de Movimento

Imagens vizinhas pertencentes à mesma cena de vídeo são muito similares, o que resulta em grande redundância temporal. As diferenças entre imagens vizinhas, quando

existem, correspondem à movimentação dos elementos da cena, à movimentação da câmera ou ainda à introdução ou saída de elementos da cena. A Figura 2.6 ilustra a diferença tomada ponto a ponto entre duas imagens deslocadas para baixo e para a direita por diferentes número de pontos. Percebe-se que as bordas dos elementos da imagem representam as maiores diferenças.

FIGURA 2.6 – Diferença entre imagens com deslocamento de: (a) um ponto, (b) dois pontos, (c) quatro pontos.

Aqueles padrões de compressão vídeo que prevêem a redução da redundância temporal adotam a técnica de estimação de movimento para tal [SHI 2000]. A estimação de movimento consiste em identificar a relação entre imagens vizinhas e mapear esta relação através de vetores denominados vetores de movimento. A informação de movimento, ou seja, os vetores de movimento, são codificados no *bitstream* que representa o vídeo compactado. Na decodificação do vídeo compactado ocorre um processo denominado compensação de movimento, que consiste em reconstituir o vídeo original utilizando a informação de movimento representada pelos vetores de movimento. A compensação de movimento exige uma pequena capacidade de processamento quando comparada com a estimação de movimento [BHA 99].

Uma vez que a estimação de movimento é o tema central deste trabalho, esta seção apresenta apenas a forma de utilização da estimação de movimento na implementação de um sistema de compressão de vídeo. A metodologia utilizada para cálculo dos vetores de movimento e os demais detalhes internos ao bloco de estimação de movimento são apresentados no Capítulo 3.

Para contextualizar esta seção a Figura 2.7 [QUE 2001] apresenta um diagrama em blocos completo da compressão de vídeo, incluindo a estimação de movimento. Nesta figura o bloco de estimação de movimento tem como entrada uma amostra do sinal após a aplicação da DCT e da quantização. Antes de iniciar a estimação de movimento o sinal de entrada deste bloco passa pela DCT inversa e pela quantização inversa, de modo a restaurar o sinal YCbCr original, exceto pelo erro de quantização. O resultado destas operações inversas é armazenado na memória de quadro.

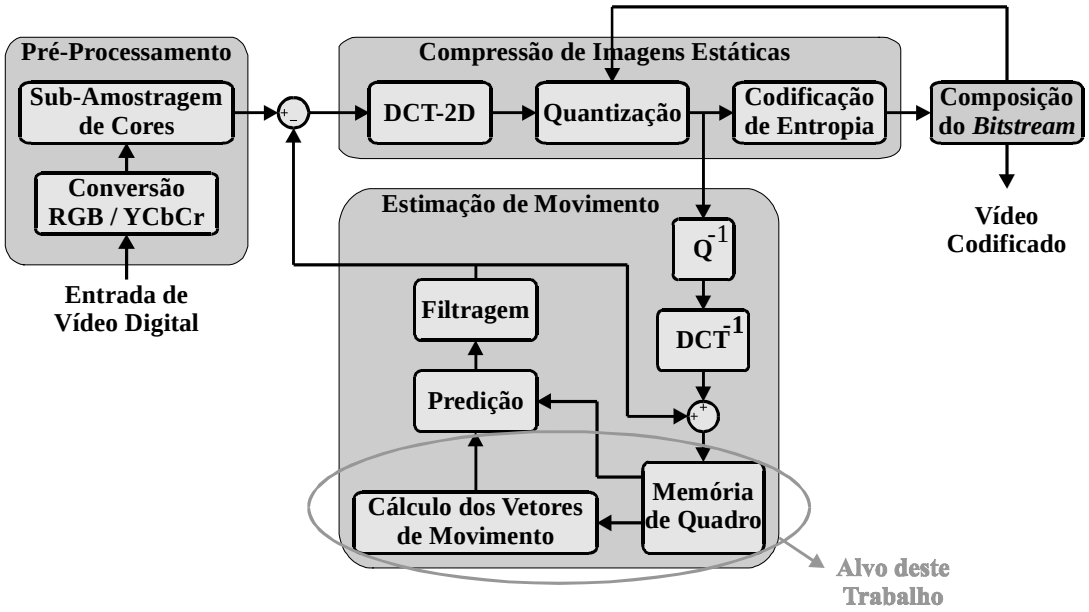

FIGURA 2.7 – Diagrama de blocos da compressão de vídeo.

Os dados armazenados na memória de quadro alimentam o bloco de cálculo dos vetores de movimento, onde é realizada a estimação de movimento propriamente dita. Este é o principal bloco da estimação de movimento, sendo o responsável por comparar as diferentes hipóteses de vetor de movimento e decidir por aquela que melhor representa o movimento da cena. Este é o alvo deste trabalho. A saída do bloco de cálculo dos vetores de movimento alimenta o bloco de predição, que é responsável por calcular o erro entre a representação utilizando vetores de movimento e o sinal original. A saída do bloco de predição passa por uma filtragem para eliminação de ruído. A seguir o sinal é subtraído do sinal original antes de passar pela compressão de imagens e seguir para a composição do *bitstream* de saída.

2.6 Padrões de Compressão de Vídeo

Após ter a taxa de dados reduzida através das técnicas de compressão de imagens estáticas e da estimação de movimento, e antes de disponibilizar o vídeo compactado na saída, é necessário compor o *bitstream* de saída de acordo com um padrão de compressão de vídeo. Este é o papel do bloco composição do *bitstream* na Figura 1.1 e na Figura 2.7. Esta seção discute brevemente alguns padrões de compressão de vídeo.

O MPEG (*Moving Picture Expert Group*) é um grupo de trabalho da ISO que define os padrões para compressão de informação audiovisual digital. O MPEG vem acompanhando e delineando a evolução da tecnologia desenvolvendo para isto novos padrões adequados à taxa de dados e qualidade exigida pelas novas aplicações de vídeo digital. Os padrões definidos pelo MPEG que são considerados estáveis são o MPEG-1 e o MPEG-2, enquanto os padrões MPEG-4, MPEG-7 e MPEG-21 encontram-se em fase de desenvolvimento [SAV 2002].

O padrão MPEG-1 foi desenvolvido para suportar o tráfego de dados a taxas da ordem de 1,5 Mbps. O MPEG-1 tem sido utilizado em muitos produtos comerciais que não dependem de compatibilidade com sistemas de televisão.

O padrão MPEG-2 foi desenvolvido para dar suporte ao vídeo entrelaçado utilizado em televisão e prevê o tráfego de dados a taxas entre 2 e 30 Mbps. O MPEG-2 está viabilizando a migração da televisão analógica para digital, permitindo a utilização de tecnologias como transmissão de áudio e vídeo multicanal e compressão escalonada [INT 94].

Os padrões MPEG-4, MPEG-7 e MPEG-21 consideram a utilização de vídeo sintético simultaneamente com o vídeo genérico (ou natural), e definem o conceito de objetos de áudio e objetos de vídeo. Estes padrões representam uma mudança conceitual com relação à metodologia de compressão utilizada no MPEG-1 e MPEG-2, definindo uma linguagem para descrição de conteúdo audiovisual na qual o vídeo é gerado de forma compacta, permitindo integrar a produção e distribuição de informação audiovisual e tornando a busca da informação mais rápida e eficiente. Estes padrões podem ser utilizados em televisão digital, aplicações gráficas interativas, aplicações multi-usuário e multimídia sobre internet, entre outras [MOV 2000].

Os padrões MPEG-1 e MPEG-2 são largamente utilizados em produtos comerciais, enquanto os demais padrões MPEG estão em fase de consolidação e são voltados a aplicações multimídia. Além dos padrões consolidados e aqueles que estão em fase de evolução deve ser considerada a existência de outros padrões menos difundidos, embora bastante utilizados em aplicações específicas, como é o caso dos padrões MJPEG, H.120, H.261 e H.263 [FUJ 97].

O sistema computacional que utilizar a arquitetura desenvolvida por este trabalho, entre outras tarefas, será responsável pela composição do *bitstream* de saída de acordo com algum padrão de compressão de vídeo. Esta arquitetura pode ser utilizada por sistemas computacionais para gerar vídeo codificado em diferentes padrões, como o MPEG-1, MPEG-2 e os demais padrões citados acima, exceto os que trabalham com o conceito de objetos de áudio e de vídeo. A seção seguinte apresenta o padrão MPEG-2, por ser o padrão mais amplo em termos de aplicabilidade dentre os citados, por ser compatível com o MPEG-1 e por ser um padrão com crescente utilização e aceitabilidade. Assim, doravante, o MPEG-2 será referenciado simplesmente por MPEG.

2.7 O Padrão de Compressão de Vídeo MPEG-2

O padrão MPEG-2 é definido na norma denominada "*Generic Coding of Moving Pictures and Associated Audio*", que é composta por dez volumes [INT 94]. O primeiro volume da norma, denominado *Systems*, define a composição do *bitstream* de dados de saída. O segundo e terceiro volumes da norma, denominados *Video* e *Audio* respectivamente, definem os parâmetros de compressão destas mídias. Os demais volumes têm papel secundário fornecendo suporte a teste, simulação, tele-difusão, armazenamento, transmissão e tempo real, entre outros recursos. Esta seção analisará o segundo volume, que trata sobre a compressão de vídeo.

A primeira parte da norma (*Systems*) pode ser entendida como uma "camada" que envolve as "camadas" de compressão de vídeo e de áudio. Nesta parte da norma são definidas duas formas de empacotamento de dados denominadas *Program Stream* e *Transport Stream*, concebidas para tráfego de dados em meios confiáveis e não confiáveis, respectivamente. A primeira foi desenvolvida para comunicação e armazenamento de dados multimídia em dispositivos locais, enquanto a segunda é útil para suportar transmissão eficiente sobre linhas sujeitas a erros físicos.

Visando formatos compatíveis e escalonáveis o MPEG define perfis (*profiles*) e níveis (*levels*). Um perfil é um sub-conjunto completo de regras de sintaxe para composição do *bitstream*. Um nível é um conjunto de restrições impostas aos parâmetros do *bitstream*.

O *bitstream* é a forma de comunicação entre o codificador e o decodificador a partir do qual é possível montar o vídeo original sem necessidade de conhecer a técnica utilizada pelo codificador para gerar o *bitstream*. A norma MPEG define o decodificador e o formato do *bitstream*, enquanto o codificador e os algoritmos de compressão não são definidos, apenas sugeridos. Para suportar isto é necessário que muitas informações relativas à maneira como os dados foram codificados, como por exemplo o tamanho de bloco utilizado para estimação de movimento, estejam codificadas dentro do *bitstream* de saída.

No MPEG-2, que suporta o formato de vídeo entrelaçado utilizado em televisão, as imagens podem ser codificadas como quadro ou como campo, ou seja, os algoritmos podem ser aplicados à estrutura de quadros ou campos. Os dados oriundos dos dois campos podem ser combinados mantendo-se o entrelaçamento, e os algoritmos são aplicados ao quadro, ou pode-se agrupar os dados referentes a cada campo desfazendo o entrelaçamento e aplicando os algoritmos ao campo, conforme é apresentado na Figura 2.8.

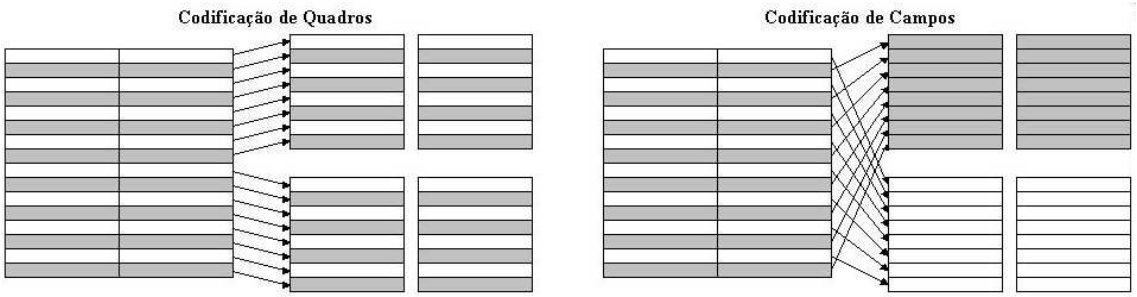

FIGURA 2.8 – Aplicação dos algoritmos de compressão a quadros e a campos.

De acordo com o tipo de redundância eliminada em cada quadro o MPEG considera a existência de três tipos diferentes de quadros, que são apresentados a seguir.

Quadro tipo I (*Intraframe*): este tipo de quadro é codificado de forma independente de outros quadros. A compressão de quadros do tipo I é fundamentada nos mesmos algoritmos que o padrão JPEG utiliza para compressão de imagens estáticas. A taxa de compressão atingida neste tipo de quadro é significativamente menor que a taxa obtida nos demais tipos.

Quadro tipo P (*Predictive*): este tipo de quadro é codificado considerando sua semelhança com quadros temporalmente anteriores, que podem ser do tipo I ou P. Se o quadro anterior também for do tipo P é codificada apenas a diferença entre os vetores de movimento.

Quadro tipo B (*Bi-directional*): este tipo de quadro é codificado considerando sua semelhança com quadros temporalmente anteriores ou posteriores, que podem ser dos tipos I, P ou B.

Dentro de um quadro, cada macrobloco pode ser de um tipo diferente. O quadro será denominado B se possuir algum macrobloco do tipo B, P se possuir algum

macrobloco do tipo P sem possuir macroblocos do tipo B, e I, se possuir apenas macroblocos do tipo I.

A estimação de movimento se aplica apenas aos quadros do tipo P e B. Nos quadros do tipo B, além das imagens anteriores são consideradas também imagens posteriores o que melhora a predição para áreas ainda não cobertas pelo movimento na imagem anterior; além disso a média de duas imagens melhora a relação sinal-ruído.

Dos três tipos de quadros apresentados, os do tipo I não apresentam predição de movimento, e por isso são utilizados para quebrar a seqüência de recursividade das imagens.

O padrão MPEG define o conceito de grupo de quadros, GOP (*Group Of Pictures*) ilustrado na Figura 2.9. A reprodução sempre é iniciada em um quadro do tipo I, uma vez que os quadros dos tipos P e B fazem referência a quadros do tipo I. Assim, existe um compromisso entre a taxa de compressão e o acesso randômico.

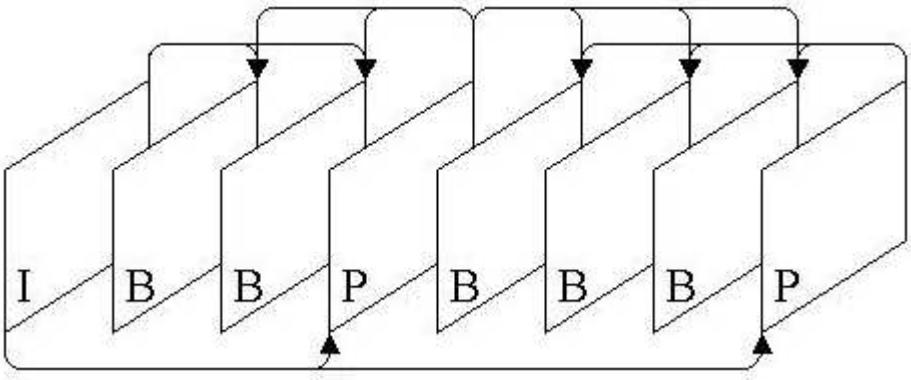

FIGURA 2.9 – Exemplo de uma estrutura temporal de quadros em GOP.

Os grupos de quadros podem ser abertos ou fechados. Em um grupo aberto, quadros do tipo P e B podem utilizar imagens que não pertencem ao grupo, o que não é permitido para grupos fechados.

Uma vez que para a decodificação de uma imagem pode ser necessário buscar informações em quadros temporalmente posteriores, a ordem de apresentação dos quadros não é necessariamente a mesma ordem em que estes são armazenados. Os quadros são armazenados de maneira que os quadros I estejam antes dos quadros P e estes antes dos quadros B dentro de uma certa seqüência com dependências.

3 Estimação de Movimento

O Capítulo 2 apresentou a maneira como o bloco de estimação de movimento se relaciona com os demais blocos da compressão de vídeo, bem como os princípios e etapas mais importantes que fundamentam esta compressão. Este capítulo detalha a estimação de movimento, apresentando inicialmente a forma de representação de um quadro em função de outro utilizando vetores de movimento. A seguir, são apresentados os principais critérios utilizados para comparar diferentes hipóteses de vetor de movimento. As duas seções finais apresentam os principais algoritmos e arquiteturas para estimação de movimento. Nestas duas seções são apresentados apenas aqueles algoritmos e arquiteturas mais importantes e relevantes para a contextualização deste trabalho, uma vez que o universo e implementações da estimação de movimento tanto em software como em hardware é extremamente amplo.

3.1 Representação de Quadros Utilizando Vetores de Movimento

A Figura 3.1 representa dois quadros consecutivos em uma cena de vídeo. O quadro que se deseja representar em função de outro é denominado quadro de referência, pois serve como referência para comparar diferentes hipóteses de vetor de movimento. O quadro onde é realizada a busca pela melhor representação do quadro de referência é denominado quadro de procura. O cálculo dos vetores de movimento é realizado a partir de regiões dos quadros de referência e de procura, denominadas região de referência e região de procura, respectivamente. A região de procura é sempre maior que a região de referência.

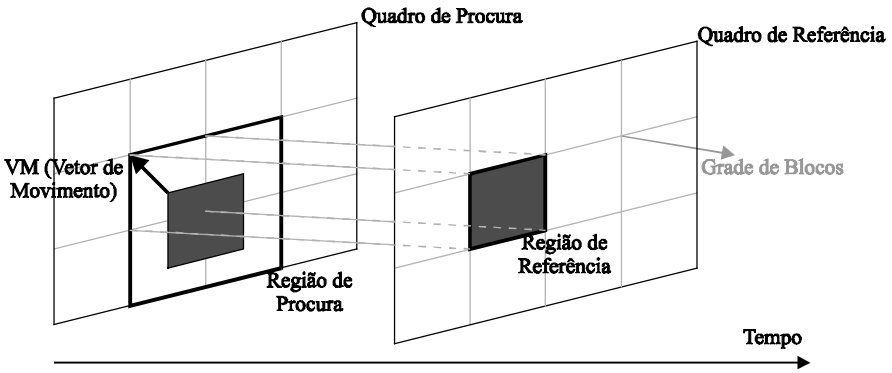

FIGURA 3.1 – Exemplo da utilização de vetores de movimento.

Na Figura 3.1, os elementos que no quadro de procura encontravam-se na região em vermelho, deslocaram-se para a região apontada pelo vetor de movimento VM no quadro de referência. Para perceber este deslocamento é preciso comparar a região de referência com diferentes possibilidades de regiões de tamanho igual ao da região de referência dentro da região de procura [MIT 2001]. O resultado desta comparação é uma grandeza denominada distorção, que é uma representação da diferença entre estas regiões.

As possibilidades que serão consideradas e o critério para cálculo da distorção utilizado para comparar estas possibilidades não são regulamentados pelos padrões de compressão de vídeo, que apenas definem a maneira de incluir esta informação no *bitstream* de saída. Estas decisões são algorítmicas e são tratadas nas Seções 3.2 e 3.3 a seguir.

Assim, um vetor de movimento é utilizado para representar uma região de referência em função de parte de uma região de procura. Isto cria uma dependência entre quadros vizinhos. Esta dependência implica em alguma carga computacional na decodificação, porém, a representação computacional de um vetor de duas componentes é significativamente mais compacta que a representação da região de uma imagem. Este é o motivo pelo qual a estimação de movimento é uma poderosa técnica de compressão de vídeo.

Em vídeos naturais existem pequenas diferenças entre a região de referência e a parte da região de procura que a representa no vídeo codificado. Esta diferença é expressa através de uma matriz de erro que é calculada no bloco de predição. Uma vez que estas regiões são muito similares, a matriz de erro é composta essencialmente por elementos de valor pequeno ou de valor nulo. Esta é uma maneira de transformar redundância temporal em redundância entrópica, que implicará em uma alta taxa de compressão quando estes dados passarem pelos blocos da compressão de imagens estáticas.

O tamanho das regiões de referência e de procura tem influência na taxa de compressão, no esforço computacional para estimação de movimento e na qualidade do vídeo reproduzido. A região de referência deve ter tamanho significativamente menor que o tamanho dos objetos que realizam movimento. O tamanho 8x8 para região de referência é muito comum, tendo sido adotado por aplicações comerciais e pelo padrão MPEG. O tamanho da região de procura não é regulamentado pelos padrões de compressão de vídeo, sendo esta uma decisão do codificador. Quanto maior a região de procura, maior a probabilidade de encontrar um bloco que representa bem a região de referência, em contrapartida, maior será a taxa de operações para a estimação de movimento. Assim, é possível encontrar diferentes tamanhos de região de procura, tais como 16x16 e 32x32 [QUE 2001] e até mesmo 256x256 [SON 2001].

3.2 Critérios para Determinação da Distorção entre Regiões

Três critérios são usuais para determinação da distorção entre regiões: o menor erro absoluto, MAE (em inglês, *mean absolute error*); a soma de diferenças absolutas, SAD (em inglês, *sum of absolute differences*) e o menor erro quadrático, MSE (em inglês, *mean square error*) [MIT 2001]. Estes critérios são utilizados para comparar a região de referência com partes diferentes da região de procura, cada uma representando uma hipóteses de vetor de movimento. Os critérios têm como resultado uma medida da diferença (ou inversamente, semelhança) entre duas regiões, denominada distorção.

A Figura 3.2 ilustra o cálculo da distorção entre duas regiões. Nesta figura a hipótese de vetor de movimento (i,j) aponta para a parte da região de procura que está sendo considerada para o cálculo da distorção. A região de referência tem dimensões MxN pontos. A região de procura é p pontos maior que a região de referência nas duas dimensões.

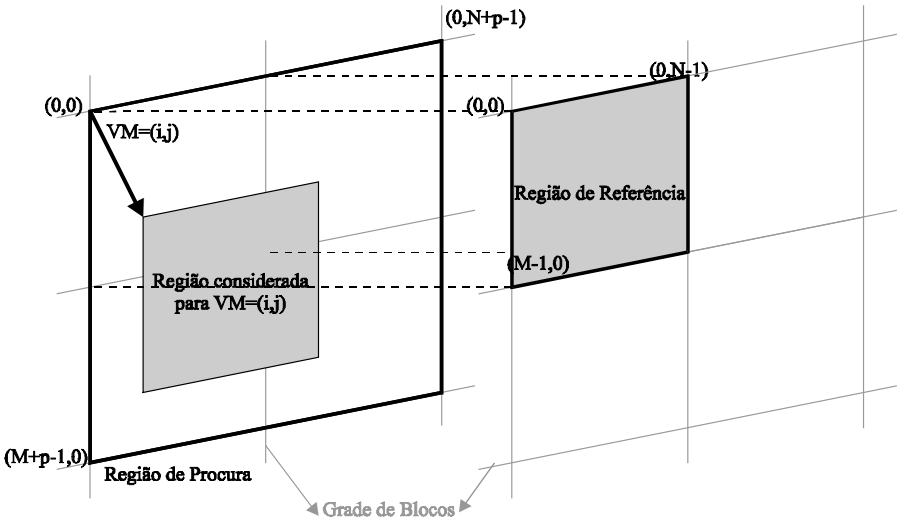

FIGURA 3.2 – Ilustração do cálculo da distorção entre duas regiões.

O critério MAE define a distorção entre duas regiões como sendo o valor médio do módulo da diferença, ou seja, da diferença absoluta entre os pontos da região de referência e os pontos correspondentes na região de procura. A distorção calculada segundo o critério MAE para a situação da Figura 3.2 é dada por:

$$D(i,j) = \frac{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left|R_{m,n} - P_{m+i,n+i}\right|}{M \times N} \quad \forall i, j \in [0, p] \tag{3.1}$$

onde:

- *D(i,j)* representa a distorção para a região apontada por VM=(i,j).
- $R_{m,n}$ representa o ponto da região de referência de coordenadas (m,n).
- $P_{m,n}$ representa o ponto da região de procura de coordenadas (m,n).

O critério SAD define a distorção entre duas regiões como sendo a somatória das diferenças absolutas entre os pontos da região de referência e os pontos correspondentes na região de procura. O critérios MAE e SAD têm resultados proporcionais que diferem por um fator *MxN*. A distorção calculada segundo o critério MAE para a situação da Figura 3.2 é dada por:

$$D(i,j) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left|R_{m,n} - P_{m+i,n+i}\right| \quad \forall i, j \in [0, p] \tag{3.2}$$

onde *D(i,j)*, $R_{m,n}$ e $P_{m,n}$ têm significados equivalentes aos da definição do MAE.

O critério MSE define a distorção entre duas regiões como sendo o valor médio da diferença elevada ao quadrado entre os pontos da região de referência e os pontos correspondentes na região de procura. A distorção calculada segundo o critério MSE para a situação da Figura 3.2 é dada por:

$$D(i,j) = \frac{\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left(R_{m,n} - P_{m+i,n+i}\right)^2}{M \times N} \quad \forall i, j \in [0, p] \qquad (3.3)$$

onde $D(i,j)$, $R_{m,n}$ e $P_{m,n}$ têm significados equivalentes aos da definição do MAE.

Em todos os critérios apresentados o valor de $D(i,j)$ é sempre maior que zero. No MAE e no SAD o valor absoluto garante a soma apenas de números positivos, enquanto no MSE o expoente quadrático tem esta função. Isto garante que erros negativos não compensem erros positivos.

As definições dos critérios para determinação do vetor de movimento apresentadas acima são dependentes de i e j. Estas variáveis têm uma faixa de variação que garante o cálculo da distorção para todas as hipóteses de vetor de movimento. O número de hipóteses de vetor de movimento depende da faixa de variação destas grandezas sendo definido pela equação:

$$H = (\Delta i + 1) \times (\Delta j + 1) \quad (3.4)$$

onde:

- H representa número de hipóteses diferentes para o vetor de movimento.
- Δi representa a faixa de variação de i.
- Δj representa a faixa de variação de j.

O critério MSE resulta em valores de distorção relacionados à potência da diferença entre as regiões analisadas, pois a potência de um sinal é proporcional ao quadrado de sua amplitude. Assim, analisando os critérios apresentados, conclui-se que o critério MSE tem significado físico mais relevante em comparação com os critérios MAE e SAD. Por outro lado, os critérios MAE e SAD necessitam significativamente menos capacidade computacional para seu cálculo. Em especial, o critério SAD conduz a um resultado equivalente ao do MAE, necessitando de menos capacidade computacional que este. Por este motivo muitas implementações da estimação de movimento são baseadas no critério SAD, como por exemplo em [ACK 97], [QUE 2001] e [TOU 99].

O vetor de movimento é definido como sendo o vetor de coordenadas (i^*,j^*) que minimiza o valor de $D(i,j)$. Matematicamente, o vetor de movimento é definido pela equação:

$$VM = (i^*, j^*) \quad | \quad D(i^*, j^*) = \min(D(i, j)), \quad \forall\, i, j \in [0, p] \qquad (3.5)$$

Os principais algoritmos utilizados para a determinação de (i*,j*) são apresentados na próxima seção.

3.3 Algoritmos para Estimação de Movimento

O algoritmo empregado para o cálculo do vetor de movimento tem influência no tempo e na capacidade computacional necessária para o cálculo dos vetores de movimento. Além disso, ainda pode ter influência na qualidade do vídeo reproduzido.

Os algoritmos para estimação de movimento se diferenciam por utilizarem maneiras diferentes de escolher as hipóteses de vetor de movimento que serão consideradas, bem como a ordem em que são calculadas. O critério de parada e o critério para cálculo da distorção adotados são considerados apenas uma variação na utilização de um algoritmo. Por isto, esta seção foca principalmente a metodologia utilizada em cada algoritmo para escolher as hipóteses de vetor de movimento a serem consideradas.

Os algoritmos aqui abordados são: busca completa [BHA 99], busca circular [TOU 2001], busca logarítmica [SHI 2000], busca por casamento de bordas [CHY 97], busca hierárquica [CHU 97] e busca por *clustering* de blocos [FUJ 97].

O algoritmo de busca completa, apresentado em [BHA 99] e [QUE 2001], consiste em calcular a distorção para todas as hipóteses possíveis de vetor de movimento e tomar como vetor de movimento o par ordenado (i,j) ao qual está associado o menor valor de distorção. O algoritmo busca completa aplicado à situação da Figura 3.2 pode ser definido pelos seguintes passos:

- Definir o critério para cálculo da distorção entre blocos.
- VM = (0,0)
- D_{min} = D(0,0)
- Para todo (i,j) $\in$ [0,p]:

  Se D(i,j) < D_{min}

  D_{min} = D(i,j)

  VM = (i,j)

No algoritmo busca completa o número de operações é proporcional ao produto da área da região de referência pela área da região de procura. Considerando a situação da Figura 3.2, utilizando-se o critério SAD e o algoritmo busca completa, o número de operações necessárias para determinar um vetor de movimento é dado por:

$$OP = M \times N \times (p+1)^2 \tag{3.6}$$

onde *OP* representa o número de operações inteiras.

Embora o algoritmo de busca completa exija grande capacidade computacional para sua implementação, ele é o algoritmo que leva ao resultado ótimo, ou seja, àquele vetor de movimento associado ao menor valor de distorção. Assim, implementações utilizando o algoritmo busca completa são mais comuns na produção de vídeo [TOU 2001], situação na qual geralmente é possível dedicar maior capacidade computacional para compressão do que em aplicações de baixo custo ou em sistemas embarcados para eletrônica de consumo de massa.

Os demais algoritmos realizam uma busca ordenada, calculando a distorção para diferentes hipóteses de vetor de movimento até encontrar um valor de distorção que satisfaça um critério de parada definido previamente. Em geral, os critérios de parada são dois: que o valor da distorção seja menor que um limiar previamente definido, ou que o número de hipóteses consideradas seja maior que um máximo previamente definido.

O algoritmo de busca circular, proposto por [TOU 2001], apresenta uma busca em uma quantidade previamente definida de zonas circulares ao redor do valor apontado pelo último vetor de movimento calculado. Sua proposta de zonas circulares supõe que cada vetor de movimento é igual ao último vetor calculado mais um deslocamento cuja probabilidade é igual em todas as direções. Esta suposição é aceitável, uma vez que existe uma correlação entre vetores de movimento adjacentes. A busca termina quando o valor de D(i,j) é menor que um limiar de parada previamente definido ou quando todas as zonas forem analisadas.

O algoritmo de busca logarítmica, apresentado em [SHI 2000], considera, a cada iteração, uma região de procura cuja área corresponde a um quarto da área utilizada na iteração anterior. Inicialmente é calculada a distorção para o ponto central e para os quatro pontos médios entre o centro e as bordas da região de procura. A cada iteração é considerada uma região de procura cujo centro corresponde ao ponto de menor distorção da iteração anterior, enquanto a distância do centro às bordas é reduzida pela metade. As iterações se repetem, reduzindo a área da região de procura em escala exponencial, e portanto aproximando-se da melhor hipótese de vetor de movimento em escala logarítmica. As iterações terminam quando a distância do centro às bordas for nula. Em caso de empate no valor das distorções uma escolha aleatória é aceita, uma vez que se sabe de antemão que o algoritmo de busca logarítmica pode não levar ao resultado ótimo.

O algoritmo de casamento de bordas proposto por [CHY 97] faz uso de um algoritmo auxiliar para detecção de bordas, o que permite identificar as regiões localizadas sobre o contorno dos elementos da cena que realizam movimento. Isto permite classificar as regiões de referência de um quadro em quatro tipos: regiões localizadas sobre o contorno dos elementos da cena que realizam movimento, regiões internas a estes elementos, regiões externas a estes elementos e regiões que não têm correspondente no quadro de procura. Inicialmente é utilizada uma região de procura pequena para o cálculo dos vetores de movimento para as regiões externas aos elementos que realizam movimento na cena. Para as regiões localizadas sobre o contorno destes elementos, Yui-Lam Chan [CHY 97] propõe uma equação para cálculo da distorção fortemente dependente das altas freqüências, que representam as bordas. Para as regiões internas a cada elemento é utilizada uma busca dependente dos vetores de movimento das regiões pertencentes às bordas do elemento correspondente. Para as regiões que não têm correspondente no quadro de procura não deve ser calculado o vetor de movimento.

O algoritmo de busca hierárquica apresentado em [CHU 97] consiste em calcular inicialmente um vetor de movimento global referente ao movimento do fundo da cena, e a seguir calcular os demais vetores de movimento utilizando o vetor de movimento global para deslocar o centro da região de procura. Para tal, Chung-Tao [CHU 97] apresenta uma equação que mostra ser bastante eficiente para o cálculo do vetor de movimento global. Esta equação pondera oito vetores de movimento calculados localmente em regiões previamente determinadas. A abordagem de dividir o movimento em dois níveis de hierarquia faz com que a busca local seja iniciada em uma região com maior probabilidade de ser o vetor de movimento. A busca local é terminada ao encontrar uma hipótese de vetor de movimento com distorção menor que um limiar previamente definido.

O algoritmo de *clustering* de blocos apresentado em [FUJ 97] considera *clusters*, ou conjuntos de regiões, para as quais é realizada a busca de maneira hierárquica similar àquela utilizada pelo algoritmo busca hierárquica. Inicialmente, os quadros de referência

e de procura são sub-amostrados por um fator *n*. A seguir, são calculados os vetores de movimento a partir destes quadros sub-amostrados. A quantidade de vetores de movimento relativos aos quadros sub-amostrados é *n* vezes menor do que seria sem a sub-amostragem. Cada *cluster*, que é um conjunto de *n* regiões pertencentes ao quadro de referência original, está associado a um dos vetores de movimento calculados utilizando os quadros sub-amostrados. O centro das regiões de procura de cada *cluster* do quadro original é deslocado por *n* vezes do vetor de movimento calculado utilizando os quadros sub-amostrados. Com isso, a área das regiões de procura deslocadas pode ser reduzida em n^2 vezes a área das regiões de procura originais. Por fim, os vetores de movimento são calculados a partir destas regiões de procura deslocadas e reduzidas utilizando o algoritmo busca completa.

A Tabela 3.1 [CHY 97] [FUJ 97] [TOU 2001] apresenta a taxa de operações prevista pelos algoritmos para estimação de movimento considerando um vídeo de resolução 640x480 à taxa de 30 quadros por segundo. Analisando esta tabela, percebe-se que os algoritmos busca circular e busca hierárquica são os que necessitam menor taxa de operações, enquanto o algoritmo de busca completa necessita a maior taxa de operações para a estimação de movimento. A diferença na qualidade dos vetores de movimento não foi considerada nesta análise uma vez que a predição de movimento (vide Figura 2.7) pode compensar esta diferença.

TABELA 3.1 – Taxa de operações para estimação de movimento por algoritmo.

Algoritmo	Taxa de Operações (MOps)
Busca Completa	530
Busca Circular	20
Busca Logarítmica	37
Busca por Casamento de Bordas	245
Busca Hierárquica	20
Busca por *Clustering* de Blocos	72

Vale observar que todos os algoritmos analisados são baseados no casamento de blocos sendo que todos eles podem ser implementados opcionalmente com qualquer um dos critérios MAE, SAD ou MSE.

3.4 Arquiteturas de *Hardware* para Estimação de Movimento

Esta seção apresenta uma amostra do universo de arquiteturas para estimação de movimento que foram encontradas na literatura. Estas arquiteturas servem como parâmetros em termos de desempenho para comparação com a arquitetura apresentada no Capítulo 4.

Inicialmente esta seção apresenta a arquitetura do circuito integrado STi3220 [QUE 2001] a fim de ilustrar as características de um circuito comercial para estimação de movimento. A seguir é exposta a arquitetura de um conjunto linear de 16 elementos de processamento, apresentada em [BHA 99]. Esta arquitetura é uma referência comumente citada na literatura e, portanto, é apresentada com bom nível de detalhes. A seguir são apresentadas outras três arquiteturas baseadas em elementos de processamento que implementam diferentes algoritmos para a estimação de movimento. Por fim, para relacionar as arquiteturas apresentadas nesta seção com a arquitetura

desenvolvida neste trabalho, é comentada a arquitetura conjunto linear de 8 elementos de processamento proposta por [ZAN 2001].

3.4.1 Arquitetura do Circuito Integrado STi3220

Em [QUE 2001] é apresentada de maneira restrita a arquitetura adotada no circuito integrado comercial STi3220, produzido pela empresa SGS-THOMSON. Os detalhes arquiteturais deste circuito não são divulgados, e portanto ele é aqui apresentado sob o ponto de vista externo. Além do STi3220, existem circuitos comerciais mais completos, que implementam todas as tarefas para compressão de vídeo, como é o caso do CXD1922Q [SON 2001] e o CS92288 [STR 2001]. Estes circuitos também realizam a estimação de movimento, porém, os detalhes de suas arquiteturas não são divulgados.

O STi3220 calcula os vetores de movimento considerando uma região de procura que seja até 16 pontos em cada direção maior que a região de referência. A região de referência pode ter 8 ou 16 linhas; os tamanhos suportados variam entre 8x4 e 8x32 ou entre 16x4 e 16x32 pontos. O critério para cálculo da distorção adotado é o SAD.

A entrada do STi3220 é composta pelo barramento R para dados de referência e pelos barramentos A, B e C para dados de procura. A Figura 3.3 [QUE 2001] apresenta seu esquema de alimentação de dados. Todos os barramentos trabalham à taxa de um dado por ciclo de processamento. A carga de dados é feita no sentido vertical, ou seja, coluna por coluna. Para tal, a região de procura é dividida em três faixas horizontais, de modo que cada uma das entradas A, B e C recebe os dados pertencentes a uma destas faixas. A faixa B possui o mesmo número de linhas que a região de referência, 8 ou 16. As faixas A e C possuem até 16 linhas, dependendo da diferença de tamanho entre a região de referência e a região de procura.

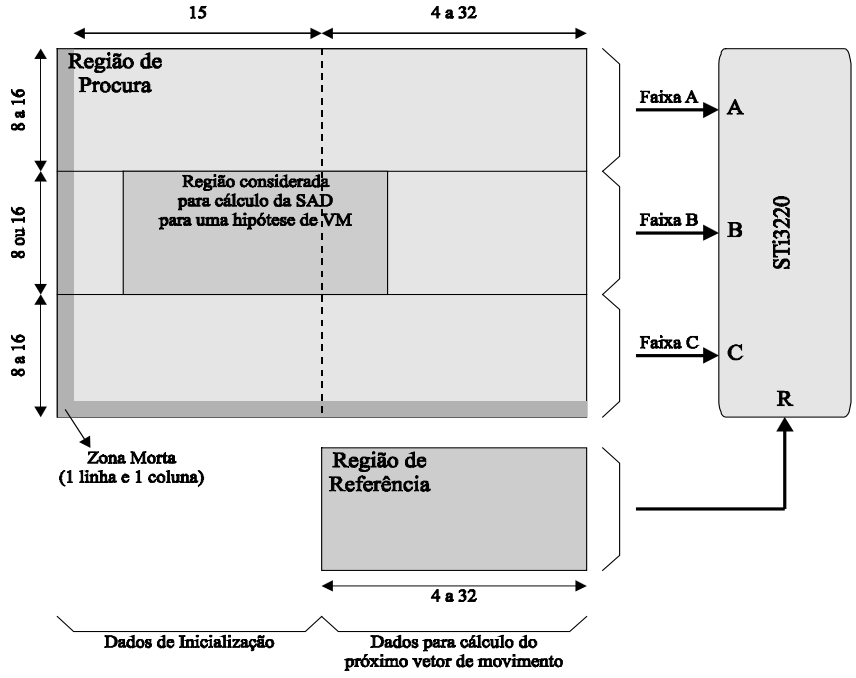

FIGURA 3.3 – Esquema de alimentação de dados do STi3220.

A região de procura possui uma zona morta de uma linha e uma coluna. Esta denominação deve-se ao fato de que as hipóteses de vetor de movimento relativas a esta

zona não são consideradas. Contudo, os dados da zona morta precisam ser carregados pois fazem parte da região considerada para cálculo da SAD para outras hipóteses de vetor de movimento.

A inicialização do STi3220 deve ser feita com os dados necessários para calcular as primeiras hipóteses de vetor de movimento, que são 16 colunas. Se o circuito for utilizado para calcular vetores de movimento referentes a regiões de referência adjacentes, os dados na memória podem ser reutilizados. Nesta situação, os dados de entrada, referentes às novas colunas, sobrescrevem os dados da memória, referentes às colunas não mais necessárias. Este esquema de alimentação de dados garante o funcionamento contínuo do circuito.

A máxima freqüência de trabalho do STi3220 é 18,25 MHz. A esta freqüência ele realiza 2,65 GOps de 8 *bits* sendo capaz de processar vídeo no formato 720x576 pontos à taxa de 44,01 quadros por segundo ou um quadro a cada 22,72 ms [QUE 2001], [SAN 98].

3.4.2 Conjunto Linear de 16 Elementos de Processamento

Em [BHA 99] é apresentada uma arquitetura que pode ser denominada conjunto linear de 16 elementos de processamento. Esta proposta de arquitetura é mostrada conceitualmente, não estando disponíveis no trabalho informações relativas da implementação. A mesma é composta por um conjunto de dezesseis elementos de processamento que trabalham em paralelo. Este conjunto é linear, o que significa que as diferenças que estão sendo calculadas em um certo momento referem-se sempre a pontos pertencentes à mesma linha. O critério utilizado para cálculo da distorção é o SAD. Os vetores de movimento são calculados a partir de uma região de referência de tamanho 16x16, sendo a busca realizada sobre uma região de procura de tamanho 31x31.

A Figura 3.4 [BHA 99] ilustra esta arquitetura. Uma vez que as linhas da região de procura têm praticamente o dobro de pontos que as linhas da região de referência, é utilizada memória de procura de porta dupla (aquela em que se pode acessar dois endereços simultaneamente). A metade esquerda das linhas armazenadas na memória de procura é acessada pelo sinal `S1` e a metade direita por `S2`. Os elementos de processamento são responsáveis por calcular e acumular a diferença entre o sinal de referência, `R`, e um dos sinais de procura, `S1` ou `S2`. As operações nos elementos de processamento (EPs) ocorrem em *pipeline*, ou seja, os dados provenientes do sinal `R` são recebidos por `EP0` e passam, a cada ciclo de processamento, para o elemento seguinte até atingir `EP15`, conforme mostra a Figura 3.4.

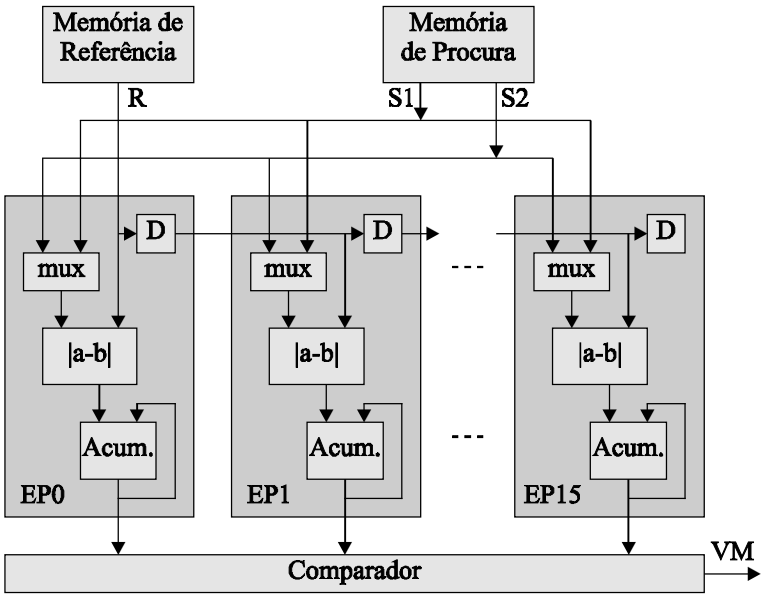

FIGURA 3.4 – Arquitetura conjunto linear de 16 elementos de processamento.

Dezesseis ciclos são necessários para processar uma linha. Após processar cada linha, a linha abaixo é tomada, até a última linha da região de procura, totalizando 256 ciclos de processamento. O processamento sobre toda a região de procura se repete para cada linha da região de referência. Assim, o cálculo de um vetor de movimento demora 4096 ciclos de processamento.

Um comparador recebe os valores de SAD calculados em cada elemento de processamento. Sua função é comparar as hipótese de vetor de movimento e apontar a de menor SAD como o vetor de movimento.

3.4.3 Arquitetura Baseadas no *Clustering* de Regiões

A arquitetura proposta por [FUJ 97] baseia-se no algoritmo *clustering* de regiões apresentado na Seção 3.3. Esta arquitetura foi projetada para realizar a estimação de movimento para o padrão H.263, que é um padrão de compressão para vídeo de baixa resolução (176x144 pontos), o que pode simplificar bastante a arquitetura. Esta arquitetura calcula os vetores de movimento a partir de uma região de referência de tamanho 16x16 pontos, realizando a busca sobre uma região de procura de tamanho 32x32 pontos. O critério para cálculo da distorção adotado é o SAD.

A arquitetura baseada no *clustering* de regiões [FUJ 97] é composta por um conjunto de oito elementos de processamento que formam um *pipeline* similar ao proposto por [BHA 99]. Além disso, existe uma unidade de geração de dados amostrados, uma unidade de acumulação e uma unidade de comparação, conforme apresenta a Figura 3.5 [FUJ 97]. A arquitetura acessa uma memória de quadro externa de 4MB através de um barramento de 16 *bits*.

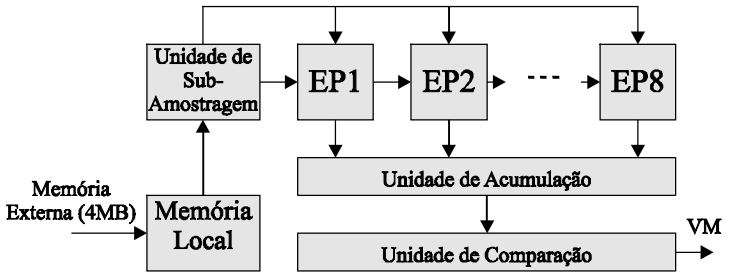

FIGURA 3.5 – Arquitetura baseada no *clustering* de regiões.

Na arquitetura baseada no *clustering* de regiões, antes dos dados da memória externa serem utilizados, eles são armazenados temporariamente em uma memória local. A seguir, estes dados passam pela unidade de sub-amostragem, onde são sub-amostrados por um fator *n* de acordo com o algoritmo apresentado na Seção 3.3. Os dados de referência percorrem um *pipeline* através dos elementos de processamento, enquanto os dados de procura podem ser acessados simultaneamente por todos os elementos de processamento. Os elementos de processamento realizam o cálculo das diferenças absolutas, que são armazenadas na unidade de acumulação. A unidade de comparação gera o vetor de movimento comparando os valores de SAD armazenados na unidade de acumulação.

A máxima freqüência de trabalho da arquitetura baseada no *clustering* de regiões é 15 MHz. A esta freqüência a arquitetura realiza 0,12 Gops, sendo capaz de processar vídeo no formato 176x144 pontos à taxa de 18,49 quadros por segundo ou um quadro a cada 54,06 ms [FUJ 97].

3.4.4 A Arquitetura EST256

Em [SAN 98] é proposta a arquitetura denominada EST256. Esta arquitetura implementa o algoritmo de busca completa apresentado na Seção 3.3, e foi desenvolvida para suportar o padrão H.261. O padrão H.261 utiliza quadros na resolução 176x144 pontos. A região de referência considerada é de tamanho 16x16 pontos, e a busca é realizada sobre uma região de procura de 32x32 pontos. O critério utilizado para o cálculo da distorção entre regiões é o SAD.

A arquitetura EST256 é composta por um conjunto de 256 elementos de processamento que trabalham em paralelo formando um *pipeline*. Cada elemento de processamento é responsável por calcular a SAD referente a uma hipótese de vetor de movimento. Esta arquitetura necessita de 256 ciclos de processamento para inicialização de seu *pipeline*, sendo ainda necessários outros 256 ciclos para o cálculo de cada vetor de movimento.

A arquitetura EST256 possui recursos de *hardware* adicionais prevendo a possibilidade de utilizar múltiplas instâncias da arquitetura para realizar a busca sobre uma região de procura maior. Vale dizer que em [SAN 98] as referências bibliográficas [BHA 99] e [QUE 2001] são citadas, a primeira como base teórica e a segunda como parâmetro de comparação.

A máxima freqüência da arquitetura EST256 é 20 MHz. A esta freqüência a arquitetura realiza 11 GOps sendo capaz de processar vídeo no formato 720x576 pontos à taxa de 48,22 quadros por segundo ou um quadro a cada 20,73 ms [SAN 98].

3.4.5 Arquitetura Baseada na Busca Hierárquica

A arquitetura proposta por [SEO 99] baseia-se no algoritmo busca hierárquica apresentado na Seção 3.3. Esta proposta de arquitetura é mostrada conceitualmente, não estando disponíveis no trabalho informações relativas da implementação.

Esta proposta é conceitual de modo que não são colocadas informações relativas à implementação. Esta arquitetura considera uma região de referência de tamanho 16x16 pontos e uma região de procura de até 32x32 pontos. O critério utilizado para o cálculo da distorção entre regiões é o SAD.

O algoritmo de busca hierárquica é baseado na sub-amostragem, assim, esta arquitetura realiza a estimação de movimento considerando diferentes taxas de sub-amostragem dos pontos das regiões de referência e de procura. Para tal, a interface de entrada e saída da arquitetura proposta por [SEO 99] admite diferentes configurações para suportar diferentes taxas de sub-amostragem.

A arquitetura baseada na busca hierárquica é composta por um conjunto de 64 elementos de processamento que trabalham em paralelo, formando um *pipeline*. O cálculo dos vetores de movimento ocorre em paralelo com a carga de dados feita por sua interface de entrada e saída (E/S). A interface de E/S também possui um *pipeline*, possibilitando o recebimento contínuo de dados. Pequenos blocos de memórias internas garantem a reutilização de dados reduzindo, assim, o acesso à memória externa. A arquitetura ainda possui recursos de *hardwar*e adicionais prevendo a utilização de múltiplas instâncias para aumentar o nível de paralelismo.

3.4.6 Conjunto Linear de 8 Elementos de Processamento

Em [ZAN 2001] é proposta uma arquitetura que pode ser denominada conjunto linear de 8 elementos de processamento. Esta arquitetura é composta por um conjunto de oito elementos de processamento muito similar àquela apresentada por [BHA 99]. Porém, na arquitetura proposta pelo autor [ZAN 2001], é utilizada uma região de referência de tamanho 8x8 pontos e região de procura de 15x15 pontos. Nesta situação o processamento para cada linha necessita de 64 ciclos de processamento, totalizando assim 512 ciclos de processamento para o cálculo de um vetor de movimento. O desempenho destas duas arquiteturas pode ser considerado equivalente. Contudo, cada uma adotando um tamanho de região de procura diferenciado.

A máxima freqüência de trabalho da arquitetura conjunto linear de 8 elementos de processamento obtida por simulação visando a implementação no dispositivo FPGA EPF10K20RC208-4 da Altera é de 8,32 MHz. A esta freqüência a arquitetura realiza 66,56 MOps sendo capaz de processar vídeo no formato 640x480 pontos à taxa de 3,38 quadros por segundo ou um quadro a cada 295,38 ms [ZAN 2001].

A arquitetura desenvolvida neste trabalho, que será apresentada no Capítulo 4, pode ser entendida como uma extensão da arquitetura proposta pelo autor [ZAN 2001] para um arranjo bi-dimensional de elementos processadores. Esta extensão visa aumentar o nível de paralelismo e diminuir, proporcionalmente, a largura dos barramentos de entrada e saída de dados.

4 A Arquitetura para Estimação de Movimento

Este capítulo apresenta a arquitetura desenvolvida para o cálculo dos vetores de movimento utilizados na compressão de vídeo digital. Esta arquitetura é inspirada na arquitetura apresentada em [BHA 99], com a diferença de que esta arquitetura tem um nível de paralelismo maior. Além disso, para a arquitetura apresentada foi desenvolvida uma interface de E/S capaz de gerenciar o recebimento contínuo de dados de vídeo.

Inicialmente este capítulo apresenta uma descrição geral da arquitetura. As seções seguintes apresentam os blocos que compõem a arquitetura, explicitando a relação entre os sinais de controle e a ordem em que as operações ocorrem nestes blocos.

4.1 Aspectos Gerais da Arquitetura

A arquitetura para estimação de movimento implementa o algoritmo de busca completa para casamento de blocos. O algoritmo de busca completa foi escolhido em função de sua regularidade, o que é muito importante para uma implementação em *hardware*. O critério utilizando para cálculo da distorção entre regiões é o SAD (soma das diferenças absolutas). Este critério é utilizado por todas as arquiteturas apresentadas na Seção 3.4 e foi escolhido por não envolver multiplicações, o que simplifica os blocos lógicos implementados.

Esta arquitetura suporta vídeo no formato 640x480 pontos, e recebe um ponto da imagem a cada ciclo de processamento. A máxima taxa de quadros por segundo que a arquitetura suporta é uma função da freqüência máxima de operação do circuito quando implementado em uma tecnologia, por exemplo, FPGA. A máxima taxa de quadros por segundo é definida pela equação:

$$qps = \frac{f_{\max}}{H \times V} = \frac{f_{\max}}{640 \times 480} \tag{4.1}$$

onde:

- *qps* é a taxa máxima de quadros por segundo;
- f_{max} é a freqüência máxima de operação do circuito implementado;
- *H* é a quantidade de pontos em uma linha (horizontal) da imagem;
- *V* é a quantidade de pontos em uma coluna (vertical) da imagem.

A estimação de movimento é realizada tomando como referência uma região da imagem de tamanho 8x8, denominada região de referência, e procurando o melhor casamento de blocos sobre uma região de tamanho 15x15, denominada região de procura. O tamanho 8x8 para a região de referência foi escolhido por ser o mesmo tamanho de bloco utilizado para cálculo da DCT em compressão de imagens, além de ser também um tamanho muito utilizado para estimação de movimento, inclusive em aplicações comerciais [MIT 2001], [FUJ 97].

A utilização de uma região de procura de tamanho 15x15 pontos é uma restrição da arquitetura desenvolvida em comparação com outras arquiteturas que utilizam região de procura de 16x16 pontos ou maior. A escolha de uma região de procura de tamanho 15x15 teve as seguintes motivações:

- Os padrões de compressão de vídeo raramente determinam tamanhos fixos para as regiões de procura, sendo este um grau de liberdade do codificador. Neste caso a região de procura de tamanho 15x15 foi uma boa solução de compromisso entre o custo computacional (em termos de área) e a taxa de compressão.
- Neste sentido, a arquitetura desenvolvida utiliza uma estratégia de endereçamento que reduz sensivelmente a quantidade de pinos de entrada e saída, a área ocupada pelos barramentos do circuito e o acesso à memória.
- A arquitetura apresentada neste capítulo pode ser replicada para suportar regiões maiores, de tamanho (15+8k)x(15+8k) para k≥0. A escolha do tamanho 15x15 garante que a distorção associada a cada hipótese de vetor de movimento é calculada apenas uma vez, o que não aconteceria para o tamanho 16x16. A possibilidade de utilizar múltiplas instâncias desta arquitetura é apresentada conceitualmente no Capítulo 6 como conclusão do trabalho.

4.2 Descrição Conceitual da Arquitetura

A função da arquitetura para estimação de movimento é calcular os vetores de movimento. A Figura 4.1 mostra como a arquitetura desenvolvida se encaixa com os demais blocos da compressão de vídeo. A parte da Figura 4.1 que foi implementada e está sendo descrita neste capítulo é o bloco de estimação de movimento e também uma parte da memória de quadro.

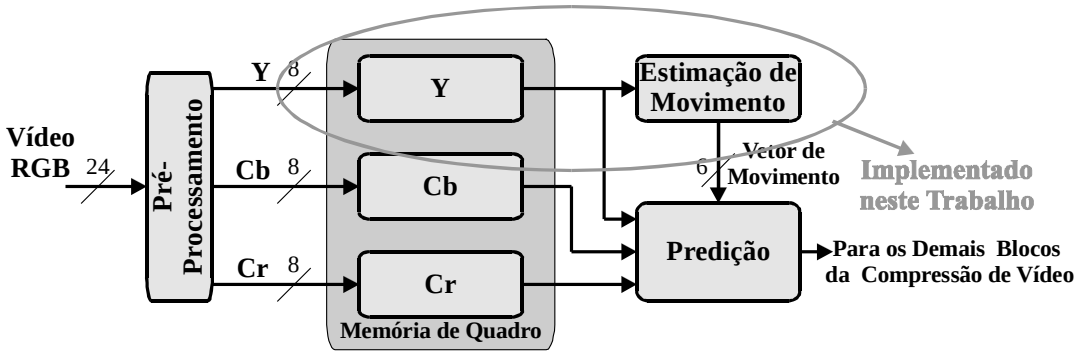

FIGURA 4.1 – Diagrama em blocos de uma aplicação da arquitetura desenvolvida.

Na Figura 4.1 o bloco de estimação de movimento é alimentado por uma amostra do vídeo de entrada tomada após o pré-processamento. Em contrapartida, na Figura 2.7, o sinal para alimentar o bloco de estimação de movimento é tomado após passar pelos blocos de DCT, quantização, DCT inversa e quantização inversa. O cálculo dos vetores de movimento a partir do sinal original tem como resultado vetores de movimento que causam a menor distorção no vídeo original, enquanto o uso do sinal quantizado tem como resultado vetores de movimento que causam a menor distorção no sinal quantizado. Para um erro de quantização pequeno, que se aplica a cenas de vídeo natural [MIT 2001], as duas implementações são equivalentes.

A entrada da arquitetura é o sinal de luminância Y de 8 *bits*, que é uma das componentes de um sinal de vídeo em representação de cores YCbCr de 24 *bits*. A taxa de entrada de dados na arquitetura para estimação de movimento é de um ponto a cada ciclo de processamento. Para tal o sinal de entrada deve, a cada ciclo de processamento, representar um ponto diferente da imagem.

A saída da arquitetura é um vetor de movimento de 6 *bits*. O vetor de movimento é um sinal atualizado a cada 64 ciclos de processamento, o que corresponde ao período de entrada para um bloco.

A arquitetura é composta por uma unidade operacional controlada por uma unidade de controle. A interface entre estas duas unidades é feita através dos sinais de controle que partem da unidade de controle para a unidade operacional, e do sinal NOVO_VM que parte da unidade operacional para a unidade de controle, conforme mostra a Figura 4.2. Esta figura mostra ainda que a unidade operacional é formada pelos seguintes blocos: interface de E/S, matriz de processamento e unidade de comparação.

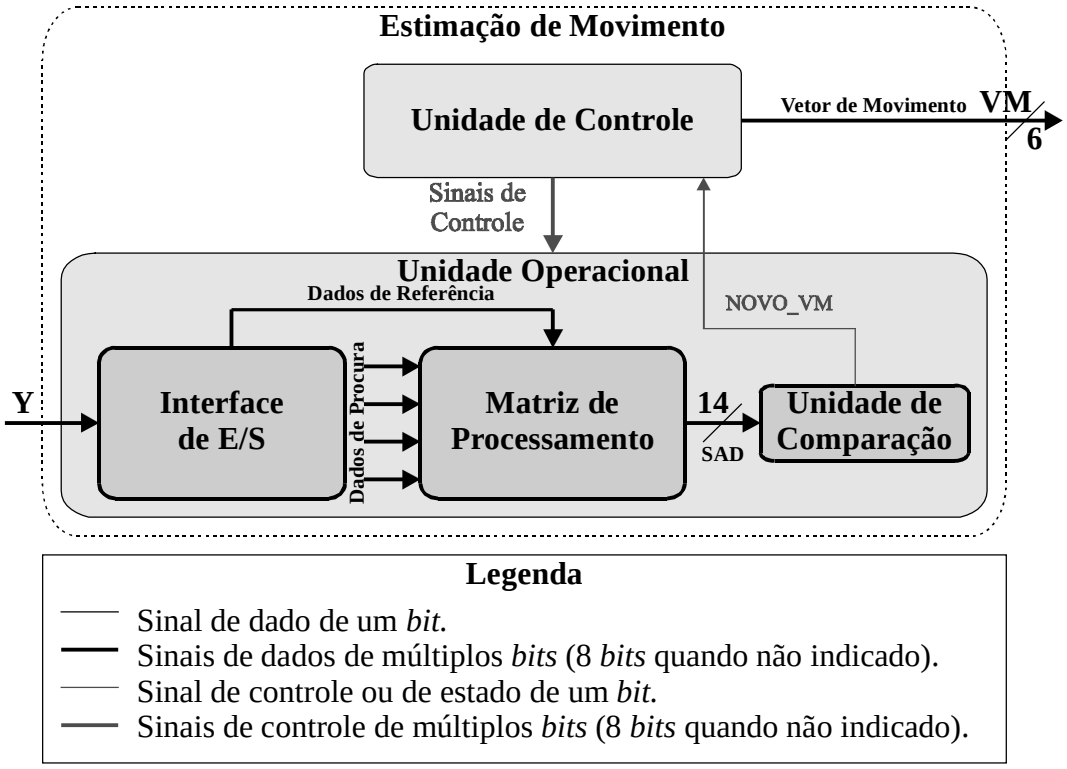

FIGURA 4.2 – Diagrama de blocos da arquitetura para estimação de movimento.

Na Figura 4.2, como nas demais que seguem, os sinais relativos ao caminho de dados são representados em preto e os sinais de controle são representados em vermelho. Os sinais de 1 *bit* são representados por linhas finas, enquanto sinais de múltiplos *bits* são representados por linhas largas. Quando não especificado, os sinais de múltiplos *bits* possuem 8 *bits*.

O papel da interface de E/S é organizar os dados na maneira esperada pela matriz de processamento. A entrada da arquitetura para estimação de movimento é a entrada da interface de E/S, que é a componente Y do sinal de vídeo de entrada. Suas saídas são quatro sinais com os dados de procura e um sinal com os dados de referência.

A matriz de processamento é o bloco no qual é realizada a principal operação para a estimação de movimento: o cálculo da SAD para as diferentes hipóteses de vetor de movimento. Este bloco tem como entrada os sinais de procura e de referência provenientes da interface de E/S. Sua saída é o sinal SAD de 14 *bits* que representa a SAD para diferentes hipóteses de vetor de movimento a cada ciclo de processamento.

Este sinal de 14 *bits* é a entrada da unidade de comparação, que analisa as diferentes SADs e envia para a unidade de controle o sinal NOVO_VM quando uma melhor hipótese para vetor de movimento for identificada. Este sinal é utilizado pela unidade de controle para determinação do vetor de movimento.

A unidade de controle gera os sinais de controle que garantem o funcionamento síncrono de todos os blocos da unidade operacional. Além disto, ela determina o vetor de movimento em função do sinal NOVO_VM e do seu estado interno, sendo portanto a unidade responsável por gerar o sinal VM que é a saída da arquitetura.

As seções que seguem apresentam os blocos que compõem a unidade operacional e a unidade de controle.

4.3 Interface de Entrada e Saída

A estimação de movimento é realizada a partir da informação de luminância do vídeo. Os dados de luminância possuem caminhos diferentes para os dados de procura e de referência pois pertencem a quadros diferentes, conforme discutido no Capítulo 2. Assim, a interface de E/S é composta pelos blocos interface de E/S para dados de procura e interface de E/S para dados de referência, conforme apresenta a Figura 4.3. As subseções que seguem apresentam estes blocos.

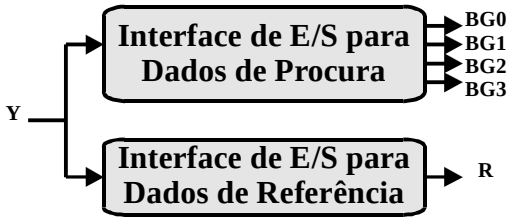

FIGURA 4.3 – Diagrama de blocos da interface de E/S.

4.3.1 Interface de Entrada e Saída para Dados de Procura

A função da interface de E/S para dados de procura é organizar os dados de procura de maneira a viabilizar a máxima utilização da capacidade computacional instalada na matriz de processamento.

Na estimação de movimento, os dados de referência pertencem ao quadro atual, enquanto os dados de procura pertencem a um quadro temporalmente anterior. Assim um dos requisitos da interface de E/S é ser capaz de lidar com uma quantidade de dados superior a um quadro de vídeo. Outro requisito é que, ao mesmo tempo em que novos dados estão chegando e devem ser armazenados, os dados de procura devem estar disponíveis para serem acessados pela matriz de processamento. Em função disso a estratégia adotada foi o uso de uma quantidade duplicada de memórias de porta simples, o que possibilita ler de uma memória enquanto se escreve em outra.

A interface de E/S para dados de procura é composta por memórias em dois níveis de granulação, denominadas memória de quadro e memória de procura. As memórias de quadro são utilizadas para armazenar um quadro completo, enquanto as memórias de procura são utilizadas para armazenar os dados referentes à região sobre a qual está sendo realizada a procura. Deve-se esclarecer que, como as memórias de quadro têm uma capacidade muito grande, estas não foram implementadas em FPGA pois isto seria inviável no dispositivo disponível. Assim, este capítulo apresenta a arquitetura da

interface de E/S supondo a existência física das memórias de quadro, enquanto o Capítulo 5 mostra como a existência destas memórias foi emulada utilizando um computador PC para viabilizar o teste da arquitetura para estimação de movimento implementada em um dispositivo FPGA.

A arquitetura da interface de E/S para dados de procura é composta por quatro memórias de quadro denominadas `MQ0`, `MQ1`, `MQ2` e `MQ3`, cada uma com capacidade de 153.600 *bytes*, o suficiente para armazenar meio quadro. Também é constituída por oito memórias de procura denominadas `MP0`, `MP1`, `MP2`, `MP3`, `MP4`, `MP5`, `MP6` e `MP7`, cada uma com capacidade de 64 *bytes*, o suficiente para armazenar uma região de 8x8 pontos. A Tabela 4.1 resume as características das memórias de quadro e de procura.

TABELA 4.1 – Níveis de memória da interface de E/S para dados de procura.

Características	**Memória de Quadro**	**Memória de Procura**
Capacidade (*bytes*)	153.600	64
Nº de instâncias	4	8
Descrição da capacidade	Meio quadro	Uma região 8x8
Nº de *bits* de endereçamento	18	6

A Figura 4.4 apresenta a arquitetura da interface de E/S para dados de procura. Na parte inferior da figura, a entrada da interface de E/S para dados de procura é o sinal de luminância `Y`. Todas as memórias de quadro são alimentadas pelo sinal `Y`, porém os dados que chegam somente serão armazenados se o respectivo sinal de habilitação para escrita estiver ativo. Estes sinais são denominados `LE_MQ0`, `LE_MQ1`, `LE_MQ2` e `LE_MQ3`. O endereçamento das memórias de quadro é feito através dos sinais `END_MQ0`, `END_MQ1`, `END_MQ2` e `END_MQ3` todos de 18 *bits*. As saídas das memórias de quadro são denominadas `Q_MQ0`, `Q_MQ1`, `Q_MQ2` e `Q_MQ3`.

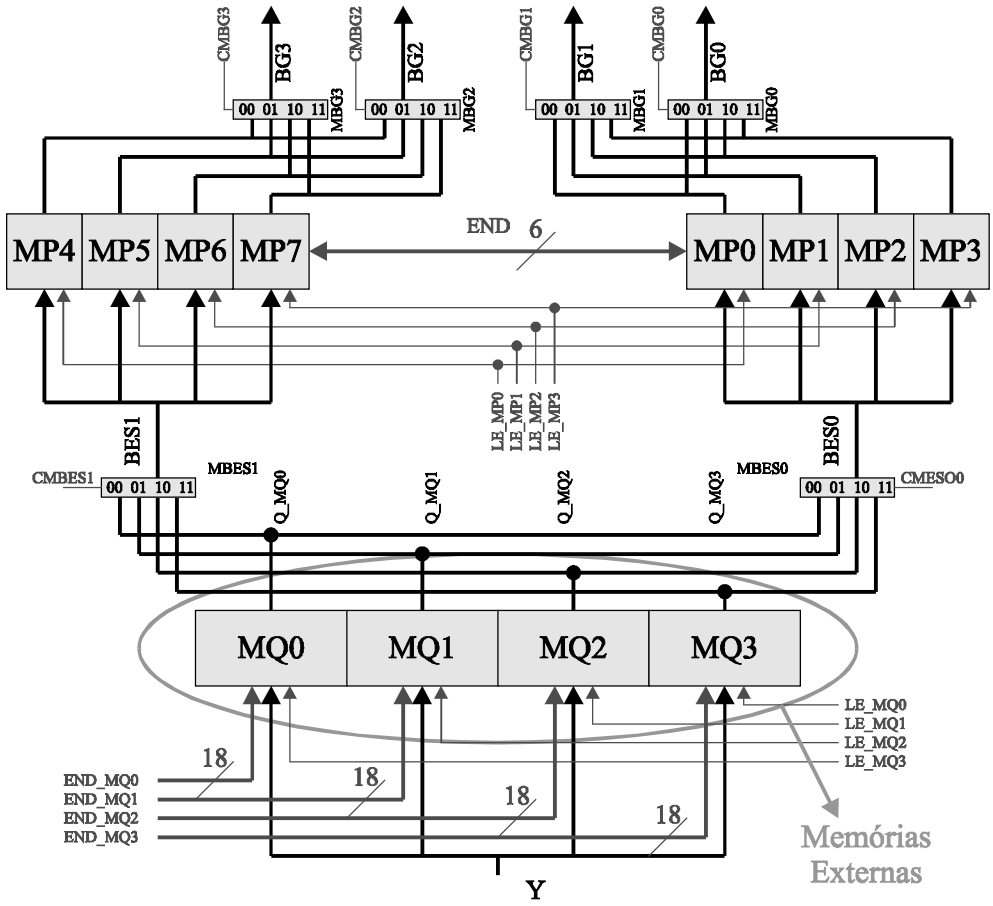

FIGURA 4.4 – Arquitetura da interface de E/S para dados de procura.

A comunicação entre as memórias de quadro e as memórias de procura é coordenada por dois multiplexadores denominados `MBES0` e `MBES1`, controlados pelos sinais `CMBES0` e `CMBES1`. Os multiplexadores `MBES0` e `MBES1` têm as mesmas entradas, que são os sinais `Q_MQ0`, `Q_MQ1`, `Q_MQ2` e `Q_MQ3`, enquanto suas saídas denominam-se `BES0` e `BES1`. Estas saídas alimentam as memórias de procura.

As memórias de procura `MP0`, `MP1`, `MP2` e `MP3` têm como entrada o sinal `BES0`, enquanto as memórias de procura `MP4`, `MP5`, `MP6` e `MP7` têm como entrada o sinal `BES1`. A escrita nas memórias de procura `MP0` e `MP4` é realizada simultaneamente, sendo habilitada pelo sinal `LE_MP0`. Da mesma maneira, a escrita também é realizada simultaneamente nos pares de memórias de procura `MP1` e `MP5`, `MP2` e `MP6`, `MP3` e `MP7`, sendo habilitadas respectivamente pelos sinais `LE_MP1`, `LE_MP2` e `LE_MP3`. Todas as memórias de procura recebem o mesmo sinal de endereçamento, denominado `END`, de 6 *bits*, armazenando assim deferentes dados em posições equivalentes das memórias de procura.

A saída da interface de E/S para dados de procura são os sinais `BG0`, `BG1`, `BG2` e `BG3`. A comunicação entre as memórias de procura e a matriz de processamento é coordenada pelos multiplexadores `MBG0`, `MBG1`, `MBG2` e `MBG3`, que são controlados pelos sinais `CMBG0`, `CMBG1`, `CMBG2` e `CMBG3`. A saída destes multiplexadores são os sinais `BG0`, `BG1`, `BG2` e `BG3`, que são entradas da matriz de processamento.

Uma vez apresentada a arquitetura da interface de E/S para dados de procura e os detalhes de todos os sinais que a compõem, será analisado na seqüência o seu comportamento.

O primeiro quadro que for recebido pela arquitetura será armazenado metade em MQ0 e metade em MQ1, sendo que as primeiras oito linhas serão armazenadas em MQ0, as oito linhas seguintes em MQ1 e assim alternadamente até o final do recebimento deste primeiro quadro. O segundo quadro será armazenado metade em MQ2 e metade em MQ3, sendo as oito primeiras linhas armazenadas em MQ2, as oito linhas seguintes em MQ3 e assim alternadamente até o final do segundo quadro. A Figura 4.5 ilustra a estratégia de armazenamento de dados nas memórias de quadro.

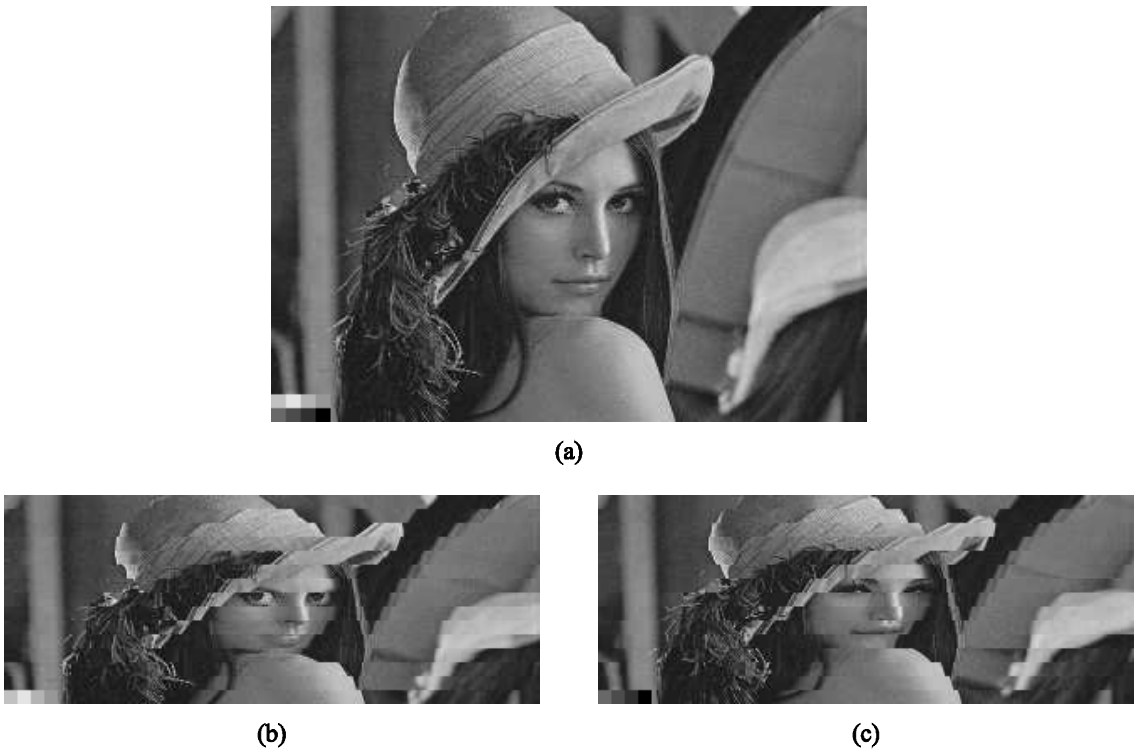

FIGURA 4.5 – (a) Quadro original. (b) Meio quadro armazenado em MQ0.
(c) Meio quadro armazenado em MQ1.

Enquanto MQ2 e MQ3 estão sendo utilizadas para escrita do segundo quadro os dados armazenados em MQ0 e MQ1 estão sendo lidos. Assim, o terceiro quadro poderá ser armazenado em MQ0 e MQ1, uma vez que os dados em MQ0 e MQ1 já foram processados. Os sinais de controle LE_MQ0, LE_MQ1, LE_MQ2 e LE_MQ3, em conjunto com CMBES0 e CMBES1, são responsáveis por determinar a troca do quadro de referência. A unidade de controle gera diferentes padrões para estes sinais dependendo do sinal externo TROCA_MQ, que determina a mudança dos dados de procura. Enquanto não for determinada a troca do quadro de referência todos os quadros que chegam são armazenados no mesmo par de memórias de quadro, por exemplo MQ2 e MQ3.

A cada ciclo de processamento o sinal Y representa um novo dado do vídeo de entrada. Para esclarecer a implementação da interface de E/S para dados de procura é importante acompanhar um dado que passa por esta interface. Para relacionar as operações com o tempo, ou mais precisamente, com o número de ciclos de

processamento que já ocorreram são definidas as variáveis *T*, *offset* e *t*. A relação entre estas variáveis é dada pela equação:

$$T = offset + t \tag{4.2}$$

onde:

- *T* é a quantidade total de ciclos de processamento que já ocorreram desde o recebimento do primeiro dado do primeiro quadro.
- *offset* representa um deslocamento em *T* até um período de interesse.
- *t* é a quantidade de ciclos de processamento que ocorreram desde a última definição da variável *offset*.

Nesta análise, inicialmente *T*, *offset* e *t* são nulos. O primeiro quadro que é recebido pela arquitetura é referenciado como quadro de ordem 0. O próximo quadro recebido é o quadro de ordem 1 e assim sucessivamente. A referência aos pontos da imagem é feita através dos dados de luminância que representam estes pontos. O dado que representa o ponto mais à esquerda da linha inferior é referenciado como dado de ordem 0. Os dados que representam os pontos da imagem são referenciados através de sua ordem, que é incrementada da esquerda para a direita e, ao final de cada linha, de baixo para cima.

Em *t*=0 o dado de ordem 0 do quadro de ordem 0 é escrito no endereço 000000 de `MQ0`; para isto o sinal de controle `LE_MQ0` vale ‘1’, os demais sinais de habilitação para escrita valem ‘0’ e `END_MQ0` vale 000000. Em *t*=1 o dado de ordem 1 é escrito no endereço 000001 de `MQ0`; para isto `LE_MQ0` permanece em ‘1’, os demais sinais de habilitação para escrita permanecem em ‘0’ e o sinal `END_MQ0` é incrementado para 000001. Este ciclo de escrita se repete até *t*=5119, quando o dado de ordem 5119 é escrito no endereço 005119 de `MQ0`. Neste momento, os primeiros 5120 dados, que representam a primeira faixa de oito linhas do quadro de ordem 0, foram escritos em `MQ0`. A próxima faixa de oito linhas será escrita em `MQ1`.

Em *t*=5120 o dado de ordem 5120 do quadro de ordem 0 é escrito no endereço 000000 de `MQ1`; para isto o sinal `LE_MQ1` passa a valer ‘1’, os demais sinais de habilitação para escrita passam a valer ‘0’ e `END_MQ1` passa a valer 000000. Em *t*=5121 o dado de ordem 5121 é escrito no endereço 000001 de `MQ1`; para isto o sinal `LE_MQ1` permanece em ‘1’, os demais sinais para habilitação de escrita permanecem em ‘0’ e o sinal `END_MQ1` passa a valer 000001. Este ciclo de escrita se repete até *t*=10239, quando o dado de ordem 10239 é escrito no endereço 005119 de `MQ1`. Neste momento os 5120 dados que representam a segunda faixa de oito linhas foram escritos em `MQ1`.

Estes ciclos de escritas alternadas em `MQ0` e `MQ1` se repetem até *t*=307199, quando o dado de ordem 307199 é escrito no endereço 153599 de `MQ1`. A Tabela 4.2 apresenta o comportamento dos sinais de endereçamento e habilitação para escrita nas memórias de quadro durante o ciclo de escrita. As memórias `MQ2` e `MQ3` seguem o mesmo esquema de endereçamento, porém seu ciclo de escrita inicia-se em *t*=307200 e encerra-se em *t*=614399, armazenando o quadro de ordem 1.

TABELA 4.2 – Sinais para escrita nas memórias de quadro.

t	END_MQ0	LE_MQ0	END_MQ1	LE_MQ1	END_MQ2	LE_MQ2	END_MQ3	LE_MQ3
0	000000	1	x	0	Ciclo de Leitura			
5119	005119	1	x	0				
5120	x	0	000000	1				
10239	x	0	005119	1				
10240	005120	1	x	0				
307199	x	0	153599	1				
307200	Ciclo de Leitura				000000	1	x	0
312319					005119	1	x	0
312320					x	0	000000	1
317439					x	0	005119	1
317440					005120	1	x	0
614399					x	0	153599	1

Na Tabela 4.2 o valor ‘x’ significa que o valor daquele sinal é irrelevante, ou seja, não influencia o comportamento do circuito naquele momento. Nas tabelas que possuem a coluna *t*, as linhas pontilhadas indicam a existência de ciclos de processamento intermediários que não estão representados.

No momento em que *t*=307200 um quadro completo está armazenado em um par de memórias de quadro, e é iniciado o ciclo de leitura. A leitura das memórias de quadro é realizada em sincronia com a escrita nas memórias de procura, ou seja, os dados são copiados das memórias de quadro para as memórias de procura. O ciclo de leitura é válido a partir de *t*=307200, quando os dados em `MQ0` e `MQ1` são válidos. Para maior clareza na explicação a variável *offset* é redefinida neste ponto do texto passando a valer 307200, que é o período de leitura de um quadro e *t* volta a valer zero. Assim, a relação entre *T* e *t* passa a ser definida pela equação:

$$T = 307200 + t \tag{4.3}$$

Inicialmente a primeira região 8x8 de `MQ0` é copiada para `MP0` enquanto, simultaneamente, a primeira região 8x8 de `MQ1` é copiada para `MP4`. Num segundo passo, a região 8x8 adjacente de `MQ0` é copiada para `MP1` enquanto, simultaneamente, a região 8x8 adjacente de `MQ1` é copiada para `MP5`. Num terceiro passo, a região 8x8 adjacente de `MQ0` é copiada para `MP2` enquanto, simultaneamente, a região 8x8 adjacente de `MQ1` é copiada para `MP6`. Num quarto passo, a região 8x8 adjacente de `MQ0` é copiada para `MP3` enquanto, simultaneamente, a região 8x8 adjacente de `MQ1` é copiada para `MP7`.

A Figura 4.6 ilustra o preenchimento da interface de E/S para dados de procura. Esta figura ilustra o conteúdo das memórias no ciclo de processamento imediatamente anterior ao início do cálculo dos vetores de movimento na matriz de processamento. Percebe-se que durante os ciclos de processamento em que os dados de `MQ0` e `MQ1`

foram copiados para as memórias de procura, de acordo com o descrito acima, as memórias de quadro MQ2 e MQ3 tiveram seu preenchimento iniciado.

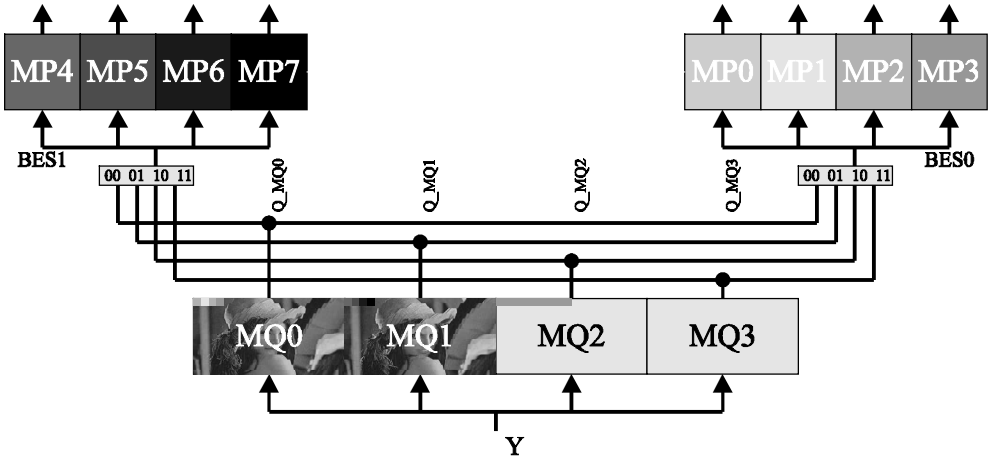

FIGURA 4.6 – Preenchimento da interface de E/S para dados de procura.

Enquanto as memórias MQ0 e MQ1 estiverem sendo utilizadas para procura, o sinal BES0 recebe o sinal Q_MQ0, saída de MQ0, e o sinal BES1 recebe o sinal Q_MQ1, saída de MQ1. Para isto o sinal de controle CMBES0 vale "00" e o sinal CMBES1 vale "01". Esta situação é mantida enquanto o sinal TROCA_MQ valer '0'. Quando o sinal TROCA_MQ passar a valer '1' as memórias MQ2 e MQ3 serão utilizadas para procura; o sinal BES0 receberá o sinal Q_MQ2, saída de MQ2, e o sinal BES1 receberá o sinal Q_MQ3, saída de MQ3. Para isto o sinal de controle CMBES0 vale "10" e o sinal CMBES1 vale "11".

Para esclarecer como os dados são transferidos das memórias de quadro para as memórias de procura é feita na seqüência uma análise temporal. A análise é iniciada em t=0. Conforme apresenta a Tabela 4.3, entre t=0 e t=63 as primeiras regiões 8x8 das memórias de quadro MQ0 e MQ1 são copiadas para as memórias de procura MP0 e MP4. Para isto o sinal END em t=0 vale 00, em t=1 vale 01 e varia seqüencialmente até t=63, quando vale 63. Nestes primeiros 64 ciclos de processamento os sinais LE_MP0 e LE_MP4 valem '1', pois estas memórias estão habilitadas a armazenar dados. Os sinais END_MQ0 e END_MQ1, que durante o ciclo de leitura são iguais, endereçam nos primeiros 8 ciclos de processamento os primeiros 8 dados da primeira linha de MQ0 e MQ1, nos 8 ciclos seguintes os primeiros 8 dados da segunda linha e assim por diante até a oitava linha. Assim, os sinais END_MQ0 e END_MQ1 valem 000000 em t=0 e variam até 000007 em t=7, quando foram copiados os 8 primeiros dados da primeira linha. Para copiar os 8 primeiros dados da segunda linha estes sinais valem 000640 em t=8 e variam até 000647 em t=15. Para copiar a última linha, estes sinais valem 004480 em t=56 e variam até 004487 em t=63.

TABELA 4.3 – Sinais para cópia dos dados das memórias de quadro para as memórias de procura.

t	END	END_MQ0 e END_MQ1	LE_MB0	LE_MB1	LE_MB4	LE_MB5
0	00	000000	1	0	1	0
7	07	000007	1	0	1	0
8	08	000640	1	0	1	0
15	15	000647	1	0	1	0
56	56	004480	1	0	1	0
63	63	004487	1	0	1	0
64	00	000008	0	1	0	1
71	07	000015	0	1	0	1
72	08	000648	0	1	0	1
79	15	000655	0	1	0	1
120	56	004488	0	1	0	1
127	63	004495	0	1	0	1

Nos sessenta e quatro ciclos de processamento seguintes, entre t=64 e t=127, as segundas regiões 8x8 são copiadas para as memórias de procura MP1 e MP5. Para tal o sinal END se repete e os sinais LE_MP1 e LE_MP5 passam a valer ‘1’. Os sinais END_MQ0 e END_MQ1 têm o mesmo comportamento descrito para as primeiras regiões 8x8 e seus valores são explicitados na Tabela 4.3. Nos próximos 64 ciclos de processamento entre t=128 e t=191, as terceiras regiões 8x8 são copiadas para MP2 e MP6; para tal os sinais END_MQ0, END_MQ1 e END têm os mesmo comportamento já descrito e os sinais LE_MP2 e LE_MP6 passam a valer ‘1’. Nos 64 ciclos de processamento seguintes, entre t=192 e t=255, as quartas regiões 8x8 são copiadas para as memórias de procura MP3 e MP7; para tal os sinais de endereçamento têm comportamento similar e o sinal LE_MP3 passa a valer ‘1’. A Tabela 4.4 explicita o comportamento dos sinais de habilitação para escrita nas memórias de procura.

TABELA 4.4 – Sinais de habilitação para escrita nas memórias de procura.

t	LE_MP0	LE_MP1	LE_MP2	LE_MP3
0 - 63	1	0	0	0
64 - 127	0	1	0	0
128 - 191	0	0	1	0
192 - 255	0	0	0	1

Este ciclo de cópia de dados se repete continuamente a cada intervalo de 256 ciclos de processamento. Uma diferença que ocorre é que as memórias MQ0 e MQ1, de acordo com o sinal TROCA_MQ, podem ser substituídas pelas memórias MQ2 e MQ3. Isto só pode ocorrer em intervalos de 307200 ciclos de processamento, que correspondem ao processamento de um quadro completo. Nesta situação os sinais

END_MQ2 e END_MQ3 passam a ter o comportamento de ciclo de leitura descrito acima para os sinais END_MQ0 e END_MQ1. A unidade de controle é responsável por esta mudança em função do sinal TROCA_MQ.

A partir de *t*=128 existem quatro memórias de procura preenchidas e o processamento pode iniciar. Para ser coerente com a análise da matriz de processamento que será feita na Seção 4.4 a variável *offset* é incrementada em 128, de modo que a relação entre *T* e *t* passa a ser dada pela equação:

$$T = 307200 + 128 + t \quad \square \quad T = 307328 + t \tag{4.3}$$

Para iniciar o processamento as memórias de procura devem ser endereçadas e os multiplexadores MBG00, MBG01, MBG10 e MBG11 controlados de maneira a escrever nos barramentos BG00, BG01, BG10 e BG11 os dados esperados pela matriz de processamento. Como os barramentos globais representam a fronteira entre a interface de E/S para dados de procura e a matriz de processamento, estes sinais são os últimos abordados nesta seção.

A Figura 4.7 ilustra o conteúdo das memórias de procura durante seis períodos de 64 ciclos de processamento. Em cada um destes períodos o sinal END varia de 00 a 64. A Figura 4.7a mostra o período anterior ao início do processamento, entre *t*=-128 e *t*=-65, quando MP0 e MP4 estavam sendo preenchidas. A Figura 4.7b também mostra um período anterior ao início do processamento, entre *t*=-64 e *t*=-1. Durante este período, MP0 e MP4 estão preenchidas enquanto MP1 e MP5 estão sendo preenchidas.

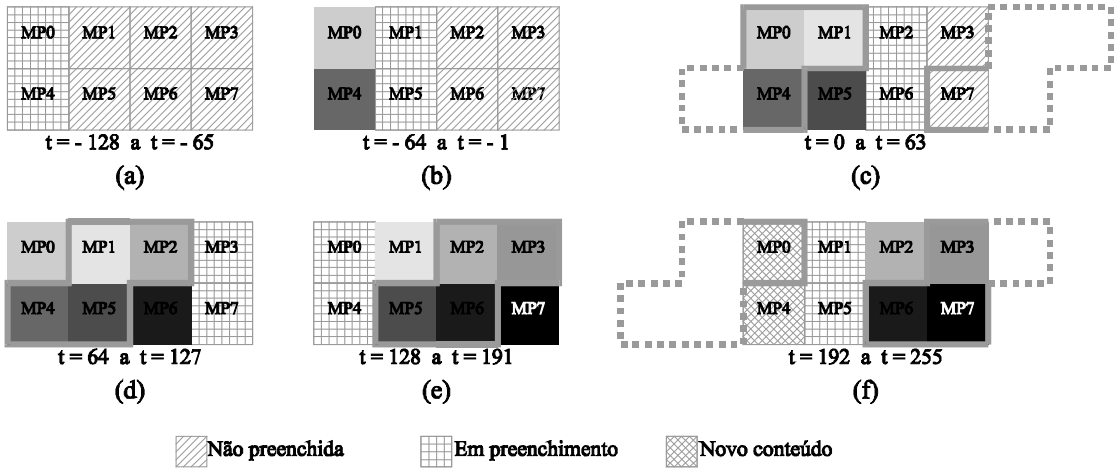

FIGURA 4.7 – Conteúdo das memórias de procura.

A Figura 4.7c mostra o período entre *t*=0 e *t*=63 no qual o processamento é iniciado. Durante este período MP2 e MP6 estão sendo preenchidas enquanto MP0, MP1, MP7 e MP4 alimentam os barramentos globais (envoltas pela linha larga). De fato, os dados de MP7 e MP4 não estão ainda sendo utilizados conforme será mostrado na Seção 4.4, pois MP7 sequer foi preenchida. Porém, em uma situação de regime, os dados das memórias MP0, MP1, MP7 e MP4 seriam válidos e todas estariam sendo utilizadas.

A Figura 4.7d mostra o período entre *t*=64 e *t*=127 no qual os barramentos globais são alimentados por MP1, MP2, MP4 e MP5. A Figura 4.7e mostra o período entre *t*=128 e *t*=191 no qual os barramentos globais são alimentados por MP2, MP3, MP5 e MP6. Assim, a linha larga salientando as memórias que alimentam os barramentos globais tem sempre o mesmo formato e se desloca para a direita a cada 64

ciclos de processamento. As memórias em preenchimento estão sempre um passo à frente deste deslocamento, uma vez que a escrita inicia 128 ciclos de processamento antes de sua utilização.

O controle deste mecanismo para alimentação dos barramentos globais é feito pelos sinais de controle dos barramentos globais `MBG00`, `MBG01`, `MBG10` e `MBG11`. A Tabela 4.5 apresenta o comportamento destes sinais. No cabeçalho desta tabela, na coluna "Escrita" aparecem os nomes dos barramentos de entrada e saída mostrando a origem dos dados, enquanto na coluna "Leitura" aparecem os nomes dos barramentos globais mostrando o destino dos dados.

TABELA 4.5 – Controle dos barramentos globais.

t	CMBG0	CMBG1	CMBG2	CMBG3	Leitura				LE_MP0	LE_MP1	LE_MP2	LE_MP3	Escrita	
					BG0	BG1	BG2	BG3					BES0	BES1
0-63	00	01	11	00	MP0	MP1	MP7	MP4	1	0	0	0	MP2	MP6
64-127	01	10	00	01	MP1	MP2	MP4	MP5	0	1	0	0	MP3	MP7
128-191	10	11	01	10	MP2	MP3	MP5	MP6	0	0	1	0	MP0	MP4
192-255	11	00	01	11	MP3	MP0	MP6	MP7	0	0	0	1	MP1	MP5

Analisando a Tabela 4.5 percebe-se que os barramentos globais `BG0` e `BG1` acessam apenas as memórias de procura `MP0`, `MP1`, `MP2` e `MP3`, que por sua vez são escritas apenas pelas memórias de quadro `MQ0` e `MQ2`, que representam as faixas pares. Da mesma forma, os barramentos globais `BG2` e `BG3` só acessam as memórias de procura `MP4`, `MP5`, `MP6` e `MP7`, que por sua vez só são escritas pelas memórias de quadro `MQ1` e `MQ3`, que representam as faixas ímpares.

4.3.2 Interface de Entrada e Saída para Dados de Referência

A função da interface de E/S para dados de referência é organizar os dados de referência em regiões 8x8, o que pode ser entendida como uma conversão de linha para bloco. A estratégia de implementação é similar à abordagem adotada no projeto da interface de E/S para dados de procura.

Como os dados de referência, que pertencem ao quadro atual, são recebidos linha a linha, é necessário armazenar uma faixa de oito linhas para viabilizar a leitura dos dados em forma de blocos. Para isto a interface de E/S para os dados de referência é composta por um par de memórias de faixa denominadas `MF0` e `MF1`, com capacidade de 5120 *bytes*, o suficiente para armazenar uma faixa de oito linhas de um quadro; e também por um par de memórias de referência com capacidade de 64 *bytes*, o suficiente para armazenar uma região 8x8, denominadas `MR0` e `MR1`. A Tabela 4.6 resume as características das memórias de faixa e de referência.

TABELA 4.6 – Níveis de memória da interface de E/S para dados de referência.

Características	Memória de Faixa	Memória de Referência
Capacidade (*bytes*)	5.120	64
Nº de blocos projetados	2	2
Descrição da capacidade	Faixa de 8 linhas	Uma região 8x8
Nº de *bits* de endereçamento	13	6

Após receber o primeiro quadro por completo, a arquitetura para estimação de movimento pode iniciar o cálculo dos vetores de movimento referentes ao quadro de ordem 1. Embora a interface de E/S tenha um funcionamento contínuo, os dados são válidos apenas a partir de T=307200, ou t=0, quando os dados de entrada estão representando o quadro de ordem 1. Assim, após o recebimento da primeira faixa de oito linhas do quadro de ordem 1, quando `MF0` está preenchida, os dados que chegam passam a ser escritos em `MF1`. Enquanto a escrita é realizada em `MF1` os dados de `MF0` são movidos em forma de bloco para `MR0` e `MR1` alternadamente.

A Figura 4.8 apresenta a arquitetura da interface de E/S para dados de referência. A entrada desta arquitetura é o sinal de luminância (`Y`). Este sinal alimenta diretamente as memórias de faixa `MF0` e `MF1`. A habilitação para escrita nas memórias de faixa é determinada pelos sinais `LE_MF0` e `LE_MF1`. O endereçamento é realizado pelos sinais `END_MF0` e `END_MF1` de 13 *bits*. As saídas das memórias de faixa são denominadas `Q_MF0` e `Q_MF1`.

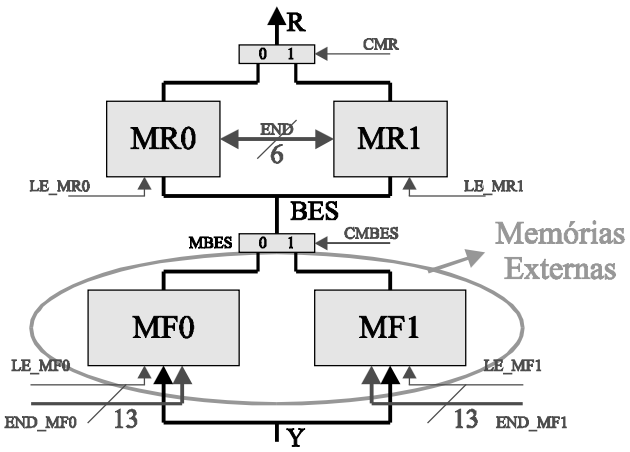

FIGURA 4.8 – Arquitetura da interface de E/S para dados de referência.

A comunicação entre as memórias de faixa e as memórias de referência é coordenada pelo multiplexador `MBES` que é controlado pelo sinal `CMBES`. As entradas do multiplexador `MBES` são os sinais `Q_MF0` e `Q_MF1`, e sua saída é denominada `BES`. As memórias de referência `MR0` e `MR1` têm como entrada o sinal `BES`. A habilitação para escrita nas memórias de referência é determinada pelos sinais `LE_MR0` e `LE_MR1`. O endereçamento é realizado pelo sinal `END` de 6 *bits*, o mesmo sinal que endereça as memórias de bloco da interface de E/S para dados de procura.

A Tabela 4.7 apresenta os sinais de endereçamento e habilitação para escrita nas memórias de faixa. Em t=0 o dado de ordem 0 do quadro de ordem 1 é escrito no

endereço 0000 de MF0; para isto o sinal LE_MF0 deve valer ‘1’ e o sinal END_MF0 deve valer 0000. Em *t*=1 o dado de ordem 1 é escrito no endereço 0001 de MF0; para isto o sinal END_MF0 passa a valer 0001 e o sinal LE_MF0 é mantido em ‘1’. Este ciclo de escrita se repete até *t*=5119, quando o último dado da oitava linha é escrito na posição 5119 de MF0. Neste momento a primeira faixa de oito linhas foi escrita em MF0; a próxima faixa será escrita em MF1.

TABELA 4.7 – Sinais para escrita nas memórias de faixa.

t	END_MF0	LE_MF0	END_MF1	LE_MF1	CBIO
0	0000	1	Ciclo de Leitura		1
5119	5119	1			1
5120	Ciclo de Leitura		0000	1	0
10239			5119	1	0

Em *t*=5120 o dado de ordem 0 da nona linha é escrito na posição 0000 de MF1; para isto o sinal END_MF1 deve valer 0000 e LE_MF1 passa a valer ‘1’, enquanto LE_MF0 passa a valer ‘0’. O ciclo de escrita se repete para MF1 até *t*=10239, quando o último dado da décima quinta linha é escrito na posição 5119 de MF1; para isto END_MF1 deve valer 5119 e LE_MF1 é mantido em ‘1’. Neste momento a segunda faixa de oito linhas foi escrita em MF1. A terceira faixa será escrita em MF0 sobrescrevendo a primeira faixa que, neste momento, já foi utilizada pela matriz de processamento. Este ciclo de escritas alternadas se repete ininterruptamente. Quando o quadro de ordem 1 terminar o ciclo continua, sem alterações para o quadro de ordem 2 e assim por diante.

O comportamento do sinal CMBES, apresentado na Tabela 4.7, mostra que o sinal BES recebe a saída da memória de faixa que está em ciclo de leitura, de maneira que a entrada das memórias de referência é sempre a saída da memória que está sendo lida. Os sinais de habilitação para escrita determinam em qual das memórias de referência estes dados serão escritos. A estratégia adotada consiste em escrever regiões 8x8 em MR0 e MR1 alternadamente.

Considere *T*=5120+*t*. A Tabela 4.8 apresenta os sinais de endereçamento e habilitação para escrita nas memórias de referência. Após MF0 receber a primeira faixa de oito linhas, os dados começam a ser transferidos de MF0 para MR1. A partir de *t*=0 os dados dos endereços entre 0000 e 0007 de MF0, que representam a primeira linha da primeira região 8x8, são transferidos para MR0; para isto o sinal END_MF0 varia de 0000 a 0007, o sinal END varia de 00 a 07, o sinal LE_MR0 vale ‘1’ e o sinal LE_MR1 vale ‘0’. A partir de *t*=8 os dados dos endereços entre 0640 e 0647 de MF0, que representam a segunda linha da primeira região 8x8, são transferidos para MR0; para isto o sinal END_MF0 varia de 0640 a 0647, o sinal END varia de 08 a 15, o sinal LE_MR0 permanece em ‘1’ e o sinal LE_MR1 permanece em ‘0’. Este ciclo de transferência de dados se repete até que, a partir de *t*=56, os dados dos endereços entre 4480 e 4487 de MF0, que representam a oitava linha da primeira região 8x8, são transferidos para MR0; para isto o sinal END_MF0 varia de 4480 a 4487, o sinal END varia de 56 a 63, o sinal LE_MR0 permanece em ‘1’ e o sinal LE_MR1 permanece em ‘0’.

TABELA 4.8 – Sinais para cópia dos dados das memórias de faixa para as memórias de referência.

t	END_MF0	END_MF1	END	LE_MR0	LE_MR1
0	0000	0000	00	1	0
7	0007	0007	07	1	0
8	0640	0008	08	1	0
15	0647	0015	15	1	0
56	4480	0056	56	1	0
63	4487	0063	63	1	0
64	0008	0064	00	0	1
71	0015	0071	07	0	1
72	0648	0072	08	0	1
79	0655	0079	15	0	1
120	4488	0120	56	0	1
127	4495	0127	63	0	1

A partir de *t*=64 a segunda região 8x8 passa a ser transferido de MF0 para MR1. A partir de *t*=64 os dados dos endereços entre 0008 e 0015 de MF0, que representam a primeira linha da segunda região 8x8, são transferidos para MR1; para isto o sinal END_MF0 varia de 0008 a 0015, o sinal END varia de 00 a 07 , LE_MR0 vale ‘0’ e o sinal LE_MR1 vale ‘1’. A partir de *t*=72 os dados dos endereços entre 0648 e 0655 de MF0, que representam a segunda linha da segunda região 8x8, são transferidos para MR1, para isto o sinal END_MF0 varia de 0648 a 0655, o sinal END varia de 08 a 15, o sinal LE_MR0 permanece em ‘01’ e o sinal LE_MR1 permanece em ‘1’. Este ciclo de transferência de dados se repete até que, a partir de *t*=120, os dados dos endereços entre 4488 e 4495 de MF0, que representam a oitava linha da segunda região 8x8, são transferidos para MR1; para isto o sinal END_MF0 varia de 4488 a 4495, o sinal END varia de 56 a 63, o sinal LE_MR0 permanece em ‘0’ e o sinal LE_MR1 permanece em ‘1’.

A leitura dos dados nas memórias de referência só representa dados válidos a partir de *t*=64, quando é iniciada a leitura de MR0. Para isto o sinal CMR deve valer ‘0’. As memórias de referência e de procura são endereçadas pelo sinal END. Para *t*=64 END já foi definido na Tabela 4.8 e vale 0000, de maneira que primeiro dado de MR0 é lido em *t*=64. Em *t*=65 o segundo dado de MR0 é lido e escrito no barramento R (vide Figura 4.8). Finalmente, em *t*=127 é lido o último dado de MR0. A partir de *t*=128 é iniciada a leitura de MR1, MR0 inicia um novo ciclo de escrita e o sinal CMR passa a valer ‘1’.

4.4 Matriz de Processamento

A função da matriz de processamento é realizar as operações de soma das diferenças absolutas para todas as hipóteses de vetores de movimento, e entregar serialmente estes resultados para a unidade de comparação. A matriz de processamento tem como entrada o sinal de dados de referência R e sinais de dados de procura BG0,

BG1, BG2 e BG3, todos provenientes da interface de E/S. Sua saída é o sinal SAD que, a cada ciclo de processamento, representa a SAD referente a uma hipótese de vetor de movimento.

A matriz de processamento é essencialmente composta por elementos de processamento. A função de um elemento de processamento é calcular a SAD para uma hipótese de vetor de movimento. A função dos demais elementos lógicos que compõem a matriz de processamento se restringe a alimentar de dados os elementos de processamento e conduzir coordenadamente os resultados dos elementos de processamento à saída SAD.

O número de elementos de processamento implementados corresponde ao número de hipóteses de vetor de movimento consideradas. Para uma região de referência de 8x8 pontos e uma região de procura de 15x15 pontos, o número de hipóteses de vetor de movimento é sessenta e quatro. Portanto, a matriz de processamento é composta por sessenta e quatro elementos de processamento.

Antes de apresentar a matriz de processamento como um todo é necessário apresentar individualmente a arquitetura de um elemento de processamento. Cada elemento de processamento tem como entrada os sinais de procura B0 e B1 e também o sinal de referência Ri. Sua saída é o sinal de referência Ro e o sinal ACC que representa a acumulação de diferenças realizada pelo elemento de processamento. A arquitetura de um elemento de processamento é apresentada na Figura 4.9.

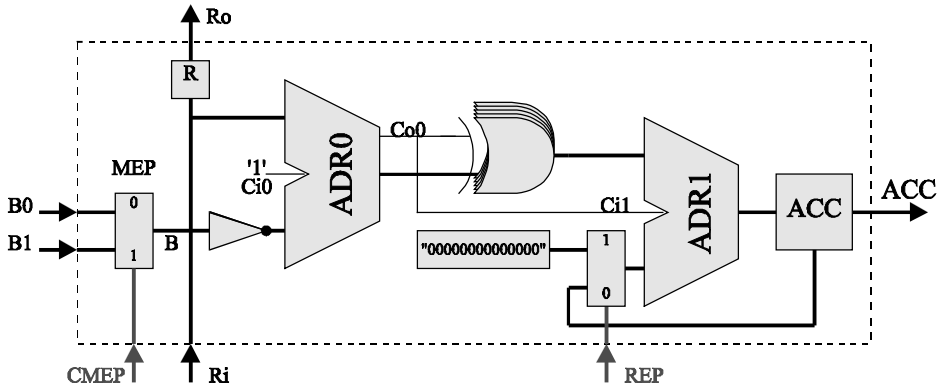

FIGURA 4.9 – Arquitetura de um elemento de processamento.

A entrada Ri alimenta o somador ADR0 e o registrador interno R. Este registrador armazena o dado de referência Ri que, no ciclo de processamento seguinte, será disponibilizado na saída Ro. A existência do registrador R permite que a saída Ro de um elemento de processamento alimente a entrada Ri de outro elemento de processamento.

As entradas B0 e B1 são ligadas à entrado do multiplexador MEP que, através do sinal de controle CMEP, seleciona qual entrada passará à saída B. A existência de duas entradas de dados de procura permite alimentar cada elemento de processamento por dois barramentos. Isto é fundamental para alimentação de dados dos elementos de processamento, conforme será apresentado ainda nesta seção.

O somador ADR0 tem largura de dados de 8 *bits*. Este somador é utilizado para implementar a função subtração para calcular a diferença entre os sinais Ri e B. A diferença entre estes dois operandos pode ser expressa como a soma do operando Ri com o complemento de dois do operando B [GAJ 97]. Como o complemento de dois é

igual ao inverso do sinal adicionado de um, o operando B passa por um inversor e, na entrada Ci0 do somador ADR0, é escrito o valor lógico ‘1’.

O resultado do somador ADR0, que representa a diferença entre os sinais Ri e B, deve ser acumulado em valor absoluto para integrar a SAD que é armazenada no registrador ACC. O somador ADR1 calcula a soma do valor armazenado no registrador ACC com valor absoluto do resultado do somador ADR0. O cálculo do valor absoluto do resultado de ADR0 é feito através de uma lógica combinacional baseada em portas XOR, comumente conhecida como inversor controlado [GAJ 97]. A Figura 4.10 apresenta o inversor controlado utilizado no cálculo do valor absoluto.

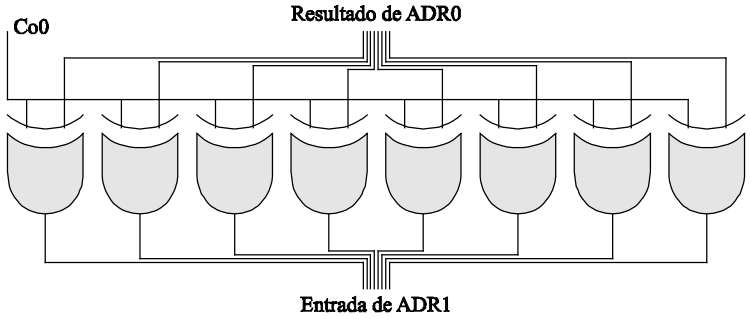

FIGURA 4.10 – Inversor controlado utilizado no cálculo do valor absoluto.

Se o resultado do somador ADR0 for positivo, ele é diretamente adicionado ao valor armazenado em ACC. Caso este resultado seja negativo, ele é invertido e o valor lógico ‘1’ deve ser escrito na entrada Ci1. O resultado de ADR0 é a entrada de um conjunto de oito portas XOR. O sinal Co0 é ligado à uma das entradas de todas as portas XOR na Figura 4.10. Considerando a tabela verdade de uma porta XOR apresentada na Tabela 4.9 [GAJ 97], percebe-se que o sinal Co0 controla a inversão do resultado de ADR0, no sentido de que os sinais de saída das portas XOR serão o inverso da entrada quando Co0 vale ‘1’ e serão iguais à entrada quando Co0 vale ‘0’. O sinal Co0 também está ligado à entrada Ci1 de ADR1; assim, quando a inversão for realizada, o valor ‘1’ é automaticamente escrito em Ci1.

TABELA 4.9 – Tabela verdade de uma porta XOR.

Co0	Resultado de ADR0	Entrada de ADR1
0	0	0	Igual a ADR0
0	1	1	
1	0	1	Inverso de ADR0
1	1	0	

O resultado do somador ADR1, que representa a SAD, é armazenado no registrador ACC, realimentando assim o processo de acumulação. O somador ADR1 e o registrador ACC foram projetados com largura de dados capaz de representar 64 vezes a maior diferença entre Ri e B. Para cada *bit* adicional a capacidade de representação é dobrada, assim para multiplicar a capacidade de representação por 64 (2^6) é necessário adicionar 6 *bits*. Como a arquitetura para estimação de movimento foi projetada para

representação em 8 *bits*, os elementos lógicos ADR1 e ACC foram implementados com 14 *bits*. Os 8 *bits* de resultado de ADR0 são concatenados com zeros para compor a entrada de ADR1, que possui 14 *bits*.

O acumulador ACC pode ser zerado através do sinal de controle REP. Quando este sinal recebe o valor lógico ‘1’ o valor zero é escrito na entrada de ADR1, de forma que o próximo valor a ser acumulado em ACC é a diferença entre Ri e B. Esta lógica garante que ACC seja zerado sem perder um ciclo de processamento para isto. Assim, no primeiro ciclo de processamento do cálculo da SAD para uma nova hipótese o sinal REP deve valer ‘1’ para garantir a validade dos dados acumulados em ACC.

Os elementos da matriz de processamento estão dispostos em oito linhas compostas por oito elementos de processamento cada uma. Estes elementos são denominados EP00, EP01, ..., EP07; EP10, EP11, ..., EP17; ...; EP70, EP71, ..., EP77. Na nomenclatura dos elementos de processamento o primeiro índice representa a linha e o segundo a coluna que este elemento ocupa na matriz de processamento.

A Figura 4.11 apresenta a arquitetura da matriz de processamento. A entrada R da matriz de processamento alimenta a entrada Ri do elemento de processamento EP00. A saída Ro de EP00 alimenta a entrada Ri de EP01 e assim, sucessivamente, a saída Ro de cada elemento de processamento alimenta a entrada Ri do elemento de processamento seguinte. A saída Ro de EP77 não é utilizada. Esta ligação em cadeia dos elementos de processamento é denominada *pipeline* da matriz de processamento.

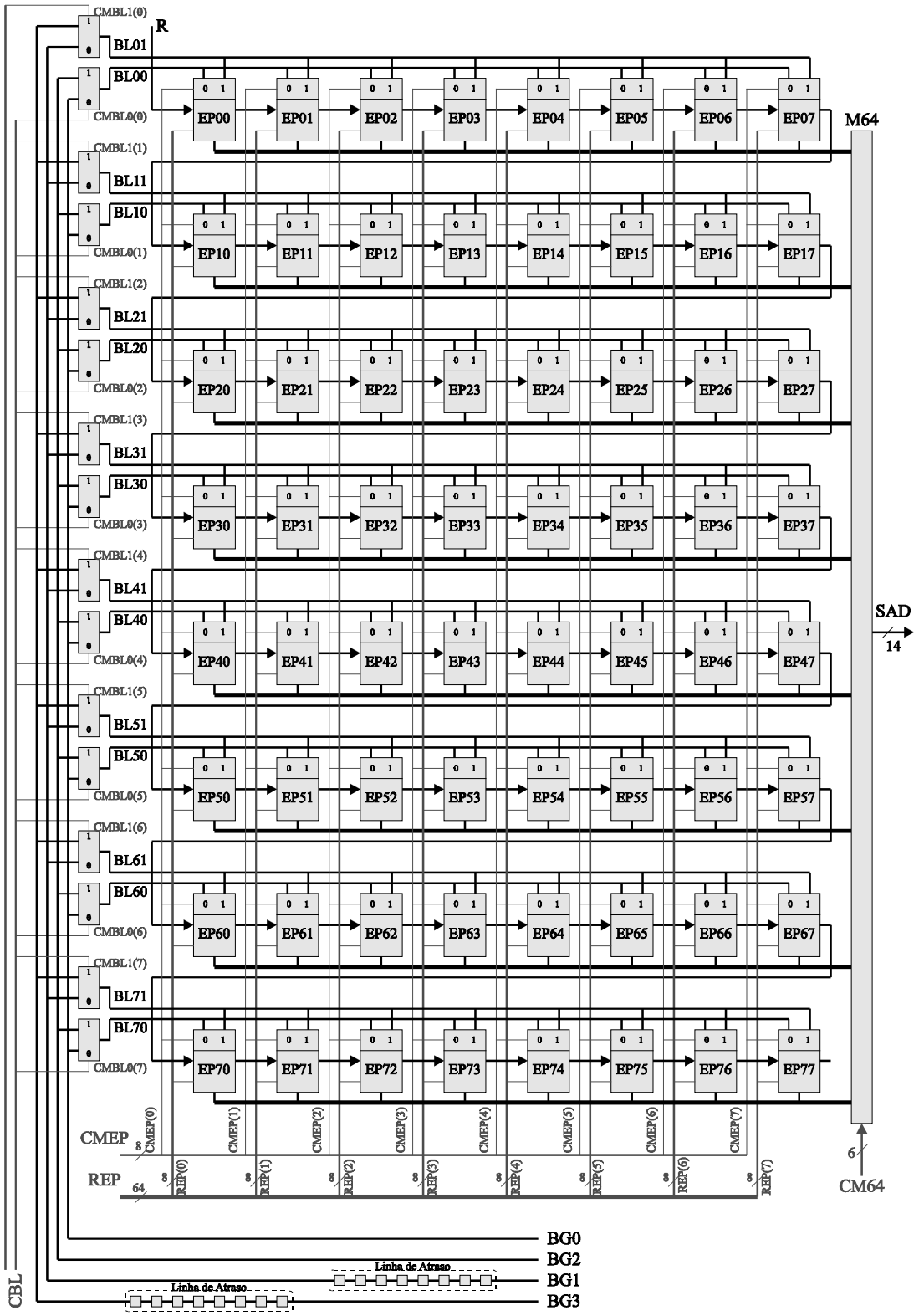

FIGURA 4.11 – Arquitetura da matriz de processamento.

Cada uma das oito linhas da matriz de processamento é alimentada por dois barramentos locais. Os barramentos locais são denominados `BL00`, `BL01`, `BL10`, `BL11`, `BL20`, `BL21`, ..., `BL70` e `BL71`. Estes barramentos são alimentados pelos quatros barramentos globais através dos multiplexadores `MBL00`, `MBL01`, `MBL10`, `MBL11`, `MBL20`, `MBL21`, ..., `MBL70` e `MBL71`, controlados pelos sinais `CMBL00`,

CMBL01, CMBL10, CMBL11, CMBL20, CMBL21, ..., CMBL70 e CMBL71, respectivamente. Os barramentos locais de ordem 0, que são BL00, BL10, ..., BL70, são alimentados pelos barramentos globais BG0 ou BG2, enquanto os barramentos locais de ordem 1, que são BL01, BL11, ..., BL71, são alimentados pelos barramentos globais BG1 ou BG3.

A entrada B0 de cada elemento de processamento está ligada a um barramento local de ordem 0, enquanto a entrada B1 está ligada ao barramento local de ordem 1. Para o elemento de processamento EP54, por exemplo, B0 está ligada a BL50 e B1 a BL51.

As saídas ACC de cada elemento de processamento são ligadas à saída SAD da matriz de processamento através do multiplexador M64, que seleciona através do sinal CM64 o resultado de um elemento a cada ciclo de processamento.

Os sinais de controle da matriz de processamento são: REP, CMEP, CMBL e CM64. Estes sinais são conjuntos de *bits*. A referência a um *bit* específico de um sinal é feita indicando a ordem deste *bit* entre parênteses. Na Figura 4.11 a quantidade total de *bits* está representada entre parênteses.

O sinal REP, cuja função é zerar os elementos de processamento, possui 64 *bits*, sendo cada *bit* ligado ao controle REP de cada elemento de processamento. O sinal REP(07) por exemplo, está ligado ao elemento EP07, REP(08) ao EP10 e assim por diante. O sinal CMEP, cuja função é controlar os multiplexadores dos elementos de processamento, possui 8 *bits*, sendo cada *bit* ligado ao controle CMEP dos elementos de uma coluna da matriz. O sinal CMEP(2), por exemplo, está ligado aos elementos EPi2, i=0...7. O sinal CMBL, cuja função é controlar os multiplexadores dos barramentos locais, possui 8 *bits*, sendo cada *bit* ligado aos dois multiplexadores de um barramento local. O sinal CMBL(2), por exemplo, está ligado aos multiplexadores MBL20 e MBL21. O sinal CM64 possui 6 *bits* e controla o multiplexador M64.

A divisão das tarefas computacionais entre os elementos de processamento é ilustrada na Figura 4.12. Cada elemento EPij acumula a SAD referente à região de procura cujo primeiro ponto tem coordenadas (i, j) e o último elemento tem coordenadas (i+7, j+7) na Figura 4.12. O elemento de processamento EP77, por exemplo, calcula a SAD para a região cujo primeiro elemento tem coordenadas (7,7) e o último elemento tem coordenadas (14, 14).

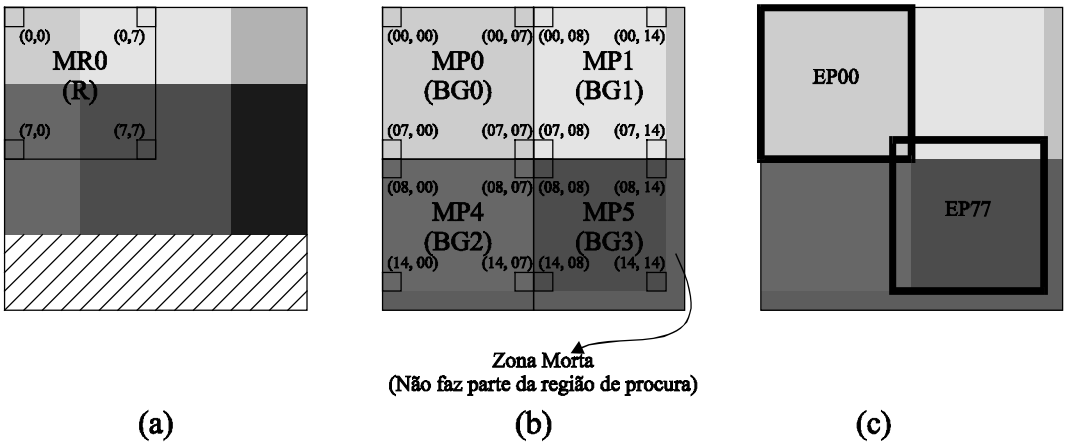

FIGURA 4.12 – (a) região de referência. (b) região de procura.
(c) Regiões consideradas por EP00 e EP77 para cálculo da SAD.

Na Figura 4.12 as regiões coloridas representam o conteúdo das memórias durante os primeiros sessenta e quatro ciclos de processamento. As coordenadas de alguns pontos são explicitadas entre parênteses. Para a região de procura as coordenadas são indexadas tendo como origem o endereço 00 da memória de procura que está ligada ao barramento BG0 que, para os primeiros sessenta e quatro ciclos de processamento representados na Figura 4.12b, é MP0. Pontos da região de procura são referenciados através da notação P(xx,yy) onde o índice xx representa a coordenada horizontal, yy representa a coordenada vertical do ponto com xx e yy variando entre 00 a 14. Linhas ou colunas podem ser referenciadas definindo o índice de linha ou de coluna respectivamente, sem definir o valor da outra coordenada. A notação P(00,yy), por exemplo, é utilizada para fazer referência à linha de ordem zero da região de procura. Estas mesmas regras são utilizadas para pontos da região de referência que são referenciados através da notação R(x,y), com x e y variando de 0 a 7.

A análise da ordem como as operações são realizadas na matriz de processamento deve começar pelo sinal REP pois este inicializa os elementos de processamento. Para que as operações realizadas em um elemento sejam válidas é necessário que o sinal REP valha ‘1’ no primeiro ciclo de processamento válido para este elemento de processamento. Uma vez que as operações na matriz de processamento ocorrem em *pipeline*, cada elemento de processamento inicia as operações um ciclo de processamento atrasado com relação ao elemento de processamento anterior. Assim, em t=0 o sinal REP(00) vale ‘1’ e os demais sinais REP valem ‘0’, inicializando assim o elemento EP00. Em t=1 o sinal REP(01) vale ‘1’, em t=2 REP(02) vale ‘1’ e assim por diante até que, em t=63, REP(63) vale ‘1’, inicializando o elemento EP77.

Durante os primeiros oito ciclos de processamento, entre t=0 e t=7, o sinal CMBL0 vale “01111111” de forma que BL00 recebe BG0. Em t=0 o elemento EP00 acumula a diferença entre R(0,0) e P(00,00). Para isto, o sinal END da interface de E/S vale 00 e CMEP vale “01111111”. Em t=1 o elemento EP00 acumula a diferença entre R(0,1) e P(00,01) enquanto EP01 acumula a diferença entre R(0,0) e P(00,01). Para isto o sinal END vale 01, CMEP vale “00111111” e o valor de R(0,0) que estava em EP00 passou para EP01. Em t=7 o elemento EP00 acumula a diferença entre R(0,7) e P(00,07), EP01 acumula a diferença entre R(0,6) e P(00,07) enquanto EP07 acumula a diferença entre R(0,0) e P(00,07). Para

isto o sinal END vale 07, CMEP vale “00000000” e o dado R(0,0) já passou por todos os elementos de processamento EP0j até atingir EP07. A Tabela 4.10 apresenta as operações que são realizadas em cada elemento de processamento durante o intervalo entre *t*=0 e *t*=7, bem como o comportamento dos sinais END, CMBL0, CLB1 e CMEP para viabilizar estas operações.

TABELA 4.10 – Operações realizadas no intervalo entre *t*=0 e *t*=7.

t	CMBL0	CMBL1	CMEP	END	EP00		EP01		EP02		EP03		EP04		EP05		EP06		EP07	
					R	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P
0	01111111	00000000	01111111	00	0,0	00,00	7,7	07,08	7,6	07,08	7,5	07,08	7,4	07,08	7,3	07,08	7,2	07,08	7,1	07,08
1			00111111	01	0,1	00,01	0,0	00,01	7,7	07,09	7,6	07,09	7,5	07,09	7,4	07,09	7,3	07,09	7,2	07,09
2			00011111	02	0,2	00,02	0,1	00,02	0,0	00,02	7,7	07,10	7,6	07,10	7,5	07,10	7,4	07,10	7,3	07,10
3			00001111	03	0,3	00,03	0,2	00,03	0,1	00,03	0,0	00,03	7,7	07,11	7,6	07,11	7,5	07,11	7,4	07,11
4			00000111	04	0,4	00,04	0,3	00,04	0,2	00,04	0,1	00,04	0,0	00,04	7,7	07,12	7,6	07,12	7,5	07,12
5			00000011	05	0,5	00,05	0,4	00,05	0,3	00,05	0,2	00,05	0,1	00,05	0,0	00,05	7,7	07,13	7,6	07,13
6			00000001	06	0,6	00,06	0,5	00,06	0,4	00,06	0,3	00,06	0,2	00,06	0,1	00,06	0,0	00,06	7,7	07,14
7			00000000	07	0,7	00,07	0,6	00,07	0,5	00,07	0,4	00,07	0,3	00,07	0,2	00,07	0,1	00,07	0,0	00,07

O sinal CMEP se repete em períodos de oito ciclos de processamento. Nos primeiros oito ciclos de processamento os valores ‘1’ do sinal CMEP não são relevantes para esta análise, pois ocorrem sempre antes da inicialização do elemento de processamento que está sendo inserido no *pipeline* (aquele que recebe em sua entrada REP o valor ‘1’). Nos oito ciclos de processamento seguintes estes valores ‘1’ do sinal CMEP são responsáveis por selecionar o sinal de BG1 para os elementos que utilizam este sinal. O sinal CMEP tem um papel de endereçamento horizontal, ou seja, este sinal trabalha selecionando os dados das linhas que chegam através dos barramentos locais.

Nos oito ciclos de processamento seguintes, entre *t*=8 e *t*=15, o sinal CMBL0 passa a valer “00111111” e CMBL1 passa a valer “0111111”, de forma que BL00 e BL10 recebem BG0 enquanto BL01 recebe BG1. Entre *t*=8 e *t*=15 o sinal CMEP repete o comportamento apresentado entre *t*=0 e *t*=7 de acordo com a Tabela 4.10. Em *t*=8 o elemento EP00 acumula a diferença entre R(1,0) e P(01,00), EP01 acumula a diferença entre R(0,7) e P(00,08), EP02 acumula a diferença entre R(0,6) e P(00,08), EP07 acumula a diferença entre R(0,1) e P(00,08) e EP10 acumula a diferença entre R(0,0) e P(01,00). A Figura 4.13 apresenta a região de procura para os elementos EP01 e EP10.

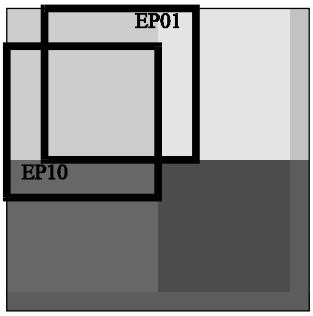

FIGURA 4.13 – Região considerada para o cálculo da SAD por EP01 e EP10.

Em *t*=8 o elemento EP01 está utilizando P(00,08) e o elemento EP10 está utilizando P(01,00). Como o sinal CMEP(1) está valendo '1' pois CMEP está valendo "01111111", o elemento EP01 está selecionando sua entrada B1 que está conectada a BG1. Embora o sinal END esteja valendo 07, o barramento local BL01 não está recebendo P(00,14), uma vez que a linha de atraso em BG1 está gerando um atraso de oito amostras fazendo com que BL01 receba P(00,08) que é o dado utilizado. As linhas de atraso inseridas nos barramentos globais BG1 e BG3 têm, portanto, função de endereçamento. Cada linha da região de procura é utilizada serialmente, sendo a seleção horizontal comandada pelo sinal CMEP. A linha P(00,yy), por exemplo, é utilizada durante o intervalo entre *t*=0 e *t*=15. Todas as memórias de procura recebem o mesmo sinal de endereçamento END, embora os dados dos barramentos BG1 e BG3 estejam deslocados oito posições à direita com relação aos barramentos BG0 e BG2 no esquema de representação apresentado na Figura 4.12. Em *t*=7 o dado P(00,15) é endereçado na memória de procura que alimenta BG1, mas sua utilização ocorrerá apenas em *t*=15. A linha de atraso é responsável por inserir este atraso entre a leitura e a utilização dos dados dos barramentos globais BG1 e BG3. Esta lógica só funciona porque a leitura da linha seguinte, por exemplo P(01,yy), ocorrerá entre *t*=8 e *t*=23, de forma que a leitura de cada linha demora dezesseis ciclos mas está atrasada de apenas em oito ciclos com relação à leitura da linha anterior. A Tabela 4.11 apresenta as operações que ocorrem nos elementos de processamento durante o intervalo entre *t*=8 e *t*=15.

TABELA 4.11 – Operações realizadas no intervalo entre *t*=8 e *t*=15.

t	CMBL0	CMBL1	CMEP	END	EP00		EP07		EP10		EP11		EP12		EP13		EP14		EP15		EP16		EP17	
					R	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P
8	00111111	01111111	01111111	00	1,0	01,00	0,1	00,08	0,0	01,00	7,7	08,08	7,6	08,08	7,5	08,08	7,4	08,08	7,3	08,08	7,2	08,08	7,1	08,08
9			00111111	01	1,1	01,01	0,2	00,09	0,1	01,01	0,0	01,01	7,7	08,09	7,6	08,09	7,5	08,09	7,4	08,09	7,3	08,09	7,2	08,09
10			00011111	02	1,2	01,02	0,3	00,10	0,2	01,02	0,1	01,02	0,0	01,02	7,7	08,10	7,6	08,10	7,5	08,10	7,4	08,10	7,3	08,10
11			00001111	03	1,3	01,03	0,4	00,11	0,3	01,03	0,2	01,03	0,1	01,03	0,0	01,03	7,7	08,11	7,6	08,11	7,5	08,11	7,4	08,11
12			00000111	04	1,4	01,04	0,5	00,12	0,4	01,04	0,3	01,04	0,2	01,04	0,1	01,04	0,0	01,04	7,7	08,12	7,6	08,12	7,5	08,12
13			00000011	05	1,5	01,05	0,6	00,13	0,5	01,05	0,4	01,05	0,3	01,05	0,2	01,05	0,1	01,05	0,0	01,05	7,7	08,13	7,6	08,13
14			00000001	06	1,6	01,06	0,7	00,14	0,6	01,06	0,5	01,06	0,4	01,06	0,3	01,06	0,2	01,06	0,1	01,06	0,0	01,06	7,7	08,14
15			00000000	07	1,7	01,07	1,0	01,07	0,7	01,07	0,6	01,07	0,5	01,07	0,4	01,07	0,3	01,07	0,2	01,07	0,1	01,07	0,0	01,07

Neste esquema de endereçamento os sinais CMBL0 e CMBL1 têm função de endereçamento vertical, ou seja, sua função é garantir que os dados referentes à linha P(00,yy), durante o intervalo entre *t*=0 e *t*=15, sejam disponibilizados nos barramentos locais BL0x. Durante o intervalo entre *t*=8 e *t*=23 as linhas de atraso garantirão que os barramentos BL0x recebam a linha P(11,yy), enquanto os barramento BL1x recebem a linha P(00,yy) e assim por diante, sempre coordenados pelos sinais CMBL0 e CMBL1. A Tabela 4.12 apresenta apenas o primeiro ciclo de processamento de cada conjunto de oito ciclos durante os primeiros sessenta e quatro ciclos de processamento. O sinal CMBL0 é igual ao sinal CMBL1, atrasado de oito ciclos de processamento. Este comportamento faz com que, a cada intervalo de dezesseis ciclos de processamento, CMBL0 selecione BG0 para os barramentos locais BLx0, enquanto o sinal CMBL1 seleciona BG1 para os barramentos locais BLx1, durante os primeiros sessenta e quatro ciclos de processamento.

TABELA 4.12 – Resumo das operações no intervalo entre *t*=0 e *t*=63.

t	CMBL0	CMBL1	END	EP00		EP10		EP20		EP30		EP40		EP50		EP60		EP70	
				R	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P
0	01111111	00000000	00	0,0	00,00	7,0	08,00	6,0	08,00	5,0	08,00	4,0	08,00	3,0	08,00	2,0	08,00	1,0	08,00
8	00111111	01111111	08	1,0	01,00	0,0	01,00	7,0	09,00	6,0	09,00	5,0	09,00	4,0	09,00	3,0	09,00	2,0	09,00
16	00011111	00111111	16	2,0	02,00	1,0	02,00	0,0	02,00	7,0	10,00	6,0	10,00	5,0	10,00	4,0	10,00	3,0	10,00
24	00001111	00011111	24	3,0	03,00	2,0	03,00	1,0	03,00	0,0	03,00	7,0	11,00	6,0	11,00	5,0	11,00	4,0	11,00
32	00000111	00001111	32	4,0	04,00	3,0	04,00	2,0	04,00	1,0	04,00	0,0	04,00	7,0	12,00	6,0	12,00	5,0	12,00
40	00000011	00000111	40	5,0	05,00	4,0	05,00	3,0	05,00	2,0	05,00	1,0	05,00	0,0	05,00	7,0	13,00	6,0	13,00
48	00000001	00000011	48	6,0	06,00	5,0	06,00	4,0	06,00	3,0	06,00	2,0	06,00	1,0	06,00	0,0	06,00	7,0	14,00
56	00000000	00000001	56	7,0	07,00	6,0	07,00	5,0	07,00	4,0	07,00	3,0	07,00	2,0	07,00	1,0	07,00	0,0	07,00

Os sinais CMBL0 e CMBL1 se repetem em períodos de sessenta e quatro ciclos de processamento. Nos primeiros sessenta e quatro ciclos de processamento os valores ‘1’ dos sinais CMBL0 e CMBL1 não são relevantes para esta análise, pois ocorrem sempre antes da inicialização do elemento de processamento que está sendo inserido no *pipeline* (aquele que recebe em sua entrada REP o valor ‘1’). Nos sessenta e quatro ciclos de processamento seguintes, entre *t*=64 e *t*=127, estes valores ‘1’ dos sinais CMBL0 e CMBL1 são responsáveis por selecionar o sinal BG2 e BG3 para os elementos que utilizam este sinal. Os sinais CMBL0 e CMBL1 têm, portanto, um papel de endereçamento vertical, ou seja, estes sinais trabalham selecionando os dados recebidos pelas linhas de elementos de processamento através dos barramentos locais.

A Tabela 4.13 apresenta o resumo das operações realizadas nos sessenta e quatro ciclos de processamento que iniciam em *t*=64. Durante este intervalo o barramento global BG0 recebe MP1, BG1 recebe MP2, BG2 recebe MP4 e BG3 recebe MB5. Na Tabela 4.13 os valores em cinza representam operações relativas ao cálculo do próximo vetor de movimento. Neste intervalo o valor ‘1’ dos sinais CMBL0 e CMBL1 são responsáveis por selecionar BG2 e BG3 para os barramentos locais que alimentam elementos de processamento que utilizam estes sinais. Em *t*=64 os elemento de processamento EPx0, exceto EP00, utilizam o dado P(08,00); para isto o sinal END da interface de E/S vale 00 e o sinal CMBL0 vale “011111111”, selecionando BG2 para os elementos de processamento EPx0, exceto EP00. Vale lembrar que no intervalo entre *t*=64 e *t*=71 os elementos de processamento EP11, ..., EP17; .....; EP71, ...,

EP77 ainda estarão utilizando BG1 e por isso o sinal CMBL1 ainda vale “00000000”, que é o valor de CMBL0 entre *t*=56 e *t*=63.

TABELA 4.13 – Resumo das operações no intervalo entre *t*=64 e *t*=127.

t	CMBL0	CMBL1	END	EP00		EP10		EP20		EP30		EP40		EP50		EP60		EP70	
				R	P	R	P	R	P	R	P	R	P	R	P	R	P	R	P
64	01111111	00000000	00	0,0	00,00	7,0	08,00	6,0	08,00	5,0	08,00	4,0	08,00	3,0	08,00	2,0	08,00	1,0	08,00
72	00111111	01111111	08	1,0	01,00	0,0	01,00	7,0	09,00	6,0	09,00	5,0	09,00	4,0	09,00	3,0	09,00	2,0	09,00
80	00011111	00111111	16	2,0	02,00	1,0	02,00	0,0	02,00	7,0	10,00	6,0	10,00	5,0	10,00	4,0	10,00	3,0	10,00
88	00001111	00011111	24	3,0	03,00	2,0	03,00	1,0	03,00	0,0	03,00	7,0	11,00	6,0	11,00	5,0	11,00	4,0	11,00
96	00000111	00001111	32	4,0	04,00	3,0	04,00	2,0	04,00	1,0	04,00	0,0	04,00	7,0	12,00	6,0	12,00	5,0	12,00
104	00000011	00000111	40	5,0	05,00	4,0	05,00	3,0	05,00	2,0	05,00	1,0	05,00	0,0	05,00	7,0	13,00	6,0	13,00
112	00000001	00000011	48	6,0	06,00	5,0	06,00	4,0	06,00	3,0	06,00	2,0	06,00	1,0	06,00	0,0	06,00	7,0	14,00
120	00000000	00000001	56	7,0	07,00	6,0	07,00	5,0	07,00	4,0	07,00	3,0	07,00	2,0	07,00	1,0	07,00	0,0	07,00

Em *t*=72 os elementos de processamento EPx0, exceto EP00 e EP01, utilizam o dado P(09,00); para isto o sinal END da interface de E/S vale 08, o sinal CMBL0 vale “00111111” e o sinal CMBL1 vale “01111111”, selecionando BG2 e BG3 para os elementos de processamento EPx0, exceto EP00 e EP01. Este esquema se repete de maneira que, a cada oito ciclos de processamento, uma linha termina o processamento relativo ao vetor de movimento anterior. Os últimos dezesseis ciclos de processamento iniciam-se em *t*=112 quando EP70 utiliza P(14,00). No intervalo entre *t*=112 e *t*=119 os elementos de processamento EP7x utilizam BG2, por isso CMBL0(7) vale ‘1’, e entre *t*=120 e *t*=127 estes elementos de processamento utilizam BG3, e por isso CMBL1(7) vale ‘1’.

A saída da matriz de processamento é determinada pelo multiplexador M64, que é alimentado pelas saídas ACC dos elementos de processamento. Em *t*=63, embora o elemento EP77 esteja recém entrando no *pipeline*, o elemento EP00 já está terminando o cálculo da SAD para a hipótese de vetor de movimento VM igual a 00. Em *t*=64 o valor da SAD calculada pelo elemento de processamento EP00 é válido em sua saída ACC, de modo que o sinal CM64 deve valer 00 para que o valor da saída ACC de EP00 seja apresentado na saída SAD da matriz de processamento. Em *t*=65 o sinal CM64 deve valer 01 para selecionar o resultado do elemento de processamento EP01 e assim por diante, até *t*=127 quando o sinal CM64 deve valer 63 para selecionar o resultado do elemento de processamento EP77. Comparando o comportamento do sinal END da interface de E/S apresentado na Tabela 4.13 com a descrição do comportamento do sinal CM64, percebe-se que os dois são iguais.

4.5 Unidade de Comparação

A matriz de processamento apresenta em sua saída o valor da SAD de uma hipótese de vetor de movimento a cada ciclo de processamento. Este valor deve ser comparado com os valores de SAD das demais hipóteses de vetor de movimento relativas à mesma região de referência para indicar como vetor de movimento a melhor hipótese. A função da unidade de comparação é, a cada ciclo de processamento, comparar a saída SAD da matriz de processamento com o menor valor já apresentado

nesta saída e, se o valor atual for menor que menor valor obtido anteriormente, indicar a existência de uma melhor hipótese para vetor de movimento.

A entrada da unidade de comparação é o sinal SAD que é a saída de matriz de processamento. A saída da unidade de comparação é o sinal NOVO_VM, que indica a existência de um novo vetor de movimento. A unidade de comparação é composta por um somador, um registrador e multiplexadores, sua arquitetura é apresentada na Figura 4.14.

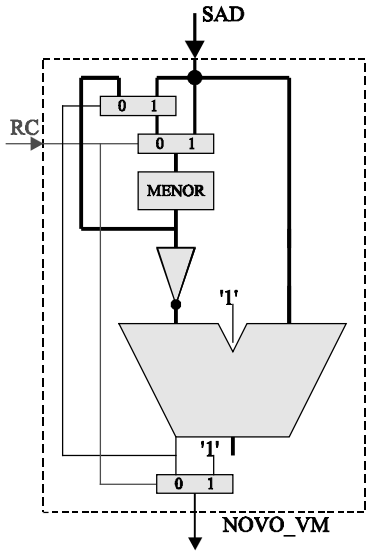

FIGURA 4.14 – Arquitetura de unidade de comparação.

A unidade de comparação possui um registrador interno denominado MENOR, onde é armazenado o menor valor recebido através da entrada SAD desde a sua inicialização. A inicialização é feita através do sinal RC. Quando sinal RC vale ‘1’ o valor da entrada SAD é escrito no registrador MENOR. Para garantir que pelo menos uma vez a saída NOVO_VM receberá o valor ‘1’, no momento da inicialização o sinal RC também comanda um multiplexador que escreve o valor ‘1’em NOVO_VM. Enquanto o sinal RC valer ‘0’ o sinal NOVO_VM receberá o sinal *carry out* do somador, conforme pode-se acompanhar na parte inferior da Figura 4.14.

O somador da unidade de comparação calcula a diferença entre a entrada SAD e o valor armazenado no registrador MENOR, utilizando a mesma metodologia adotada nos elementos de processamento. Quando esta diferença for negativa, o que indica que o valor de SAD é menor que o de MENOR, o sinal de saída NOVO_VM (que é o *carry out* do somador) vale ‘1’, sinalizando assim a existência de um novo vetor de movimento. Além disso, o sinal NOVO_VM também comanda a atualização do registrador MENOR com o valor de SAD através de multiplexador.

Na hipótese de o valor do sinal SAD ser igual ao valor armazenado no registrador MENOR, a diferença entre estes dois valores é nula. Nesta situação não será considerada a existência de um novo vetor de movimento. Esta implementação prioriza vetores de movimentos mais próximos à origem, representada pelo ponto P(00,00) na Figura 4.12.

Em t=64 a unidade de comparação recebe no sinal SAD a SAD relativa à hipótese VM=00. Neste momento o sinal RC vale ‘1’, de modo que o valor de SAD é armazenado

no registrador MENOR e o sinal NOVO_MV vale ‘1’, indicando que a hipótese VM=00 é o atual vetor de movimento. Entre *t*=65 e *t*=127 a atualização do registrador MENOR e o valor ‘1’ do sinal NOVO_MV acontecem apenas quando o valor do sinal SAD é menor que o valor armazenado no registrador MENOR. O sinal NOVO_MV é tratado então pela unidade de controle, apresentado a seguir.

4.6 Unidade de Controle

Inicialmente, esta seção apresenta um resumo das análises temporais realizadas ao longo deste capítulo. Este resumo é focado nos sinais de controle e visa analisar isoladamente o seu comportamento. Isto permite uma análise da periodicidade destes sinais, motivando assim a implementação da unidade de controle baseada em um contador que é apresentada no decorrer da seção.

Uma vez que as memórias de quadro e de faixa (vide Figura 4.4 e Figura 4.8, respectivamente) não são implementadas em FPGA, o projeto da unidade de controle não prevê a geração dos sinais de endereçamento e habilitação para escrita destas memórias. Também não é prevista a geração do sinal TROCA_MQ.

A entrada da unidade de controle é o sinal NOVO_VM, gerado pela unidade de comparação. Suas saídas são apresentadas na Tabela 4.14. A função da unidade de controle é gerar todos os sinais que controlam os blocos da unidade operacional e também gerar o sinal MV que é a saída da arquitetura para estimação de movimento.

TABELA 4.14 – Sinais de saída da unidade de controle.

Interface de E/S para Dados de Referência		**Interface de E/S para Dados de Procura**		**Matriz de Processamento**		**Unidade de Comparação**		**Arquitetura para Estimação de Movimento**	
Sinal	**Nº de *Bits***	**Sinal**	**Nº de *Bits***	**Sinal**	**Nº de *Bits***	**Sinal**	**Nº de *Bits***	**Sinal**	**Nº de *Bits***
END	6	END	6	REP	64	RC	1	VM	6
LE_MR0	1	LE_MP0	1	CMBL0	8				
LE_MR1	1	LE_MP1	1	CMBL1	8				
CMR	1	LE_MP2	1	CMEP	8				
		LE_MP3	1	CM64	6				
		CMBG0	2						
		CMBG1	2						
		CMBG2	2						
		CMBG3	2						

Cada coluna da Tabela 4.14 apresenta os sinais que controlam um dos blocos da arquitetura para estimação de movimento (vide Figura 4.2), exceto pela última coluna, que se refere à arquitetura como um todo. O sinal END aparece repetido nas duas primeiras colunas pois este sinal é utilizado por aqueles dois blocos da interface de E/S (vide Figura 4.3). Para simplificar a análise a ser feita nesta seção são definidos dois sinais: o sinal LE_MR, de 2 *bits*, que representa os sinais LE_MR0 e LE_MR1, e o sinal LE_MP, de 4 *bits*, que representa os sinais LE_MP0, LE_MP1, LE_MP2 e LE_MP3. Portanto, em se tratando da unidade de controle, as referências são feitas sempre aos sinais LE_MR e LE_MP.

Os sinais da Tabela 4.14 podem ter periodicidade de 8, 64 ou 256 ciclos de processamento. O único sinal de período 8 ciclos de processamento é o CMEP, cujo comportamento é apresentado na Tabela 4.15. Nesta tabela, assim como na Tabela 4.16, o número de *bits* de um sinal é apresentado entre parênteses ao lado do mesmo.

TABELA 4.15 – Sinal de controle com período 8 ciclos de processamento.

t	CMEP
0	01111111
1	00111111
2	00011111
3	00001111
4	00000111
5	00000011
6	00000001
7	00000000

Os sinais de período 64 ciclos de processamento são apresentados na Tabela 4.16. Nesta tabela as linhas pontilhadas indicam ciclos de processamento que não estão representados. Vale observar que os sinais END e CM64 possuem igual comportamento. Para simplificar a notação utilizada na Tabela 4.16, todos os *bits* do sinal REP valem ‘0’ exceto pelo *bit* cuja ordem é indicada entre parênteses, que vale ‘1’. Os sinais CMBL0 e CMBL1 são atualizados a cada 8 ciclos de processamento, por isso, aparecem em células mescladas na Tabela 4.16. O sinal RC vale ‘1’ no ciclo de processamento em que *t*=0, nos demais ciclos vale ‘0’.

TABELA 4.16 – Sinais de controle com período 64 ciclos de processamento.

t	END	REP	CMBL0	CMBL1	CM64	RC
0	00	REP(00)=1	01111111	00000000	00	1
7	07	REP(07)=1			07	
8	08	REP(08)=1	00111111	01111111	08	
15	15	REP(15)=1			15	
16	16	REP(16)=1	00011111	00111111	16	
23	23	REP(23)=1			23	
24	24	REP(24)=1	00001111	00011111	24	
31	31	REP(31)=1			31	
32	32	REP(32)=1	00000111	00001111	32	0
39	39	REP(39)=1			39	
40	40	REP(40)=1	00000011	00000111	40	
48	48	REP(48)=1			48	
49	49	REP(49)=1	00000001	00000011	49	
55	55	REP(55)=1			55	
56	56	REP(56)=1	00000000	00000001	56	
63	63	REP(63)=1			63	

A Tabela 4.17 apresenta o comportamento dos sinais de período 256 ciclos de processamento. Todos estes sinais estão representados em valores binários, por isso, não é necessário indicar o número de *bits*.

TABELA 4.17 – Sinais de controle com período 256 ciclos de processamento.

t	LE_MR	CMR	LE_MP	CMBG0	CMBG1	CMBG2	CMBG3
0 - 63	01	0	1000	00	01	11	00
64 - 127	10	1	0100	01	10	00	01
128 - 191	01	0	0010	10	11	01	10
192 - 255	10	1	0001	11	00	01	11

Devido à periodicidade dos sinais de controle apresentados, a unidade de controle é baseada em um contador denominado contador de controle. Como o maior período possível é 256 ciclos de processamento, é necessário que o contador de controle possua 8 *bits* para representar este intervalo.

O contador de controle também pode ser entendido como uma máquina de estados com 256 estados. Nesta máquina, os estados dos sinais são gerados a partir de alguns dos 8 *bits* do contador de controle. Os *bits* utilizados para gerar cada sinal, se necessário, podem passam por uma lógica combinacional para gerar o sinal da maneira esperada pelos blocos controlados. A Figura 4.15 ilustra a arquitetura da unidade de controle, mostrando os *bits* do contador de controle utilizados para a geração de cada sinal.

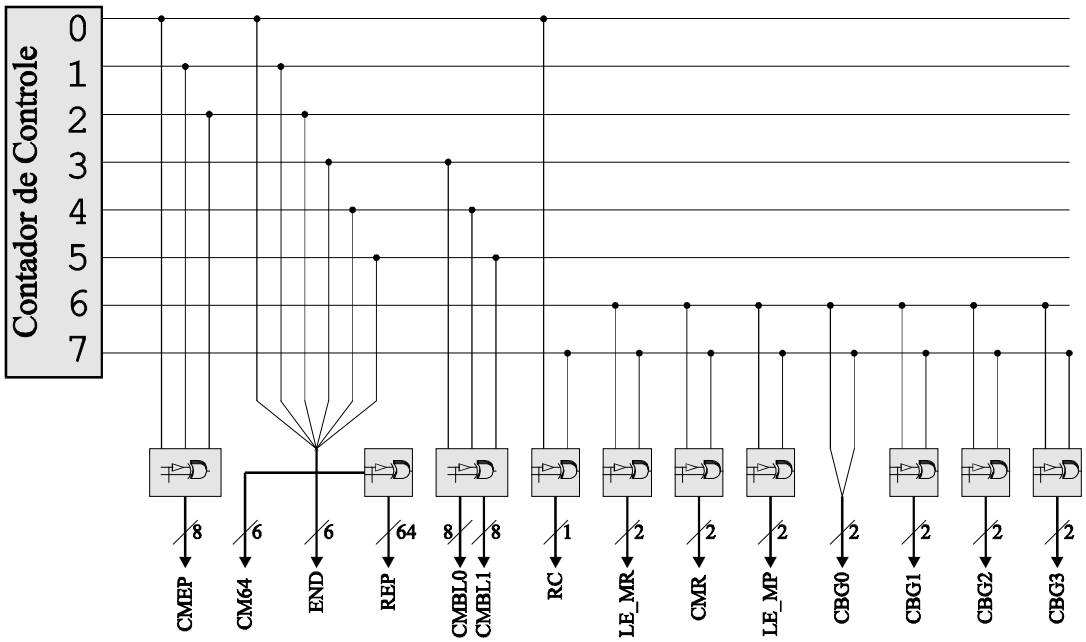

FIGURA 4.15 – Arquitetura para geração dos sinais de controle.

Analisando a Figura 4.15 observa-se que os sinais `CMBL0` e `CMBL1` não dependem dos 3 *bits* menos significativos do contador de controle. Isto explica o fato destes sinais serem constantes durante períodos de 8 ciclos de processamento. Da mesma maneira os sinais `LE_MR`, `CMR`, `LE_MP`, `CMBG0`, `CMBG1`, `CMBG2`, e `CMBG3` não dependem dos 6 *bits* menos significativos do contador de controle. Isto explica o fato destes sinais serem constantes durante períodos de 64 ciclos de processamento.

Observando a arquitetura de um elemento de processamento apresentada na Figura 4.9 percebe-se que, no ciclo de processamento imediatamente posterior ao cálculo da última diferença absoluta, o valor da SAD é disponibilizado em sua saída `ACC`. O elemento de processamento `EP00`, por exemplo, disponibiliza a SAD referente à hipótese de vetor de movimento `MV` igual a 00 nos instantes em que *t* é múltiplo de 64, ou seja, *t*=64, *t*=128, etc. Observando a Tabela 4.16 percebe-se que o sinal `END` também pode ser utilizado para indicar o elemento de processamento que está disponibilizando um valor válido de SAD. O sinal `CM64` atua em sincronia com este mecanismo para garantir que a saída `SAD` da matriz de processamento receba um valor de SAD válido.

Observando a arquitetura da unidade de controle apresentada na Figura 4.14 percebe-se que, durante o mesmo ciclo de processamento em que o sinal `SAD`, saída da matriz de processamento, possuir um valor menor que aquele armazenado no registrador `MENOR`, o sinal `NOVO_VM` terá valor ‘1’. Isto significa que, no instante em que o sinal `NOVO_VM` valer ‘1’, a melhor hipótese de vetor de movimento é aquela referente ao elemento de processamento que terminou o cálculo da SAD no instante anterior e portanto está iniciando o cálculo de uma nova SAD no instante atual.

A unidade de controle possui um registrador denominado `VM`, no qual é armazenado o valor do sinal `END` no instante em que o sinal `NOVO_VM` valia ‘1’. A Figura 4.16 apresenta a parte da arquitetura da unidade de controle responsável por armazenar o valor do sinal `VM` ou, em outras palavras, gerar a saída da arquitetura para estimação de movimento. O sinal `RC` (o mesmo que controla a unidade de comparação) é utilizado para controlar a inicialização do registrador `VM`. Nos instantes em que *t* é

múltiplo de 64 o sinal RC vale ‘1’, de modo a armazenar no registrador VM o valor de END igual a 00, que é o valor inicial para o vetor de movimento.

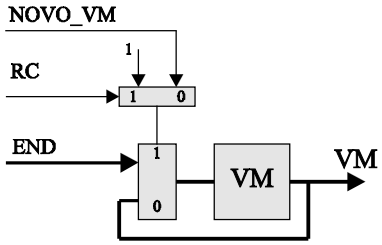

FIGURA 4.16 – Detalhe do registrador VM da unidade de controle.

Para a arquitetura desenvolvida neste trabalho o vetor de movimento representa a ordem do elemento de processamento que apresentou a menor SAD. Outra interpretação é que o vetor de movimento representa o endereço do ponto onde inicia a região que gerou o menor valor de SAD.

A comparação dos valores de SAD demora 64 ciclos de processamento, sendo iniciada no instante em que *t*=64. Neste instante, de acordo com a Figura 4.16, o valor de END igual a 00 é armazenado em VM e estará disponível em *t*=65. Nos ciclos de processamento seguintes, entre *t*=65 e *t*=127, o valor de END será armazenado no registrador VM apenas se o sinal NOVO_VM valer ‘1’. Ao final das comparações, no instante em que *t*=127, o valor de END igual a 63 poderá ser armazenado no registrador VM e, neste caso, estaria disponível em *t*=128. Segundo este esquema de geração do vetor de movimento, percebe-se que o valor de VM é válido nos instantes em que *t* é múltiplo de 64 a partir de *t*=128, ou seja, *t*=128, *t*=192, etc. Nos demais instantes o sinal VM representa a melhor hipótese de vetor de movimento encontrada até o instante atual.

Uma vez que o sinal RC é utilizado para inicializar o registrador VM, ele também indica que o valor de VM é válido pois estas duas situações ocorrem no mesmo instante. O valor de VM deve ser lido no mesmo instante em que RC vale ‘1’, pois no ciclo de processamento seguinte este valor será sobrescrito.

5 Implementação e Validação da Arquitetura para Estimação de Movimento

Este capítulo apresenta a implementação e a validação da arquitetura para estimação de movimento, incluindo sua prototipagem no dispositivo FPGA (*Field Programmable Gate Array*) XC2S150 da Xilinx [XIL 2002].

Inicialmente o capítulo apresenta os detalhes relativos à implementação da arquitetura, tais como os recursos utilizados, o fluxo de projeto adotado e alguns comentários sobre a sua descrição VHDL. A validação da arquitetura foi realizada através da análise de resultados obtidos em simulação e em prototipagem.

5.1 Recursos Utilizados

Os recursos utilizados para a implementação e validação da arquitetura para estimação de movimento são:

- Ferramenta para desenvolvimento de *hardware WebPack 4.2*, da Xilinx [XIL 2002].
- Ferramenta para simulação de *hardware ModelSim 5.5*, da Mentor [MEN 2002].
- *Kit* para prototipagem *Xilinx Spartan-II Evaluation Kit* da Avnet, cujo principal componente é o dispositivo FPGA XC2S150, da Xilinx.
- Ferramenta para desenvolvimento de *software Visual C++*, da Microsoft.
- Câmera de vídeo *Take It 350*, da Microtec.
- Computador PC com processador Pentium III.

A ferramenta para desenvolvimento de *hardware WebPack* é fornecida gratuitamente pelo fabricante do dispositivo FPGA utilizado. O *WebPack* é um pacote de aplicativos no qual cada aplicativo é responsável por uma das etapas do fluxo de projeto. Estes aplicativos são integrados no ambiente de desenvolvimento apresentado na Figura 5.1. Nesta figura, à esquerda são apresentadas as etapas do fluxo de projeto; à direita, são apresentados os arquivos que compõem o projeto como código fonte VHDL e descrições de forma de onda para simulação.

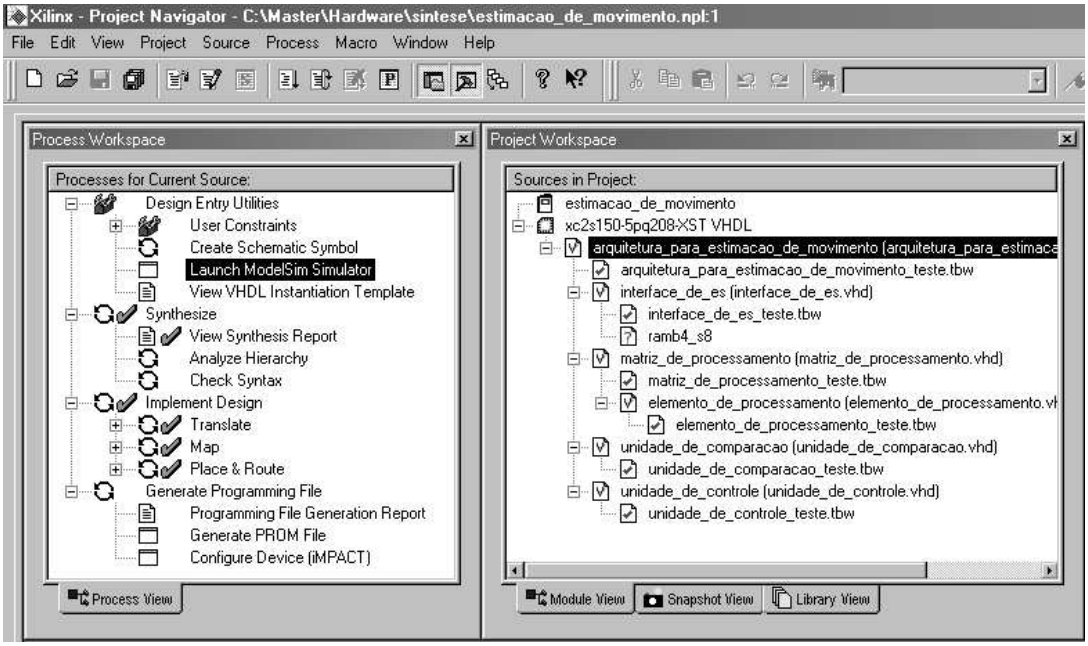

FIGURA 5.1 – Ferramenta para desenvolvimento de *hardware WebPack*.

Na ferramenta *WebPack*, um "projeto" é um ambiente de trabalho no qual são associados arquivos de descrição de *hardware*, arquivos de descrição de formas de onda para simulação e um arquivo de descrição de um dispositivo físico FPGA, além de outros tipos de arquivos que não foram utilizados neste trabalho.

De maneira integrada ao *WebPack* é distribuída a ferramenta de simulação *ModelSim*. Esta ferramenta pode ser acessada a partir do ambiente integrado do *WebPack* (vide item selecionado no fluxo de projeto na Figura 5.1). O *ModelSim* possibilita a realização de simulações meramente funcionais ou simulações completas considerando os atrasos. O resultado das simulações pode ser analisado através de um recurso para visualização de formas de onda incluído no *ModelSim*.

Para prototipagem foi utilizado o *Xilinx Spartan-II Evaluation Kit* apresentado na Figura 5.2 Este *kit* é vendido pelo distribuidor Avnet apenas para fins de prototipagem. O *kit* consiste em uma placa de circuito impresso na qual encontra-se o dispositivo FPGA XC2S150, a memória EEPROM XC18V01 para configuração do dispositivo FPGA, circuito de alimentação, chaves do tipo liga-desliga e circuito para geração de relógio. O dispositivo FPGA XC2S150 possui 150 mil portas lógicas equivalentes distribuídas em 1728 blocos lógicos programáveis ou *slices*, segundo a denominação adotada em [XIL 2002], e 12 blocos de memória de 4096 *bits* por bloco.

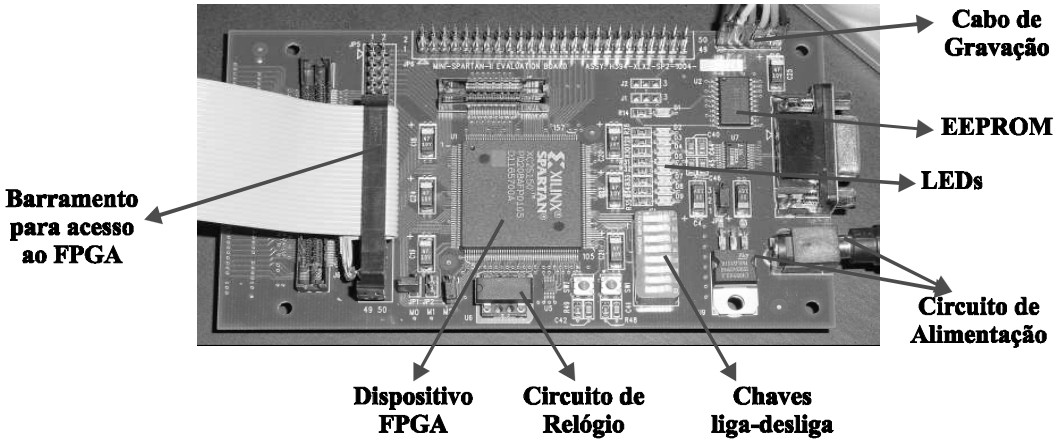

FIGURA 5.2 – *Kit* para prototipagem *Xilinx Spartan-II Evaluation Kit*.

O desenvolvimento do *software* para auxiliar na validação da arquitetura foi feito utilizando a ferramenta *Visual C++*. Esta ferramenta proporciona um ambiente integrado para desenvolvimento de *software* com o qual é possível ter acesso à arquitetura de aplicação MFC (Microsoft *Foundation Classes*) com a qual foram desenvolvidos os componentes gráficos do *software*.

Este *software* possibilita a captura de imagens externas, o que é feito através da câmera de vídeo *Take It 350*. Esta é uma câmera do tipo *Web Cam* de resolução máxima 640x480 pontos.

5.2 Fluxo de Projeto

O fluxo de projeto é dependente da ferramenta para desenvolvimento de *hardware*. O fluxo de projeto adotado neste trabalho foi definido pela ferramenta *WebPack*. A Figura 5.3 apresentada de maneira simplificada este fluxo de projeto.

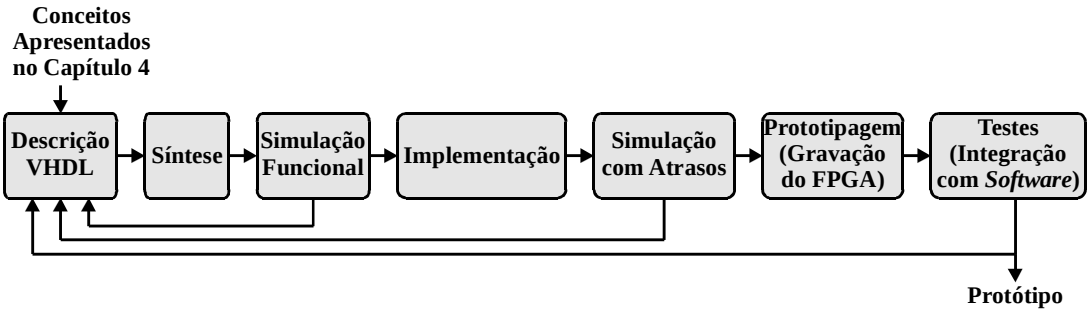

FIGURA 5.3 – Fluxo de projeto adotado.

A etapa inicial do fluxo de projeto consiste em descrever em linguagem VHDL a arquitetura apresentada no Capítulo 4. A seguir, na etapa de síntese, a ferramenta *WebPack* gera uma descrição intermediária onde são representadas todas as unidades funcionais presentes na descrição VHDL de entrada. A partir do resultado da síntese pode ser realizada uma simulação funcional.

A etapa seguinte é a implementação. Nesta etapa o conjunto de unidades funcionais presentes na representação intermediária gerada pela síntese é mapeado para as células lógicas do dispositivo FPGA. Para isto a ferramenta *WebPack* possui uma

descrição física do dispositivo XC2S150, incluindo os dados de atraso entre as conexões internas. O papel do *WebPack* durante a implementação é alocar as unidades funcionais da arquitetura nos blocos lógicos do FPGA, prevendo também as conexões entre estas unidades. Nesta etapa, o *WebPack* gera uma segunda descrição intermediária, onde são representadas as unidades funcionais já associadas aos blocos lógicos do FPGA e os atrasos entre suas conexões. A partir do resultado da implementação pode ser realizada uma simulação considerando os atrasos.

A seguir, na etapa de prototipagem, o dispositivo FPGA é gravado com uma imagem da descrição intermediária gerada na etapa de implementação. Para realização dos testes com o protótipo é necessária a integração com o *software* desenvolvido para auxiliar na validação. Este *software* permite alimenta o protótipo com dados e lê o valor dos vetores de movimento calculados.

Conforme mostra a Figura 5.3, os resultados observados durantes as duas etapas de simulação e durante os testes com o protótipo servem para a correção de erros e o refinamento da descrição VHDL inicial.

5.3 Descrição em Linguagem VHDL

A descrição em linguagem VHDL da arquitetura para estimação de movimento constitui um projeto na ferramenta *WebPack* denominado `estimacao_de_movimento`. Esta descrição é composta por seis arquivos, cada um descrevendo uma "entidade", isto é, uma `entity` da linguagem VHDL. Estes arquivos levam o mesmo nome da entidade que descrevem adicionado da extensão `.vhd`. A Tabela 5.1 apresenta as entidades que compõem o projeto `estimacao_de_movimento` associadas ao bloco da arquitetura que descrevem.

TABELA 5.1 – Entidades que compõem a descrição VHDL da arquitetura.

Entidade	Bloco da Arquitetura Descrito
`arquitetura_para_ estimacao_de_movimento`	Arquitetura para Estimação de Movimento
`interface_de_es`	Interface de Entrada e Saída
`matriz_de_processamento`	Matriz de Processamento
`elemento_de_processamento`	Elemento de Processamento
`unidade_de_comparacao`	Unidade de Comparação
`unidade_de_controle`	Unidade de Controle

Nas entidades do projeto `estimacao_de_movimento` o número de *bits* dos sinais é definido em função de um parâmetro em VHDL do tipo `generic` denominado `n`. Este parâmetro tem valor igual ao número de *bits* do sinal de entrada, podendo variar entre 1 e 8. Visto de outra forma, `n` é o número de *bits* de cada ponto da imagem de entrada, e determina o número de níveis de cinza da imagem. O número de *bits* dos sinais que carregam dados de entrada, como por exemplo os barramentos de entrada e saída, barramentos globais, barramentos locais, `B0`, `B1`, `Ri` e `Ro` é igual ao valor de `n`. O número de *bits* dos sinais que carregam valores de SAD, como por exemplo os sinais `ACC` e `SAD` é igual ao valor de `n` + 6. O parâmetro `n` não tem influência sobre os sinais de controle.

Em todas as entidades foi adicionado um sinal de relógio denominado `CLK` cuja borda ascendente determina o final de um ciclo de processamento e início de outro.

A Figura 5.4 mostra a janela do *WebPack* na qual são apresentados de maneira hierárquica os arquivos que compõem o projeto `estimacao_de_movimento`. No topo desta figura aparece um ícone que representa o projeto, seguido por um ícone que representa o dispositivo FPGA associado a este projeto. Abaixo aparecem ícones com detalhes em azul que representam os arquivos contendo as descrições VHDL das entidades que compõem o projeto. A cada entidade está associado um ícone com detalhes em verde, que representa o arquivo contendo as descrições das formas de onda utilizadas para simulação daquela entidade. Os arquivos de descrição de formas de onda levam o mesmo nome da entidade ao qual estão associados acrescentado de `_teste.tbw`.

FIGURA 5.4 – Arquivos que compõem o projeto `estimacao_de_movimento`.

A hierarquia de entidades apresentada na Figura 5.4 mostra que as entidades `arquitetura_para_estimacao_de_movimento`, `interface_de_es` e `matriz_de_processamento` possuem instâncias de outras entidades. A `matriz_de_processamento`, por exemplo, possui instâncias da entidade `elemento_de_processamento`.

A entidade `arquitetura_para_estimacao_de_movimento` é peculiar pois ela não é utilizada para acrescentar elementos lógicos à arquitetura. Seu papel é o de acomodar as instâncias e conexões das entidades `interface_de_es`, `matriz_de_processamento`, `unidade_de_comparacao` e `unidade_de_controle`, representando assim a arquitetura para estimação de movimento como um todo.

A entidade `interface_de_es` possui instâncias da entidade `ramb4_s8`, que descreve uma memória do tipo RAM de 512 posições de 8 *bits*. Esta descrição de memória é fornecida pelo fabricante do FPGA juntamente com a ferramenta *WebPack*. Sua descrição é fechada, isto é, seu código fonte não está acessível ao usuário da ferramenta.

A entidade `ramb4_s8` possui 512 posições porque as opções de descrição de memória oferecidas pelo fabricante para o dispositivo XC2S150 totalizam sempre 4096 *bits*, o que corresponde a um bloco de memória interno do dispositivo FPGA. Esta

descrição de memória foi utilizada para implementar as memórias de procura e de referência, que são de 64 posições. Para tal, nos 3 *bits* superiores do sinal de endereçamento das instâncias de `ramb4_s8` a ferramenta *WebPack* escreveu automaticamente o valor ‘0’.

Vale lembrar que, neste trabalho, as memórias de quadro e de faixa não foram descritas em VHDL. Assim, as entradas da arquitetura para estimação de movimento passam a ser os sinais `BES`, `BES0` e `BES1`.

Uma vez que a descrição da entidade `ramb4_s8` é fechada, o número de *bits* dos sinais nas instâncias de `ramb4_s8` não pode ser definido em função do parâmetro `n`. Assim, quando este parâmetro recebe um valor inferior a 8, a ferramenta *WebPack* atribui automaticamente o valor ‘0’ aos *bits* não utilizados nos sinais de acesso às instâncias de `ramb4_s8`.

5.4 Resultados de Simulação

Esta seção apresenta os resultados obtidos na simulação das entidades que compõem o projeto `estimacao_de_movimento`. Ao final da seção são apresentados alguns resultados da implementação deste projeto.

Para garantir a validade dos resultados as simulações foram realizadas na ordem inversa da hierarquia apresentada na Figura 5.4, ou seja, as entidades que são instanciadas são apresentadas antes das entidades nas quais estas instâncias são feitas. Desta maneira, inicialmente é apresentada a simulação da interface de E/S. Em seguida é apresentada a simulação de um elemento de processamento individual, seguida pela simulação da matriz de processamento. As simulações das unidades de comparação e de controle são então apresentadas, e por fim é apresentada uma simulação da arquitetura para estimação de movimento como um todo.

As simulações apresentadas são aquelas realizadas a partir dos resultados da implementação na qual os atrasos são considerados. Estas simulações foram realizadas na segunda etapa de simulação dentro do fluxo de projeto. Os resultados obtidos na primeira etapa de simulação não são apresentados, uma vez que os resultados da simulação com atrasos garantem os resultados da simulação funcional.

Nas simulações do elemento de processamento e da unidade de comparação foi atribuído aos sinais de controle um comportamento diferente do apresentado na Seção 4.6. Isto permite focar a análise nas funcionalidades destas entidades e reduzir o número de ciclos de processamento simulados. Nas demais entidades esta estratégia não pôde ser utilizada pois alteraria os resultados.

A Figura 5.5 apresenta a simulação da interface de E/S. Esta simulação tem como entradas o sinal de relógio `CLK`, os sinais de dados `BES`, `BES0` e `BES1` e os sinais de controle `END`, `LE_MP`, `CMBG0`, `CMBG1`, `CMBG2`, `CMBG3`, `LE_MR` e `CMR`. Suas saídas são os sinais `BG0`, `BG1`, `BG2`, `BG3` e `R`. Nesta simulação é realizado um ciclo de escrita e um ciclo de leitura em cada uma das memórias de procura e de referência. Os sinais de controle dos multiplexadores são sincronizados com o sinal `END` para comprovar a existência dos dados nos endereços em que estes foram anteriormente escritos. Note que, no segundo ciclo de processamento, o valor 5 é escrito simultaneamente em `MP0` e `MP4` sendo lido no sexto ciclo de processamento e enviado para as saídas `BG0` e `BG1`, respectivamente.

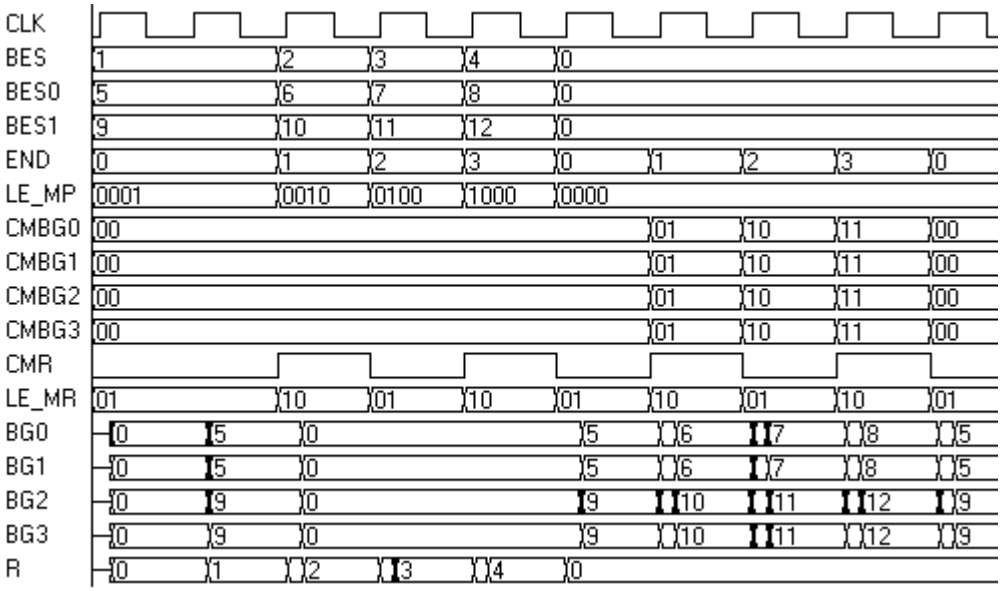

FIGURA 5.5 – Simulação da interface de E/S.

A Figura 5.6 apresenta a simulação de um elemento de processamento. Esta simulação tem como entradas o sinal de relógio `CLK`, os sinais de dados `Ri`, `B0` e `B1` e os sinais de controle `REP` e `CMEP`. Suas saídas são os sinais `Ro` e `ACC`. Nesta simulação é realizada a acumulação de algumas diferenças. Esta simulação comprova que enquanto o sinal `REP` é mantido em '1', o valor de `ACC` é igual à diferença entre `Ri` e `B0`. Ela também mostra o correto comportamento do elemento de processamento na situação em que o sinal `B0` (ou `B1` para `CMEP` valendo '1') tem valor maior que `Ri`. Note que, no quarto ciclo de processamento, o valor do sinal `ACC` é incrementado por 3 que é a diferença absoluta entre o sinal `Ri` que vale 7 e o sinal `B0` que vale 10. No ciclo de processamento seguinte o sinal `CMEP` muda de valor de modo que o sinal `B1` passa a ser tomado em lugar de `B0`.

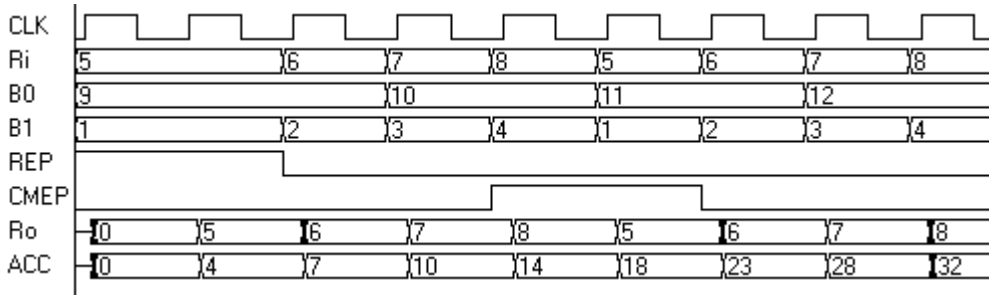

FIGURA 5.6 – Simulação de um elemento de processamento.

A simulação da matriz de processamento é apresentada a seguir. Esta simulação tem como entradas o sinal de relógio `CLK`, os sinais de dados `BG0`, `BG1`, `BG2`, `BG3` e `R` e os sinais de controle `REP`, `CMBL0`, `CMBL1`, `CMEP` e `CM64`. Sua saída é o sinal `SAD`. Esta simulação foi realizada em um período superior a 128 ciclos de processamento, porém é apresentada apenas a segunda metade deste período, durante o qual o sinal `SAD` é válido.

Para viabilizar a análise desta simulação os sinais de entrada de dados foram mantidos constantes. Assim, aos sinais `BG0`, `BG1`, `BG2`, `BG3` e `R` foram atribuídos os

valores 2, 3, 4, 5 e 1, respectivamente. Com base na descrição da arquitetura apresentada na Seção 4.4 foi construída a Tabela 5.2, onde são apresentados os resultados esperados para a saída SAD nesta simulação.

TABELA 5.2 – Resultados esperados na simulação da matriz de processamento.

	EPx0	**EPx1**	**EPx2**	**EPx3**	**EPx4**	**EPx5**	**EPx6**	**EPx7**
EP0x	64	72	80	88	96	104	112	120
EP1x	80	88	96	104	112	120	128	136
EP2x	96	104	112	120	128	136	144	152
EP3x	112	120	128	136	144	152	160	168
EP4x	128	136	144	152	160	168	176	184
EP5x	144	152	160	168	176	184	192	200
EP6x	160	168	176	184	192	200	208	216
EP7x	176	184	192	200	208	216	224	232

Cada célula da Tabela 5.2 representa o resultado esperado para a SAD calculada pelo elemento de processamento que encontra-se na posição dentro da matriz de processamento que corresponde à posição da célula na Tabela 5.2. A Figura 5.7 apresenta a simulação da matriz de processamento. Nesta simulação pode-se comprovar que os valores do sinal SAD correspondem aos valores esperados apresentados na Tabela 5.2.

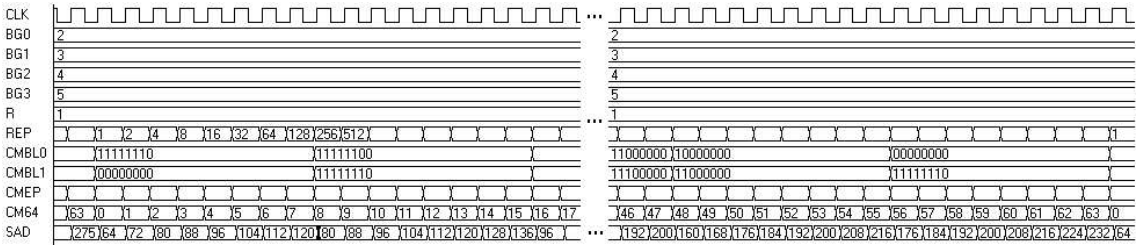

FIGURA 5.7 – Simulação da matriz de processamento.

A Figura 5.8 apresenta a simulação da unidade de comparação. Esta simulação tem como entrada o sinal de relógio CLK, o sinal de dados SAD e os sinal de controle RC. Sua saída é o sinal NOVO_VM. Para auxiliar na análise da simulação o sinal MENOR, que não é saída da arquitetura da unidade de comparação, também é apresentado. Esta simulação comprova que para RC igual a ‘1’ o sinal MENOR recebe o valor do sinal SAD e o sinal NOVO_VM vale ‘1’. Para RC igual a ‘0’, isto somente acontece na situação em que o valor de SAD é menor que o valor armazenado no registrador MENOR.

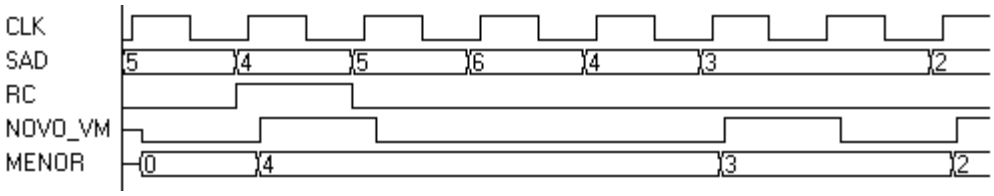

FIGURA 5.8 – Simulação da unidade de comparação.

A Figura 5.9 apresenta a simulação da unidade de controle. Esta simulação tem como entradas o sinal de relógio `CLK` e o sinal `NOVO_VM`. Suas saídas são os sinais de controle `END`, `LE_MP`, `CMBG0`, `CMBG1`, `CMBG2`, `CMBG3`, `LE_MR`, `CMR`, `REP`, `CMBL0`, `CMBL1`, `CMEP`, `CM64` e `RC`. Esta simulação foi realizada em um período superior a 256 ciclos de processamento. O objetivo desta simulação é comprovar que os sinais de controle ocorrem na ordem correta, conforme apresentado na Seção 4.6. Para melhor visualização a Figura 5.9a apresenta os 256 ciclos de processamento da simulação da unidade de controle. Na Figura 5.9b é apresentado um detalhe de 64 ciclos de processamento e na Figura 5.9c é apresentado um detalhe de 8 ciclos de processamento.

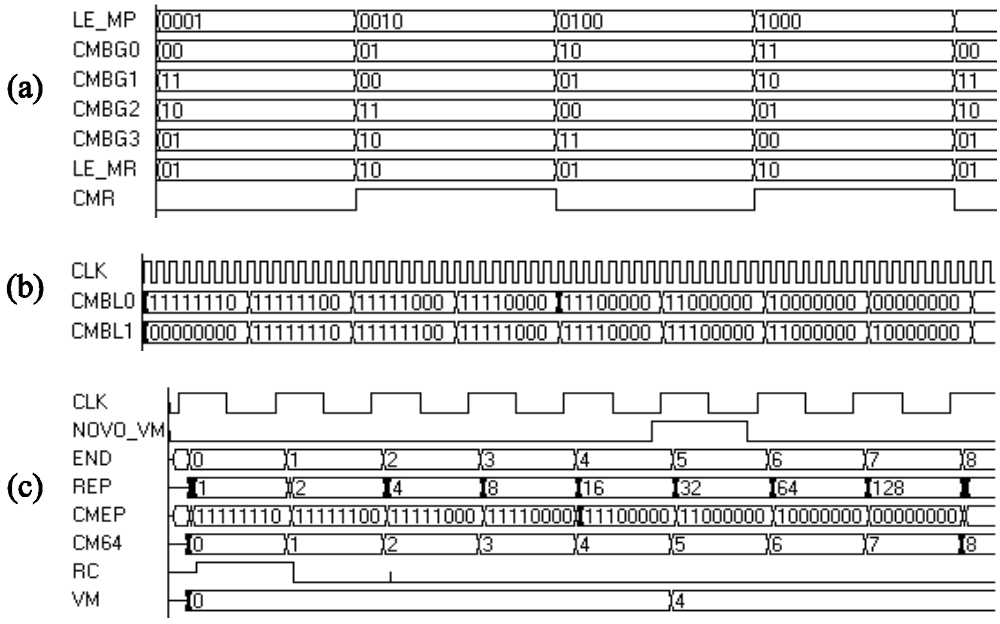

FIGURA 5.9 – Simulação da unidade de controle. (a) 256 ciclos.
(b) 64 ciclos. (c) 8 ciclos.

A seguir é apresentada a simulação da arquitetura para estimação de movimento como um todo. Esta simulação tem como entradas o sinal de relógio `CLK` e os sinais de dados `BES`, `BES0`, `BES1`. Sua saída é o sinal `VM`. Para auxiliar na análise da simulação também são apresentados os sinais internos `BG0`, `BG1`, `BG2`, `BG3`, `R`, `SAD`, `NOVO_VM` e `END`. Nesta simulação as 64 posições das memórias de referência e de procura foram preenchidas com valores iguais. Na memória `MR0` foi escrito o valor 0 e em `MR1` o valor 1. Na memória `MP0` foi escrito o valor 0, em `MP1` o valor 1 e assim por diante, até que em `MP7` foi escrito o valor 7.

A Figura 5.10a apresenta os primeiros 192 ciclos de processamento da simulação da arquitetura para estimação de movimento. Nos primeiros 64 ciclos de processamento, as memórias `MR0`, `MP2` e `MP6` são utilizadas para escrita, enquanto as memórias `MR1`,

MP0, MP1, MP7 e MP4 alimentam a matriz de processamento, porém seus dados não são válidos ainda. Nos próximos 64 ciclos de processamento as memórias MR1, MP3 e MP7 são utilizadas para escrita, enquanto as memórias MR0, MP1, MP2, MP4 e MP5 alimentam a matriz de processamento, porém seus dados não são válidos ainda. Nos próximos 64 ciclos de processamento, as memórias MR0, MP0 e MP4 são utilizadas para escrita, enquanto as memórias MR1, MP2, MP3, MP5 e MP6 alimentam a matriz de processamento, desta vez com dados válidos. Assim, após estes 192 ciclos de processamento os valores do sinal SAD passam a ser válidos. A Figura 5.10b apresenta um detalhe desta simulação, onde pode-se confirmar que os valores do sinal SAD correspondem aos valores esperados apresentados na Tabela 5.2.

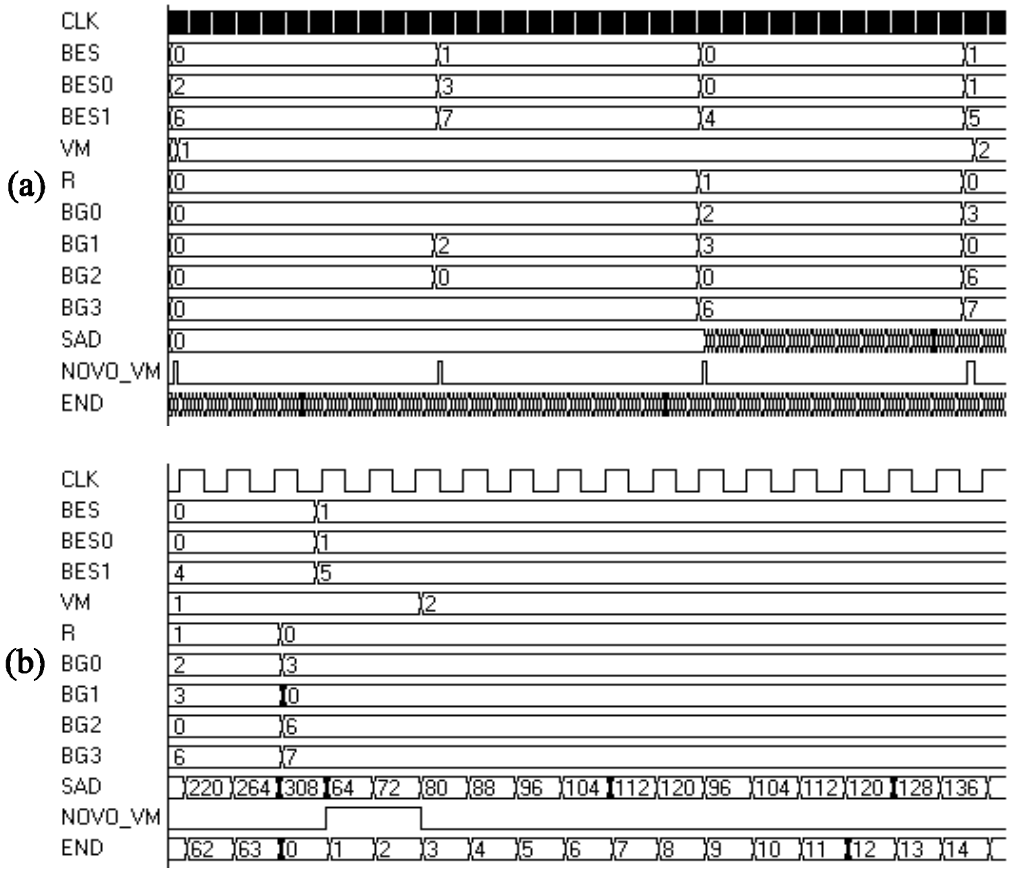

FIGURA 5.10 – Simulação da arquitetura para estimação de movimento.
(a) 192 ciclos. (b) 16 ciclos.

Uma vez comprovada a funcionalidade do circuito através das simulações apresentadas, a implementação ainda provê dois resultados que merecem análise: a ocupação de blocos lógicos e os atrasos dos sinais.

A Tabela 5.3 apresenta a quantidade de blocos lógicos necessários para implementar cada uma das entidades que compõem a arquitetura para os valores de n=8 e n=4, em valor absoluto e em percentual do total de células lógicas do dispositivo XC2S150. Deve-se notar que todas as entidades, exceto a unidade_de_controle, utilizam uma quantidade de blocos lógicos do dispositivo FPGA proporcional ao valor de n. A entidade interface_de_es, além dos blocos lógicos, utiliza 10 blocos de memória, o que corresponde a 83,3% dos 12 blocos presentes no dispositivo XC2S150.

TABELA 5.3 – Quantidade de blocos lógicos necessários para implementação das entidades do projeto `estimacao_de_movimento`.

Entidade	Utilização dos Blocos Lógicos do Dispositivo FPGA			
	n=8		n=4	
`arquitetura_para_ estimacao_de_movimento`	1918	111,0%	1228	71,1%
`interface_de_es`	36	2,1%	18	1,0%
`matriz_de_processamento`	1795	103,9%	1113	64,4%
`elemento_de_processamento`	16	0,9%	10	0,6%
`unidade_de_comparacao`	15	0,9%	11	0,6%
`unidade_de_controle`	65	3,8%	65	3,8%

É importante notar, na primeira linha da Tabela 5.3, que para implementar a arquitetura para estimação de movimento em 8 *bits* são necessários 1918 blocos lógicos, o que corresponderia a 111,0% da capacidade do dispositivo XC2S150. Portanto, por limitação deste dispositivo FPGA disponível na placa onde foram realizados os experimentos com esta arquitetura, não é possível nele implementar a arquitetura projetada com codificação da luminância em 8 bits. Na Figura 5.11 é apresentado um gráfico que mostra o comportamento do número de blocos lógicos utilizados na implementação da arquitetura para estimação de movimento em função do número de *bits* de entrada do sinal de luminância.

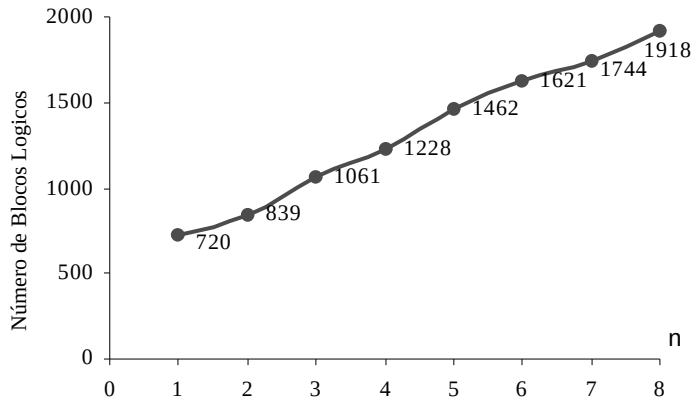

FIGURA 5.11 – Número de blocos lógicos necessários para a implementação da arquitetura para estimação de movimento em função de `n`.

A Tabela 5.4 apresenta o maior atraso entre conexões e a máxima freqüência de trabalho para cada uma das entidades do projeto `estimacao_de_movimento`. A entidade `elemento_de_processamento` é a que possui o menor atraso. A máxima freqüência com que o dispositivo FPGA poderá trabalhar é a máxima freqüência de trabalho da entidade `arquitetura_para_estimacao_de_movimento`, que de acordo com a Tabela 5.4 é de 33 MHz.

TABELA 5.4 – Atrasos relativos às entidades do projeto.

Entidade	Atraso (ns)	Máxima Freqüência de Relógio (MHz)
arquitetura_para_ estimacao_de_movimento	30,263	33,04
interface_de_es	6,696	149,34
matriz_de_processamento	18,853	53,04
elemento_de_processamento	5,371	186,19
unidade_de_comparacao	8,217	121,70
unidade_de_controle	5,958	167,84

5.5 Resultados de Prototipagem

Esta seção inicialmente apresenta e justifica pequenas modificações que foram realizadas na arquitetura para fins de prototipagem. A seguir é apresentado o *software* utilizado para auxiliar na análise dos resultados da prototipagem. Por fim são realizadas algumas considerações sobre os resultados obtidos.

O protótipo gerado consiste no *kit* de desenvolvimento *Xilinx Spartan-II Evaluation Kit* conectado a PC através da porta paralela e este conectado à câmera *Take It 350* por um cabo USB. A Figura 5.12 apresenta o ambiente de desenvolvimento utilizado. Neste ambiente, o PC é responsável por enviar dados para o protótipo e ler os resultados. Os dados, em geral, são gerados pela câmera, porém, é possível utilizar arquivos de imagem pré-armazenados no PC.

FIGURA 5.12 – Ambiente de desenvolvimento.

A prototipagem foi realizada utilizando uma descrição em 4 *bits*, ou seja n=4. A motivação para isto está fundamentada em dois aspectos. O primeiro é que, de acordo com a Tabela 5.3, o dispositivo XC2S150 possui uma quantidade de blocos lógicos suficiente para prototipagem da arquitetura para estimação de movimento com 4 *bits*. O segundo é que a porta paralela do PC possui, no máximo, 12 sinais que podem ser utilizados para escrita, de modo que, com uma representação em 4 *bits*, é possível

representar os três sinais dos barramentos de entrada e saída simultaneamente. Para ter uma idéia do prejuízo visual da representação em 4 *bits* a Figura 5.13 apresenta uma comparação entre imagens representadas em 8 e em 4 *bits*.

FIGURA 5.13 – (a) Imagem representada em 8 *bits*. (b) A mesma imagem representada em 4 *bits*.(c) Apenas a luminância representada em 4 *bits*.

O *software* para auxiliar na validação emula a existência das memórias de quadro e de faixa, ou seja, os dados que ele escreve na porta paralela correspondem aos sinais `BES`, `BES0` e `BES1`. A Figura 5.14 apresenta os sinais de comunicação entre o protótipo e a porta paralela do PC. A leitura do valor do vetor de movimento é feita em duas etapas. Para isso foi criado o sinal `VM_HI_LO` de 3 *bits*. Este sinal é utilizado para passar para o PC o valor do sinal `VM` de 6 *bits*. Assim, o sinal `VM_HI_LO` apresenta ou os *bits* 2, 1 e 0 ou os *bits* 5, 4 e 3 do sinal `VM`, alternando a cada ciclo de processamento. O PC concatena estes *bits* para reconstituir o valor do sinal `VM`.

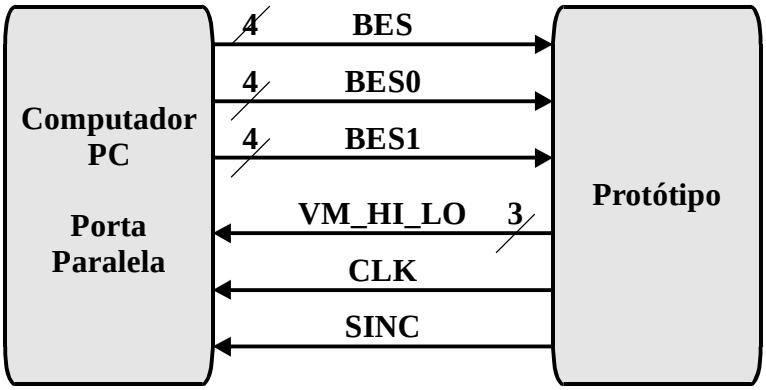

FIGURA 5.14 – Sinais de comunicação entre o PC e o protótipo.

Os sinais `CLK`, de relógio, e `SINC`, de sincronia, são gerados pelo protótipo e utilizados pelo PC para sincronia. O menor período possível para este sinal depende do atraso interno do FPGA e da taxa de atualização da porta paralela. No que depende do atraso interno do FPGA, a máxima freqüência de `CLK` seria de 33 MHz. Porém, a baixa taxa de atualização da porta paralela [ELS 2000] é o fator limitante. O *kit* possui um oscilador de 40 MHz que é utilizado como referência para gerar o sinal `CLK`. Constatou-se experimentalmente que o menor período para o sinal `CLK` com o qual o PC ainda garante a estabilidade dos dados na porta paralela é 102,37 μs, o que corresponde a uma freqüência de 9,76 KHz.

A ferramenta *WebPack* possui um recurso que permite mapear os sinais de entrada e saída de uma descrição em VHDL para os pinos do dispositivo FPGA associado ao projeto. Os pinos do FPGA utilizados para comunicação com o PC estão ligados ao conector do *kit* de desenvolvimento denominado JP5. A Tabela 5.5 apresenta

a maneira como os pinos do FPGA foram mapeados para o conector JP5 e estes aos pinos da porta paralela do PC. Cada pino da porta paralela possui um nome que está associado a sua função quando esta porta é utilizada conectada a uma impressora.

TABELA 5.5 – Mapeamentos dos sinais para conexão com a porta paralela.

PC		FPGA		
Pino da Porta Paralela	**Função**	**Pino do Conector JP5**	**Pino do FPGA**	**Função**
1	/ reg_ctl 0	41	P68	BES1(0)
2	reg_data 0	39	P63	BES(0)
3	reg_data 1	37	P61	BES(1)
4	reg_data 2	35	P59	BES(2)
5	reg_data 3	33	P57	BES(3)
6	reg_data 4	31	P44	BES0(0)
7	reg_data 5	29	P42	BES0(1)
8	reg_data 6	27	P37	BES0(2)
9	reg_data 7	25	P35	BES0(3)
10	reg_est 6	23	P33	VM_HI_LO(0)
11	/ reg_est 7	21	P30	VM_HI_LO(1)
12	reg_est 5	19	P27	VM_HI_LO(2)
13	reg_est 4	17	P23	CLK
14	/ reg_ctl 1	40	P67	BES1(1)
15	reg_est 3	38	P62	SINC
16	reg_ctl 2	36	P60	BES1(2)
17	/ reg_ctl 3	34	P58	BES1(3)
18 a 25	GND		-	-

A Figura 5.15 apresenta o *software* desenvolvido para auxiliar na validação do protótipo. A interface deste *software* apresenta dois campos de visualização de imagens e alguns botões de controle. O maior campo de visualização de imagens, à esquerda, apresenta o quadro de trabalho, ou seja, o retorno visual das operações realizadas. No outro campo, no canto superior direito, é apresentado o vídeo proveniente da câmera *Take It 350* em tempo real. No canto inferior direito encontram-se os botões de controle divididos em três grupos: os botões associados ao quadro de referência, os botões associados ao quadro de procura e os botões associados a ambos. Abaixo da imagem de trabalho existe uma caixa de mensagens onde são apresentadas confirmações das ações realizadas.

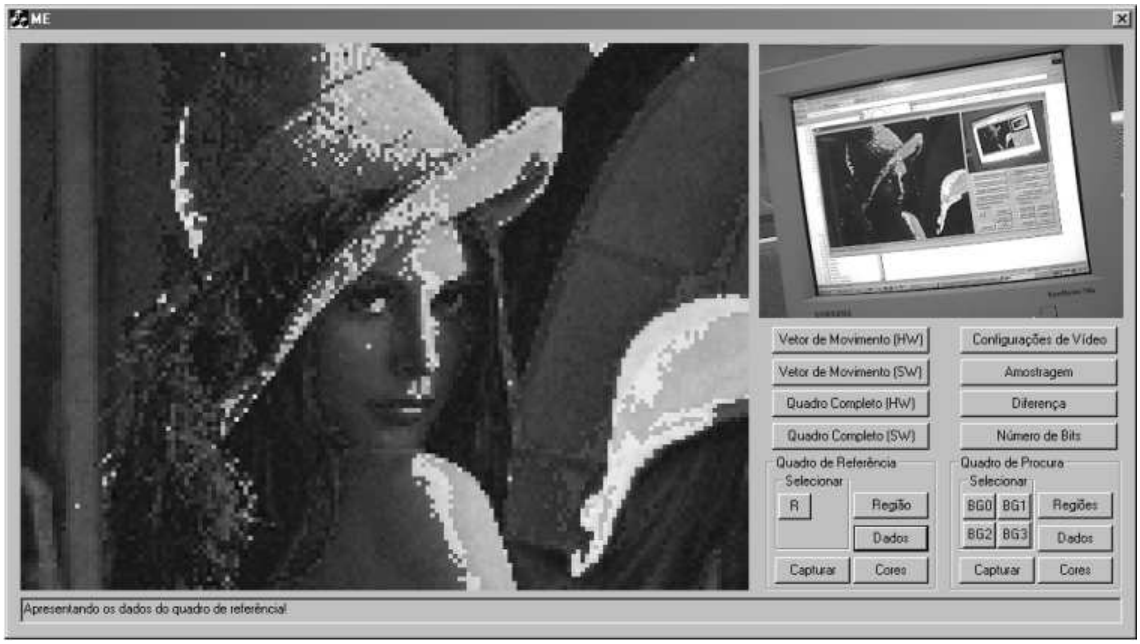

FIGURA 5.15 – *Software* desenvolvido para auxiliar na validação do protótipo.

O botão `Configurações de Vídeo` permite ajustar as configurações do vídeo de entrada. A configuração utilizada em todos os testes é resolução de 640x480 em 24 *bits* RGB.

O botão `Diferença` permite calcular a diferença ponto a ponto entre o quadro apresentado na área de trabalho e o quadro de procura, exibindo o resultado na imagem de trabalho. Esta funcionalidade permite ter uma idéia visual da diferença entre dois quadros. Ela é útil para avaliação dos vetores de movimento resultantes e também foi utilizada para construir as imagens apresentadas na Figura 2.6.

Os botões associados ao quadro de referência possuem correspondentes associados ao quadro de procura. Os dois botões `Capturar` são utilizados para capturar um quadro do vídeo ao vivo, guardando-o como quadro de referência ou quadro de procura, de acordo com o botão apertado. Os dois botões `Cores` são utilizados para visualizar os quadros capturados. Os dois botões `Dados` são utilizados para visualizar uma representação dos dados de luminância utilizados para o cálculo dos vetores de movimento.

O cálculo de um único vetor de movimento pode ser realizado em *software* ou em *hardware* utilizando os botões `Vetor de Movimento (SW)` ou `Vetor de Movimento (HW)`, respectivamente. O valor do vetor de movimento calculado é apresentado na caixa de mensagens. Para o cálculo do vetor de movimento em *software* são utilizados dados representados em 8 *bits*, enquanto para o cálculo em *hardware* os dados são representados em 4 *bits*, devido às restrições do protótipo. Os dados utilizados para o cálculo de um único vetor de movimento devem ser selecionados utilizando os botões `R`, `BG0`, `BG1`, `BG2` e `BG3`. Cada um destes botões é responsável por selecionar os dados que irão alimentar os barramentos de mesmo nome na arquitetura para estimação de movimento. O botão `Região` é utilizado para visualizar os dados que irão alimentar o barramento `R`, enquanto o botão `Regiões` é utilizado para visualizar os dados que irão alimentar os barramentos `BG0`, `BG1`, `BG2` e `BG3`.

O cálculo de todos os vetores de movimento referentes a um quadro pode ser realizado em *software* ou em *hardware* utilizando os botões `Quadro Completo (SW)` e `Quadro Completo (HW)`, respectivamente. Este cálculo é uma implementação cíclica do cálculo de um único vetor de movimento para todas as possíveis regiões `R` do quadro de referência, considerando as correspondentes regiões `BG0`, `BG1`, `BG2` e `BG3` do quadro de procura. Os dados para o cálculo em *software* são representados em 8 *bits*, enquanto para o cálculo em *hardware* são representados em 4 *bits*. O resultado, apresentado na imagem de trabalho, é uma representação de todos os vetores de movimento calculados. Os vetores de movimento são representados através de uma semi-reta que liga o canto superior esquerdo de cada região de referência considerada ao ponto final do vetor de movimento que é obtido através do cálculo realizado em *software* ou em *hardware*.

O botão `Amostragem` permite ajustar a taxa de sub-amostragem dos dados utilizados para cálculo do vetor de movimento. A sub-amostragem não havia sido originalmente prevista no trabalho, porém ela se mostrou fundamental durante a fase de desenvolvimento para permitir melhor visualização. Esta variável pode assumir os valores 1, 4 e 20 correspondentes às taxas de sub-amostragem 1:1, 1:16 e 1:400, respectivamente. Os valores 1, 4 e 20 foram escolhidos pois são divisores comuns de 480 e de 640. Na imagem de trabalho da Figura 5.15 são apresentados os dados utilizados para cálculo do vetor de movimento com taxa de sub-amostragem de 1:16.

No sentido de validar a arquitetura para estimação de movimento com o auxílio do *software* desenvolvido foram realizados alguns testes. Inicialmente, foram realizados os cálculos de um único vetor de movimento em *software* e em *hardware*; para tal, foram utilizados os botões `Vetor de Movimento (SW)` e `Vetor de Movimento (HW)`, respectivamente. Nos testes seguintes foram realizados os cálculos de todos os vetores de movimento para um quadro completo em *software* e em *hardware*; para tal, foram utilizados os botões `Quadro Completo (SW)` e `Quadro Completo (HW)`, respectivamente.

O cálculo de um único vetor de movimento é ilustrado na Figura 5.16. Este teste foi realizado com sub-amostragem de 1:400 para permitir que os resultados sejam observados de maneira intuitiva. Inicialmente, o quadro de referência apresentado na Figura 5.16a foi gerado a partir de um arquivo de imagem. O quadro de procura foi gerado a partir da edição do quadro de referência, embutindo um deslocamento de um ponto (em amostragem 1:400) para baixo e para a direita, o que pode ser observado através das faixas de coloração uniforme na parte superior e na parte esquerda da imagem, apresentada na Figura 5.16b. A seguir, a região de referência foi selecionada utilizando o botão `R` e a região de procura foi selecionada utilizando os botões `BG0`, `BG1`, `BG2` e `BG3`. A Figura 5.16c foi gerada utilizando o botão `Região` e apresenta a região de referência, ou seja, os dados que alimentam o barramento `R`. A Figura 5.16d foi gerada utilizando o botão `Regiões` e apresenta a região de procura, ou seja, os dados que alimentam os barramentos `BG0`, `BG1`, `BG2` e `BG3` da arquitetura para estimação de movimento.

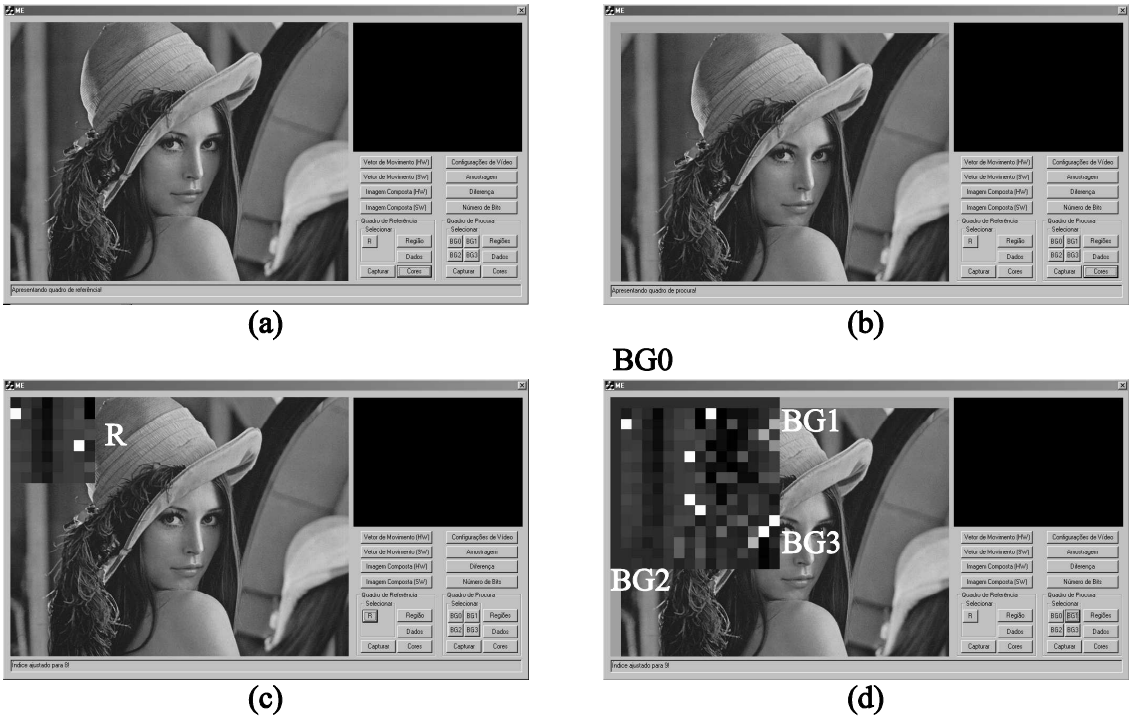

FIGURA 5.16 – Exemplo de cálculo de um vetor de movimento. (a) quadro de referência. (b) quadro de procura. (c) região de referência. (d) região de procura.

O resultado do cálculo do vetor de movimento para a situação apresentada na Figura 5.16 foi VM igual a 9. Isto significa que a 9ª hipótese de vetor de movimento considerada foi a que apresentou menor SAD. Para interpretar o valor de VM como um vetor bidimensional deve-se lembrar que as hipóteses de vetor de movimento são consideradas na ordem ilustrada pela Figura 5.17.

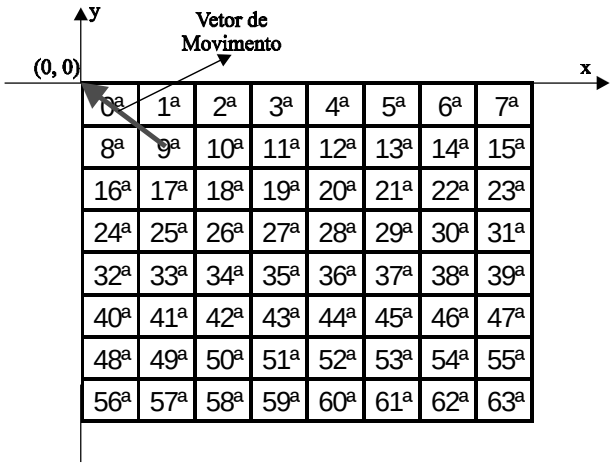

FIGURA 5.17 – Ordem em que as hipóteses de vetor de movimento são consideradas.

O vetor de movimento representa o movimento da parte da região de procura que possui menor SAD com relação à região de referência. Portanto o vetor de movimento tem origem em um ponto da região de procura e final no ponto (0, 0), conforme ilustra a Figura 5.17. O ponto de origem do vetor de movimento é dado em função do valor de VM pelo par ordenado:

$$(x, y) = \left(resto\left(VM/8\right), -\left\lfloor VM/8 \right\rfloor \right) \quad (5.1)$$

Utilizando a Equação 5.1, conclui-se que o valor de `VM` igual a 9 representa um vetor de movimento dado pelo par ordenado (1, –1). O segundo termo no lado direito da equação 5.1 refere-se ao arredondamento para o menor inteiro da razão VM/8, representado pelo símbolo $\lfloor VM/8 \rfloor$. Ou seja, trata-se de um deslocamento de 1 ponto (ou pixel) para a direita e 1 ponto para baixo, que foi o movimento artificialmente introduzido no quadro apresentado na Figura 5.16b.

Em uma extensão deste teste, ainda considerando a situação da Figura 5.16, foram utilizados os botões `Quadro Completo (SW)` e `Quadro Completo (HW)` para o cálculo de todos os vetores de movimento para aquele quadro de referência. A Figura 5.18 apresenta os vetores de movimento resultantes deste teste. Todos estes vetores de movimento referem-se a valores de `VM` igual a 9. Os resultados obtidos por *software* são os mesmos que os obtidos por *hardware*. A precisão dos resultados obtidos é atribuída à regularidade do movimento analisado neste teste.

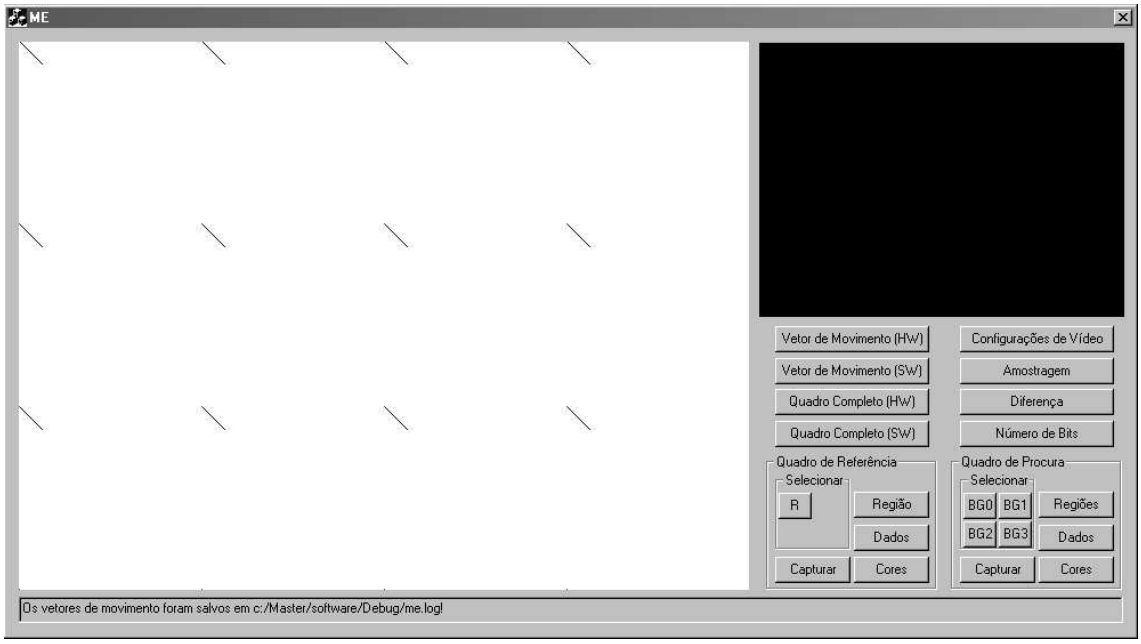

FIGURA 5.18 – Vetores de movimento obtidos para a situação da FIGURA 5.16.

Os demais testes realizados têm por objetivo exercitar a arquitetura desenvolvida com dados menos regulares, simulando situações reais de utilização. Foram consideradas quatro cenas denominadas C0, C1, C2 e C3. Estas cenas foram capturadas sob as seguintes condições:

- C0 – A câmera *Take It 350* foi utilizada para capturar uma cena real na qual o movimento foi gerado artificialmente, deslocando-se os objetos da cena. A diferença de tempo entre a captura dos dois quadros não é relevante, uma vez que o movimento foi produzido artificialmente.
- C1 – A câmera foi utilizada para capturar uma cena real na qual é apresentado o movimento natural do deslocamento de dois automóveis. A diferença de tempo entre a captura dos dois quadros é de 0,16 s, o que corresponde a um quadro à taxa de 6 qps.

- C2 – Esta cena foi capturada a partir de um vídeo em DVD e apresenta o movimento de translação de um satélite. A diferença de tempo entre a captura dos dois quadros é de 0,1 s, o que corresponde a 3 quadros à taxa de 30 qps.
- C3 – Esta cena foi capturada a partir de um vídeo em DVD e apresenta o movimento de uma pessoa falando. A diferença de tempo entre a captura dos dois quadros é de 0,03 s, o que corresponde a um quadro à taxa de 30 qps.

Nestes quatro testes com cenas reais não foi realizada sub-amostragem. Os vetores de movimento de valor nulo não são representados para melhor visualização dos resultados. Como as cenas provenientes do vídeo em DVD apresentam tarjas pretas acima e abaixo da imagem, os vetores de movimento relativos a estas regiões foram desconsiderados nas avaliações realizadas.

Os resultados dos testes são apresentados de forma padronizada conforme ilustrado na Figura 5.19.

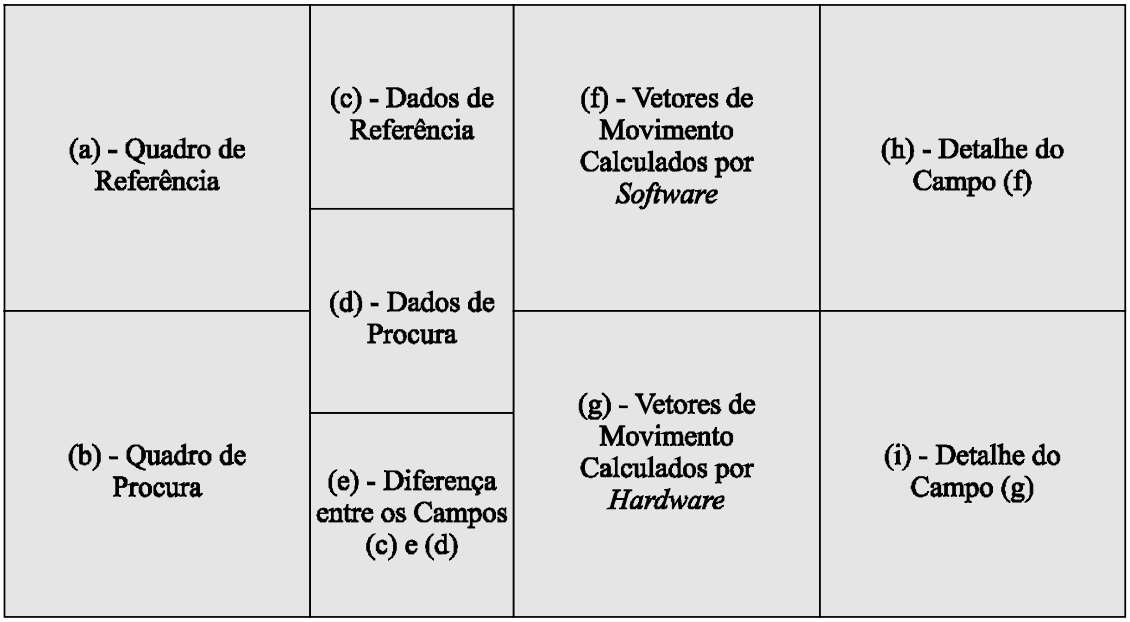

FIGURA 5.19 – Formato de apresentação do resultado dos testes.

Para as quatro cenas foi realizado o mesmo procedimento de teste. Neste procedimento, inicialmente são capturados os quadros de referência (campo (a)) e de procura (campo (b)) através dos botões `Captura`. A seguir, utilizando os botões `Dados`, são observados os dados utilizados como referência (campo (c)) e os dados utilizados para procura (campo (d)). No passo seguinte, utilizando o botão `Diferença`, é observada a diferença ponto a ponto entre o quadro de referência e o quadro de procura (campo (e)). Por fim são calculados os vetores de movimento em *software* (campos (f) e (h)) e em *hardware* (campos (g) e (i)) utilizando os botões `Quadro Completo (SW)` e `Quadro Completo (HW)`, respectivamente.

O procedimento de teste inclui ainda uma análise que mensura a diferença entre os resultados obtidos pelo cálculo em *software* e em *hardware*. A partir do valor de `VM` foi calculado o valor das componentes do vetor de movimento em cada direção utilizando a Equação 5.1 para todos os vetores do quadro. A diferença absoluta média entre as componentes dos vetores em cada direção calculados por *software* (representação de 8 *bits* por ponto) e por *hardware* (representação de 4 *bits* por ponto) foi acumulada e ao

final calculada a média. O resultado é expresso em percentual do tamanho da região de referência na direção considerada (8 pontos para ambas).

A Figura 5.20 apresenta o resultado do teste com a cena C0. A maior parte dos vetores de movimento observados neste resultado estão inscritos no contorno dos objetos da cena. Nota-se a redução da quantidade de vetores de movimento nas regiões internas aos objetos em que não houve variação de cor. Ainda é possível notar a presença de vetores de movimento nas regiões onde houve variação gradual da luminosidade, como os cantos e sombras da cena. Nos campos (h) e (i) da Figura 5.20 é possível perceber uma pequena diferença entre os vetores de movimento calculados por *software* e por *hardware*. Esta diferença é de 14,5% na direção vertical e 15,4% na direção horizontal.

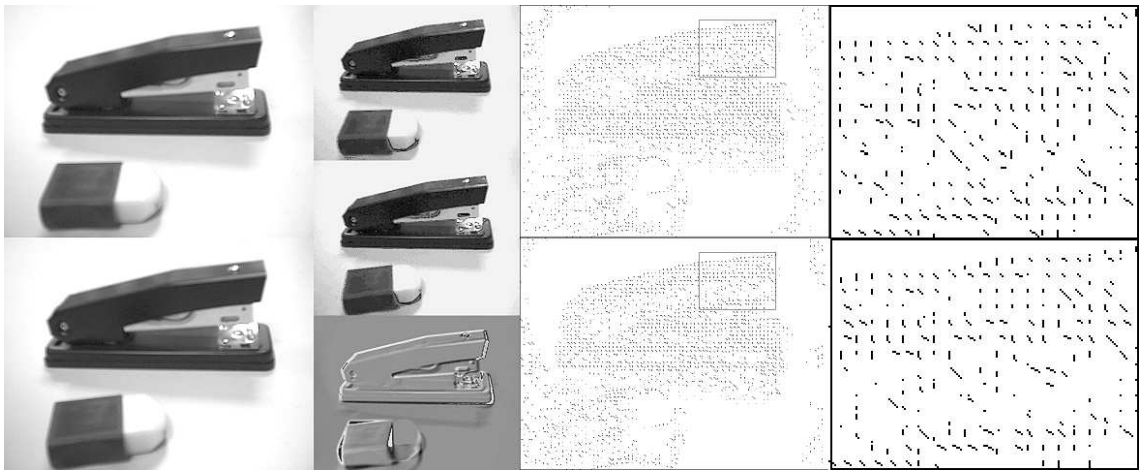

FIGURA 5.20 – Resultado do teste com a cena C0.

A Figura 5.21 apresenta o resultado do teste com a cena C1. Antes de analisar os vetores de movimento é importante observar através da diferença entre os quadros (campo (e)) que houve uma pequena movimentação da câmera no momento da captura. Esta movimentação implica na existência de vetores de movimento de pequena amplitude no contorno de alguns elementos da cena, principalmente nos postes e ao longo da faixa central da cena, composta por edificações e árvores. O resultado do teste identificou o movimento realizado pelo automóvel à esquerda, como pode-se observar nos detalhes apresentados nos campos (h) e (i) da Figura 5.21. Já o movimento do automóvel à direita não está explicito nos resultados devido à sua amplitude, que foge aos limites da região de procura. No céu não são observados vetores de movimento devido à sua uniformidade, porém, no asfalto, de pigmentação menos uniforme, é observada uma concentração moderada de vetores de movimento. A diferença entre os vetores de movimento calculados por *software* e por *hardware* é de 7,6% na direção vertical e 6,3% na direção horizontal.

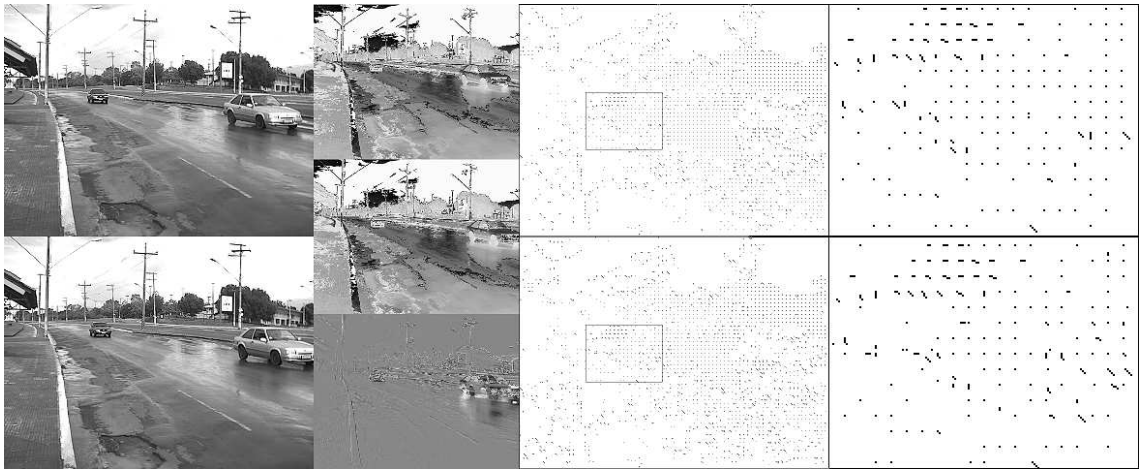

FIGURA 5.21 – Resultado do teste com a cena C1.

A Figura 5.22 apresenta o resultado do teste com a cena C2. Nestes resultados observa-se uma concentração moderada de vetores de movimento de pequena amplitude nas regiões referentes à movimentação do planeta ao fundo. Na parte redonda do satélite observa-se uma redução do número de vetores de movimento devido à baixa luminosidade desta região, que faz com que uma grande região apareça uniformemente preta. Na parte retangular do satélite os vetores de movimento não possuem um comportamento definido dada a grande variação de cores e luminosidade desta região. Em uma análise qualitativa dos detalhes apresentados nos campos (h) e (i) da Figura 5.22 é possível perceber que os vetores de movimento calculados por *software* refletem melhor o movimento da região próxima à borda do satélite do que aqueles vetores calculados por *hardware*. A diferença entre os vetores de movimento calculados por *software* e por *hardware* é de 15,7% na direção vertical e 9,1% na direção horizontal.

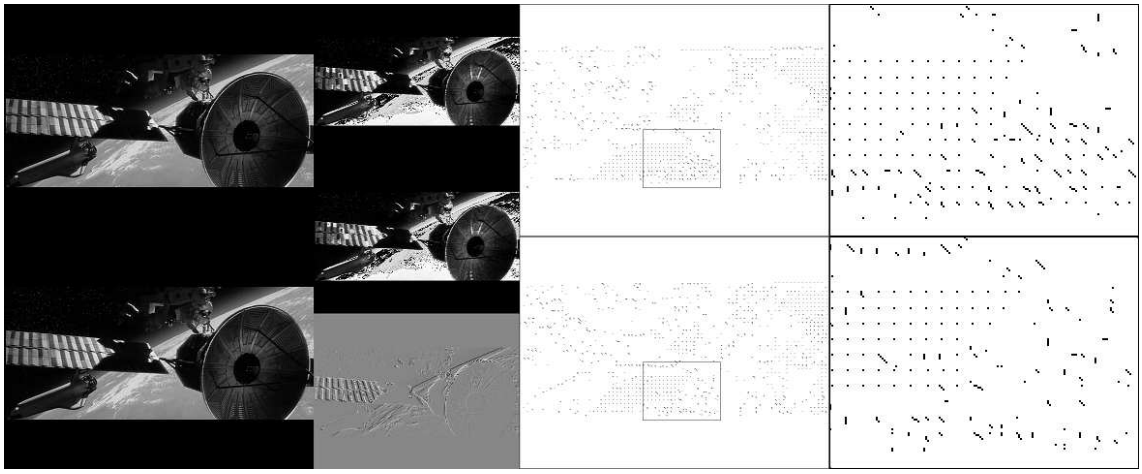

FIGURA 5.22 – Resultado do teste com a cena C2.

A Figura 5.23 apresenta o resultado do teste com a cena C3. Neste resultado observa-se uma concentração moderada a alta de vetores de movimento nos lados direito e esquerdo da pessoa que está falando. Estes vetores, embora intuitivamente não esperados, correspondem a uma pequena movimentação de câmera, que, devido à existência de grades e de um fundo não uniforme, implicou na detecção de tais vetores. Na região da cena relativa ao queixo da pessoa também é observada uma concentração de vetores de movimento que correspondem às diferenças observadas no campo (e) da

Figura 5.23. Estes vetores refletem o movimento de abertura da boca para falar realizado pela pessoa, conforme pode ser observado nos detalhes apresentados nos campos (h) e (i) da Figura 5.23. Ainda observou-se neste teste que no resultado obtido por *hardware* foi detectada menor quantidade de vetores de movimento nas zonas escuras da cena quanto comparado com o resultado obtido por *software*. Isto ocorreu porque os 4 *bits* utilizados no *hardware* não são suficientes para representar as pequenas diferenças de cor observadas na cena. Isto implicou em um pequeno aumento da diferença entre os vetores de movimento calculados por *software* e por *hardware* que ficou em 19,1% na direção vertical e 8,3% na direção horizontal.

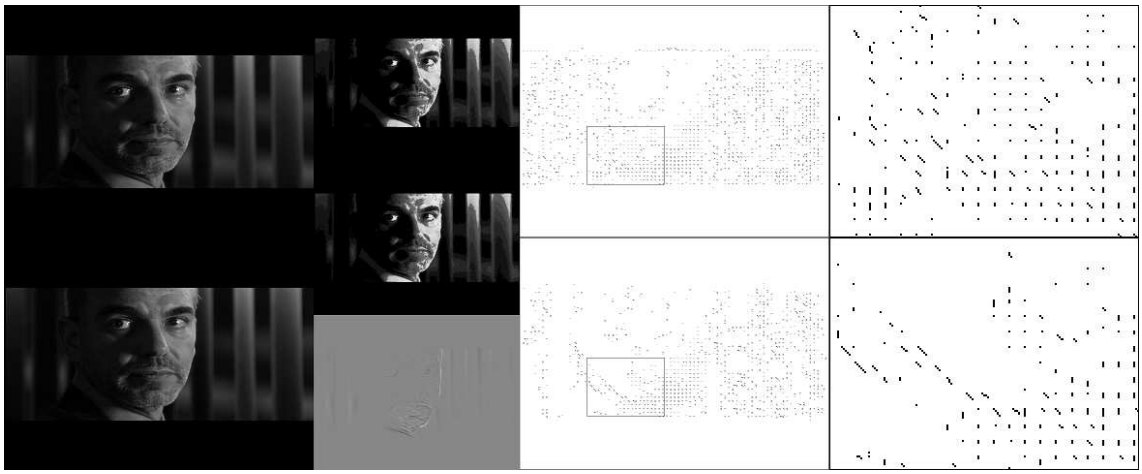

FIGURA 5.23 – Resultado do teste com a cena C3.

Qualitativamente os testes realizados são importantes no sentido de avaliar a diferença entre os vetores de movimento calculados pelo protótipo, cujos dados são representados em 4 *bits*, e os vetores calculados por *software*, cujos dados são representados em 8 *bits*. Neste sentido, uma comparação visual mostra que os resultados obtidos por *hardware* são muito similares àqueles obtidos por *software*.

A Tabela 5.6 apresenta um resumo da comparação quantitativa realizada para os resultados obtidos com cada cena. Esta tabela mostra que existem diferenças entre os resultados obtidos com a versão em *software* e em *hardware*. Estas diferenças ocorrem devido ao diferente número de *bits* utilizados para a representação de dados nestas duas versões. Embora estas diferenças sejam significativas, deve-se lembrar que o *layout* do resultado obtido por *hardware* é muito similar àquele obtido por *software*, o que mostra que em ambos resultados foi identificado o mesmo padrão de movimento. Vale lembrar que a decisão de representar os dados em 4 *bits* foi tomada em função da quantidade restrita de blocos lógicos no dispositivo FPGA XC2S150. Utilizando-se um dispositivo FPGA de maior capacidade, a arquitetura pode ser implementada com representação de dados em 8 *bits*. Neste caso, de acordo com as simulações realizadas, os resultadas obtidos por *software* ou por *hardware* seriam exatamente iguais.

Tabela 5.6 – Diferenças entre os resultados obtidos por *software* e por *hardware*.

	C0	**C1**	**C2**	**C3**
Vertical	14,5%	7,6%	15,7%	19,1%
Horizontal	15,4%	6,3%	9,1%	8,3%

O protótipo, operando à freqüência de 9,76 KHz por limitações da porta paralela utilizada, necessita de 31,47 s para o cálculo dos 4800 vetores de movimento referentes a um quadro completo na resolução 640x480 pontos. A implementação em *software*, rodando em um processador Pentium III 733 MHz, necessita em média 1,8 s para realizar esta mesma tarefa, um tempo 17 vezes menor. Esta diferença evidencia o grande impacto na performance do protótipo causado pela comunicação através da porta paralela. Na hipótese de que a comunicação não implicasse em limitações na freqüência de trabalho do protótipo, este poderia rodar a até 33 MHz, de acordo com as simulações realizadas. Neste caso, o cálculo de todos os vetores de movimento para um quadro seria realizado em 9,3 ms, um tempo 193 vezes menor do que aquele obtido com o Pentium III.

A arquitetura desenvolvida opera à taxa de 64 operações de acumulação por ciclo de processamento. Considerando a freqüência de 733 MHz, o PC opera, em média, à taxa de 0,02 acumulações por ciclo de processamento ou 37,28 ciclos para cada acumulação. Assim, a arquitetura desenvolvida apresenta uma performance em termos de taxa de operações por ciclo de processamento 2386 vezes maior que o PC.

Nas comparações de tempo de execução e taxa de operações deve-se considerar que, no caso do PC, o processamento concorre com outros processos, como a carga da imagem, sistema operacional, entre outros. Ainda assim, esta comparação é considerada justa uma vez que em qualquer situação real de utilização do PC estas tarefas seriam concorrentes.

6 Conclusão

6.1 Análise do Trabalho Realizado

Neste trabalho foi desenvolvida uma arquitetura de *hardware* para o cálculo dos vetores de movimento utilizados em compressão de vídeo. Esta arquitetura é composta por uma interface de E/S, uma matriz de processamento com 64 elementos de processamento, uma unidade de comparação e uma unidade de controle. A arquitetura para estimação de movimento foi descrita em linguagem VHDL. Nesta descrição foi definido um parâmetro que permite configurar o número de *bits* utilizados para representação dos dados. Foi gerado um protótipo da arquitetura para dados representados em 4 *bits* em um *kit* de desenvolvimento baseado no dispositivo FPGA XC2S150 da Xilinx. Para validação, além da simulação, foi desenvolvido um *software* que roda em plataforma PC capaz de alimentar o protótipo com dados e de ler o valor dos vetores de movimento utilizando a porta paralela para comunicação.

O *software* desenvolvido se mostrou um importante instrumento para validação da arquitetura, pois permite a visualização dos vetores de movimento, possibilitando uma interpretação visual do resultado. A comprovação do correto funcionamento da arquitetura foi feita com base em valores de entrada constantes, quando através da simulação, ou com base em imagens reais, quando utilizando o *software* desenvolvido. Os resultados obtidos foram satisfatórios, uma vez que os vetores de movimento obtidos através da simulação são iguais aos valores manualmente calculados e os vetores de movimento obtidos utilizando o protótipo são iguais aos valores obtidos utilizando o *software* (quando ambos operam com o mesmo número de *bits*).

A freqüência máxima de trabalho da arquitetura para estimação de movimento implementada no dispositivo FPGA utilizado neste trabalho é de 33 MHz. Porém, em função das limitações impostas pela comunicação através da porta paralela do PC, a freqüência máxima de trabalho do protótipo é de 9,76 KHz. Assim, o protótipo deve ser entendido como uma prova de conceito feita em baixa freqüência, realizada sem otimização da interface entre o PC e o protótipo. Considerando o vídeo de entrada no formato 640x480 e a arquitetura trabalhando a 33 MHz, os vetores de movimento são calculados à taxa de 107,32 quadros por segundo, ou um quadro a cada 9,3 ms. Estendendo estes resultados para outros formatos de vídeo, obtém-se as taxas apresentadas na Tabela 6.1. Nesta tabela percebe-se que para a resolução 1024x768 a arquitetura é capaz de manter uma taxa superior a 30 quadros por segundo, que é a taxa utilizada em televisão.

TABELA 6.1 – Taxa de quadros por segundo para diferentes formatos de vídeo.

Formato de Vídeo	Resolução	Máxima Taxa de Quadros por Segundo
CIF	352x288	325,52
VGA	640x480	107,42
SVGA	800x600	68,75
SVGA	1024x768	41,96
SXVGA	2048x1536	10,49

A latência total, ou seja, o número de ciclos de processamento antes da apresentação do primeiro vetor de movimento para a arquitetura completa é de 312.512 ciclos. Destes, 307.200 ciclos são gastos para escrita das memórias de quadro, 5120 ciclos para escrita nas memórias de faixa, 128 ciclos para escrita nas memórias de procura e de referência e 64 ciclos para inicializar o *pipeline* da matriz de processamento. Para a arquitetura utilizada na prototipagem, isto é, sem as memórias de quadro e de faixa, a latência é de 192 ciclos, sendo 128 ciclos gastos para escrita nas memórias de referência e de procura e 64 ciclos gastos para a inicialização do *pipeline* da matriz de processamento. Após apresentado o primeiro vetor de movimento, a arquitetura gasta 64 ciclos para o cálculo de cada vetor de movimento. Tanto a latência quanto a freqüência de trabalho não são influenciadas pelo número de *bits* de representação dos dados.

A arquitetura utilizada para prototipagem, na qual os dados são representados em 4 *bits*, utiliza 1228 blocos lógicos do FPGA, o que corresponde a 71,1% dos 1728 blocos lógicos disponíveis no dispositivo XC2S150. Caso o protótipo fosse gerado para dados representados em 8 *bits*, ele utilizaria 1918 blocos lógicos, o que corresponderia a 110,0% dos blocos lógicos disponíveis. Independentemente do número de *bits* com que os dados são representados, a arquitetura ainda utiliza 10 blocos de memória, o que corresponde a 83,3% dos 12 blocos disponíveis no dispositivo XC2S150. O protótipo utiliza 16 pinos de entrada e saída, incluindo o sinal de relógio. Caso o protótipo fosse gerado para dados representados em 8 *bits*, ele utilizaria 31 pinos.

A Tabela 6.2 apresenta uma comparação de desempenho entre as arquiteturas apresentadas na Seção 3.4 e a arquitetura desenvolvida. Esta tabela apresenta a taxa de quadros por segundo em valores absolutos calculados considerando os tamanhos de imagem para os quais as arquiteturas foram projetadas, e também em valores normalizados para a resolução de 640x480, região de referência 8x8 e região de procura 16x16. Assim, as arquiteturas STi3220 e EST256 podem ser consideradas arquiteturas de alto desempenho e a arquitetura desenvolvida neste trabalho se mostrou comparável a estas.

TABELA 6.2 – Comparação de desempenho da arquitetura desenvolvida com outras arquiteturas.

Arquitetura	Referência	Máx. Freq. de Trabalho (MHz)	Taxa de Operações (GOps)	Valores Absolutos		Valores Normalizados para 640x480	
				qps	ms	qps	ms
STi3220	[QUE 2001]	18,25	2,65	44,01	22,72	237,65	4,21
Arquitetura Baseada no *Clustering* de Regiões	[FUJ 97]	15,00	0,12	18,49	54,08	6,10	163,89
EST256	[SAN 98]	20,00	11,00	48,22	20,74	260,39	3,84
Conjunto Linear de 8 Elementos de Processamento	[ZAN 2001]	8,32	0,06	3,38	295,86	3,38	295,86
Arquitetura Desenvolvida	-	33,00	2,11	107,32	9,32	107,32	9,32

A estratégia adotada para distribuição de dados para os elementos de processamento, além de garantir a sua plena utilização, reduz a largura da entrada de dados. Considerando dados representados em 8 *bits*, para alimentar diretamente todos os elementos de processamento seria necessário um barramento de 1024 *bits*, enquanto a entrada de dados implementada é de 40 *bits*. A utilização de dois níveis de memória na interface de E/S potencializa o acesso a mais dados simultaneamente, a simplificação do endereçamento das memórias de procura e o uso de memórias de procura de pequena capacidade, e que portanto podem ser implementadas em FPGA.

6.2 Trabalhos Futuros

Alguns trabalhos futuros de baixa complexidade consistem em variações baseadas na arquitetura desenvolvida neste trabalho.

O primeiro trabalho consiste na utilização de múltiplas instâncias da arquitetura desenvolvida para busca sobre regiões de procura maiores, já que a arquitetura utilizada para prototipagem pode ser diretamente instanciada para suportar tais regiões de procura. Para tal seriam necessárias modificações apenas na parte da interface de entrada e saída, que não foi implementada no protótipo. O número de instâncias deve ser k^2 para k igual a 1, 2, 3, etc. O tamanho da região de procura pode ser expresso em função de k por (k*8+7)x(k*8+7). A possibilidade de instanciação é apresentada conceitualmente na Figura 6.1 para k=2, isto é, são utilizadas quatro instâncias, o tamanho da região de referência permanece em 8x8 e o tamanho da uma região de procura é 23x23. Cada instância deve receber através de seus barramentos globais parte da região de procura 23x23. Vale notar que a zona morta de uma instância é considerada por outra instância, exceto pelas instâncias que recebem dados pertencentes à zona morta da região de procura 23x23.

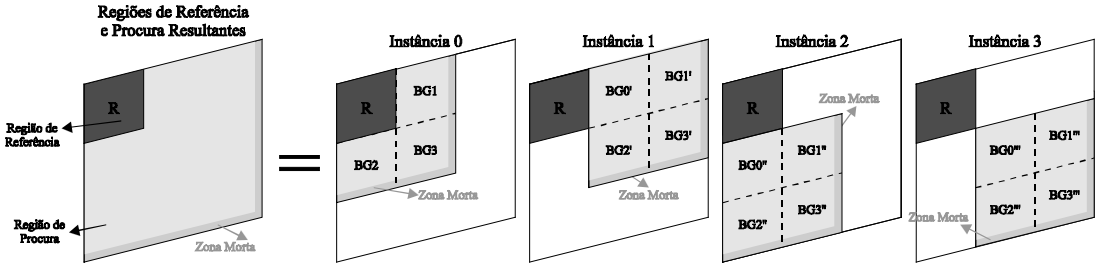

FIGURA 6.1 – Regiões de referência e procura para a utilização de múltiplas instâncias da arquitetura desenvolvida.

O segundo trabalho consiste na utilização da arquitetura desenvolvida para implementar outros algoritmos, como por exemplo os de busca hierárquica, circular ou logarítmica. A sub-amostragem implementada no *software* para auxiliar na validação mostra que é possível implementar o algoritmo de busca hierárquica utilizando a arquitetura desenvolvida. Para tal seria necessário implementar, junto à interface de E/S, uma unidade de sub-amostragem.

O terceiro trabalho consiste na implementação da arquitetura desenvolvida utilizando uma arquitetura alternativa para o elemento de processamento. A Figura 6.2 apresenta uma proposta de arquitetura alternativa. Externamente, a arquitetura alternativa possui os mesmos sinais que a arquitetura de elemento de processamento apresentada na Figura 4.9. Existem duas principais diferenças entre estas arquiteturas: a primeira é que a arquitetura alternativa necessita de dois ciclos de processamento para cálculo de cada SAD, e a segunda é que a arquitetura alternativa suporta apenas sinais representados em 4 *bits*. Estas modificações visam a redução do número de elementos lógicos de um elemento de processamento, o que se reflete na redução da área de silício necessária para sua implementação.

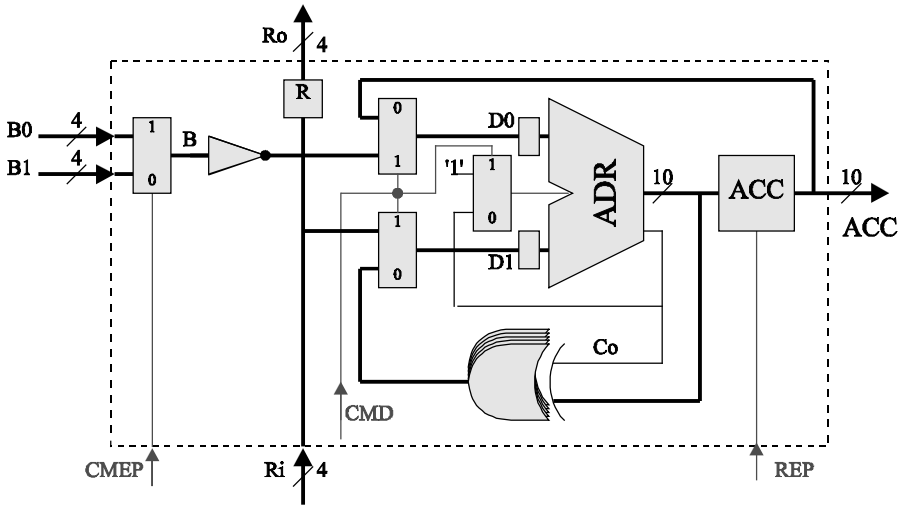

FIGURA 6.2 – Arquitetura alternativa de elemento de processamento.

A arquitetura alternativa de elemento de processamento, apresentada na Figura 6.2, utiliza apenas um somador denominado `ADR`. Dois multiplexadores são utilizados para selecionar os sinais `D0` e `D1`, entradas do somador `ADR`. Estes multiplexadores são comandados pelo sinal de controle `CMD` que também alimenta a entrada `Ci`, *carry in* do somador `ADR`.

Na arquitetura alternativa, a acumulação de uma diferença absoluta é realizada em duas etapas. Na primeira etapa, o somador ADR calcula a diferença entre os sinais Ri e B. Para tal, o sinal CMD vale ‘1’ de modo que D0 recebe o valor de Ri, D1 recebe o valor de B e a entrada Ci do somador ADR recebe o valor ‘1’. Na segunda etapa este somador é utilizado para incrementar o registrador ACC com o valor absoluto da diferença calculada na primeira etapa. Para tal, o sinal CMD vale ‘0’, de modo que D0 recebe o valor de ACC, D1 recebe o resultado da diferença calculada na primeira etapa (invertido, se for o caso) e Ci recebe o valor de *carry* gerado na primeira etapa.

O trabalho futuro mais complexo, continuidade natural deste trabalho, é a integração da arquitetura desenvolvida com uma arquitetura de *hardware* para compressão de vídeo. Inicialmente, para que a arquitetura desenvolvida possa ser independente de outras plataformas como o PC, é necessário sua integração com memórias externas e com um sensor de imagem. A partir desta integração ainda será necessária a integração com outras arquiteturas para realizar as demais tarefas da compressão de vídeo, que são essencialmente o cálculo da matriz de erro, a compressão de imagens estáticas e a composição do *bitstream*. Possíveis componentes para este trabalho são a memória UT6164C32Q-6 da Utron e o sensor de imagem OV7411 da Omnivision. A vantagem da utilização de sensores de imagem está no fato de que a leitura de dados é simples e não é necessária a implementação de um protocolo de comunicação, como o USB.

Referências

[ACK 97] ACKLAND, Bryan. VLSI Architectures for Multimedia and Video Conferencing. In: INT. SYMP. ON CIRCUITS AND SYSTEMS, 1997, Hong Kong. **Tutorials …** [S.l.]: IEEE Press, 1997. p. 147-163.

[AGO 2002] AGOSTINI, Luciano V. **Projeto de Arquiteturas Integradas para a Compressão de Imagens JPEG.** 2002. 143f. Dissertação (Mestrado em Ciência da Computação) – UFRGS, Porto Alegre.

[BHA 99] BHASKARAN, Vasudev; KONSTANTINIDES, Konstantinides. **Image and Video Compression Standards:** algorithms and architectures. 2nd ed. Massachusetts: Kluwer Academic, 1999. 454p.

[BIO 2002] BIOTEMAS. **Olho Humano:** Visão. [S.l.], 2002. Disponível em: <http://intermega.globo.com/biotemas/olhohumano.htm>. Acesso em: 18 jul. 2002.

[CHY 97] CHAN, Yui-Lam; SIU, Wan-Chi. Block Motion Vector Estimation Using Edge Matching: an Approach With Better Frame Quality as Compared to Full Search Algorithm. In: INT. SYMP. ON CIRCUITS AND SYSTEMS, 1997, Hong Kong. **Proceedings …** [S.l.]: IEEE Press, 1997. p. 1145-1148.

[CHU 97] CHU, Chung-Tao; ANASTASSIOU, Dimitris; CHANG, Shih-Fu. Hierarchical Global Motion Estimation/Compensation in Low Bitrate Video Coding. In: INT. SYMP. ON CIRCUITS AND SYSTEMS, 1997, Hong Kong. **Proceedings …** [S.l.]: IEEE Press, 1997. p. 1149-1152.

[ELS 2000] ELSON, Jeremy. **Parapin:** a Parallel Port Pin Programming Library for Linux. Marina del Rey: University of Southern California, 2000. Disponíel em: <http://www.circlemud.org/pub/jelson>. Acesso em: 18 jul. 2002.

[ELY 95] ELY, S. R. MPEG Video Coding: a Simple Introduction. **EDU Technical Review**, [S.l.], 1995. Disponível em: <http://www.ebu.ch/trev_266-ely.pdf>. Acesso em: 18 jul. 2002.

[FUJ 97] FUJITA, Gen; ONOYE, Takao; SHIRAKAWA, Isao. A New Motion Estimation Core Dedicated to H.263 Video Coding. In: INT. SYMP. ON CIRCUITS AND SYSTEMS, 1997, Hong Kong. **Proceedings …** [S.l.]: IEEE Press, 1997. p. 1161-1164.

[GAJ 97] GAJSKY, Daniel D. **Principles of Digital Design.** New Jersey: Prentice-Hall, 1997. 468p.

[GON 92] GONZALEZ, Rafael C.; WOODS, Richard E. **Digital Image Processing.** Massachusetts: Addison-Wesley, 1992. 716p.

[HOF 2001] HOFFMANN, Gustavo A. et al. The binDCT Processor. In: WORKSHOP IBERCHIP, 7., 2001, Montevideo, Uruguai. **[Memórias].** Montevideo, Uruguai: Universidad de la Republica, 2001. 1 CD-ROM.

[INT 94] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO/IEC 13818 – JTC1/SC29/WG11:** Generic Coding of Moving Pictures and Associated Audio. Singapura, 1994.

[MAT 2002] THE MATWORKS, INC. **Matlab 5.3.** Disponível em: <http://www.matworks.com>. Acesso em: 11 jul. 2002.

[MIT 2001] MITCHELL, Joan L. et al. **MPEG Video Compression Standard.** Massachusetts: KAP, 2001. 467p.

[MEN 2002] MENTOR, INC. **ModelSim PE/SE.** Portland: Mentor Graphics Inc., 2002. Disponível em: <http://www.model.com/products/modelsim_pe_se.asp>. Acesso em: 14 dez. 2002.

[MOV 2000] MOVING PICTURE EXPERTS GROUP. **The MPEG Home Page.** [S.l.], 2000. Disponível em: <http://mpeg.telecomitalialab.com>. Acesso em: 18 jul. 2002.

[OPP 99] OPPENHEIM, Alan V.; SCHAFER, Ronald W. **Discrete-time signal processing.** 2nd ed. New Jersey: Prentice-Hall, 1999. 870p.

[QUE 2001] QUEROL, Marc. **STI3220:** Motion Estimation Processor Codec. [S.l.]: SGS-Thomson Microelectronics, 2001. Disponível em: <http://www.st.com/stonline/books/pdf/docs/1648.pdf>. Acesso em: 18 jul. 2002.

[SAN 98] SANZ, César; GARRIDO, Matías J.; MENESES, Juan M. VLSI Architecture for Motion Estimation Using the Block-Matching Algorithm. In: AUTOMATION AND TEST EUROPE CONF., 1998, Paris. **Proceedings …** [S.l.: s.n.], 1998. p. 45-49.

[SAV 2002] SAVATIER, Tristan. **MPEG Pointers and Resources.** [S.l.]: MPEG-TV, 2002. Disponível em: <http://www.mpeg.org>. Acesso em: 18 jul. 2002.

[SEO 99] SEO, Young San; YOU, Jae Hee. VLSI Design of Hierarchical Search Motion Estimation Processor Chip. In: ASIA-PACIFIC CONF. ON ASIC, 1999, Seoul, Coréia. **Proceedings …** [S.l.]: IEEE Press, 1999.

[SHI 2000] SHI, Yun Q.; SUN, H. **Image and Video Compression for Multimedia Engineering:** fundamentals, algorithms and standards. United State: CRC Press, 2000. 480p.

[SON 2001] SONY SEMICOND. CO. OF AMERICA. **CXD1922Q:** MPEG-2 Video Encoder. San Jose, 2001. Disponível em: <http://www.sel.sony.com/semi/PDF/CXD1922Q.pdf>. Acesso em: 18 jul. 2002.

[STR 2001] STREAM MACHINE CO. **CS92288:** MPEG-2 Audio/Video codec. Milpitas, 2001. Disponível em: <http://www.cirrus.com/pubs/cs92288prodbull.pdf?DocumentID=883>. Acesso em: 18 jul. 2002.

[TOU 99] TOURAPIS Alexis M.; AU, Oscar C.; LIOU, Ming L. Fast Motion Estimation Using Circular Zonal Search. In: VISUAL COMMUNICATIONS AND IMAGE PROCESSING, 1999, San Jose. **Proceedings …** [S.l.: s.n.], 1999. v. 02, p. 1496-1504.

[TOU 2001] TOURAPIS Alexis M.; AU, Oscar C.; LIOU, M. L. Predictive Motion Vector Field Adaptive Search Technique (PMVFAST) Enhancing Block Based Motion Estimation. In: VISUAL COMMUNICATIONS AND IMAGE PROCESSING, 2001, San Jose. **Proceedings …** [S.l.: s.n.], 2001.

[XIL 2002] XILINX, INC. **Xilinx Spartan II Data Sheet.** San Jose: Xilinx Inc., 2002. Disponível em: <http://www.xilinx.com/partinfo/ds001.pdf>. Acesso em: 18 jul. 2002.

[ZAN 2001] ZANDONAI, Diogo et al. An architecture for MPEG motion estimation. In: WORKSHOP IBERCHIP, 7., 2001, Montevideo, Uruguai. **[Memórias].** Montevideo, Uruguai: Universidad de la Republica, 2001. 1 CD-ROM.